Faculteit Wetenschappen
Departement Informatica

Applying Machine Learning in Business Process Monitoring

Proefschrift

voorgelegd tot het behalen van de graad van
Doctor in de Wetenschappen: Informatica
aan de Universiteit Antwerpen
te verdedigen door

Stephen PAUWELS

Promotor: prof. dr. Toon Calders

Antwerpen, 2022

Applying Machine Learning in Business Process Monitoring
Nederlandse titel: *Toepassen van Machinaal Leren bij het Monitoren van Bedrijfsprocessen*

If you always do what you've always done, you'll always get what you've always got.

— Henry Ford

Acknowledgements

I compare the journey of finishing my Ph.D. with the journey Frodo had to take to Morder, but only this time I did not have to destroy anything but rather create a scientific contribution. This journey was not always easy, especially the beginning of my quest was a hard one. It was only after winning the BPI Challenge in 2018 that things really started to move forward. But just like Frodo, I had people who supported me along the way. I would not have been able to complete this thesis without them.

A special thank goes out to my supervisor prof. dr. Toon Calders, more than six years age our journey together started. First with my weekly visits to you at the Université Libre de Bruxelles and starting from 2016 full-time in Antwerp. Although not having a choice with whom I would do my research, I am very happy and grateful to have had the chance to work with you for so many years both on research and education. I like the way you care about all your students (even the non-Ph.D. candidates), and the amount of time you are willing to spend with them every week. Thank you, Toon, for the infinite amount of patience you showed and the occasional psychological therapies.

I would also like to thank the members of the jury; dr. Annalisa Appice, dr. Boudewijn Van Dongen, dr. Jochen De Weerdt, dr. Bart Goethals and dr. Floris Geerts. Your helpful suggestions and remarks together with some interesting (short) discussions have ensured that this thesis became a work on which I am very proud.

If the pandemic taught me one thing, then it is that colleagues are an important part of a job. I had the luxury of working with some wonderful people at Adrem Data Lab. Len, to say that we had a strange (but dynamic) working relationship would be an understatement. Arguing with you about various topics made some of our lunch breaks legendary. Joeri, you were given the hard task of sharing an office with me. It was a pleasure of working in the same office, talking and complaining about students when we needed it, or sitting all day in silence listening to our own music. It were big shoes to fill, but Ewoenam, even though we did not work often together in the office due to known reasons, it was a pleasure of having you in the office. And of course, I would also like to thank all other members: Boris, Daphne, Hassaan, Jan, Joey, Koen, Lien, Marco, Mozghan, Nikolaj, Olivier, Robin, Sam, Sandy and the entire team at Biomina.

The computational resources and services used in Chapter 3 were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government – department EWI. I would like to thank Franky for his support and patience when I was using the supercomputer.

Thanks to Timo Nolle, Alexander Seeliger, and all others who made the BPM Conferences memorable. Especially Timo, as we worked on the same topic and were direct “competitors”, our discussions in various hotel bars were always interesting and fun to do.

I would also like to thank all authors of methods used for providing access to their code. In special we would like to thank Kristof Bohmer, Farbod Taymouri, and Liu Lin for providing extra information that helped us clarify some implementation details.

Getting the opportunity to start working as a Ph.D. student would not have been possible without the group of friends I met here at the University of Antwerp during my first year studying Computer Science: Yannick, Timothy, Tom, Tom, Alexander, Maarten, and Mats. Thanks for having this group dynamic of pushing each other further every time and keeping challenging each other. Without you all, this would not have been possible.

Thank you also to my parents and family, as they provided me with the opportunity to go to university (even though I always said that I was not going to). And especially thanks to my godson Finn, even though you do not realize it (yet), seeing you, playing, and interacting with you has given me so much energy over the past three years. Last but certainly not least, I would like to thank my girlfriend, Marianna. Your support for what I was doing was crucial. Thanks for allowing me to attend various conferences (all the way to Sydney) and seeing something from the world, while at the same time you had to stay home and work. The freedom and opportunities you give me are just some of the things that make you a fantastic girlfriend, and soon, a wonderful mother for our little girl.

Thank you all!

Stephen Pauwels
Antwerpen, 2022

Contents

Publications

Journals

- **Stephen Pauwels** and Toon Calders. Detecting anomalies in hybrid business process logs. In *ACM SIGAPP Applied Computing Review 19.2*, pages 18–30, 2019.
- **Stephen Pauwels** and Toon Calders. Evaluating Next Activity Predictions: What Could Go Wrong? *Submitted*, 2022.

Conferences

- **Stephen Pauwels** and Toon Calders. An anomaly detection technique for business processes based on extended dynamic bayesian networks. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (ACM SAC '19)*, pages 494–501, 2019.
- **Stephen Pauwels** and Toon Calders. Bayesian network based predictions of business processes. In *International Conference on Business Process Management (BPM Forum '20)*, pages 159–175, 2020.
- **Stephen Pauwels** and Toon Calders. Incremental Predictive Process Monitoring: The Next Activity Case. In *International Conference on Business Process Management (BPM '21)*, pages 123–140, 2021. *Best Student Paper Award.*

Workshops, Demos & Prizes

- **Stephen Pauwels** and Toon Calders. Mining multi-dimensional complex log data. Benelearn, 2016.
- **Stephen Pauwels** and Toon Calders. Detecting and explaining drifts in yearly grant applications. In *Business Process Intelligence Challenge (BPI Challenge '18)*, winner in the Academic category, 2018.
- **Stephen Pauwels** and Toon Calders. ACD2: A Tool to interactively Explore Business Process Logs. In *BPM (PhD/Demos)*, pages 129–133, 2019.

Code & Software

- **Stephen Pauwels**. Github repository - Bayesian Networks in Business Processess. `https://github.com/StephenPauwels/edbn`
- **Stephen Pauwels**, Armando Alliu, and Toon Calders. ACD2 - Software Tool `https://adrem.uantwerpen.be/conceptdrift`, 2019

CHAPTER 1

Introduction

1.1 Business Process Management

1.1.1 History

Processes are everywhere around us. They are an important aspect in different organizations, from big international businesses to governmental and local organizations. Yet, it is only recently that processes are considered to be an important part of these businesses. Businesses used to focus mainly on the end product itself, neglecting the difficulties in setting up and improving the various processes leading to this end product. Often a *best effort* approach regarding all processes within their organization was used, but as Elon Musk has stated: "*Creating a manufacturing process is more difficult than creating the product*"[1].

This lack of process awareness can be explained by looking at the history and evolution of how products are being produced or services are being delivered. Starting in prehistoric times, humans needed to support themselves or the small group they lived in. They had to produce their own food, their own tools, and everything they needed to survive. People needed to have all the knowledge about all the items or services they needed. Or to put it differently; knowledge about all aspects of all processes was a necessity to survive. Later, when cities started to arise, people started to be more specialized and work was divided between different craftsmen. People did not have to know all the skills required to survive, as people were able to buy food at the market without having to go hunting themselves. Growing crops and trading food had become two of the specializations. People living in this age had a very good understanding of the entire process they were involved in. With the

[1] `https://everydayastronaut.com/starbase-tour-and-interview-with-elon-musk/`, consulted on 14/02/2022

Industrial Revolution, however, this shifted further towards a form of pure specialization. Workers would only be involved in a single step of the entire process. To coordinate all these different parts of the processes the role of *manager* emerged. These managers are not necessarily masters in their craft; they rather look at how to improve and optimize how the job is done, using the resources available to them.

With work getting more and more divided, organizations started to be structured more using the principle of labor division. Workers with a similar focus on a part of the production process were grouped in different work units, each having its manager(s). These units were further organized hierarchically in different groups. For example, units were placed under departments, departments under faculties, etc. The advantage of this way of working is that people can have a large amount of knowledge about the small part of the entire process they are working on. One of the downsides was that optimizing a production process was often limited to only optimizing the work of a single department or work unit. To be able to analyze how the entire end-to-end process works, more powerful tools, and analysis were needed. Often Real performance gain can be achieved only when optimizing the processes over the boundaries of the work units. Example 1 shows that to achieve true improvements, changes have to be made to the overall process and organization, and not only within a single work unit.

Example 1. *During the 1980s, Ford acquired a big financial stake in Mazda. After visiting the sites of Mazda, one of the main things observed by the Ford representatives was how understaffed their work units seemed in comparison to their own. Yet, these units operated perfectly normally. This was studied by Michael Hammer [44] who focused on the purchasing process within the two companies.*

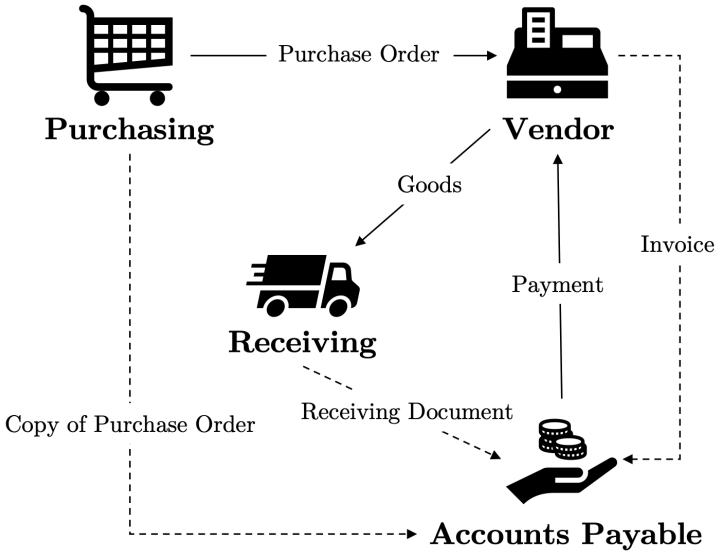


Figure 1.1: Representation of the purchasing process as used by Ford as shown by Mazda as shown by Hammer [44].

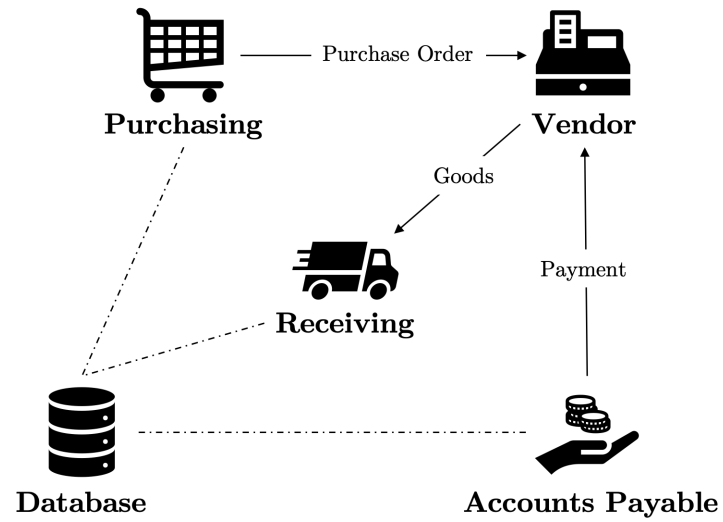


Figure 1.2: Representation of the purchasing process as used by Mazda as shown by Hammer [44].

Every purchase that was made by Ford needed to go through the purchasing department; they would send the order to the vendor and a copy to the accounting department. After the goods were delivered to the Ford warehouse, the shipping notice was passed on to the accounting department. Besides that, the vendor would also send an invoice to the accounting department. The task of the accounting department was to check the consistency between all these documents. Numerous discrepancies were discovered and sorting these out occupied several hundred people within Ford. An overview of this process is given in Figure 1.1. In contrast, at Mazda, only 5 people worked in this department. The big difference was that instead of detecting and resolving discrepancies one by one, Mazda avoided these discrepancies.

At Mazda, they used a central database to store all information on the purchases. This database was then used by the purchasing department to store all information about the order. When the goods arrived, the people at the warehouse could immediately check if the delivered goods matched the ordered goods. If this was not the case, they would simply not accept the goods. Only when the goods matched the order, the warehouse would accept and the accounting department would only need to pay those orders which were accepted. This process is depicted in Figure 1.2.

The transition to better monitoring and being able to optimize processes within organizations came with the availability of more powerful IT systems such as Enterprise Resource Planning (ERP) and Workflow Management Systems (WfMSs). These systems later became known as Business Process Management Systems (BPMSs). Nowadays we see even more advanced tools, algorithms, and insights becoming available that can help organizations improve their processes. These improvements range from improving the efficiëncy of the overall process (reducing the manufacturing time) to lower the cost of producing a product to improving the general quality of the end product. With organizations starting to actively embrace Business Process

Management, it becomes more and more clear that process-oriented organizations perform better than non-process-oriented ones [68].

1.1.2 Business Processes

A nice and understandable definition of a business process was given by Marlon Dumas[2]:

A Business Process is a chain of events, activities, and decisions
involving a number of actors and objects
triggered by a need and leading to an outcome
that is of value to a customer

This definition covers all different aspects of a business process. A business process consists of activities performed by some workers, (automated) events that get raised to which the workers need to react, and decisions workers need to make based on the information they have. Every process has workers or automated systems that perform the above-mentioned activities. Example 2 describes some typical types of processes within an organization.

Example 2. *Some typical examples of processes:*

- ***Quote-to-order:*** *When receiving the request for a quote, a company will start this process which ends in the customer either placing an order or rejecting the quote.*
- ***Order-to-cash:*** *When ordering something the customer sets in motion an entire process that ensures that she gets the goods and that the store receives the payment for the goods. Typical activities within this kind of processes are* order received, shipping, sending an invoice, payment received. *Together with the* Quote-to-order *process this combination is often called the* Quote-to-cash *process.*
- ***Issue-to-resolution:*** *When a customer raises a problem or issue, such as a malfunctioning piece of hardware that was recently delivered, this process is started. The process stops when the customer and, preferably, the organization agree that the issue has been resolved.*

To be able to talk about, create, analyze, and improve processes, it is important that we can write them down as accurately as possible. Different models exist, each with its benefits and difficulties. Some examples of these modelling techniques are *Transition Systems*, *Petri Nets*, *Workflow Nets*, *Causal Nets*, *YAWL* and *Business Process Modeling Notation (BPMN)* [108]. By using these models we can more easily detect problems within the process or try to optimize the different processes within an organization. Another advantage of these formal models such as Petri Nets and Transition Systems is that there exists a solid theoretical background with techniques that can be applied to detect problems within the process. These models can also

[2] Business Process Mining Course - Lecture 1: Introduction - `https://www.youtube.com/watch?v=LrEYPx0Rqxk`

Table 1.1: Extract from an event log from the Helpdesk dataset [115]. Events are grouped by case.

Timestamp	Case	Activity	Resource
2012/10/09 13:50:17	Case1	Assign seriousness	Value 1
2012/10/09 13:51:01	Case1	Take in charge ticket	Value 1
2012/10/12 14:02:56	Case1	Take in charge ticket	Value 2
2012/10/25 10:54:26	Case1	Resolve ticket	Value 1
2012/11/09 11:54:39	Case1	Closed	Value 3
2012/04/03 07:55:38	Case2	Assign seriousness	Value 4
2012/04/03 07:55:53	Case2	Take in charge ticket	Value 4
2012/04/05 08:15:52	Case2	Resolve ticket	Value 4
2012/05/19 08:00:28	Case2	Closed	Value 5
2012/04/03 12:08:32	Case5	Assign seriousness	Value 9
2012/04/03 12:45:33	Case5	Take in charge ticket	Value 10
2012/04/03 12:47:22	Case5	Resolve ticket	Value 10
2012/04/03 13:15:02	Case5	Take in charge ticket	Value 10
2012/04/03 15:07:28	Case5	Resolve ticket	Value 10
2012/05/25 12:10:28	Case5	Closed	Value 5
⋮	⋮	⋮	⋮

be used to check if instances of the processes are following the correct steps; we call this conformance checking.

The events produced by a running process are often captured in an event log. This log records several different features for every executed event within the process. Three features always have to be present in an event log to correctly reflect all instances of a given process:

- **Timestamp:** When was the event executed?
- **Activity:** Which action was executed?
- **Case:** To which instance of the process does this event belong?

Extra information, such as the worker executing the event or the department responsible for the event, can also be recorded in the event log. Some event logs additionally keep track of when an activity was started and completed. This enables us to analyze both the time needed to complete a task and the idle time between two tasks. An example event log with one extra feature, an anonymized resource executing the activity, can be found in Table 1.1.

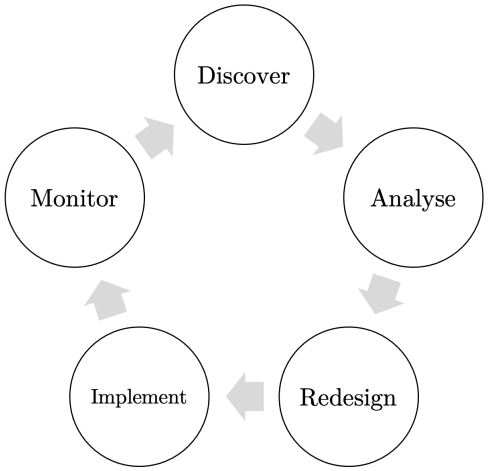


Figure 1.3: The Lifecycle of a Process.

We can describe all steps involved in describing, analyzing, and improving processes using the BPM lifecycle, as shown in Figure 1.3. The different stages are:

- **Discover** Based on an event log we can create a process model that captures the behavior seen in the log. The earliest example of such an algorithm is the α-algorithm [106]. This algorithm takes an event log as input and outputs a Workflow net that can replay (a subset of) cases within the event log. This Petri net is constructed based on particular patterns that occur in the event log. For example, if a certain activity A is followed by B, but B is never followed by A, then we can assume that there is a causal dependency between A and B.

- **Analyse** After obtaining models from the process as they are currently implemented and being executed, we can start identifying and analyzing the issues within the process during the *Process Analysis* step. Such an issue can be that the *cycle time* (the time needed to execute the entire process) is too high and has to be improved.

- **Redesign** When we have identified potential issues we can move to the step of *Process Redesign*, in which we search for possible remedies and analyze these proposed changes to the process. When solving known issues we have to make sure that we do not introduce new issues into the process.

- **Implement** Once a solution has been found and is approved, we move to the *Process Implementation* phase. During this step, we make the actual changes to how the process is executed, by updating the workflows used by the workers and by adapting the IT systems of the organization.

- **Monitor** When a process has been updated and is running, we have to keep an eye on detecting new issues and ensuring that the process is running as intended, we call this step *Process Monitoring*. In this thesis, we are mainly interested in this last step.

Detecting and analyzing event logs is called *Process Mining*. The Process Mining Manifesto [107] gives a valuable overview of important characteristics and possible challenges of applying process mining. One of the most important aspects, when we want to use process mining to get more insights into the execution of the processes, is the systematical and detailed gathering of information in the event logs. Using wrong, incomplete, or noisy data leads to models and insights of lower quality.

Another challenge that is described in [107] is that of concept drift, i.e. how to update, adjust, and extend models for a process that is dynamic and is likely to change during its lifetime.

A useful model should be easy to read and understandable (*simplicity*), yet contain enough detail to allow for a correct representation of the model (*precision*). Often a trade-off has to be made between simplicity and precision. A model that is too simple can be too general, and will thus allow for too many different variations. On the other hand, a model that is too precise will be overfitted for the data used to train the model and will not be useful for analyzing new data.

1.1.3 Process Monitoring

To ensure that the implemented process is running as intended and results in a satisfied customer, it is important to keep track of some key metrics such as *cycle time, customer satisfaction, resource utilization, ...*. To get these insights, we capture the data generated by the execution of the business processes in event logs. We then use these event logs to verify the conformance with the intended process execution, organization policies, or regulations.

There are two main types of process monitoring techniques: *Offline Process Monitoring* and *Online Process Monitoring*. Offline monitoring uses the available historical data to analyze the performance of the process. Using offline techniques we can get an overview of the performance of the process, reasons for good or bad performance, or compliance with the process. Online monitoring is concerned with the assessment of the performance of currently running processes. We want to be able to predict as soon as possible if a certain execution will end in a negative outcome, or if the completion will be delayed. When the online analysis detects one of these possibilities, alarms or triggers are generated. These alarms should give the workers or automated system the possibility of detecting the potential problem and correcting it.

In this thesis we focus on three specific aspects of process monitoring: *Predictive Monitoring, Concept Drift Detection*, and *Anomaly Detection*.

Predictive Monitoring is a purely online technique in which we want to predict what the next steps within the execution are. This is useful to proactively detect bottlenecks before they even occur and take the appropriate actions to prevent this from happening. We can also use this technique to already allocate some resources that are needed to perform the next activity/activities.

Anomaly Detection on the other hand, takes the known or expected behavior into account and tries to indicate which cases are showing abnormal behavior and should therefore be investigated. This technique can be applied both in offline and online situations. Using it in the offline mode gives insights in how well the process execution followed the expected steps, while the online mode gives workers and systems the ability to correct anomalies as quickly as possible.

Detecting Concept Drift is typically an offline monitoring technique in which we want to check if there were any changes to the process. When changes are detected we also want to know if they were caused by accident or are more fundamental within the process, thus indicating if a new analysis and possible redesign of the process is needed. We are also interested in concept drift in an online situation, in which we want to detect drifts to know when to update the models used.

1.2 Structure of the Thesis

In this section, we give an overview of our contributions and the structure of this thesis.

1.2.1 Predictive Monitoring

In this thesis, we are interested in predicting the next activities, given a sequence of completed events. We do so by calculating the probabilities for all possible activities, as shown in Figure 1.4. In this figure, we have a partially completed sequence of events *A*, *B*, *C* and we want to predict the next event. In **Chapter 3** we introduce the use of Dynamic Bayesian Networks (DBNs) for predicting the next activity in a running instance of a business process. A Bayesian Network is a graphical way of representing a probability distribution. This distribution is built using the different correlations between variables that are present in the data. A Dynamic Bayesian Network provides more insights to the user on why a certain activity was predicted in a sequence of events. We show that our model performs on par with current state-of-the-art methods while having lower runtimes for learning. We add extra functionality to the DBNs to make sure they can handle unseen situations. The results in Chapter 3 were published as *Stephen Pauwels and Toon Calders. Bayesian network based*

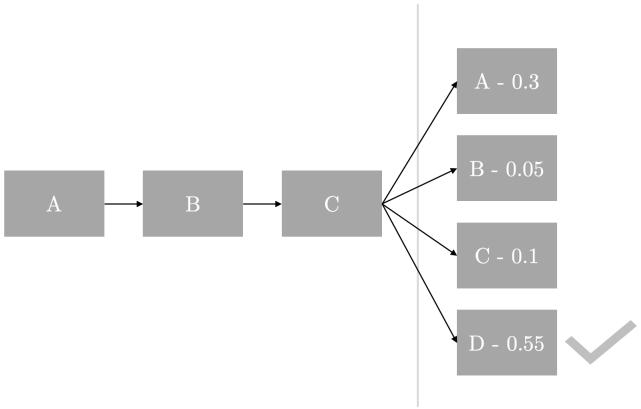


Figure 1.4: After observing A, B and C, the model predicts the probability for every possible next activity (A, B, C or D). Here D has the highest probability and is selected as the best candidate.

predictions of business processes in International Conference on Business Process Management (BPM Forum '20), pages 159–175, 2020..

Most prediction methods are created for a static setting; the model gets trained on a fixed part of the data and tested on the other part. Real-life processes, however, can change slightly or significantly over time. In **Chapter 4** we explore the different possibilities of how to update existing models. We show how to update our proposed DBN method and take a look at how neural networks can be updated after they have been trained. We investigate the difference between using a window-based and a Concept Drift-based approach.

Current state-of-the-art neural network methods require some time to learn their models. To better investigate the incremental capabilities of neural networks, we created an easy baseline network that can be learned much faster than existing methods. The network only consists of basic layers, in contrast to the Long-Short-Term-Memory or Convolution layers used by state-of-the-art methods. We show that this model performs on par or even outperforms existing methods in both static and dynamic settings. The results in Chapter 4 were published as *Stephen Pauwels and Toon Calders. Incremental predictive process monitoring: The next activity case in International Conference on Business Process Management, pages 123–140. Springer, 2021*

While performing the experiments for the two previous chapters, we came across multiple inconsistencies between the accuracy numbers reported in the various papers and the numbers we were able to reproduce. This led us to investigate the experimental setups used in the various papers. We saw that different settings are used between papers. Although all papers use their own, unique, experimental setup, most papers copy/paste results from other papers to compare their results with. Therefore, in **Chapter 5** we take a close look at these different experimental setups and perform a series of experiments to show the impact some of the settings have on the obtained accuracy. To be able to compare different existing methods, we implemented a uniform testbed that can be used by authors to compare their new methods with existing work. The use of such a uniform test environment reduces the risks of comparing apples with oranges which is important, as we show that small differences in the evaluation setup can lead to slightly different conclusions when comparing different methods.

1.2.2 Anomaly Detection

In **Chapter 6** we extend the Dynamic Bayesian Network method to use its full potential of being able of giving the probability for every single variable in the model. Using these probabilities we can calculate scores for individual attributes, the full event, and the total case. These scores can then be used to rank all cases based on how likely they are to be anomalous. Using this ranking-based approach, workers can concentrate their efforts on those cases which have the biggest anomaly. Figure 1.5 shows how we use the same principles as in Chapter 3 to detect anomalies. In this example, we use the calculated probabilities to determine how likely it is for event B to occur in the given case.

In business process event logs, some features have much stronger relations than only the conditional dependencies captured by the DBN. For example, a given value of the parent attribute may imply a certain value for the child attribute. To be

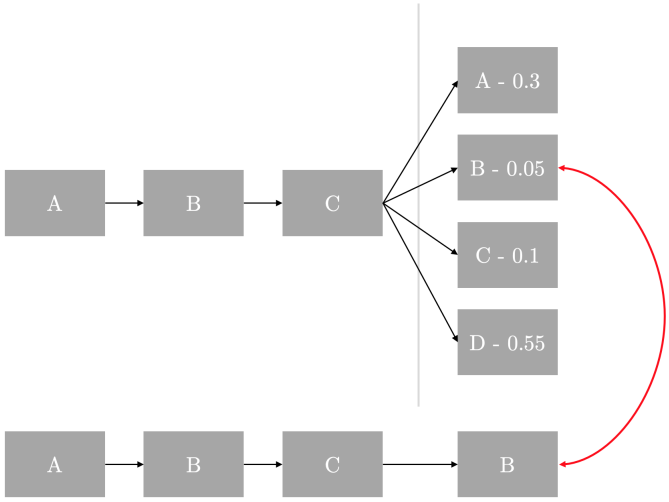


Figure 1.5: After activities A, B and C are executed, we see that the predicted probability for B as next activity equals 0.05. Adding this probability to the overall probability for the case can cause it to fall below a certain anomaly-threshold, indicating that this case (and specifically the execution of activity B) is likely an anomaly.

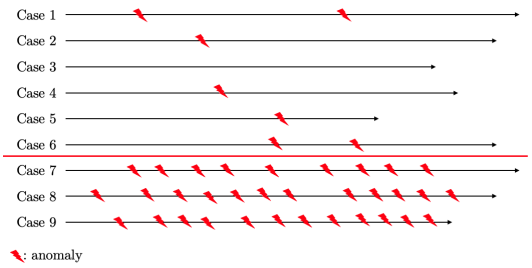
able to better make use of this property, we add Functional Dependencies to the original dynamic Bayesian Networks. A Functional Dependency exists between two attributes when one attribute completely determines the other attribute.

We show that our model can indicate what attribute is responsible for the high anomaly score. This reduces the amount of data that has to be analyzed manually.

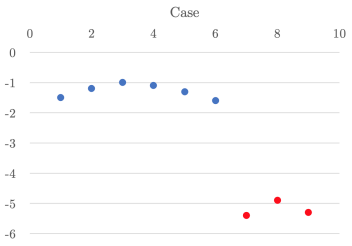
In **Chapter 6** we explain how we learn the different dependencies, and show how to calculate the anomaly score for event logs containing both categorical and numerical attributes. We use Kernel Density Estimators to extend the use of conditional probabilities to numerical attributes. The results in chapter 6 were published as *Stephen Pauwels and Toon Calders. An anomaly detection technique for business processes based on extended dynamic bayesian networks in Proceedings of the 2019 ACM Symposium on Applied Computing, 2019* and *Stephen Pauwels and Toon Calders. Detecting anomalies in hybrid business process logs in ACM SIGAPP Applied Computing Review, 19(2):18–30, 2019*

1.2.3 Concept Drift

Generating an anomaly score based on historical data is the same as assigning a score that indicates how normal the given case and events are, compared to a reference window. In **Chapter 7** we test our existing model in a different scenario; Concept Drift Detection and Explaining. We do this by utilizing the BPI Challenge 2018 dataset, which was part of the Business Process Intelligence workshop organized together with the BPM conference. We explain the different drifts occurring in the event log generated by yearly grant applications. Figure 1.6 illustrates why we can use anomaly detection for concept drift detection. Drift can be detected when



(a) Representation of the anomalies found per case.



(b) Overview of the overall anomaly score for every case in the event log. These scores are used for detecting drifts bases on their statistical properties.

Figure 1.6: Detecting concept drift based on the amount of anomalies found per case. When most cases only contain anomalies, we can assume that drift has occurred (as can been seen starting from case 7).

multiple anomalies occur in almost all new cases, as we can see from case 7. In **Chapter 7** we use our extended dynamic Bayesian network to detect and explain drifts in event logs. One of the benefits of our model is that we can easily change the perspective that we want to investigate.

Normally we try to detect drifts within the activity perspective, but we could also use our model to search for drifts within the resource perspective or to investigate the difference between departments. In **Chapter 8** we introduce a tool we created using the anomaly and concept drift detection techniques from the last chapters. We show the possible benefits of the methods we introduced by incorporating them in a visual and intuitive web application.

A complete overview of the structure of this thesis can be found in Figure 1.7.

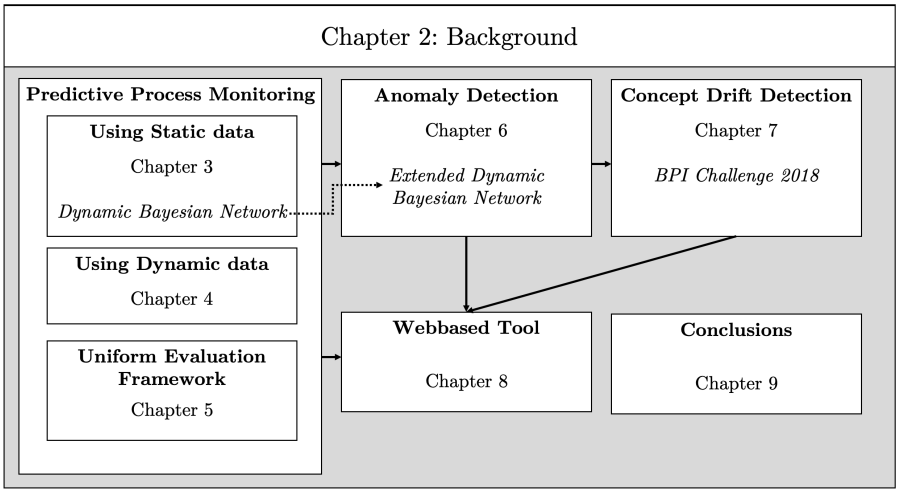


Figure 1.7: Structure of the Thesis.

1.2.4 Contributions

The contributions of this thesis are:

- **Predictive Process Monitoring**
  - We introduce the use of Dynamic Bayesian Networks for Process Monitoring.
  - We extend DBNs to be able to predict the next activity in a running case.
  - We propose general strategies to allow incremental updates to prediction models.
  - We propose a simple neural network that performs at par with existing, more complicated models.
  - We look at how evaluations are performed in other next activity prediction papers and compare them and show the differences that can occur when a different experimental setup is used.
  - We performed elaborate comparative studies for the next activity prediction case.
- **Anomaly Detection**
  - We extend the DBN to be able to score events based on how anomalous they are.
  - We extend the DBN to be able to handle both categorical (e.g. activity) and numerical (e.g. duration) attributes.
- **Concept Drift Detection**
  - We show that our model can be used for explainable concept drift detection.
  - We created a web tool, that showcases the various possibilities of the model.

CHAPTER 2

Preliminaries

Before we dive into the deep, we introduce some of the fundamental aspects we use throughout the entire thesis. In this chapter, we introduce what an event log is, how we transform this event log into a prefix-based log, and we introduce existing machine learning techniques that deal with the same or equivalent problem settings as our thesis. Finally, we introduce the Dynamic Bayesian Networks which play a major role in our proposed methods.

2.1 Business Process Logs

A Business Process is a series of ordered structured events that are required to perform a certain task [116]. Such a sequence of events that together form a single instantiation of a business process is called a *case*. Running processes are monitored by logging the events in an *event log*. Every event in this event log can contain different attributes. Such attributes are the activity, timestamp, duration, resource, documents used, department, sensor data, ... The attributes can be either categorical or numerical. In general, event logs indicate which events belong together in the same case. If not, we can apply a clustering algorithm as described in [80] for identifying and grouping different events into cases. In the remainder of this thesis, we assume that all events belong to a case and contain a case identifier.

We formally introduce the concepts of *an event*, *a case*, and *an event log* as follows:

Definition 1. An event $e = (eID, act, t, data)$ *consists of a unique identifier eID, an associated activity act, a timestamp t, and a tuple data* $= (a_0, \ldots, a_n)$ *denoting the values of (possible) extra attributes* $\mathscr{A} = (A_0, \ldots, A_n)$*, an ordered set of attributes. We denote the ordered set containing only the categorical attributes as* $\mathscr{A}^c$*, and the numerical attributes as* $\mathscr{A}^n$.

Definition 2. A case $c = (cID, [e_1, e_2, \ldots, e_{m_c}])$ *has a unique identifier cID and consists of an ordered list of events. The order in the list is determined by the timestamp of the events. We use* m_c *to denote the number of events present in case c.*

Table 2.1: Example event log. Events are ordered by timestamp. cID denotes the case identifier, events with the same cID together form a case. The axtra attributes are the ID of the user performing the activity (*UID*), the name of the user (*UName*), and the role of the user (*URole*).

eID	Activity	Time	Type	UID	UName	URole	cID
0	Log in	0	System	001	Joe	employee	1
1	Logged in	1	System	001	Joe	employee	1
2	Create	1	Request	001	Joe	employee	1
3	Send Mail	2	Request	001	Joe	employee	1
4	Log in	3	System	002	Linda	manager	2
5	Logged in	4	System	002	Linda	manager	2
6	Create	6	Request	001	Joe	employee	2
7	Send Mail	7	Request	001	Joe	employee	2
8	Disapproved	8	Request	002	Linda	manager	2
9	Log in	9	System	003	Marc	employee	3
10	Logged in	10	System	003	Marc	employee	3
11	Create	10	Request	003	Marc	employee	3
12	Approved	11	Request	002	Linda	manager	1
13	Send Mail	12	Request	003	Marc	employee	3
14	Approved	17	Request	004	Bob	sales	3
15	Log in	18	System	001	Joe	manager	4
16	Logged in	19	System	001	Joe	manager	4
17	Create	20	Request	001	Joe	manager	4
18	Approved	21	Request	001	Joe	manager	4
19	Send Mail	21	Request	001	Joe	manager	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Definition 3. *An* event log *is a finite set of cases.*

When not enough attributes are present for a given event, we can add dummy values. We assume that for every event the cID of the case to which it belongs is known.

Example 3. *The event log in Table 2.1 was generated by a process where an employee needs to log in to a system to create a request. This request is then sent to his or her manager who can approve or reject the request. The log contains 8 attributes: Time, (event)ID, Type, Activity, UID, UName, URole, and cID. We use cID to keep track of the case to which an event belongs. In total, we have 4 users, each with a unique ID and Name. Every user has a role from a limited set of roles. For the sake of simplicity, we have only captured a subset of all possible actions that can or should occur.*

2.1.1 The k-prefix and k-context

To model the behavior of a business process we need to take the order of the events into account. Therefore we introduce a prefix-based system, in which we capture the preceding events (the prefix) of every event in the event log.

Definition 4. *Let $k \geq 0$. The k-prefix of an event e_i, the i-th event of a case, is defined as a tuple $\mathscr{P}_k(e_i) = (X_{i-k}, \ldots, X_{i-1})$ where*

$$X_{i-l} = \begin{cases} e_{i-l} & \text{if } i-l > 0 \\ None & \text{otherwise} \end{cases}$$

None is a special dedicated value that should not occur in the log. We use $\mathscr{P}_k(e_i).A_l$ for $l = 1 \ldots k$ to denote the value of attribute A of the l-th event before e_i.

We introduce the k-context of an event to link the event with its k-prefix. We do this by concatenating the entries of the event and its k previous events. By doing this, our learning algorithms do not have to take the sequential nature of event logs into account, making it possible to start from standard, existing learning approaches.

Definition 5. *Let $k \geq 0$. The k-context of an event e is defined as $\mathscr{C}_k(e) = (\mathscr{P}_k(e), e)$. We use the following notations:*

$$\mathscr{C}_k(e).A := e.A$$
$$\mathscr{C}_k(e).A_l := \mathscr{P}_k(e).A_l$$

To easily work with all events and their associated prefixes, we transform the original event log into a *k-contextlog*, where we collect the k-context of every event. This k-contextlog can contain additional information that is not linked to a single event, such as the duration between two events. We can add this extra information in the form of additional attributes to the k-contextlog.

To indicate attributes and their corresponding values, we use uppercase letters to indicate attributes and lowercase to denote the value of an attribute.

Example 4. *When we look at Table 2.1, the* 2-context *of event 3 consists of events 1, 2 and 3. To be able to use all attributes from all events in our model, we convert the 3 events into a single tuple:* (System_2, Logged in_2, 001_2, Joe_2, employee_2, Request_1, Create_1, 001_1, Joe_1, employee_1, Request, Send Mail, 001, Joe, employee). *We added the subscript i to increase readability; System_2 for instance indicates that Type_2 = System in the* 2-context. *For the current event, we omit the subscript 0. We further added extra attributes to indicate the duration between the different events; $dur_2 = 0$ (the duration between events 1 and 2) and $dur_1 = 1$ (the duration between events 2 and 3).*

2.2 Methods for Learning from Data

Now that we have defined the data our methods will have to work on, in this section we take a look at the machine learning approaches that can be used to solve the problems we try to solve. We pay special attention to Bayesian networks and Dynamic Bayesian networks, which we use extensively in this thesis.

The use of machine learning to make all kinds of predictions, such as the outcome of a certain situation or a score for a customer, has been studied in different fields. Examples are the financial sector [63], the stock market [129] and public health [85].

We distinguish two types of machine learning methods:

- **Supervised methods** These methods require labeled data to train their model, e.g. the outcome for a certain event has to be known while learning.
- **Unsupervised methods** The data needed to train these models does not need to contain specific labels. These methods look at similarities between the data and assume that similar data points behave in the same way.

2.2.1 Predictive Monitoring

Making predictions requires a labeled data set, otherwise, the model has no way of knowing what to predict. Typical examples of such supervised learners are:

- **Decision Trees** [47, 58] A tree-like structure composed of different nodes, where each node represents a binary decision based on the values of different attributes. These nodes are created by selecting the features which carry maximum information for the particular classification/prediction task.
- **Support Vector Machines** [69, 70] An SVM is a binary classifier that uses a hyperplane to divide the data. This division is made by making the distance between the hyperplane and the given data as large as possible. For new data, one has to look at which side of the hyperplane the data lies to get the class label. When the data is represented in only two dimensions, we get a single line as the boundary between the two classes.
- **Neural Networks** [3] Neural Networks consist out of several cells, each with input and output, and connections between the different cells. These cells can perform calculations on the input they receive and return it via their output. In general, a neural network is a function that maps a given input to some output values. Given a set of known input/output pairs, a neural network learns the optimal weights for all cells in the network. In the following sections, we take a look at how neural networks can also be used in business process monitoring.
- **K-Nearest Neighbor methods** [76, 90, 127] When using KNN methods we label data based on the k examples of known data that are the closest to this new data. How this distance gets calculated and what features are used depends on the problem domain and proposed solution.

We can also use methods that are typically used in an unsupervised setting such as K-Mean Clustering [61], and Expectation Maximization [123]. These methods are often used as a first stage in which similar data are grouped, after which a supervised method can be applied to this simplified data.

Besides these machine learning techniques, we can also make use of more statistics-based learning techniques such as Logistic Regression [27], or Bayesian Networks [98]. Every technique has its advantages and disadvantages. To improve

the quality of predictions one can combine multiple techniques in an ensemble [5, 60]. This reduces the impact of a single, possibly bad-performing technique on the overall prediction.

2.2.2 Anomaly Detection

Anomaly detection tries to identify abnormal occurrences in data, given a reference window. In the survey presented by Chandola et al. [26] about anomaly detection for discrete sequences, anomaly detection is divided into three different sub-problems:

1. Identifying anomalous sequences with respect to a database of normal sequences
2. Identifying an anomalous subsequence within a long sequence
3. Identifying a pattern in a sequence whose frequency of occurrence is anomalous

The first type is a semi-supervised way of detecting anomalies as we explicitly have a database with normal executions. The other two are unsupervised as no extra information (label) is available. Even though all these three types of anomaly detection approaches can be of interest in the business process setting, our research mainly focuses on the first type of approach. We try to identify the cases within the execution of a process that show anomalous behavior. Outside of business processes, many different algorithms are proposed. These algorithms can be roughly divided into four groups, based on the used technique:

1. **Similarity-based techniques** These techniques consider the entire sequence as a single point (a single entity) and then apply a proximity-based point anomaly detection technique that uses an appropriate similarity measure [19, 86]. Instances far away from found clusters are marked as anomalous. Figure 2.1 gives an example of this technique.
2. **Window-based techniques** These techniques only analyze a short window of symbols/activities within the sequence at a time. In contrast to the previous techniques, window-based techniques consider these different windows as single points by extracting certain features from this window. This gives more

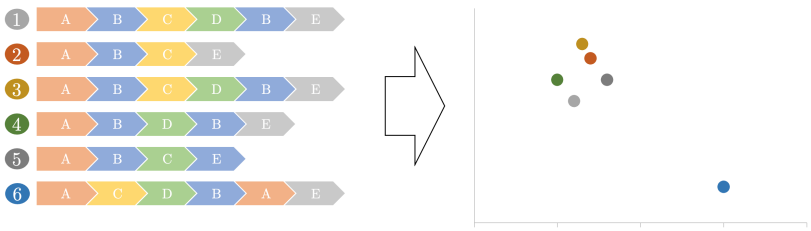


Figure 2.1: Example of a similarity-based techniques where every sequence is transformed in a single anomaly score. An outlier indicates an anomalous sequence.

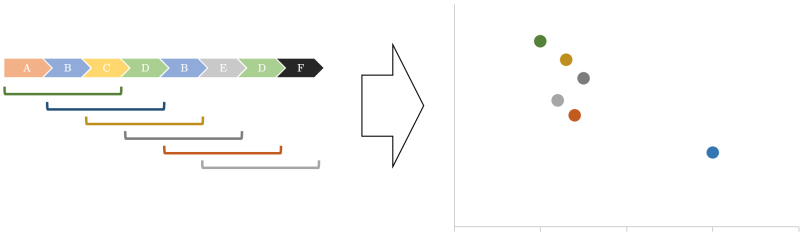


Figure 2.2: Example of calculating anomaly scores for windows of size 3 in a single sequence.

flexibility and improves the ability to detect smaller, more local anomalies [46, 117]. An example of this technique is shown in Figure 2.2.

3. **Markovian techniques** These techniques use a probabilistic model that predicts the probability of observing each symbol and uses a per-symbol probability to obtain the total anomaly score for the entire sequence. Every symbol/event is analyzed with respect to a few previous symbols [39, 126].

4. **Hidden Markov model (HMM)-based techniques** This technique learns an HMM that best describes the normal training sequences. For every test sequence, we can then calculate the probability using our learned HMM [84, 124].

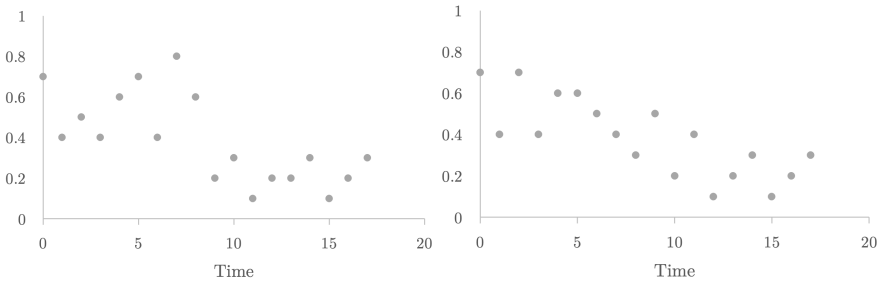
The main application field of these anomaly detection techniques lies in intrusion detection, where we want to detect inappropriate access to a computer system or network infrastructure by possible malicious software or users. Although our main focus lies on business processes, we show that some of the basic ideas and techniques can be adapted to work in the field of anomaly detection within business process monitoring.

2.2.3 Concept Drift Detection

Concept drift has also been studied in Machine Learning, where concept drift refers to a non-stationary learning problem over time. This means that the training data does no longer match the data that is being analyzed [131]. Typical situations in which concept drift occurs are changes in legislation, merging of departments, policy changes, ...

New data arrives at different times t, we assume that all this data has been generated by a source. We state that every instance at time t (X_t) has been generated by source s_t. Data comes from the same source when the distribution of the data is the same. Otherwise, they come from different sources and we say that we have found *concept drift*.

Two examples are shown in Figure 2.3. Figure 2.3a shows a sudden drift in which the source immediately changes from one point in time to the next. In Figure 2.3b we can see a more gradual change from one source to the other source. Concept drift learners divide the problem into the following four sub-problems:



(a) Sudden drift (b) Gradual drift

Figure 2.3: Two examples of concept drift occurring. We assume that the source is a random number generator and drifts occur when the bounds for generated numbers change. The y-axis denotes the generated values.

1. **Future assumption** This is the core assumption that has to be made. The designer has to make an assumption about the future data source. *How is the data going to behave in the future, how can we identify it?*

2. **Change type** The possible change patterns have to be identified. *Do we have a sudden or more gradual drift?*

3. **Learner adaptivity** Based on the previous two selected solutions one has to choose the mechanism that makes the learner adaptive. *How and when do we transition to take the new data distribution into account?*

4. **Model selection** Lastly one has to determine the criteria to choose a particular parametrization of the used learner at every time step.

Schlimmer and Granger [91] introduced the term *Concept drift* in 1986 when they presented their algorithm STAGGER that can incrementally learn from noisy data.

In general two main approaches have been developed: an *Evolving* and *Trigger* approach. The evolving approach constantly updates itself according to the observed behavior in the data. A popular technique for this approach is the use of *classifier ensembles*. The outputs of several models are combined to get a final decision. These models can range from SVMs [50], to Gaussian mixture models [122], and k-Nearest Neighbor methods [56].

The second approach uses triggers that determine how and when the model should be changed. These models can make use of change detectors, when a change has been detected the training window is cut at this point [37, 79]. Another method is to adaptively change the window size when drift has been discovered [12, 121].

In Chapter 4 we look at both evolving and trigger approaches to update existing models. In Chapter 7 we introduce our own Concept Drift detection method using a trigger approach that makes use of our DBN model.

2.2.4 Bayesian Networks

A *Bayesian Network* (BN) [81] represents a joint distribution for a set of random variables $(X_1, \ldots, X_n)$. It does so by capturing the relations between the random variables. The different relations are represented in a *directed acyclic graph*. Every variable is represented by a node in the graph. When a variable Y conditionally depends on another variable X there exists an edge in the network from X to Y. If there is an edge from X to Y, we call X a *parent* of variable Y, a single variable can have multiple parents. We denote the parents of a node Y as $Pa(Y)$ and the actual values of the parents as $v\_Pa(Y)$. The joint distribution of a Bayesian Network with variables $X_1, \ldots, X_n$ can be decomposed as follows:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | Pa(X_i)) \tag{2.1}$$

The joint distribution thus gets decomposed into different factors, one factor for every variable in the model. This factor can be calculated in different ways depending on the type of variable. Default Bayesian Networks use Conditional Probability tables to model the conditional probability distribution $P(X_i|Pa(X_i))$. In **Chapter 6** we extend this default model by adding Functional Dependencies and Kernel Density Estimates.

Example 5. *The Bayesian network presented in Figure 2.4 models whether the grass in our garden is wet. The grass can become wet because it rains or because our sprinkler system is working. We model this by using three boolean variables:*

- ***Rain**: Is it raining outside?*
- ***Sprinkler**: Are the sprinklers working?*
- ***Grass Wet**: Is the grass wet?*

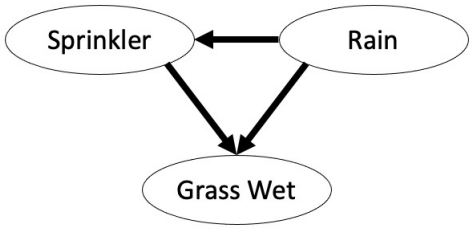


Figure 2.4: Bayesian Network with three variables: Sprinkler, Rain and Grass Wet.

It is intuitively clear that these different variables all influence each other. The arrows in the model indicate how these variables influence each other. It is clear that when it rains or when the sprinklers are working the grass will be wet. We also have that the sprinklers are most likely not working when it rains and we thus also have an extra

Table 2.2: Conditional Probability Tables for Rain (a), Sprinkler (b), and Grass Wet (c).

(a) Rain

P(True)	P(False)
0.2	0.8

(b) Sprinkler

Rain	P(True)	P(False)
False	0.4	0.6
True	0.01	0.99

(c) Grass Wet

Sprinkler	Rain	P(True)	P(False)
False	False	0.0	1.0
False	True	0.8	0.2
True	False	0.9	0.1
True	True	0.99	0.01

dependency between Rain and Sprinkler. We can then describe the total probability distribution using three tables (one for every variable in the model) as shown in Table 2.2.

2.2.5 Dynamic Bayesian Network

A Dynamic Bayesian Network (DBN) [89] incorporates the sequential aspect of the data by adding extra variables to the network. For the previous k time steps, variables are added that represent the values of the attributes in the previous time steps. We call them the *past-variables*. An example of a DBN with $k = 1$ can be found in Figure 2.5. In this diagram, the different time steps are indicated with the dotted boxes. Dependencies can exist within one time step (Activity → Type) or between time steps ($\text{Activity}_{\text{previous}} \rightarrow \text{Activity}_{\text{current}}$). Because we are interested in learning how variables in the current time step depend on variables from the previous time steps, we only allow arrows going to variables in the current time step.

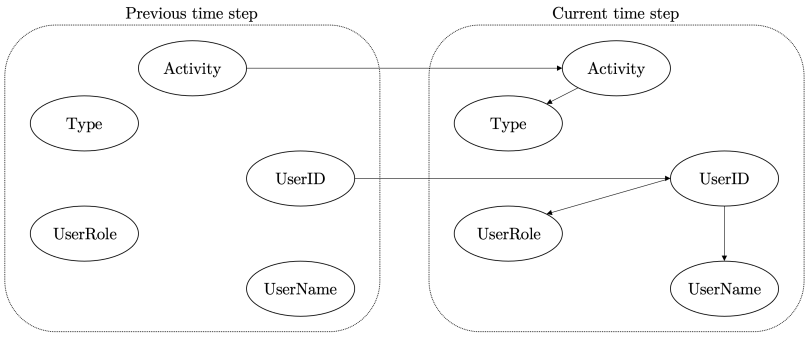


Figure 2.5: Example Dynamic Bayesian Network consisting of the current and one previous time step.

(Dynamic) Bayesian networks can be used to infer the probability distribution for a set of variables, given a separate set of attributes for which we know the exact values. This distribution can then be used to select the most likely value or to compute the probability for a given value.

2.3 Conclusion

In this chapter, we introduced the basic definitions of what an event log is and what the typical elements present in these logs are. We explained how we capture the history of events by introducing the k-prefix and k-context. By using the k-context we link every event with its k previous time steps, making it possible to learn a model using all attributes in all available time steps.

Furthermore, we looked at existing machine learning techniques that have been used to solve the same or equivalent problems as we try to solve in this thesis. We introduced the different problems we tackle in this thesis and the different approaches that can be used to solve them. To conclude, we introduced Dynamic Bayesian Networks that we will use extensively in the remainder of this thesis.

CHAPTER 3

Bayesian Network-based Predictions of Business Processes

Making predictions about a running business process case is becoming more important as more and more systems are getting automated. Predicting deviating behavior early on in a process can ensure that possible errors are identified and corrected or that unwanted delays are avoided. We propose to use Bayesian Networks to capture dependencies between the attributes in a log to obtain a fine-grained prediction of the next activity. Elaborate comparisons show that our model performs on par with the state-of-the-art methods. Our model, however, has the additional benefit of explainability; due to its underlying Bayesian Network, it is capable of providing a comprehensible explanation of why a prediction is made. Furthermore, the runtimes of our learning algorithm are orders of magnitude lower than those of state-of-the-art methods that are based on deep neural networks.[1]

[1] This chapter is based on *Stephen Pauwels and Toon Calders. Bayesian network based predictions of business processes. In International Conference on Business Process Management (BPM Forum '20), pages 159–175, 2020.*

3.1 Introduction

Di Francescomarino et al. [33] show the growing importance and interest in predicting events within Business Processes. Different methods have been proposed based on different techniques; using probabilistic models, machine learning, or Deep Learning. All these methods have in common that they use historical data to create a reference model that subsequently can be used in new cases to determine the next activity before it occurs.

Commonly used process models such as workflow nets or BPMN diagrams are less suitable for predictions within event logs as they fail to capture transition probabilities and as a result, have less accurate predictions. Most of the state-of-the-art prediction models rely on Deep Artificial Neural Networks. Despite that these Deep Learning methods can provide accurate predictions, they also have some issues. A first issue is that training the models requires extensive resources such as time and computation power, necessitating the use of specialized hardware. Typically these methods also require excessive amounts of training data for learning an accurate model. Secondly, neural networks are so-called black-box models, in the sense that the predictions they make cannot be explained easily, making them unsuitable for many applications.

To overcome these issues, we introduce an alternative method based on *Dynamic Bayesian Networks* for making predictions in an event log. Using the Conditional Dependencies present in the event log, we can create a probabilistic model that can predict different attributes, including the next activity. In comparison with the state-of-the-art deep learning methods, our method has the advantage that it produces predictions that can be explained and takes orders of magnitude less time and resources to train while performing with comparable predictive accuracy. An elaborated use case showing the explainability can be found in Chapters 7 and 8.

This chapter is structured as follows: Section 3.2 gives an extensive overview of the related work. Section 3.3 describes how we use Dynamic Bayesian Networks for predicting the next event(s). Extensive evaluation and comparisons are performed in Section 3.4, where we look at the performance for both the next activity prediction problem and the suffix prediction problem. Next to comparing the accuracy of the different methods, we also compare the runtime of both training and testing for all methods.

3.2 Related Work

We can distinguish different types of predictions that are mainly researched within the business process field:

- **Next-activity Prediction** predicts only the next activity that will occur within a running case
- **Suffix Prediction** predicts all following activities that will occur within a running case
- **Remaining time Prediction** predicts the time that remains for completing the entire case

- **Outcome Prediction** predicts what the general outcome of a case will be

In this thesis, we mainly focus on predicting the next activity.

Table 3.1 contains an overview of the described methods. We indicate for every method which type of prediction problem it can solve and what type of input they require. Do they only take the activity of the event into account (the activity-perspective) or do they also use other attributes such as resource, department (the data-perspective)? We also indicate if the source code from the authors is publicly available.

3.2.1 Probabilistic Models

There already exists ample research on applying probabilistic models for predictions in Business Processes. Becker et al. [6] 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), where every transition is linked with a probability for going to a specific state. An advantage of these models is their explainability and possibility of visualization, but on the other hand, they do not take multiple attributes into account and are not able to find long-term dependencies; i.e., dependencies between events that are far apart in a case. These PFAs were alse used by Breuker et al. [16], they improve this model by incorporating more event data.

Lakshmanan et al. [55] propose an instance-specific Probabilistic Process Model (PPM) that calculates the likelihood for all possible events to occur next. They show that under some non-restrictive assumptions, their model can be transformed into a Markov Chain, allowing them to rely on standard Markovian techniques. To create their model, a Petri Net process model is mined based on the given traces. This model is then extended by adding a probabilistic model for all XOR (choices) and AND (parallel gateways) splits that allow it to predict the next event.

Polato et al. [82] propose a method that uses a Data-Aware Transition system. This transition system not only takes the activity perspective into account but can also use extra data attributes that are present in the process. Their method can work in a changing environment and can handle unexpected scenarios.

An important factor in these probabilistic methods is that they are comprehensible and can explain why certain predictions were made, but only use limited information of the events. As a result, they encounter the same challenges as most process models; i.e.: these models fail at detecting and correctly predicting long-term dependencies.

Table 3.1: Overview of related work. Indicating which model type was used: Dynamic Bayesian Network (DBN), Hidden Markov Model (HMM), Probabilistic Finite Automaton (PFA), Probabilistic Process Model (PPM), Data-Aware Transition system (DAT), Long-Short-Term-Memory Neural Network (LSTM), Convolutional Neural Network (CNN), or Generative Adversarial Network (GAN).

Method	Model type	Input data	Predict Next Activity	Predict Suffix	Predict Duration	Code Available
Ours	DBN	multivariate	✓	✓	✓	✓
Becker et al.	HMM	activity	✓			
Breuker et al.	PFA	activity	✓			
Lakshmanan et al.	PPM	activity	✓			
Polato et al.	DAT	multivariate	✓	✓	✓	✓
Everman et al.	LSTM	activity, resource	✓	✓	(✓)	
Tax et al.	LSTM	activity, duration	✓	✓	✓	✓
Lin et al.	LSTM	multivariate	✓	✓		
Camargo et al.	LSTM	activity, resource and duration	✓	✓	✓	✓
Di Mauro et al.	CNN	activity, duration	✓		✓	✓
Pasquadibisceglie et al.	CNN	activity, duration	✓			✓
Taymouri et al.	GAN with LSTM	activity, duration	✓			✓

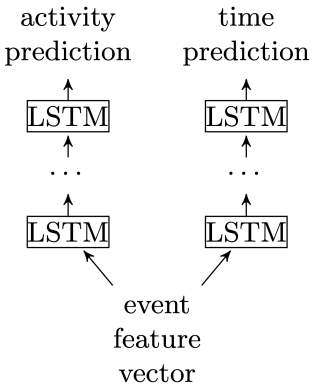


Figure 3.1: Overview of the architecture used by Tax et al. [101].

3.2.2 Deep Learning

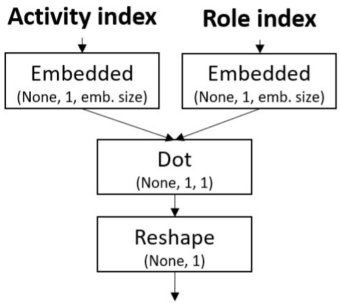
With the growing popularity of Deep Learning, which is achieving great results in the field of Natural Language Processing (NLP) and image processing, researchers proposed the usage of Neural Networks in predictive process monitoring as well. Most of the state-of-the-art methods that are recently proposed use Neural Networks after Evermann et al. [40] showed that these models can be successfully applied within the context of business processes.

Architectures The first networks for next event prediction use Long-Short-Term-Memory (LSTM) layers. LSTM layers can selectively remember parts of the sequence and use this memory for determining the output. Most of the methods that use these LSTMs are very similar and differ mainly with respect to the architecture of the network; e.g., extra LSTM cells can be added or they are organized in different ways.

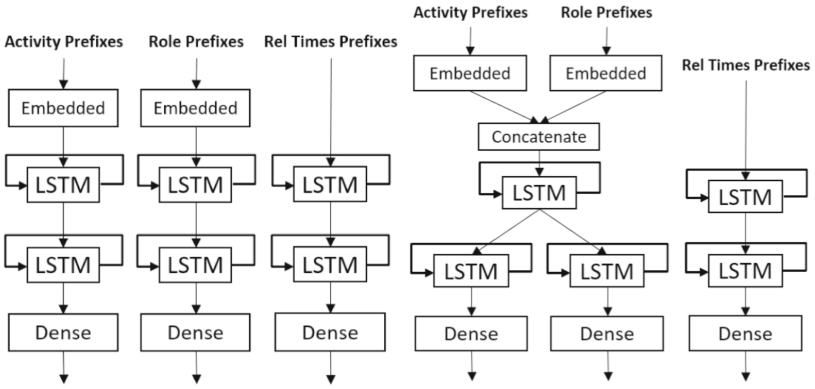
Tax et al. [101] propose an LSTM-based neural network to predict the next activity, the full suffix, and the remaining time. The cases (sequences of events) are encoded by using one-hot encoding to represent the activities. Besides the activity, they also take the time since the previous event, the hour of the day, and the day of the week into account. Depending on the exact prediction task, a slightly different architecture can be chosen. All the possible architectures consist of several connected LSTM layers, the number of layers can vary between datasets and has to be chosen by the user. This network is shown in Figure 3.1.

Camargo et al. [23] added a separate neural net for learning a non-trivial embedding, that takes both the activity and resource into account. Furthermore, different network architectures were proposed by Camargo with different hyperparameter settings. These architectures range from combining activity, resource and duration together to keeping all attributes separate. Figure 3.2 gives an overview of the used architectures.

To further incorporate extra attributes in an LSTM-based model, Lin et al. [62] propose to include a custom layer (the modulator) to take advantage of these extra attributes present in the data. This modulator layer learns how important the



(a) Embedding network



(b) Prediction network - specialized (c) Prediction network - shared

Figure 3.2: Overview of the architectures and embedding network used by Camargo et al. [23].

attributes are for our predictions and assigns weights to these attributes. This network is shown in Figure 3.3.

The LSTM architecture has also been used in combination with Generative Adversarial Networks by Taymouri et al. [103], as shown in Figure 3.4. In this type of model, the network exists of two parts: the first part tries to predict the best activity as well as possible, while the other tries to divide the real activities that happened from the predicted ones. Both parts of the model compete with each other. In this way, the predictive model gets more accurate feedback about its performance.

An alternative for an LSTM-based model is a Convolutional Neural Network (CNN) to deal with the sequential nature of business processes. These CNNs are often used in Computer Vision [99] and have proven to perform better than recurrent neural network [4], such as LSTMs. Another benefit of these convolutional networks is that they are trained in a more efficient way than the previously introduced LSTM-based methods.

Both Di Mauro et al. [35] and Pasquadibisceglie et al. [78] use CNNs to predict the next activity. Di Mauro et al. use the prefixes consisting of the events as the input sequences for the model. A typical problem when using a CNN is the kernel size to use. This problem was solved in [99] by introducing an inception module

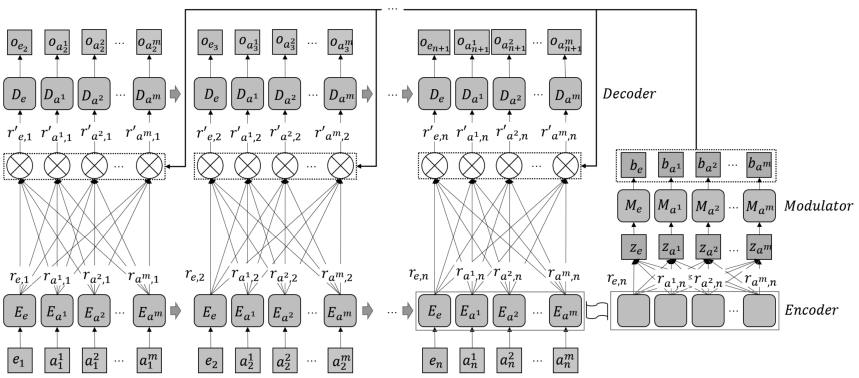


Figure 3.3: Overview of the architecture used by Lin et al. [62].

that runs several convolutions with all different kernel sizes on the same input. The architecture used by Di Mauro et al. is shown in Figure 3.5.

Pasquadibisceglie et al. convert the input events into an image-like input, with features indicating the number of times a certain activity has occurred in the case, extended with extra timing information. An overview of their network can be found in Figure 3.6.

Input representation Most deep learning methods use a one-hot encoding feature vector as input for the network. This vector has the same size as the input domain of the input and every position in the vector corresponds to a possible value of this input. We have a 1 on position i in the vector when the actual value equals the value that is linked with position i and 0 otherwise. Some approaches take multiple attributes of the events into account. For instance, Camargo et al. [23] first create equal-size embeddings for both activity and resource. Next, they use the dot product to combine these two features into a single feature vector which has lower dimensionality than the original feature vectors. Lin et al. [62] on the other hand, have adapted their network to take multiple attributes into account. 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 number of event attributes and event values is one of the biggest challenges for the methods based on

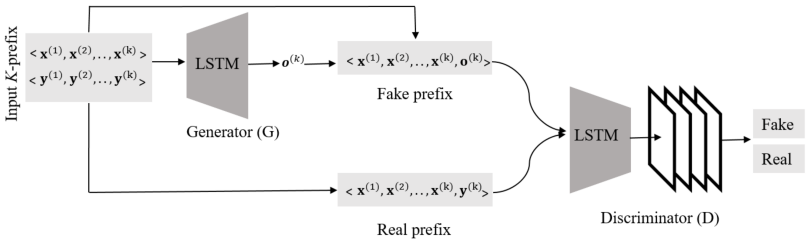


Figure 3.4: Overview of the architecture used by Taymouri et al. [103].

Recurrent Neural Nets. To tackle this problem, Hinkka et al. [45] propose a Recurrent Neural Net method where they first cluster events according to their attribute values to lower the overall complexity.

As the used Neural Networks need sequences of the same length, two solutions were proposed. The first solution is to pad all sequences with 0 until they all match the length of the longest sequence [35, 101]. The other solution is to divide the sequences in different windows with a fixed window-size [23]. The advantage of this second solution is that the complexity of the neural network does not depend on the length of the longest sequence, as we fix the length of the input sequences (windows) beforehand.

3.3 Dynamic Bayesian Network-based Predictions

Before we explain how we extend and use Bayesian Networks for event prediction, we first introduce the basic problem settings. Predicting the suffix of a case is defined as follows:

Problem Statement 1. *Given a partially completed case* $C = (cID, [e_1, \ldots, e_i])$*, with* $0 < i < m_c$*, we want to predict the list* $[a_{act}^{i+1}, \ldots, a_{act}^{m_c}]$ *of the activities of* all *following events.*

We express the problem of finding the next activity as a special case of suffix prediction as follows:

Problem Statement 2. *Given a partially completed case* $C = (cID, [e_1, \ldots, e_i])$*, with* $0 < i < m_c$*, we want to predict the activity* a_{act}^{i+1} *of the immediately next event* e_{i+1}*.*

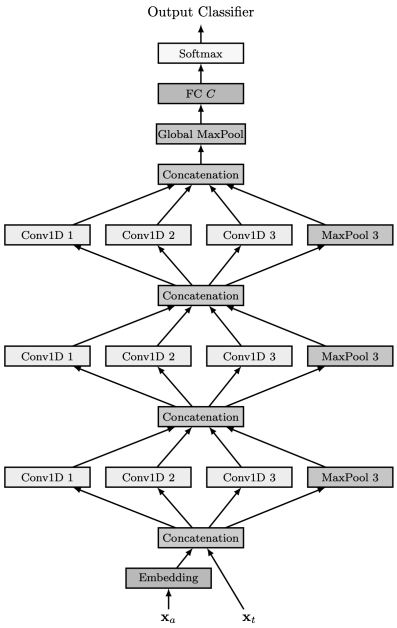


Figure 3.5: Overview of the architecture used by Di Mauro et al. [35].

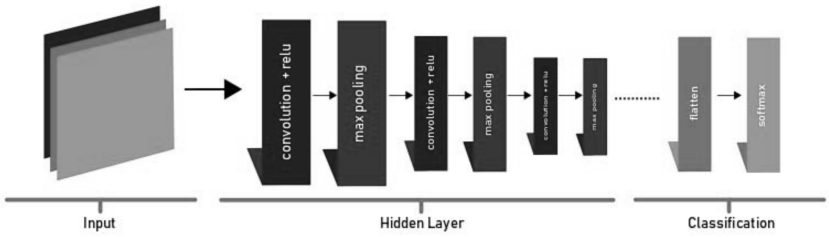


Figure 3.6: Overview of the architecture used by Pasquadibisceglie et al. [78].

We use historical data to train our probabilistic model. With the available data we can train a Bayesian network using existing learners and populate the Conditional Probability Tables. In the next sections, we explain how we use this model for event prediction. Note that, although we only consider event prediction using categorical attributes, DBNs can be extended to also be able to handle numerical attributes and predict numerical values as is shown in later chapters of this thesis.

3.3.1 Dynamic Bayesian Network

We use Bayesian Networks (BNs) [81] to model the dependencies between attributes present in an event log as introduced in Section 2.2.4. All attributes present in the k-contextlog (including timestamp, activity and all data attributes) correspond to random variables within the Bayesian Network.

To learn a BN we first need to learn the structure of the model (the dependencies between attributes). At the same time we learn the Conditional Probability Tables (CPTs) that capture $P(X_i|Pa(X_i))$ for all random variables X_i. These tables contain the probabilities for all possible values given the occurrence of values of the parents. Existing algorithms already exist in the literature that can learn the structure of such a BN given a reference dataset [89]. We introduce the dynamic aspect into the Bayesian Network by only allowing dependencies between attributes from a previous time step to attributes in the current time step, as we want to capture real-life dependencies that do not break the chronological order of the event log.

3.3.2 Next Activity Prediction

To predict the activity attribute *act*, we calculate $P(act)$ for all possible (known) values of the activity attribute. Following Equation 2.1 we estimate the probability for the activity as $P(act|Pa(act))$, where $Pa(act)$ are the attributes on which the current activity depends according to our learned model.

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 predicted probability for being the next activity.

Table 3.2: Example CPT based on event log from Table 2.1; assume activity depends on activity and resource from the previous time step. This CPT gives the empirical conditional probabilities $P(activity \mid activity_1, resource_1)$, where $activity_1$ and $resource_1$ denote respectively the activity and resource of the previous time step.

$activity_1$	$resource_1$	$activity$	$P(activity \mid activity_1, resource_1)$
Log in	Joe	Logged in	1.0
Log in	Linda	Logged in	1.0
Send mail	Joe	Approved	0.7
Send mail	Joe	Disapproved	0.3
Send mail	Marc	Approved	1.0
Create	Joe	Rejected	0.4
Create	Joe	Send mail	0.6
Approved	Linda	Send mail	1.0
Approved	Bob	Send mail	1.0

Example 6. *Suppose we learned that the activity attribute depends on activity and resource from the previous time step. The probability* $P(activity \mid activity_1, resource_1)$ *can be read from the CPT shown in Table 3.2.*

Given the values Create *and* Joe *for activity and resource respectively from the previous time step, we can see in the table that two activities for the next event are possible:* Rejected *(with a probability of 0.4) and* Send mail *(with a probability of 0.6). We return the value with the highest probability, which is* Send mail*, as the prediction for the next event.*

Because we populate the conditional probability tables 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 that are present to avoid having to use a probability of 0. We use marginalization techniques 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 have been observed in the training data, but not together.

For the first case, when there are new values for some attributes, we calculate the marginalized probability where we iterate over all known values for these attributes. We let the new attributes vary over all possible values, which occur in the CPT, while we keep the known attributes fixed. That is: Let $Pa(act)_u$ be the set of parent attributes of act for which we observe an unseen value and $Pa(act)_k$ the other parents with known attribute values. We thus have that $Pa(act) = Pa(act)_u \cup Pa(act)_k$. For determining the probability of the activities we need to calculate $P(act|Pa(act)_u, Pa(act)_k)$. We calculate alternatively $P(act|Pa(act)_k)$ as follows:

$$\sum_{a_u \in dom(Pa(act)_u)} P(a_u|Pa(act)_k) * P(act|a_u, Pa(act)_k) \tag{3.1}$$

Notice that we used $a_u \in dom(Pa(act)_u)$ to denote the domain of $Pa(act)_u$, which contains all previously observed combinations of the attributes $Pa(act)_u$.

Example 7. *Consider the CPT from Table 3.2, suppose that we need to predict the next activity, based on the values* System failed *and* Linda *for attribute and resource. The value for the resource has already been seen, but the value for the activity is new. We thus marginalize over the activity attribute. The seen attribute* resource *only occurs together with the values* Log in *and* Approved. *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:*

$$\begin{aligned} P(A_{act}|Linda) &= P(Log\ in \mid Linda) * P(A_{act} \mid Log\ in, Linda) \\ &\quad + P(Approved \mid Linda) * P(A_{act} \mid Approved, Linda) \end{aligned}$$

Assume $P(Log\ in \mid Linda) = 0.4$ *and* $P(Approved \mid Linda) = 0.6$. *We can now calculate the probabilities for all activities as follows:*

$$\begin{aligned} P(Log\ in \mid System\ failed, Linda) &= 0.4 * 1.0 + 0.6 * 0 = 0.4 \\ P(Logged\ in \mid System\ failed, Linda) &= 0.4 * 0 + 0.6 * 0 = 0 \\ P(Send\ mail \mid System\ failed, Linda) &= 0.4 * 0 + 0.6 * 0 = 0 \\ P(Create \mid System\ failed, Linda) &= 0.4 * 0 + 0.6 * 0 = 0 \\ P(Approved \mid System\ failed, Linda) &= 0.4 * 0 + 0.6 * 1.0 = 0.6 \end{aligned}$$

From these results, we can see that the next activity with the highest probability is Approved.

For the second case, when all values occur in the training data but the combination of values does not occur in the CPT, we take the average probability of marginalizing over every attribute. We use the following formula to estimate the probabilities for the activities:

$$\frac{\sum_{A \in Pa(A_{act})} \sum_{a \in dom(A)} P(a|Pa(A_{act}) \setminus A) * P(A_{act}|a, Pa(A_{act}) \setminus A)}{\#Pa(A_{act})} \tag{3.2}$$

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

Example 8. *Consider again the CPT from Table 3.2, suppose this time we have the values* Send mail *and* Linda *for activity and resource. Both values already occur in the CPT but never together. We thus calculate the following formula for every possible activity:*

$$\begin{aligned} P(X_{act}|Send\ mail, Linda) = (&P(Log\ in|Linda) * P(X_{act}|Log\ in, Linda) \\ &+ P(Create|Linda) * P(X_{act}|Create, Linda) \\ &+ P(Approved|Linda) * P(X_{act}|Approved, Linda) \\ &+ P(Joe|Send\ mail) * P(X_{act}|Send\ mail, Joe) \\ &+ P(Marc|Send\ mail) * P(X_{act}|Send\ mail, Marc)) \\ &+ P(Bob|Send\ mail) * P(X_{act}|Send\ mail, Bob))/2 \end{aligned}$$

Assume:

$$\begin{aligned} P(\textit{Log in} \mid \textit{Linda}) &= 0.2 \\ P(\textit{Create} \mid \textit{Linda}) &= 0.4 \\ P(\textit{Approved} \mid \textit{Linda}) &= 0.6 \\ P(\textit{Joe} \mid \textit{Sendmail}) &= 0.4 \\ P(\textit{Marc} \mid \textit{Sendmail}) &= 0.3 \\ P(\textit{Bob} \mid \textit{Sendmail}) &= 0.3. \end{aligned}$$

We then get the following probabilities for the possible activities:

$$\begin{aligned} P(\textit{Log in} \mid \textit{Send mail}, \textit{Linda}) = \\ (0.2*0+0.4*0+0.6*0+0.4*0+0.3*0+0.3*0)/2 = 0 \\ P(\textit{Logged in} \mid \textit{Send mail}, \textit{Linda}) = \\ (0.2*1.0+0.4*0+0.6*0+0.4*0+0.3*0+0.3*0)/2 = 0.1 \\ P(\textit{Send mail} \mid \textit{Send mail}, \textit{Linda}) = \\ (0.2*0+0.4*0+0.6*0+0.4*0+0.3*0+0.3*0)/2 = 0 \\ P(\textit{Create} \mid \textit{Send mail}, \textit{Linda}) = \\ (0.2*0+0.4*0+0.6*0+0.4*0+0.3*0+0.3*0)/2 = 0 \\ P(\textit{Approved} \mid \textit{Send mail}, \textit{Linda}) = \\ (0.2*0+0.4*0+0.6*0+0.4*1.0+0.3*1.0+0.3*0)/2 = 0.35 \end{aligned}$$

From these results, we can see that the next activity with the highest probability is Approved.

3.3.3 Suffix Prediction

To predict the suffix of cases we also have to predict other attributes, as we need these attributes to be able to keep predicting the activities in the suffix. First, 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.

Example 9. *Consider the example from the previous section. In this example,* activity *depends on* activity *and* resource *in the previous time step. Assume now that* resource *itself also depends on* 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 3.3.2. As these formulas are attribute-independent we can also use them for predicting other attributes. We can use the same marginalization techniques when new combinations of parent values occur.

Table 3.3: Conditional Probability Table for activity used in example 11.

$activity_1$	$activity$	$P(activity \mid activity_1)$
start	a	1.0
a	a	0.6
a	b	0.4
b	end	0.8
b	c	0.2

Example 10. *To predict the suffix for our example we have to predict values for activity and resource. We thus need to calculate the following probabilities:*

$$P(activity \mid activity_1, resource_1)$$
$$P(resource \mid activity_1, resource_1)$$

The disadvantage of always selecting the activity with 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. To solve this issue, we restrict 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 11. *Consider a situation where activity only depends on the previous activity. The CPT for activity is shown in Table 3.3. Assume we have* $[start]$ *as the current prefix, when we would predict the suffix without limitations on the self-loops we would get the following sequence:* $[start, a, a, a, a, a, \ldots, a]$ *until we reach the maximum allowed suffix length. When we set the maximum size for self-loops to 3, we get the following sequence:* $[start, a, a, a, b, end]$.

3.4 Evaluation

In this section, we describe the different experiments we performed to assess the quality of our method. First, we describe the different datasets used for performing the experiments. Next, we describe how we performed the experiments. The results are split into an evaluation of our method, where we test the influence of the number of time steps used, and a comparison with state-of-the-art methods. We perform both runtime and accuracy experiments to fully compare the different methods. Furthermore, we perform qualitative experiments on the explainability in Chapter 7.

3.4.1 Datasets

For our evaluation, we selected different well-known datasets that are often used in related work.

Table 3.4: Characteristics of the datasets after preprocessing.

Dataset	# Events	# Cases	# Activities	Avg. events 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
BPIC18	2,514,266	43,809	41	57.4	2,973
Helpdesk	20,722	4,371	14	4.7	15

- **BPIC12** [111]: a log from a Dutch financial institution, containing applications for personal loans. The event log contains three intertwined processes, one of which was used to create an additional event log (**BPIC12W**).
- **BPIC15** [112]: a log containing building permit applications from five different Dutch municipalities, where the data of every municipality is saved in a single event log. We denote these 5 datasets respectively **BPIC15_1** to **BPIC15_5**.
- **BPIC18** [110]: a log containing applications for EU agricultural grants. This is the most challenging dataset and not all state-of-the-art methods could be executed using this event log.
- **Helpdesk** [115]: a log containing ticket requests of the helpdesk from an Italian Software Company.

Following the same preprocessing steps used by Camargo et al. [23], we first remove all cases with less than 5 events from the logs. Only for the Helpdesk dataset, we set this threshold to 3 because of the smaller average length of the cases in this dataset.

Table 3.4 gives the characteristics of the used datasets after preprocessing. The table shows that the datasets used in our experiments exhibit a wide variety of characteristics leading to an evaluation as complete as possible.

3.4.2 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 GPUs with 16GB of memory for some of the experiments. Most of the experiments were conducted using only CPU nodes, as our experiments showed that training LSTMs on a GPU is less efficient than using a CPU. We include some of these results in our general runtime results. We explicitly mention when the experiment was run on a GPU node.

We split all datasets into a training and test part. We have chosen to split the datasets as is done in the implementation of the experiments by Camargo et al. [23]. The first 30% of cases are used for testing and the remaining 70% for training.

In this chapter we are only interested in predicting activities, therefore we have chosen to only use architectures that do not use time.

All Neural Network learning algorithms use an early stopping method with a patience of 42, meaning that when no improvement occurs for 42 epochs the learning algorithm stops. Increasing or decreasing this parameter has a direct influence on runtimes, but can also influence the obtained accuracy. The value of 42 was chosen because of its consistent use in all the implementations made by other authors. We did not perform hyperparameter optimization for the neural network methods, but rather used the parameter settings as proposed by the authors in their implementations.

To measure the performance of the next activity prediction we use accuracy, which gives us the fraction of activities 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 distance [29] between the predicted trace s_p and the correct trace s_e as the 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:

$$S(s_p, s_e) := 1 - \frac{DL\_distance(s_p, s_e)}{\max(len(s_p), len(s_e))} \tag{3.3}$$

An implementation of our method and code to execute the experiments can be found in our GitHub repository[2].

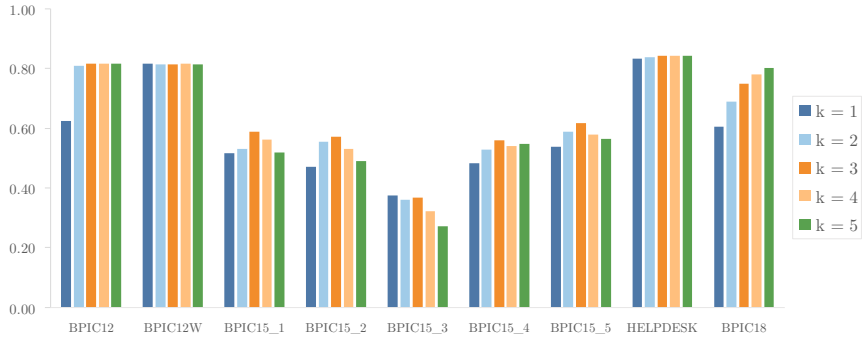
3.4.3 Evaluation

First, we examine the influence of k, which determines how many previous time steps the model takes into account, on the accuracy. We performed this test both for next event prediction and suffix prediction.

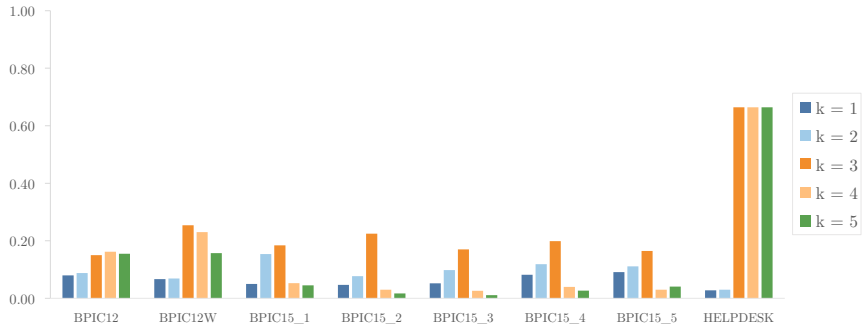
Figure 3.7a and 3.7b show the performance of our method for different k values. As DBNs only learn those dependencies that are most informative to describe the data, we expect that the accuracy would converge when the value of k becomes larger, as the model would no longer detect dependencies from attributes after a certain time step back. However, for the BPIC15 datasets, we see that the accuracy decreases for a higher k value. As these datasets are more dynamic, in that their underlying distribution changes drastically over time, this decrease can be explained by the fact that the model is overfitting the training data. Because for the other datasets the training data is a better representation for the data present in the test set, we do not observe this problem there. In the next chapter, we look at this problem and how we can update our model so that we can use a fixed value for k for any dataset.

Given that the optimal value of k for both prediction tasks widely differs between datasets, we have opted to include finding the optimal value of k as part of the

[2] https://github.com/StephenPauwels/edbn/tree/master/Predictions/BPM2020



(a) Next activity prediction



(b) Suffix prediction

Figure 3.7: Obtained accuracy on different datasets for varying k-values.

training process. That is, we iterate over different k values and use a separate validation set, extracted from the training data, to determine the best k value.

Figure 3.8a and 3.8b give a visual representation of the models learned for the Helpdesk and BPIC15_1 datasets using the 5 previous time steps available. These visual representations give us extra insight into the complexity of the data. When we compare both figures we can see that the more complex event log (containing more different activities) corresponds to a more complex model. The models can also confirm our initial explanation of why increasing the value of k leads to a decreased accuracy for the BPIC15_1 dataset but not for the Helpdesk dataset. As the DBN learns that the activities in the Helpdesk dataset only depend on the previous time step, the learning algorithm does not add any other time step to the dependencies no matter how many of these time steps are available in the data.

3.4.4 Comparison

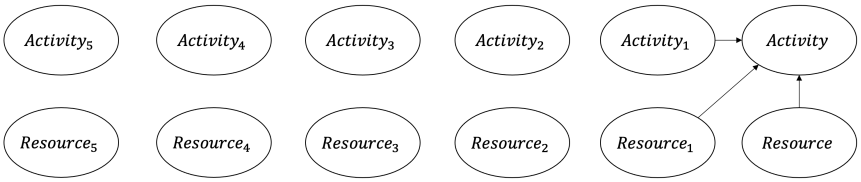
We compare our method with four well-known, state-of-the-art methods, all of which use neural networks. We used the implementations provided by Camargo et al. [23], Tax et al. [101] and Di Mauro et al. [35], which were slightly adapted to match our input format. We implemented the method described by Lin et al. [62] as correctly as possible according to the paper, as no implementation was

Table 3.5: Comparison of runtimes to train the model (in seconds). The first result for Camargo et al. corresponds to the shared category architecture, and the second to the specialized one.

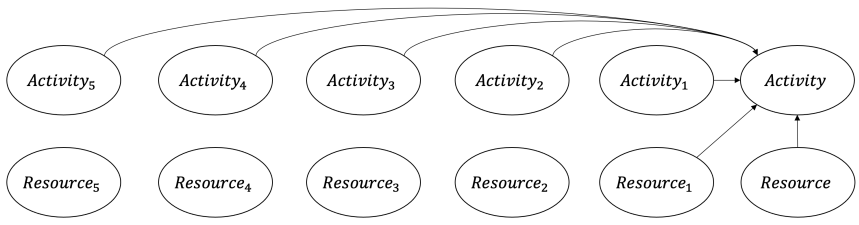
Dataset	DBN	Camargo	Lin	Tax	Di Mauro
BPIC12	66	16,681 - 13,251	5,736	14,123	1,418
BPIC12W	30	4,957 - 4,722	1,734	6,331	489
BPIC15_1	16	4,309 - 4,369	1,660	5,742	1,524
BPIC15_2	13	3,198 - 2,698	2,089	3,724	712
BPIC15_3	17	5,292 - 4,298	2,360	6,204	1,233
BPIC15_4	15	4,300 - 4,552	3,339	4,118	1,103
BPIC15_5	18	4,412 - 3,868	3,134	9,280	5,107
BPIC18	1,704	107,574 - 117,707	32,249	-	-
Helpdesk	11	1,951 - 3,516	730	1,273	45
BPIC12 (GPU)	-	> 21,600	> 21,600	13,969	328

available. All used implementations have been added to the GitHub repository. For the method proposed by Camargo et al., we ran both the *shared category* and *specialized* architectures. We used the activity and the resource of an event as input data for all datasets.

When comparing the runtimes of training the various models in Table 3.5 we can see that our method needs orders of magnitude less time to train than other methods. The training of our model consists of learning the structure and populating



(a) Helpdesk



(b) BPIC15_1

Figure 3.8: Models learned for the Helpdesk and BPIC15_1 dataset using a k-context with $k = 5$. The subscripts indicate the number of steps into history, with subscript 1 indicating the first previous time step and 5 the last available time step.

Table 3.6: Comparison of runtimes to predict the next events (in seconds). The first result for Camargo et al. corresponds to the shared category architecture, and the second to the specialized one.

Dataset	DBN	Camargo	Lin	Tax	Di Mauro
BPIC12	39	447 - 456	204	902	12
BPIC12W	8	166 - 174	82	243	8
BPIC15_1	9	133 - 145	71	258	9
BPIC15_2	8	144 - 149	72	386	10
BPIC15_3	10	163 - 166	79	389	9
BPIC15_4	9	116 - 116	63	269	9
BPIC15_5	12	165 - 165	78	510	10
BPIC18	490	4,994 - 5,396	2,410	-	-
Helpdesk	5	56 - 61	37	27	6

the different CPTs. The table confirms that a CNN can be trained more efficiently than the LSTM-based networks. A CNN has the added advantage that it can be learned efficiently using a GPU, whereas training LSTMs on a GPU is less efficient. Table 3.6 contains the runtimes for predicting the next event itself. We see that our model, together with the CNN model introduced by Di Mauro et al., outperforms the other methods.

We also tested training the models using GPU compute nodes. The runtimes for the BPIC12 dataset can also be found in Table 3.5. As our computing time on the nodes was limited we did not get results for some of the methods. The results show that there is no benefit in using GPUs over CPUs when training networks consisting of LSTM layers.

Next Activity Prediction

When looking at the comparison in Table 3.7 we see that our method performs on par with current state-of-the-art methods for predicting the next activity. 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.

Suffix Prediction

Table 3.8 shows that our method is not suitable for predicting the entire suffix of a running case. Our model uses extra attributes for predicting the next activity, we thus also need to predict the values for all attributes that belong to the parent set of the activity attribute. For every next event that we want to predict, we need to start from all these predicted values. This increases the probability that we predicted a wrong value which has a large impact on the activity that is predicted next. One idea to increase the performance of the predictions for the extra attributes is to allow them to use the recently predicted activity. Instead of predicting all attributes at the

Table 3.7: Comparison of the accuracies for next activity prediction. The first result for Camargo et al. corresponds to the shared category architecture, and the second to the specialized one.

Dataset	DBN	Camargo	Lin	Tax	Di Mauro
BPIC12	0.82	0.80 - 0.79	**0.83**	0.81	0.82
BPIC12W	**0.82**	0.80 - 0.80	0.81	0.80	0.74
BPIC15_1	0.53	0.46 - 0.50	0.60	**0.64**	0.60
BPIC15_2	0.55	0.44 - 0.45	0.45	**0.56**	**0.56**
BPIC15_3	0.36	0.33 - 0.35	0.38	**0.46**	0.40
BPIC15_4	0.56	0.44 - 0.48	0.52	**0.59**	0.52
BPIC15_5	0.62	0.45 - 0.47	0.61	**0.64**	0.58
BPIC18	**0.80**	**0.80** - 0.76	0.79	-	-
Helpdesk	**0.84**	0.80 - 0.76	0.82	0.80	0.83

same time, we can perform a two-step prediction. First, we predict the next activity, based on only the previous time steps. And next, we predict the extra attributes in the current time step by using all attributes from previous time steps and the activity of the current time step.

The runtimes shown in Table 3.9 are orders of magnitude higher than for predicting the next activity. We need to remark that much of the code provided by the different methods have significant room for efficiency improvements. Currently, most suffix prediction methods predict every next event separately, although the libraries used are optimized for batch prediction. An improvement would be to alter the code to facilitate batch prediction for all cases which have not yet been ended. This includes adding some extra housekeeping, but the cost for this housekeeping

Table 3.8: Comparison of the accuracies for suffix prediction. The first result for Camargo et al. corresponds to the shared category architecture, and the second to the specialized one. As the implementation by Di Mauro et al. does not provide the possibility of predicting the suffix, we did not perform these experiments using their method.

Dataset	DBN	Camargo	Lin	Tax
BPIC12	0.15	0.55 - **0.58**	0.18	0.15
BPIC12W	0.25	0.45 - **0.47**	0.12	0.09
BPIC15_1	0.18	0.49 - **0.57**	0.55	0.50
BPIC15_2	0.22	0.56 - **0.57**	0.51	0.39
BPIC15_3	0.17	**0.55** - 0.51	0.54	0.42
BPIC15_4	0.20	0.47 - 0.53	**0.58**	0.37
BPIC15_5	0.16	0.51 - 0.49	**0.58**	0.44
Helpdesk	0.66	**0.90** - 0.88	0.62	0.87

Table 3.9: Comparison of runtimes to predict the suffixes (in seconds). The first result for Camargo et al. corresponds to the shared category architecture, and the second to the specialized one. As the implementation by Di Mauro et al. does not provide the possibility of predicting the suffix, we did not perform these experiments using their method.

Dataset	DBN	Camargo	Lin	Tax
BPIC12	911	6,666 - 6,913	10,731	58,386
BPIC12W	273	2,369 - 2,586	3,833	66,752
BPIC15_1	34	1,113 - 1,015	778	8,568
BPIC15_2	33	1,090 - 1326	1,102	9,501
BPIC15_3	35	1,295 - 1,451	1,076	9,080
BPIC15_4	35	952 - 1,057	758	9,760
BPIC15_5	49	1,387 - 1,278	1,204	19,763
Helpdesk	26	50 - 54	67	76

is insignificant in comparison with the gained efficiency of not having to call the prediction function for every event separately.

3.5 Conclusion

In this chapter, we introduced our model based on Dynamic Bayesian Networks that is capable of predicting next events by modeling the known behavior in Business Processes starting from a reference event log. The model has the advantage of being easy to learn without the need for specialized hardware (such as GPUs). Another advantage over existing methods is that our model can handle unseen values, both unseen activities, and extra attributes. Where neural networks work like a black box, our model can be visually represented easily. This visual overview of the model can then be inspected by users and be used to learn extra aspects of the data. This could lead to users having more confidence in the learned model. In Chapter 7 we show in more detail how our model can be used to explain the returned results.

We looked at the performance, in terms of accuracy, of our model for both next activity and suffix prediction. Our experiments show that our model performs on par with state-of-the-art methods for predicting the next activity. While achieving good results for predicting the next activity, we also see that our model is, in its current form, not suitable for predicting the suffix of a running case. This is because it depends on extra attributes which in their turn have to be predicted for every event in the suffix. Besides the qualitative experiments, we also compared the methods in terms of the time needed for training. These experiments show that our model outperforms all other methods.

An interesting observation made during our experiments is that our DBN learns reasonable simple models, with not many dependencies between the various attributes. This may raise the question of whether more complex models such as neural networks are the best choice to solve this problem. In the next chapter, we

will look further into this by introducing a simple, less complex neural network architecture.

We 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 prediction techniques for business processes. Also, a better understanding of the different experimental designs is needed; in Chapter 5 we take a first look at how evaluations for next activity predictions are performed across different papers. To further improve our DBN method we show in the next chapter that our model is easily updatable and can adapt to occurring drifts in the data. As already indicated we could also improve the suffix prediction using our DBN by introducing a two-step prediction phase, where we first predict the activity of the next event and, using this predicted activity, predict the remaining attributes from the event. For future research, it would be interesting to investigate how DBNs and the other next activity prediction methods could be used in the setting of outcome prediction. We could use the same mechanics as explained in this chapter, only changing the target the model learns from the activity to the outcome of the case.

CHAPTER 4

Incremental Predictive Process Monitoring

Next activity prediction methods for business processes are always introduced in a static setting, implying a single training phase followed by the application of the learned model during the test phase. Real-life processes, however, are often dynamic and prone to changes over time. Therefore, all state-of-the-art methods need regular retraining on new data to be kept up to date. It is, however, not straightforward to determine when to retrain nor what data to use; for instance, should all historic data be included or only new data? Updating models that still perform at an acceptable level wastes a potentially large amount of computational resources while postponing an update too much will deteriorate model performance. In this chapter, we present incremental learning strategies for updating these existing models that do not require fully retraining them, hence reducing the computational resources needed while still maintaining a more consistent and correct view of the process in its current form. As an alternative to adapting existing methods, we also introduce a basic neural network method consisting of a single dense layer. This architecture makes it easier to perform fast updates to the model and enables us to perform more experiments. We investigate the differences between the proposed incremental approaches. Experiments with a prototype on real-life data show that these update strategies are a promising way forward to further increase the power and usability of state-of-the-art methods.[1]

[1] This chapter is based on *Stephen Pauwels and Toon Calders. Incremental Predictive Process Monitoring: The Next Activity Case. In International Conference on Business Process Management (BPM '21), pages 123–140, 2021.*

4.1 Introduction

Predictive process monitoring uses historical data to predict several aspects of ongoing business processes, such as remaining time prediction, outcome prediction, and next activity prediction. Recently proposed next activity prediction methods always assume a static setting, where we divide the datasets into fixed training and test parts. One important aspect of Business Processes, however, is that they are inherently dynamic and that different time periods in the log can have different characteristics. Although some authors propose to retrain the model regularly to incorporate the changes in the data, this might not be the most efficient way of updating a model in terms of both runtime and accuracy. Furthermore, no method describes which events to use to retrain the model. It can be beneficial not to use all available historical events as they are no longer relevant for future activities, due to drifts in the processes [15]. No state-of-the-art method proposes how to perform these updates or has been evaluated under these dynamic circumstances.

In this chapter, we explore different strategies that can be used for incremental learning of next activity prediction models, ranging from completely retraining the models for every new batch of data to only using the newly arrived data to update the existing model. These update strategies can shed new light on the performance of the different methods, as not all methods are equally well suited to be adapted for a dynamic environment.

It is important to note that the proposed strategies apply to a variety of existing methods, especially neural networks. We aim at showing that the update strategies have significant benefits in a dynamic environment independent of which of the described models is better.

To visualize the performance over time (with varying amounts of data used for training/updating) we propose a graphical representation of the average accuracy on a given time within a given window. This sliding window technique gives an accurate view of how the predictive performance of the models changes over time.

We show that some incremental methods outperform the completely retrained models despite the *catastrophic forgetting* [94] property of neural networks. Catastrophic forgetting is the phenomenon that a neural network increasingly forgets and ignores the original input when retraining the model with new data. This phenomenon is a potential risk when updating neural networks. However, in this chapter we swill show that we can leverage catastrophic forgetting to gradually forget older, less relevant, events in the presence of concept drift.

Because we needed a lightweight neural network for our initial experiments, we created a basic neural network architecture that takes a k-prefix as input, has a single dense layer, and gives a probability distribution over all possible activities for the next time step as output. During the experiments, we show that this new architecture (which requires only a limited amount of computational resources) performs on par and in some cases even outperforms the selected more complex state-of-the-art architectures.

Our contributions are the following:

1. We introduce a simple, but accurate, neural network architecture for next-activity prediction.
2. We compare different update strategies in terms of accuracy and runtime.
3. We establish that updating models in case of drift can have a significant effect on performance on the one hand, and does not hurt performance in the absence of drift on the other.

The next section gives an overview of related work on incremental learning, concept drift, and predictive process monitoring and positions our work within the field of incremental predictive process monitoring. In Section 4.3 we explain the different strategies that can be used for updating. In Section 4.4 we introduce our basic neural network architecture. Experiments on all possible update strategies are performed in Section 4.5.

4.2 Related Work

Incremental Learning Algorithms

Different applications exist that can deal with the dynamic nature of business processes. Some of these applications that are adapted for use in an online setting are: business process discovery [21], conformance checking [20] and concept drift detection [75].

Learning in a setting in which data changes its nature and characteristics over time is a known and well-studied problem within machine learning [42, 88]. A popular solution is to use a sliding window to move over the data that indicates which data to use for building a model at a particular moment. One of the downsides of such a sliding window is that they are often of a fixed size, which is determined a priori by the user. This can lead to window sizes that are too big, and thus too insensitive for changes, or too small, in which case too many changes are detected. The correct window size depends on the data itself, and can also vary over time. Therefore, Bifet and Gavalda proposed a learning technique that uses an adaptive window size (`ADWIN`) [12]. On the one hand, when the data is stationary the window size grows, and on the other hand, the window size shrinks when changes are detected. In contrast to other proposed adaptive methods, Bifet and Gavalda show that the performance of their adaptive window is guaranteed by providing bounds on the false positive and false negative rates.

Another application of incremental learning algorithms is when the data arrives in the form of a data stream in which the compute resources are not able to keep all the arrived data in memory. Hoeffding trees, as proposed by Domingos and Hulten [36], are incremental decision trees that are learned from a massive data stream. This method does, however, assume that the distribution generating the arriving samples does not change over time. Hoeffding trees can be learned in a constant time proportional to the number of attributes.

Gama et al. [41] consider concept drift as described by Bose et al. [14, 15] but propose the use of different incremental algorithms to deal with these changes.

This incremental learning overcomes, by constantly updating the learned models, the issue of concept drift often being unexpected and unpredictable. Incremental learning is thus able to update the model on time, well ahead of models using a concept drift detection method. We continue some of the ideas presented by Gama et al. and further elevate them for use with neural networks and next activity prediction.

Incremental Predictive Process Monitoring

Maisenbacher et al. [66] and Di Francescomarino et al. [34] use the incremental learning algorithms mentioned above to create incremental models that can predict the outcome of a running case. The main focus of their work is using incremental classifiers that can classify ongoing traces based on their predicted outcome. Maisenbacher et al. look at different existing approaches, like the ones described above, and explore the performances of these approaches when applied to business processes. Di Francescomarino et al. focus on a clustered-based and index-based technique to predict the outcome for an ongoing trace. One of the disadvantages of their work is that they do not show if existing methods could be adapted to incorporate the incremental learning aspect. Their work does indicate the potential benefit of incremental learning in predictive process monitoring.

Berti et al. [9] propose a method for remaining time prediction that can deal with concept drift in data, by only training on the relevant part of the data that correctly behaves according to the current business process. One disadvantage of their approach is that they need existing concept drift detection methods (like the one proposed by Bose et al. [15]). Knowing which intervals behave in a static way they propose to use distance functions between an ongoing trace and the traces present in the current static interval. Using this distance function they calculate the reliability of a trace in the context of predicting the remaining time for an ongoing trace. In contrast to the approach proposed by Maisenbacher et al., Berti et al. require some a priori knowledge about the different drifts present in the data, making it less suitable for online use.

4.3 Update Strategies

Different strategies exist that deal with the presence of drifts in the data. We divide them into two main categories; the first category trains a new model (*reset*), the second category updates the existing model (*update*). In the remainder of this chapter. Next, we take a look at how we can select the data used for learning the model.

4.3.1 Data Selection

Relearning a model after every event is both infeasible and unnecessary. Therefore we divide the event log into windows of a certain size. These windows can be divided by specifying the number of events per window, or by specifying a time interval for every window (days, weeks, months, ...). A log consists of an ordered list of windows $w_0, w_1, \ldots, w_n$ where w_t is the window of all events that occured at timestamp t and

n the latest time present in the log. We have for w_i and w_j with $i < j$ that all events in window w_i occurred before all events in window w_j.

Using these windows we can define three different strategies to determine which data to use for learning a model at time t. The first strategy uses all historical data to learn the model (w_0, w_1, …, w_{t-1}). The next strategy only takes a limited, fixed amount of the immediate history (windows) into account ($w_{t-\ell}$, $w_{t-\ell+1}$, …, w_{t-1}) with ℓ the number of windows to consider). The last method retrains its model when a drift occurred and updates the model after every window with all windows starting from the last drift point up to the current window (w_{d_t}, w_{d_t+1}, …, w_{t-1}) with d_t the time of the latest detected drift.

4.3.2 Update Existing Methods

We can combine all data selection strategies with both the reset and update methods. This leads to six different options for incremental predictive models. The options using the reset strategy can be used straightforwardly with existing methods. When using the update options, existing approaches possibly need extra adaptation. In this chapter, we consider both neural net methods and Dynamic Bayesian Networks. Both these types of methods are already learned in an iterative process, and can thus easily be extended to our incremental approach. Every iteration during training results in a (slightly) adapted model. When, according to a selected loss function, this model performs better than the previous one, we keep this model to start the next iteration with. Updating these models thus only implies that we have to perform extra iterations on the existing model using our updated data.

Next, we describe in more detail how the incremental aspect can be added to both neural networks and dynamic bayesian networks based on how these models are learned from data.

Drift-based Predictions

We use the method proposed by Bose et al. [15] to detect drifts present in the data. This method generates features on which we can apply statistical tests to test if two samples originate from the same distribution or not. To update the model, we retrain the model after every batch using all available data starting from the last seen drift until the most recent used batch of events.

Incremental Neural Networks

Training a neural network is an iterative procedure that tries to optimize a certain model score. This score indicates how accurate the current model is for predicting events (validates the model). Neural network learners use a subset of events from the training data for this validation and the remaining events for actually updating the weights of the model. To update a neural network, we start from the existing model and perform a fixed number of additional training epochs using a different set of training data, which is again split into a training part and a validating part. While performing these additional epochs, we keep the hyperparameters fixed to their initial values.

Using new data for updating the model causes the model to diverge from what it originally learned, as it is now validating using new data. Therefore, the model can potentially be less optimal for the original data that was used for the initial training. This is the typical catastrophic forgetting phenomenon that occurs when updating a neural network. For a dataset that contains drift, this is, however, a positive feature rather than a weakness.

As all considered neural network methods train their model in the same way, we can use this incremental method on every proposed neural network model such as the Single Dense Layer (SDL), LSTM, and CNN-based methods.

Incremental Dynamic Bayesian Networks

Dynamic Bayesian Networks consist of a model structure and model parameters. Both the structure and parameters are learned from data and can be trained/updated separately.

The structure of a DBN describes the conditional dependencies between attributes in the data. We learn the model using a hill-climbing algorithm, where we perform iterations of adding or removing dependencies as long as these actions improve the model score. To update the structure we run this algorithm on the existing model. While improvements can be made to the structure, the algorithm will continue to perform iterations.

The parameters of the DBN describe the conditional probabilities for all attributes, and can be calculated as follows:

$$P(A|Pa(A)) = \frac{P(A \cap Pa(A))}{P(Pa(A))} \tag{4.1}$$

Where $Pa(A)$ are the attributes on which A depends.

To easily compute these probabilities, we create an inverted index for every attribute. Instead of keeping track of the different values that occur in a single row, we keep track of the rows in which a certain value occurs. Due to this inverted index, the DBN method is very suitable for an incremental setting. Updating a CST is easy, but when a dependency is added or removed CSTs have to be recomputed from scratch. By using inverted indexes for the different attributes we do not have to maintain an explicit model of the CSTs, as the probabilities are computed on-the-fly.

Example 12. *Consider the event log as shown in Table 4.1a. We can create an inverted index for the attributes activity, role, and department by listing for every value of the attributes in which event they occur using the event ID. The inverted indexes are shown in Table 4.1b, 4.1c, and 4.1d.*

For calculating a conditional probability we can use set operations when dealing with categorical values only:

$$P(act = A|role_1 = r_1, dept_1 = d_0) = \frac{P(act = A, role_1 = r_1, dept_1 = d_0)}{P(role_1 = r_1, dept_1 = d_0)} \tag{4.2}$$

$$= \frac{\#(\{0\} \cap \{0,1\} \cap \{0,1,3\})}{\#(\{0,1\} \cap \{0,1,3\})} = \frac{1}{2} \tag{4.3}$$

Table 4.1: Example log with extra attributes role and department (a) and the inverted indexes for activity (b), role (c) and department (d). We use the subscript 1 to indicate the attribute in the previous time step.

(a) Example event log

eventID	*activity*	$role_1$	$department_1$
0	A	r_1	d_0
1	B	r_1	d_0
2	C	r_2	d_1
3	B	r_2	d_0

(b) Activity

Value	IDs
A	{0}
B	{1,3}
C	{2}

(c) Role

Value	IDs
r_1	{0,1}
r_2	{2,3}

(d) Department

Value	IDs
d_0	{0,1,3}
d_1	{2}

To further improve our implementation we can only keep track of the counts we need in Equation 4.3. To update the model, we increment the values that correspond with the combinations of the nominator and denominator.

4.4 Reference Model: Single Dense Layer (SDL)

Incrementally updating existing models adds a level of complexity and cost. To reduce the overall cost of learning and updating the models we propose a neural network based on a single dense layer for predicting the next event. We did not find any paper exploring the use of simple NNs with only dense layers, without the need of using Petri Nets as intermediate structure [105]. As we show in Section 4.5, such a shallow fully-connected network benefits from low runtimes and high accuracy both in an incremental and non-incremental situation.

We use the k-prefix as input to capture the sequential nature of the event log in the network. The SDL network consists of a layer of input cells, one cell for every attribute in every time step in the k-prefix. The selection of these attributes depends on the data used and can vary significantly between datasets. We encode the data using an encoding layer, which encodes the data using one-hot encoding. In the case of numeric attributes, we do not have to encode the values and can use them as-is. These cells are then concatenated before they are linked with a dense layer with as many cells as there are activities and a dropout of 0.2, as also used by other methods. As the output layer, we use a softmax layer, ensuring that the network returns a probability distribution over all possible events. All input cells $i_0, i_1, \ldots, i_n$ represent the activities and extra attributes present in the entire prefix. We thus have $n = |\mathscr{A}|k$ input cells in the network, with $\mathscr{A}$ the set of all considered attributes (from both activity and data perspective), and k the size of the history taken into account. The encoding cells $e_0, e_1, \ldots, e_n$ create a one-hot encoding of all inputs.

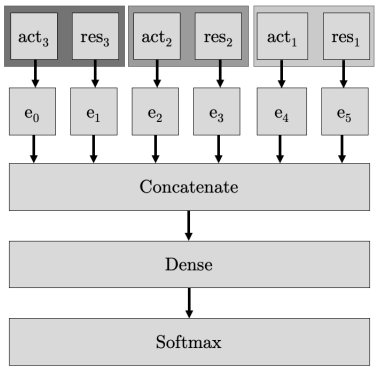


Figure 4.1: Example SDL network using two attributes, activity and resource, from 3 previous time steps.

Example 13. *Suppose we have an event log containing both the activity and resource executing the activity. If we want to create a model with prefix size* $k = 3$*, we have the following input cells:* $i_0, i_1, i_2, i_3, i_4, i_5$*.* i_0, i_2, i_4 *represent the activities and* i_1, i_3, i_5 *the resources for the 3 time steps in the prefix. A visual overview of the resulting network can be found in Figure 4.1.*

4.5 Experiments

To test the different update strategies we selected 4 methods, each using a different type of model. These methods were selected based on their diversity to cover a wide variety of state-of-the-art methods with shown performance. For this study we use the following methods:

- **Dynamic Bayesian Networks (DBN)**
- **Single Dense Layer NN (SDL)**
- **Long-Short-Term-Memory NN (LSTM)**: Tax et al. [101]
- **Convolutional Neural Network (CNN)**: Di Mauro et al. [35]

In the first place, we are interested in how the accuracy of the different methods changes over time. We define accuracy as the portion of correctly predicted activities. For this purpose, we use a sliding window of fixed size for which we calculate the accuracy obtained within this window. We then use a graphical representation with the event index (in chronological order) on the X-axis and the window accuracy on the Y-axis. This *accuracy-plot* gives us an easy tool to compare different methods and to see how the accuracy changes over time. We can use these graphs to see if the tested model does suffer from drifts in the data and if it can recover from changes.

As described in section 4.3, first a batch size has to be determined that indicates the frequency of performing an update to the model. In our study we tested three different ways of dividing the data into batches; by day, week, or month. Preliminary

tests show only minor differences between these batch sizes. Therefore, we decided to use monthly batches for our experiments.

During our experiments, we are interested in the difference between all update strategies and how a model that uses updates, performs in contrast to a static one. We are thus less interested in determining the single best model to use when incorporating updates in our next activity predictor. We set the various hyperparameters as proposed by the authors in their implementation. These hyperparamaters remained fixed during all experiments to ensure that we can observe the influence of our update strategies independent of other model specific settings.

When evaluating the performance of the update strategies we use an *interleaved-test-then-train* approach. Each batch of data is first used to test the model before we use it for updating the model. All code used for the experiments can be found in our Github Repository[2].

4.5.1 Dataset Selection

To test the update capabilities of the models in the best possible way, we need datasets where some drifts occur in the activity perspective. We start with looking at the following datasets which are often used in the literature and have different characteristics:

- **Helpdesk** [115]: a log containing ticket requests of the helpdesk from an Italian software company
- **BPIC11** [113]: a log of a Dutch academic hospital. It shows the different activities and phases the patients go through.
- **BPIC12** [111]: a log containing applications for personal loans. The log contains three intertwined processes.
- **BPIC15** [112]: a log containing building permit applications from five different Dutch municipalities. The log is splitted into five sublogs, one for every municipality (**BPIC15_1** to **BPIC15_5**).

The events in the datasets were first sorted according to timestamp and then split 50/50 in train and test set in chronological order. The details of the different datasets can be found in Table 4.2. The train set is used to train the initial model and the test set is first used to test the method and then to update the initial models. To answer our research questions, we use datasets that do contain variation over time in the activity perspective. To find out if any drifts occur in the datasets we train a model on the first half of the data. We then test the remaining half using this static model. Using our accuracy-plot, which uses a sliding window to calculate the accuracy, we can then look for changes in accuracy over time.

Figure 4.2 shows some (small) changes in accuracy over time for the Helpdesk, BPIC11, and BPIC15 datasets when not using any update strategy (the darker colored graphs). The BPIC15 datasets suffer the highest loss of accuracy. This experiment also shows the constant performance of the BPIC12 data. This dataset is useful to see the performance of our incremental learners when no drifts are present. Ideally, the incremental algorithms should perform similarly to the non-incremental ones.

[2] https://github.com/StephenPauwels/edbn/tree/master/Predictions/BPM2021

Table 4.2: Number of days, weeks and month in the train and test parts used for all datasets.

	Train			Test		
Dataset	#days	#weeks	#months	#days	#weeks	#months
Helpdesk	623	97	23	683	112	27
BPIC11	622	90	21	550	79	19
BPIC12	87	14	3	80	12	4
BPIC15_1	562	112	26	595	122	30
BPIC15_2	555	116	28	550	117	28
BPIC15_3	678	117	29	604	117	28
BPIC15_4	621	126	30	516	107	26
BPIC15_5	566	112	27	569	121	29

4.5.2 Baseline Comparison

A first question that needs to be answered is how much accuracy gain there is when using an incremental algorithm in contrast to the non-incremental ones. Figure 4.2 shows the evolution of the accuracy over time for the four methods on all datasets. With the accuracy calculated using a sliding-window of size 1000. This experiment shows the need for an incremental approach when utilizing prediction models in a real-life setting. The results from the BPIC15 datasets show a large increase in accuracy. Although the models first seem to suffer from the drift in the data, they all recuperate fast and can keep fairly high accuracies, much higher than the ones seen in the literature for this dataset.

This graph also shows another important aspect of the incremental algorithms; as we do not know in advance if and when drift occurs, the data may contain no drift at all. Our results on BPIC12 show that the incremental approaches have the same accuracy as the non-incremental ones. This indicates that we can use the incremental versions in all situations, without having to compromise on accuracy.

4.5.3 Update Strategy

To compare the different incremental approaches we introduced, we selected a single dataset and ran all options for all methods. Initial experiments show comparable behavior for other datasets. Figure 4.3 shows three different observations.

1. We can see that there is little difference between the options for the DBN method. This can be explained by the fact that the DBN model has no way to forget older events.

2. The SDL and LSTM (Tax et al.) methods show similar behavior. Using all data to update the model consistently shows the worst performance, as using the full dataset ensures that the model is unable to forget the older events that became irrelevant. We see that using an update strategy using a window scores the best for both methods, thus making use of the catastrophic forgetting to

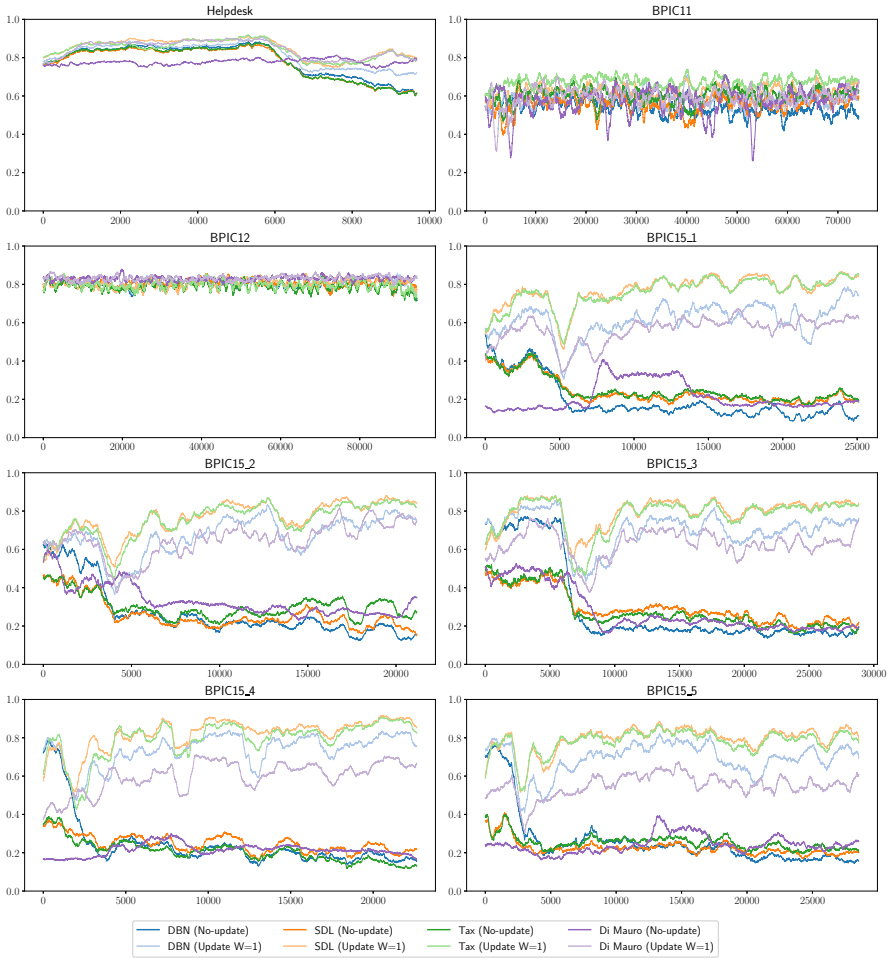


Figure 4.2: Accuracy-plot for both a static model and an incremental model using montly batches and a window size of 1. The graphs are created using a sliding-window of 1000 events.

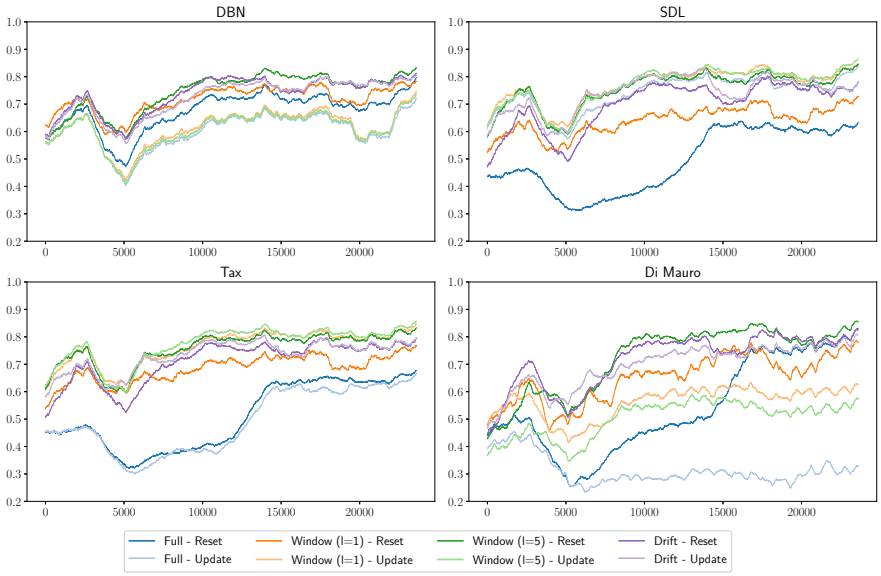


Figure 4.3: Accuracy-plot for the different update strategies for the BPIC15_1 dataset. The graphs are created using a sliding-window of 1000 events. For every method we test the various strategies that can be applied. For the window-based strategies we look at windows of only a single batch and five batches.

gradually replace the older with newer knowledge. The use of drift detection also gives good results, but this involves running an extra algorithm for every batch to decide whether a drift has occurred.

3. The architecture using convolutional neural networks (Di Mauro et al.) shows different behavior; in the accuracy plot linked with this method in Figure 4.3 we see that the reset strategies achieve the highest accuracy, while the update strategies fail to improve their overall accuracy. This is visible by the large difference between the darker and lighter curves in the graph. This can be due to the difference in nature (e.g. the ability to forget less) of the convolution layers used in this model.

4.5.4 Runtime Results

Table 4.3 shows the average time for each update using the different strategies. Overall we see that SDL is the fastest algorithm to learn, followed by DBN, Di Mauro, and the LSTM architecture of Tax show to be the slowest. These results are in line with earlier made observations that models that use the LSTM architecture scale worse than other methods. The SDL architecture is thus well suited for use within an incremental situation.

Table 4.3: Runtime (average *(stdev)*) for the update iterations for the BPIC15_1 dataset (in seconds)

Strategy	Batch	DBN	SDL	Tax	Di Mauro
Training		48	78	433	188
Reset	Full	106 *(25)*	73 *(18)*	985 *(394)*	450 *(132)*
	Window (size 1)	17 *(4)*	5 *(1)*	37 *(10)*	20 *(9)*
	Window (size 5)	26 *(6)*	14 *(1)*	175 *(39)*	68 *(14)*
	Drift	28 *(15)*	14 *(10)*	187 *(178)*	98 *(87)*
Update	Full	60 *(12)*	56 *(12)*	505 *(101)*	132 *(27)*
	Window (size 1)	1 *(0.5)*	1 *(0.5)*	11 *(4)*	3 *(1)*
	Window (size 5)	7 *(0.8)*	6 *(1)*	59 *(7)*	15 *(2)*
	Drift	3 *(5)*	2 *(1)*	13 *(6)*	6 *(7)*

4.5.5 Overall Results

Table 4.4 shows an overview of the accuracy obtained by all methods, using the different update strategies on all datasets. This table confirms the behavior that we saw in the previous experiments. We see consistent results for all different methods.

These results also show the performance of our SDL method in comparison to existing methods (both with and without the incremental aspect). We see that our new architecture performs on par with existing methods. On top of that, as the complexity of our model is fairly low, training this model takes considerably less time than training the existing methods.

Table 4.4: Average accuracy for all methods on all different update settings (batches are grouped by month).

Dataset	Strategy	Batch	DBN	SDL	LSTM	CNN
Helpdesk	No-update		0.78	0.77	0.77	0.78
	Reset	Full	0.82	0.80	0.78	0.81
		Window (size 1)	0.82	0.83	0.83	0.82
		Window (size 5)	0.84	0.84	0.82	0.83
		Drift	0.83	0.82	0.81	0.82
	Update	Full	0.81	0.80	0.77	0.80
		Window (size 1)	0.81	0.85	0.84	0.84
		Window (size 5)	0.81	0.84	0.83	0.84
		Drift	0.83	0.85	0.84	0.84

Table 4.4: (continued)

Dataset	Strategy	Batch	DBN	SDL	LSTM	CNN
BPIC11	No-update		0.54	0.56	0.60	0.57
	Reset	Full	0.58	0.62	0.66	0.63
		Window (size 1)	0.51	0.56	0.58	0.55
		Window (size 5)	0.58	0.63	0.66	0.62
		Drift	0.56	0.60	0.64	0.60
	Update	Full	0.60	0.62	0.60	0.64
		Window (size 1)	0.58	0.62	0.67	0.59
		Window (size 5)	0.58	0.63	0.68	0.60
		Drift	0.57	0.60	0.62	0.58
BPIC12	No-update		0.80	0.81	0.79	0.83
	Reset	Full	0.81	0.81	0.79	0.83
		Window (size 1)	0.79	0.79	0.80	0.82
		Window (size 5)	0.81	0.81	0.80	0.83
		Drift	0.81	0.81	0.79	0.83
	Update	Full	0.81	0.81	0.79	0.83
		Window (size 1)	0.81	0.80	0.80	0.83
		Window (size 5)	0.81	0.81	0.80	0.83
		Drift	0.81	0.80	0.79	0.83
BPIC15_1	No-update		0.20	0.24	0.25	0.21
	Reset	Full	0.68	0.50	0.52	0.55
		Window (size 1)	0.71	0.64	0.68	0.65
		Window (size 5)	0.74	0.75	0.75	0.71
		Drift	0.73	0.69	0.70	0.71
	Update	Full	0.61	0.74	0.50	0.32
		Window (size 1)	0.62	0.76	0.76	0.56
		Window (size 5)	0.61	0.76	0.77	0.50
		Drift	0.72	0.73	0.73	0.69
BPIC15_2	No-update		0.27	0.26	0.30	0.34
	Reset	Full	0.68	0.48	0.51	0.74
		Window (size 1)	0.71	0.60	0.66	0.61
		Window (size 5)	0.73	0.73	0.74	0.74
		Drift	0.73	0.67	0.69	0.71
	Update	Full	0.67	0.73	0.49	0.42
		Window (size 1)	0.67	0.76	0.75	0.65
		Window (size 5)	0.66	0.76	0.76	0.65
		Drift	0.73	0.72	0.71	0.66

Table 4.4: (continued)

Dataset	Strategy	Batch	DBN	SDL	LSTM	CNN
BPIC15_3	No-update		0.30	0.30	0.28	0.28
	Reset	Full	0.71	0.50	0.53	0.51
		Window (size 1)	0.74	0.69	0.71	0.71
		Window (size 5)	0.76	0.77	0.78	0.72
		Drift	0.76	0.72	0.72	0.74
	Update	Full	0.70	0.76	0.51	0.40
		Window (size 1)	0.70	0.79	0.78	0.63
		Window (size 5)	0.70	0.79	0.79	0.58
		Drift	0.76	0.77	0.76	0.75
BPIC15_4	No-update		0.25	0.25	0.21	0.21
	Reset	Full	0.74	0.52	0.54	0.60
		Window (size 1)	0.73	0.68	0.71	0.69
		Window (size 5)	0.79	0.79	0.79	0.74
		Drift	0.75	0.58	0.58	0.61
	Update	Full	0.74	0.79	0.52	0.40
		Window (size 1)	0.74	0.81	0.79	0.60
		Window (size 5)	0.73	0.81	0.81	0.54
		Drift	0.75	0.79	0.79	0.65
BPIC15_5	No-update		0.27	0.23	0.26	0.24
	Reset	Full	0.68	0.50	0.52	0.58
		Window (size 1)	0.74	0.68	0.71	0.71
		Window (size 5)	0.75	0.76	0.77	0.71
		Drift	0.76	0.71	0.72	0.74
	Update	Full	0.67	0.75	0.50	0.37
		Window (size 1)	0.69	0.79	0.78	0.55
		Window (size 5)	0.68	0.78	0.78	0.50
		Drift	0.76	0.77	0.76	0.72

4.6 Conclusion

In this chapter, we looked at the situation in which we want to predict the next activity in a business process when the underlying process can be subject to change. We changed the way we evaluate the performance of the various models from the standard setting of using a static train and test set to a dynamic setting where we update the model after testing a certain batch of events. We create these distinct batches based on the day, week, or month of the events. This setting reflects more realistic situations as we now use all data available for predicting and not only a fixed reference event log that might be out-of-sync with the current events arriving.

To update the models we looked at two different aspects. The first aspect considers how to update the models themselves, and the second considers what data we use for performing the update. Both model update options can be combined with all data selection methods. The data selection methods are the same for all considered methods, but how we update the model is different between the DBN and the Neural Network methods.

Updating the DBN model is easy and can be performed rapidly, as the different Conditional Dependency Tables are implemented based on inverted indexes. The results show that both update strategies improve the overall accuracy. However, the reset method performs better than the update method. We can explain this by the fact that our DBN is not able to forget events he has previously learned when using the update method. Only when using the reset method does it reset its CPTs and fill them back up with only the new data.

To be able to quickly test the performance of Neural Networks in this dynamic setting, we implemented a model with a simple architecture that takes the different values in the k-prefix as inputs and mainly consists of a single dense layer with the size equal to the number of activities that are present in the training log. Our results show that this model performs on par with existing methods, both in a static and dynamic situation. These results raise the question of whether complex architectures are beneficial for the prediction problem we want to solve. The results show that using the update method performs best in the dynamic setting. But they also show an improvement in quality for all methods that use any of our newly proposed incremental strategies. A major advantage of this method is the time required for training the model. We show that it outperforms the other Neural Network methods, without the need for specialized hardware (such as GPUs).

When we look at the results obtained by the other Neural Networks architectures we can see that the obtained quality of the results for both methods increases for all tested incremental methods. We do see that using the update strategy is more suitable for the LSTM method, while a reset method is more suitable for the CNN method.

We performed our experiments on a variety of datasets, each with its own characteristics. Some of these datasets contain less drift than others. The results on these datasets (such as the BPIC12 data) show that our incremental method also performs well (has no significant loss in accuracy) when no drift is available, making the method suitable to run when we do not know if drift is going to occur or not.

We see an interesting avenue for future research in improving the incremental DBN method. In Chapter 7 we will show that our model can also be used for drift detection. By combining the drift detection with these incremental update strategies

we could improve the quality of the DBN model. The main concept would be to perform an update on the model when no change has been detected, and reset the model when a change has been detected. Using the same model for both tasks has the additional advantage of not needing to perform a different (potentially complex) drift detection method.

As we only use four different methods, we cannot make strong conclusions about the best way of solving the incremental next activity prediction problem. This is often highly dependent on the characteristics of the considered process. We showed that we can leverage existing state-of-the-art methods to cope with variation and drifts in the data by adding a basic incremental framework. Some of the datasets used in our experiments often get ignored in the literature due to their dynamic nature. We show that, when using a suitable update strategy, most methods are ready to be used in a more challenging environment than the test settings and datasets used most of the time in literature.

CHAPTER 5

Evaluating Next Activity Prediction Methods

Recently several new methods for next activity prediction in business process event logs have been proposed in the literature. In this chapter, we critically evaluate the reported performance figures and identify and illustrate several pitfalls in the way methods are evaluated in the field. Experiments are set up differently by different authors, often without knowing what consequences the different choices can have on the reported accuracy of the methods. We show that seemingly small variations in how the data is prepared and how experiments are set up can lead to significant changes in the resulting predictions. These changes can lead to misleading claims even more so as there is a tendency in this domain to copy accuracy results from other papers that were produced under different settings. We show the impact of this practice; based on a rigorous study of the papers and supplementary code of other authors, a multitude of setup choices is found, some of which have a large impact on the overall reported accuracy of several state-of-the-art methods. For instance, the way data is split in training and test sets or whether or not artificial start and end events are introduced turn out to have a significant impact on the reported accuracy numbers, rendering the copying of performance figures from other papers an unreliable basis for comparison. A major contribution is the comparison of several state-of-the-art methods against a well-motivated, uniform testbed, leading to new insights regarding the perceived progress of the field.[1]

[1]This chapter is based on *Stephen Pauwels and Toon Calders. Evaluating Next Activity Predictions:What Could Go Wrong? Submitted, 2022.*

5.1 Introduction

In recent years, many new methods for predicting the next activity have been proposed, mostly based on Deep Neural Networks (DNNs). DNNs already proved their usefulness in other domains that consider sequential data [32]. Different surveys in the field of next activity prediction have been conducted to get a clear overview of what exists and how each of the methods performs on the standard datasets [33, 51, 102, 104, 118]. The main goal of these papers is to get an overview of the used methods and techniques and determine which setting is best for which type of data.

One aspect that is often overlooked in these surveys is a detailed overview and in-depth analysis of the evaluation techniques used to test next activity prediction algorithms. With new methods appearing frequently, a plethora of evaluation techniques and preprocessing choices can also be found in the literature. Some authors use the implementations of existing techniques to rerun experiments using the same setup they use for testing their own method. However, an often used method is to copy published performance figures from the papers introducing the existing methods, without taking into account what preprocessing steps were performed on the data or how the data was split into train and test data. Although at first the influence of these choices may seem negligible, in this chapter we show that changes to the data can have a significant influence on the obtained score, making it dangerous to directly compare performance figures from different publications.

Ketyko et al. [49] and Weytjens and De Weerdt [119] confirm these initial observations and propose their solutions to the problem. Ketyko et al. focus on predicting the suffix of a running case. They introduced a framework in which existing methods fit, and compare how different methods perform. Besides the standard metrics they introduce graphs that can give more detailed information about the performance of the methods in various situations. Weytjens and De Weerdt propose to create fair datasets by eliminating bias that can be present in real-life event logs. The problems they describe are more related to remaining time prediction and suffix prediction, and thus less relevant for next activity prediction. In this chapter, we show the actual differences that occur when the experiments are not performed in the same way. We show that we can create fair comparisons by following some simple guidelines to ensure that we simulate real-life scenarios.

To get an overview of what choices are made when designing evaluations, we first take a look at some state-of-the-art methods and how the authors performed their experiments. Based on these findings we formulate five choices that every paper has made (either implicitly or explicitly). For every choice, we take a look at what decision was taken, showing a wide variety of different settings used throughout the literature.

Our objective is to investigate how these choices affect the performance measurements. Based on our initial findings of how evaluations are performed, we introduce a basic test setting and strategy. We use this baseline strategy for testing the influence on the obtained prediction score for different state-of-the-art methods. A direct consequence of our work is a testbed with various implementations available to be used by others to ensure that comparisons between methods are performed correctly and without much effort for new authors. This testbed is made available at `https://github.com/StephenPauwels/edbn`.

Our main contributions are:

1. We introduce a uniform testbed with an easy interface that can be used to add new methods and compare them in a uniform way to a variety of existing methods that are implemented in the testbed.
2. All implementations used in this chapter are available in our testbed.
3. We show the effect of different evaluation setups on the obtained results for several state-of-the-art methods.
4. We show the danger and wrong conclusions that can be the result of copying results from existing papers.

The next section gives an overview of the considered methods. In Section 5.3 we look at how authors add the historical aspect to an event by using prefixes of events of a fixed size. The different ways of evaluating the models and the baseline setting we use are presented in Section 5.4. Section 5.5 explains the different experiments and in Section 5.6 we discuss the results and insights found.

5.2 Methods

In our work we do not consider older methods based on Bayesian methods and Markov Chains to model and predict the continuation of ongoing cases [6, 16, 55, 82], as we are interested in how experiments are performed in state-of-the-art methods. Therefore we only consider the recent work done in the field. This recent work uses Neural Networks (NNs) for predicting the next activities. This choice was made after the successful use of these NNs in natural language processing, which uses sequence modeling based on Recurrent Neural Networks (RNNs) consisting of Long-Short-Term-Memory (LSTM) layers [43]. Evermann et al. [40] show that these models can also be used within the context of business processes. Specifically, we consider the following algorithms:

- Tax et al. [101] propose an LSTM-based neural network to predict the next activity, the full suffix, and the remaining time. The cases are encoded by using a one-hot encoding to represent the activities. Besides the activity, they also take the time since the last event, the hour of the day, and the day of the week into account. Depending on the exact prediction task, a slightly different architecture can be chosen. All the possible architectures consist of several connected LSTM layers, the number of layers can vary between datasets and has to be chosen by the user.
- Camargo et al. [23] add a separate neural net for learning a non-trivial embedding, that takes both the activity and resource into account. Furthermore, different network architectures have been proposed by Camargo with different parameter settings. These architectures range from combining activity, resource and duration together to keeping all attributes separate. As no indication was given on how to select the best network before running the experiments we have chosen to use a single architecture for all our experiments.

- To further incorporate extra attributes in an LSTM-based model, Lin et al. [62] propose to include a custom layer (the modulator) to take advantage of these extra attributes present in the data. This modulator layer learns how important the attributes are for our predictions and assigns weights to these attributes.
- An alternative to an LSTM-based model is a Convolutional Neural Network (CNN) that can deal with the sequential nature of business processes. These CNNs are often used in Computer Vision [99] and have proven to perform better than RNNs [4]. Another benefit of these convolutional networks is that they are trained in a more efficient way than the previously introduced LSTM-based methods [35].
- Both Di Mauro et al. [35] and Pasquadibisceglie et al. [78] use CNNs to predict the next activity. Di Mauro et al. use the prefixes consisting of the events as the input sequences for the model. A typical problem when using a CNN is the kernel size to use. This problem was solved in [99] by introducing an inception module that runs several convolutions with all different kernel sizes on the same input.
- Pasquadibisceglie et al. [78] convert the input events into an image-like input, with features indicating the number of times a certain activity has occurred in the case, extended with extra timing information.
- More recently, LSTMs have been used by Taymouri et al. in a Generative Adversarial Network [103]. The first part of this network tries to predict the next activity as good as possible. Then, a second part of the network receives both the real activity and the predicted activity and tries to identify the predicted activity as well as possible. The feedback generated by the second part is then fed back in the first part to improve the quality of its predictions. This is repeated until a certain quality threshold is reached.
- In contrast to using more specific architectures, we introduced a Neural Network consisting of a single dense layer in Chapter 4. The network incorporates the sequential nature of business processes by adding extra input cells, depending on the size of the prefix. One of the biggest advantages of this simple architecture is the low runtime for training the model. We show that this simple network performs on par with existing, more complex models.

Besides the deep learning methods, we use Dynamic Bayesian Networks (DBN) to predict the next activity, as introduced in Chapter 3. DBNs have the advantage of being fast to learn, and being able to naturally explain the returned prediction. This type of model learns the dependencies between attributes at different time steps in the log. Based on these dependencies, probability tables are constructed indicating the conditional probability of a certain activity occurring, given that other activities have occurred.

The above-mentioned methods were chosen to perform our experiments as they are all recent methods that show to outperform the competition. With this combination of methods we also have a wide variety of architectures used. As mentioned before, our primary intent is not to find the best method currently available, but rather to test the influence of the evaluation method on the overall

Table 5.1: Overview of the considered methods. With an indication of which model type was used: Dynamic Bayesian Network (DBN), Long-Short-Term-Memory Neural Network (LSTM), Convolutional Neural Network (CNN), Generative Adversarial Network (GAN), or standard Neural Network (NN).

Method	Architecture	prefix	attributes	code available
Tax et al.[101]	LSTM	pre		✓
Lin et al.[62]	LSTM	pre	✓	
Camargo et al.[23]	LSTM	pre	✓	✓
Di Mauro et al.[35]	CNN	pre		✓
Pasquadibisceglie et al.[78]	CNN	pre		✓
Taymouri et al.[103]	GAN with LSTM	post		✓
DBN [Chapter 3]	DBN	pre	✓	✓
SDL [Chapter 4]	NN	pre	✓	✓

accuracy of the models. Table 5.1 gives an overview of all methods and their key characteristics.

5.3 Prefixes

One aspect of designing a next activity predictor is how to incorporate the known history in the model. Although all considered papers denote it differently, all methods use the same concept of prefixes.

Definition 6. *A* prefix *is a consecutive segment of a case for which we want to predict the next event.*

One of the biggest differences between all methods lies in how they select the number of historical events to use and how to deal with padding when insufficient events are available to fill the entire prefix. Some use a fixed prefix size, which can be set as a parameter when training the model. Others use the length of the longest case as prefix size. This last method can potentially lead to using a large prefix size, although only a couple of cases in the data reach this length. This can lead to an overly complex model and decreased performance.

No previous events are available to create the prefix for the first events in a case. To be able to use the prefix approach for these events with insufficient previous events, we introduce the use of padding events. One of the choices we have to make when using prefixes is how to construct the prefix-log. Table 5.2 shows all possibilities of padding prefixes. Prefixes 1, 2, and 3 use *pre-padding* as there are not enough past events available to fill up the prefix. Prefixes 4, 5, and 6 use *no-padding* and only contain events that are present in the given event log. Prefixes 7 and 8 use *post-padding* to deal with the situation that not enough future events exist.

Post-padding should not be used when creating the prefixes but is added in this section as it is used by Taymouri et al. [103]. These entries should, however, not be

used for evaluating the predictions, as they always expect the padding event as the next event (0 in our example from Table 5.2). Because this padding event is present in the prefix itself, it is easy for prediction models to learn that the expected event should also be a padding event. This leads to an artificially inflated accuracy that is often significantly higher than when not using these post-padded events.

With this notion of prefixes we can rewrite our original definition of next activity prediction using the notion of prefixes:

Problem Statement 3. *Given a prefix of size k, $\langle e_{i-k}, \ldots, e_{i-2}, e_{i-1} \rangle$, predict the activity of the immediate next event e_i.*

Problem Statement 4. *A* Prefix-log *is a list of mappings between prefixes and their associated next event. For every mapping, we keep track of the case identifier cID of the case it originated from.*

Using this definition of next activity prediction, we can consider more than only the next activity. We could also use the different prefixes to predict the general outcome for the case (e.g. success or failure), or the expected cost (e.g. the actual cost in dollars for completing the process). For most of the methods, we only have to replace the prediction target (the next activity) in the prefix-log with the new target.

5.4 Evaluating Prediction Models

In this section, we take a closer look at how the evaluations are performed by the selected methods. We separate this overview into two parts. In the first subsection we take a look at the characteristics of the datasets that are used, and use this information to select the datasets we will use in our experiments. Secondly, we extract the key differences in experimental design that have been used that may impact the achieved results.

5.4.1 Dataset Selection

A total of eight different datasets were used to evaluate the models, Table 5.3 gives the main characteristics of these datasets. Most of the datasets originate from the

Table 5.2: Different prefixes for case $\langle a, b, c, d, e \rangle$ and a prefix size of 3.

	prefix	predict
1	$< 0, 0, 0 >$	a
2	$< 0, 0, a >$	b
3	$< 0, a, b >$	c
4	$< a, b, c >$	d
5	$< b, c, d >$	e
6	$< c, d, e >$	0
7	$< d, e, 0 >$	0
8	$< e, 0, 0 >$	0

Table 5.3: Main characteristics of the datasets considered.

Dataset	# Cases	# Events	Events per case	# Activities
BPIC11	1,143	150,291	131	624
BPIC12	13,087	262,200	20	36
BPIC13	7,554	65,533	9	13
BPIC14	46,616	466,737	10	39
BPIC15_1	1,199	52,217	44	398
BPIC15_2	832	44,354	52	410
BPIC15_3	1,409	59,681	42	383
BPIC15_4	1,053	47,293	45	356
BPIC15_5	1,156	59,083	51	389
BPIC17	31,509	1,202,267	38	26
BPIC18	43,809	2,514,266	57	41
Helpdesk	4,580	21,348	5	14

BPI Challenges[2]. The advantage of these datasets is that they contain a realistic reflection of ongoing business processes and that these datasets are well-known within the community. Every dataset has different characteristics on which the difficulty of correctly predicting the next activity depends.

Two datasets have been used by all authors to evaluate their methods: the BPI Challenge 2012 dataset [111] and the Helpdesk dataset [115]. As all methods have reported results on these datasets, often these results get copied to compare the performance with a newly proposed method. This is for instance the case for the results of the method proposed by Tax et al. [101] reported by Camargo et al. [23], Lin et al. [62], and Pasquadibisceglie et al. [78]. As we show, this copying of results can lead to misleading results as seemingly unimportant, small changes to how the data is prepared and used can lead to a significant change in the obtained accuracy.

After close inspection of the Helpdesk dataset, we saw that two different versions of this dataset have been used and compared to each other as if they were the same dataset. The first version of the dataset contains 3,804 cases, 13,710 events, and 9 activities (version A). The second version contains 4,580 cases, 21,348 events, and 14 activities (version B). This second version also contains extra attributes that could be used to improve predictions, whereas the first version does not contain these extra attributes.

Table 5.4 shows a detailed overview of which datasets were used by the different methods and Table 5.3 gives an overview of the most important characteristics of the different datasets. To get the best overall view of the behavior of the different methods, we selected 4 datasets for our experiments. They were selected to get a wide variety of datasets with different characteristics. The **BPIC11** dataset originates from a Dutch academic hospital, showing the different activities and phases the patients go through. It was chosen because of the high number of different activities

[2]https://www.tf-pm.org/newsletter/newsletter-stream-2-05-2020/bpi-challenges-10-years-of-real-life-datasets

Table 5.4: Overview of the datasets used by the various authors.

	Tax	Lin	Camargo	Di Mauro	Pasquadibisceglie	Taymouri	DBN	SDL
BPIC11								✓
BPIC12	✓	✓	✓	✓	✓	✓	✓	✓
BPIC13			✓					
BPIC14		✓						
BPIC15			✓				✓	✓
BPIC16		✓						
BPIC17		✓				✓		
BPIC18							✓	
Helpdesk	$\checkmark_A$	$\checkmark_B$	$\checkmark_B$	$\checkmark_A$	$\checkmark_A$	$\checkmark_A$	$\checkmark_B$	$\checkmark_B$

that are present in the data. This dataset also contains longer than average traces. As a dataset of a larger size, we selected the **BPIC12** data. This dataset contains the applications for personal loans from a Dutch financial institution. The log contains three intertwined processes, one of which was used to create an additional, frequently used, event log (**BPIC12W**). **Helpdesk** was selected as a small dataset that we can use to quickly perform different experiments. The challenge in this dataset is the limited amount of information that is available due to short cases. Unless otherwise specified, we use the version with 4,580 cases (version B). This log contains ticket requests from the helpdesk of an Italian Software Company. As the last dataset, we selected the **BPIC15** dataset, which is a log containing building permit applications from five different Dutch municipalities, where the data of every municipality is saved in a single event log. We denote these five datasets respectively **BPIC15_1** to **BPIC15_5**. These logs contain many activities with known drift behavior, making these datasets more complex to accurately predict. Using this variety of datasets we can extract interesting conclusions from our experiments.

5.4.2 Evaluation Setup

When looking for the different settings used, we make a distinction between *model-specific* choices and *data-specific* choices. With model-specific choices, we mean all choices that impact the creation of the model itself, e.g. the encoding used, the prefix size used, the hyperparameter settings, ... These choices are made by the authors when designing their models and are outside of the scope of this study. Other authors already examined these different choices and their respective impact [100, 118]. We do not consider these variations in the models as errors in the evaluation procedure. These variations reflect correct evaluations with different models. However, in our setting, we want to evaluate all models in the versions as they are intended and

proposed by the authors.

Data-specific choices alter some characteristics of the data itself. The big difference with model-specific choices is that these data-specific choices are made, possibly implicitly, by all authors for their evaluations. Due to the specific nature and structure present in event logs, these choices can potentially have unforeseen consequences.

Scrutinizing the papers and code provided by the authors of the aforementioned methods, we identified the following, often implicit choices that have been made in the experimental setup in these papers. We study their influence systematically in Section 5.5:

- **How to split the data into a train and test set?** In the literature, we see two visions on how to split the event log. The first vision maintains the chronological aspect of business process logs while the second does not. Both of these visions are correct, but evaluate different aspects of a method. When maintaining the chronological aspect, the evaluation is concentrating on how the predictor scores in a realistic, changing environment. The standard division under that setting is to select the events for training from the first events in the log and use the remainder of the log for evaluation. However, we also saw in the literature the opposite selection where the first events in the event log are used for evaluating [23].

  When using a random division, the use of k-fold cross-validation is standard. Under such a schema, the emphasis is on the performance of the model itself without having to deal with the typical chronological aspect of business processes such as concept drift.

- **Divide based on event or case?** Is it allowed for some events from a single case to be placed in the train set and others in the test set? When both splitting train and test set chronologically and keeping cases together, one has to make sure none of the cases in the training set overlap in time with cases in the test set to avoid anachronisms where events in the test set need to be predicted that come before some of the events in the training set. This is, again, a choice that has a potentially large impact on the chronological order of the events used for training the model. Dealing with incomplete cases is less of a problem for all methods as they all make use of prefixes of a certain length. If we want to put our focus on how methods perform within the field of business processes, we have to make sure that we do not break the chronological structure of the log as predicting the current event using the future is not possible in a realistic environment.

- **How many events to use for training?** For every way of creating the train and test logs, one has to decide how many events should be in every part. This can significantly change the characteristics of the data, as predicting becomes easier with more events to train the model. Another important aspect is the presence of drifts in the data, and more specifically, drift between the train and test log. Using a different split may affect the number of similar events in both parts. When the data is statically split into train and test, we set a percentage of the data in the log we use for training and the remaining for testing. For

the k-fold cross-validation, we select the k to use. When k becomes larger, the number of events in the training part becomes larger.

- **Add artificial end event?** Some logs contain a limited number of fixed end events, to indicate the end of a case. In the literature, a separate artificial end event is often added to the log to ensure the model can determine the end of a case. This is done without taking into account if the original log already contains such end events. This can artificially increase the accuracy as the artificial end event always follows the original end event and is hence straightforward to predict.

- **Filter cases on case length?** Case length in real-life event logs can vary greatly. To reduce the impact of small cases (that probably contain different behavior than cases of a larger size) some authors propose to filter the data on minimum case length, thus eliminating these (potentially) dissimilar cases. This also can have a large impact on the accuracy, as less data is used in both training and testing and potential anomalous cases are not taken into account, making both training and testing of the model easier because of less variance in the cases.

Table 5.5: Evaluation setup as used by the different methods.

Method	Own results	Split	How split?	Train-amount	Add end event?	Filter length
Tax		Case	Train-Test	66	✓	-
Lin		Case	Train-Test	70		5
Camargo		Case	Test-Train	70	✓	-
Di Mauro		Case	K-Fold	3-fold	✓	-
Pasquadibisceglie		Case	Train-Test	66	✓	-
Taymouri	✓	Event	Train-Test	80	✓	-
DBN	✓	Case	Test-Train	70		5
SDL	✓	Event	Train-Test	70		-

Based on the evaluations used in the literature, we selected different variations to consider in our experiments. Table 5.6 shows all possible values for these different settings we consider. Table 5.5 maps the papers we have included in the comparison to the settings used in these papers. As can be seen, there is a large variety of settings used, which should already be a clear indication that performance figures in these papers cannot be directly compared.

Running experiments where we compare all possible settings would lead to 120 experiments per method per dataset, thus totaling 8,640 different experiments. Therefore, we introduce a baseline setting which we use as a starting point for

Table 5.6: Overview of setup choices and used values.

Choice	Possible values
How to split dataset?	{Train-Test, Test-Train, Random, K-fold}
Split events or cases?	{Event, Case}
Amount of training data	{60%, 66%, 70%, 80%} or {3-fold, 5-fold, 10-fold}
Add artificial end-event?	{True, False}
Filter cases	{None, 5}

testing the impact of the different choices separately. This decreases the amount of experiments we have to run.

5.4.3 Baseline Setting

Table 5.7 summarizes the settings we consider as our baseline for testing next activity prediction. These settings were carefully chosen to ensure that all experiments reflect a realistic scenario. An important first aspect is that only historical data gets used for training a model. Therefore, we have to ensure that all events in the test data occurred after any event in the training data. To achieve this goal we split the event log into two parts based on a specific point in time. Events that happened before this point are considered in the training set, the other events are considered in the test set. To avoid possible anachronisms we thus split by only looking at the event level. Secondly, some datasets already contain end-events, adding an extra artificial end-event to the dataset is not needed. It will even give a wrong picture of the performance of the model tested, as all state-of-the-art models can learn to perfectly predict this extra end-event, given that the original end-event just happened. Thirdly, to ensure our test data contains enough samples to get meaningful results, we choose to select 70% of events for training the model and 30% for testing. Finally, we do not add a filter on the case length because we believe that these smaller cases contribute to getting a complete picture of the performance of a model.

Table 5.7: Our baseline setting.

	Split	How split?	Train-amount	Add end event?	Filter length
Baseline	Event	Train-Test	70	False	-

Table 5.8: The different quality metrics used. With n the total number of events, l the number of activities, s_i the true number of events with activity i, tp_i the number of true positive predicted activities for activity i, fn_i the number of false negatives, and fp_i the number of false positives.

Metric	Formula
Accuracy	$\frac{1}{n}\sum_{i=1}^{l} s_i \frac{tp_i}{tp_i+fn_i} = \frac{1}{n}\sum_{i=1}^{l} tp_i$
Average Precision	$\frac{1}{n}\sum_{i=1}^{l} s_i \frac{tp_i}{tp_i+fp_i}$

5.5 Experiment Design

To understand the influence of the differences in the experimental setup, we designed different types of experiments. To test the impact of the different evaluation choices one at a time we start by using a baseline setting, which was introduced in Section 5.4.3. Starting from this setting, we alter one of the choices to evaluate the influence of this choice on the overall accuracy. We use this experiment to determine which choices have a significant impact on the obtained results. Some of the choices may seem to be important but do not influence the obtained accuracy. Or vice-versa, a seemingly insignificant choice can lead to a significant change in accuracy for some methods.

The next type of experiment looks at how the ranking of the models changes based on which setting is used. We use different settings based on the setups used by the different methods. An overview of these settings can be found in Table 5.6.

Different evaluation metrics are used throughout the literature. Accuracy is the most popular one, this metric shows how many events were predicted correctly. Besides the accuracy, we also take a look at the influence of the experimental setup on the precision. The precision looks at how many of the events that are predicted to be of a certain activity are actually correctly predicted. We use the accuracy for all our experiments, and also report the precision for the ranking experiment.

An overview of these metrics can be found in Table 5.8, where n denotes the total number of events, l the number of activities, s_i the true number of events with activity i, tp_i the number of true positive predicted activities for activity i, fn_i the number of false negatives, and fp_i the number of false positives. Note that $s_i = tp_i + fn_i$. These metrics are calculated per activity and summed up, weighted by the true frequency of this activity.

The reported metrics for a method are the average of the obtained metrics over all datasets. By using this average, we level out any data-specific issue or outliers and get a more robust general overview of the behavior of the models. Some of the datasets contain drift and are thus harder to predict while others represent a more stable process. By combining these datasets we get a view of the overall performance of the methods. We provide the complete overview of all separate results in our repository.

All detailed results and code used for running the experiments and the imple-

Table 5.9: Average accuracy for all methods using the baseline settings.

Method	Baseline
Tax	0.56
Lin	0.56
Camargo	0.45
Di Mauro	0.59
Pasquadibisceglie	0.55
Taymouri	0.38
DBN	**0.63**
SDL	0.57

mentations for the various methods can be found in our GitHub repository[3]. To compare all methods with exactly the same data, we used the code provided by the authors. Lin et al. did not provide publicly available code, and therefore we use our own implementation of their method. We adapted all methods to fully work with our testbed. By using the adapter pattern in our implementation, we made sure that the original workings of the methods are preserved. In our experiments we did not look at how to set the different hyperparameters for all models, we kept the default settings as they were used in the original implementations. This can lead to models being trained in a sub-optimal setting. Therefore we do not make any conclusions on absolute results in our experiments. After all, we are more interested in how various changes to the data influence these results.

5.6 Results

5.6.1 Influence of Different Evaluation Aspects

In this section, we first discuss the effect on the obtained accuracy when we change one single aspect of the evaluation setup. In the next section, we take a look at the case when multiple choices are different.

Baseline

Before we test the influence of the different choices, we ran all methods using the baseline setting we use throughout this chapter. These baseline results can be found in Table 5.9. As the performances of most methods are similar, only small changes in accuracy can already lead to a different ranking of the different methods, leading to different conclusions. This experiment shows, again, the possible danger when authors compare methods using different setups.

[3] `https://github.com/StephenPauwels/edbns`

Table 5.10: Average accuracy for all methods for different ways of splitting the data.

Method	Train-Test	Test-Train	Random	3-fold CV
Tax	0.56	0.54	0.76	0.53
Lin	0.56	0.45	0.75	0.53
Camargo	0.45	0.39	0.71	0.20
Di Mauro	0.59	0.50	**0.77**	0.58
Pasquadibisceglie	0.55	0.52	0.75	0.57
Taymouri	0.38	0.36	0.42	0.38
DBN	**0.63**	0.46	0.74	0.56
SDL	0.57	**0.55**	0.76	**0.59**

How to split the data?

When looking at Table 5.10, it is clear that randomly splitting the data in train and test yields the best accuracy for most methods. One of the interesting and more difficult parts about predicting in a business process environment is the time aspect of the events. Events arrive at certain points in time and in a certain order. Taking this chronology into account is an important part of making predictions for business processes. Some datasets contain drifts over time, making it harder to predict events after this drift when only the first events are used for training. Using the random division of events, events from both before and after the drift are used to predict the other events. This leads to achieving higher accuracies as more diverse information is used. This way of evaluating does however not correspond to the chronological aspect of business processes. Using the first events for the test set shows to perform worse than the other settings while, again, breaking the chronological aspect of the event log and thus simulating an unrealistic situation.

K-fold cross-validation is a popular method outside of business processes to evaluate the performance of a prediction model. A first important observation is that the CV method does not at all preserve the chronological aspect of the business processes. We assume that the different folds do preserve the chronological aspect. All events in fold i must happen before all events in fold $i+1$ etc. Using cross-validation does again not fully capture the typical behavior of business processes, but is useful for capturing the performance of the model itself. The second observation is that the results obtained are similar to those for train-test and test-train, this is due to the fact that for 3-fold cross-validation, the first fold corresponds to the test-train case, while the last fold corresponds to the train-test situation. Also important to note is that k-fold predicts the next activity for every event in the log, while the other methods only predict this for a smaller set of events (the test set only).

Table 5.11: Average accuracy for all methods for different percentages of training data.

Method	60%	66%	70%	80%
Tax	0.50	0.53	0.56	0.64
Lin	0.49	0.54	0.56	0.62
Camargo	0.41	0.43	0.49	0.49
Di Mauro	0.51	0.56	0.59	**0.64**
Pasquadibisceglie	0.47	0.50	0.55	0.62
Taymouri	0.36	0.38	0.38	0.40
DBN	**0.54**	**0.61**	**0.63**	0.62
SDL	0.49	0.52	0.57	0.63

Amount of data used for training

In this section, we look at the difference in performance when a different amount of data is used for training the model. First, we look at the situation where we split the data using the train-test method. Thus, we look at different percentages of the data to use for training. Next, we look at the cross-validation method and compare different values for k, as a higher k means more events to use for training the models.

Percentage used for training Table 5.11 shows that, as expected, the more data can be used for training, the better the predictions are. This is, again, due to the chronological nature and the presence of drifts. With more data to train, the model is able to learn the behavior after these drifts and thus predict the next events more accurately. Interesting is to see the difference between using 66% and 70%, even though this is only a small difference, it gives a relatively large impact on the prediction quality. This can be due to the fact that by changing the amount of training data from 66% to 70% we include more events after a certain drift, ensuring

Table 5.12: Average accuracy for all methods for different k folds of training data.

Method	3-fold	5-fold	10-fold
Tax	0.53	0.62	0.67
Lin	0.53	0.60	0.70
Camargo	0.20	0.20	0.20
Di Mauro	0.58	0.67	**0.74**
Pasquadibisceglie	0.57	0.67	0.73
Taymouri	0.38	0.41	0.45
DBN	0.56	0.61	0.69
SDL	**0.59**	**0.67**	0.73

Table 5.13: Average accuracy for all methods for splitting based on events or cases.

Method	Events	Cases
Tax	0.56	0.58
Lin	0.56	0.51
Camargo	0.49	0.46
Di Mauro	0.59	0.60
Pasquadibisceglie	0.55	0.41
Taymouri	0.38	0.38
DBN	**0.63**	**0.64**
SDL	0.57	0.56

that the models can predict the last events more accurately.

Number of folds used As for the previous experiment, Table 5.12 shows the expected behavior of more training data leading to a more accurate result. Again, this shows the importance of comparing methods using the exact same setting, as these changes have a significant impact on the prediction quality.

Split based on events or cases?

Table 5.13 shows that some of the models score significantly lower when splitting based on cases (and thus not preserving the chronological nature of the log). We see a drop in the average accuracy as the accuracy for the datasets in which the chronological aspect is an important part of the data drops. For Pasquadibisceglie and Lin, we see that the obtained scores for the BPIC15 datasets drop significantly (often dividing the score by 2). These datasets contain drifts at certain points in time and thus the chronological aspect of the data has a significant impact on the results. The other methods are more robust to this phenomenon.

Add artificial end-event?

Overall we see only minor changes when adding an artificial end event when comparing the results in Table 5.14. But even with these small changes, comparing SDL with Lin in different settings can lead to different conclusions. One could say that SDL achieves 0.55, while Lin achieved 0.56. Thus Lin outperforms the SDL method. But when comparing the results with the same settings, we see that SDL outperforms Lin for both the setting with and without an artificial end event.

Filter cases

The results in Table 5.15 show only a very limited influence when filtering out cases with a length smaller than 5. The general behavior that we see is a small increase in accuracy, as shorter cases are harder to predict (because of less information

Table 5.14: Average accuracy for all methods for adding extra end-event.

Method	With end event	Without end event
Tax	0.55	0.56
Lin	0.52	0.56
Camargo	0.47	0.45
Di Mauro	0.57	0.59
Pasquadibisceglie	0.54	0.55
Taymouri	0.38	0.38
DBN	**0.62**	**0.63**
SDL	0.55	0.57

available). Most datasets are however not influenced by this choice, as most logs contain many more events per case on average than our threshold.

Accuracy differences

One of our main objectives is to show how much the obtained scores change when evaluation setups are different. Table 5.16 shows the minimum and maximum obtained average accuracy for all methods. We can see that the obtained results can vary greatly, thus clearly illustrating the danger of copying results from existing

Table 5.15: Average accuracy for all methods for filtering cases.

Method	No filter	Filter length 5
Tax	0.56	0.57
Lin	0.56	0.53
Camargo	0.45	0.48
Di Mauro	0.59	0.60
Pasquadibisceglie	0.55	0.55
Taymouri	0.38	0.39
DBN	**0.63**	**0.64**
SDL	0.57	0.57

Table 5.16: Minimum and maximum average accuracy obtained for all methods over all datasets.

Method	Minimum	Maximum	Difference
Tax	0.44	0.76	0.32
Lin	0.32	0.75	0.43
Camargo	0.20	0.71	0.51
Di Mauro	0.50	0.77	0.27
Pasquadibisceglie	0.41	0.75	0.34
Taymouri	0.36	0.45	0.09
DBN	0.46	0.74	0.28
SDL	0.49	0.76	0.27

papers without taking the correct evaluation setup into account.

5.6.2 Ranking Comparison

To see the impact when multiple options have different choices we ran the experiments using the settings we found in the literature, as presented in Table 5.5. Table 5.17 shows the average accuracies obtained for all methods over all datasets using the different settings. We added the obtained ranks (based on the accuracy obtained before rounding) for all models given a certain setting.

We see that the ranking of the models differs significantly when different settings are used. For some methods, the difference is rather small (Camargo and Taymouri for instance are always ranked 7th and 8th), while for other methods the rankings differ much more. When we take a look at the DBN method, we can see that it varies between rank 1 and rank 6.

When we take a look at the average accuracy obtained by the methods using different settings, we, again, see a large difference in absolute value. The accuracies for the method proposed by Tax et al. vary between 0.44 and 0.59. Where we saw minor differences when only a single setting was changed, we now see much larger differences as the different evaluation setups differ in more than one setting. This experiment only changes the experimental setup, while all parameters and settings of the models are kept the same. It shows that using a different experimental setup can lead to a different result when comparing prediction models.

We also computed the ranking scores based on the precision metric to show that no matter which metric is used, the experimental setup has a significant impact on the results. In Table 5.18 we can see that in general, the ranking does not change much in comparison with the accuracy. Again, we do see some significant differences in precision between different setups.

Table 5.17: Comparison with settings used by the different methods. The reported accuracy is the average over all datasets. The labels on the left indicate the tested models, while the labels at the top indicate from which papers the used setting originates.

	Tax	Camargo	Lin	Di Mauro	Pasquadibisceglie	Taymouri	DBN	SDL
Method	Rank *(Accuracy)*							
Camargo	8 *0.20*	8 *0.21*	8 *0.23*	8 *0.21*	8 *0.20*	8 *0.14*	8 *0.21*	7 *0.45*
Di Mauro	2 *0.53*	3 *0.54*	2 *0.58*	2 *0.56*	2 *0.53*	**1** ***0.63***	3 *0.53*	2 *0.59*
Lin	4 *0.51*	5 *0.49*	5 *0.53*	6 *0.50*	4 *0.51*	5 *0.60*	5 *0.48*	4 *0.57*
Pasquadibisceglie	5 *0.49*	2 *0.55*	4 *0.54*	3 *0.55*	5 *0.49*	2 *0.62*	2 *0.55*	6 *0.55*
DBN	**1** ***0.58***	6 *0.48*	**1** ***0.65***	4 *0.54*	**1** ***0.58***	4 *0.60*	6 *0.47*	**1** ***0.64***
SDL	3 *0.51*	**1** ***0.57***	3 *0.57*	**1** ***0.57***	3 *0.51*	3 *0.62*	**1** ***0.57***	3 *0.57*
Tax	6 *0.44*	4 *0.50*	6 *0.50*	5 *0.51*	6 *0.44*	6 *0.59*	4 *0.49*	5 *0.56*
Taymouri	7 *0.37*	7 *0.37*	7 *0.35*	7 *0.38*	7 *0.37*	7 *0.40*	7 *0.34*	8 *0.39*

5.6.3 Impact of Using Extra Features

Another aspect that can influence the results is the use of extra features (besides activity and time). Table 5.19 shows the baseline results again, with the indication if the method uses the values of extra attributes (for example the resource or role executing the activity). We see that there is no connection between the quality of the predictions and the use of extra features for the datasets used. We consider the use of extra features as a more fundamental choice when designing the model itself. Therefore, we do not change this behavior in our experiments, as we consider this to be a model-specific choice and not a data-specific one. Also, the table shows no clear benefit in results when using extra features, indicating that adding extra features to obtain a higher score does not help for the selected datasets.

Table 5.18: Comparison with settings used by the different methods. The reported precision is the average over all datasets. The labels on the left indicate the tested models, while the labels at the top indicate from which papers the used setting originates.

	Tax	Camargo	Lin	Di Mauro	Pasquadibisceglie	Taymouri	DBN	SDL
Method	Rank *(Precision)*							
Camargo	8 *0.20*	8 *0.20*	8 *0.23*	8 *0.21*	8 *0.20*	8 *0.12*	8 *0.20*	7 *0.47*
Di Mauro	2 *0.58*	4 *0.59*	2 *0.62*	2 *0.61*	2 *0.58*	2 *0.68*	4 *0.59*	2 *0.63*
Lin	4 *0.53*	5 *0.53*	5 *0.55*	6 *0.52*	4 *0.53*	5 *0.62*	6 *0.50*	5 *0.58*
Pasquadibisceglie	3 *0.54*	3 *0.60*	4 *0.58*	3 *0.57*	3 *0.54*	4 *0.65*	3 *0.59*	4 *0.59*
DBN	**1** ***0.69***	**1** ***0.63***	**1** ***0.72***	**1** ***0.65***	**1** ***0.69***	**1** ***0.71***	**1** ***0.63***	**1** ***0.73***
SDL	5 *0.52*	2 *0.61*	3 *0.59*	4 *0.56*	5 *0.52*	3 *0.65*	2 *0.61*	3 *0.59*
Tax	6 *0.48*	6 *0.51*	6 *0.49*	5 *0.53*	6 *0.48*	6 *0.59*	5 *0.51*	6 *0.57*
Taymouri	7 *0.33*	7 *0.34*	7 *0.31*	7 *0.33*	7 *0.33*	7 *0.36*	7 *0.32*	8 *0.35*

5.6.4 Threat to Validity

Despite our efforts to conduct the most complete set of experiments, there are some aspects of our study that the reader should keep in mind when interpreting the different results.

This is not a complete survey of existing state-of-the-art methods, we rather opted to select this subset of methods as they contain a wide variety of different methods and architectures, based on giving a general overview of the differences when applying different settings.

When using neural networks, different hyperparameters have to be set to guarantee the best results. As it was not possible to find the optimal set of parameters for all our experiments, we resorted to using the standard settings as they were provided in the papers and implementations of the different networks. Another aspect of neural networks is that they rely on a random aspect. And thus using the same method on the same data twice can lead to significant differences in their results. To get an overview of what methods are more stable and what the differences can be, we

Table 5.19: Average accuracy for all methods using the baseline settings.

Method	Baseline	Extra features
Tax	0.56	
Di Mauro	0.59	
Taymouri	0.38	
Lin	0.56	✓
Pasquadibisceglie	0.55	✓
Camargo	0.45	✓
DBN	**0.63**	✓
SDL	0.57	✓

included extra experiments to investigate the stability in the next subsection.

Stability

In this section, we take a look at how stable and consistent the different neural network methods score. As learning these methods involves some randomness, this can lead to different results for different runs. To test this we used the Helpdesk dataset to run 10 different runs of every method. The mean, standard deviation, minimum and maximum scores are presented in Table 5.20. This table shows that in general, the methods are stable, except for the method proposed by Camargo et al. The reason we found for this is that Camargo also uses a neural network to learn an integrated embedding using both the activity and resource. Further experiments showed that, when the used embedding was kept the same over different runs, the final result of the method was much more consistent. We did not perform these stability experiments for the DBN method, as this learning algorithm is deterministic and thus always returns the same model with the same quality.

5.7 Choices and Guidelines

After performing our various experiments, we propose the following choices and guidelines:

- **How to split the data into a train and test set?** A first possibility is to use a train-test setting. This respects the chronological order of the event log and tests the model in a realistic setting. The other option that can be chosen is to use k-fold cross-validation, this method breaks the chronological order and can thus only be used when we want to know the actual predictive quality of our model in a setting without concept drift.

Table 5.20: Mean, standard deviation, minimum and maximum for 10 runs using the Helpdesk dataset for all neural network methods.

Method	Mean	Standard Deviation	Minimum	Maximum
Tax	0.744	0.004	0.736	0.750
Lin	0.782	0.013	0.750	0.792
Camargo	0.626	0.107	0.429	0.779
Di Mauro	0.785	0.010	0.767	0.798
Pasquadibisceglie	0.756	0.012	0.745	0.787
Taymouri	0.721	0.022	0.681	0.755
SDL	0.781	0.007	0.763	0.788

- **Split based on event or case?** When using a train-test split, splitting should always occur based on events. Dividing the event log based on cases can result in some cases in the train set containing events that happen after some cases in the test set. When using the k-fold cross-validation, or a random split, one needs to split the data according to cases. Otherwise, we can have incomplete data to make a prediction.

- **How many events to use for training?** This depends on the model itself and is a choice that should be made by the authors themselves. However, the chosen values should be reported clearly as more data means better models.

- **Add artificial end event?** This should only be added when the model needs it, for instance when we want to predict the suffix of a running case. But then we should make sure that the prediction of this event is not taken into account for determining the quality of our predictions. Because datasets can already contain such a uniform end event and therefore the prediction of an additional artificial end event would be trivial and artificially increase the obtained results.

- **Filter cases on case length?** This depends on the model itself and is a choice that should be made by the authors themselves. However, the chosen values should be reported clearly.

It is significant that all methods use the same datasets. Experiments show almost no variation in the performance of the methods on both the *Helpdesk* and *BPIC12* datasets. This could indicate that these datasets are less suitable for comparing methods. Further investigation of these datasets to determine if they are suited for next event prediction evaluation is needed, as these two datasets are often the only datasets used for evaluations. When looking at the results from the *BPIC15* datasets

we see a much larger difference between all methods. This dataset contains some drifts and has more different activities than the other datasets. This might indicate that some of the state-of-the-art methods are not very well suited to handle this larger and more complex type of event log.

5.8 Conclusion

We showed that the performed experiments reported upon in the literature to assess the quality of next activity predictions are often flawed or incomplete. In this chapter, we first take a close look at how new methods for next event prediction are evaluated by the authors and how comparisons between methods are made. Even though everybody intuitively knows that preprocessing data differently changes the results, most of the recently published papers perform comparisons with existing methods by copying their initial results as most of the papers use the same datasets. One regularly overlooked aspect, however, are the various choices that, implicitly or explicitly, have to be made when designing experiments. Authors seem insufficiently aware of the possible consequences of the used evaluation setup, leading to unforeseen changes in the results. Secondly, we test the influence of these different preprocessing choices and formulate five different choices that have to be made. After performing our tests we can now also present some guidelines on how to make every choice. Some of the possible choices are wrong and should be avoided, whereas other choices only affect the aspect of the model we are testing. Some of the choices have less importance but should be described in the papers. It is clear that when comparing models, all these choices have to remain the same during testing. We proposed a set of guidelines that ensure that experiments can be performed correctly.

We showed the importance of reproducibility in research. Having quality code available for every method makes it possible to conduct fair comparison experiments. It also gives a clear and correct overview of how the experiments were performed. To facilitate the fair comparison of new methods. All our code is available in the form of a testbed that can be used to compare any method with all other implemented methods, making sure the same data and experimental setup are used. This testbed is available at `https://github.com/StephenPauwels/edbn`. We hope that the availability of this uniform testbed will allow authors of future papers to perform fair comparisons much more easily.

CHAPTER 6

Detecting Anomalies in Event Logs

Checking and analyzing various executions of different Business Processes can be a tedious task as the logs from these executions may contain thousands or millions of events, each with a (possibly large) number of both numerical and categorical attributes. In this chapter, we develop a way to automatically model the behavior captured in event logs with dozens of these attributes. The advantage of our method is that we do not need any prior knowledge about the data or the attributes. We introduce a new algorithm that learns a model of a log starting from the data itself. The learned model can then be used to detect anomalous executions in the data. To achieve this, we extend Dynamic Bayesian Networks with numerical attributes and functional dependencies to better model the normal behavior found in event logs. The model is capable of scoring events and cases, even when previously unseen values or new combinations of values appear in the log. An important benefit of our model is the ability to give a decomposition of the score that indicates the root cause of the anomalies. We also conducted a comparison with other state-of-the-art algorithms for detecting anomalies in Business Processes which shows that our approach outperforms other algorithms.[1]

[1]This chapter is based on *Stephen Pauwels and Toon Calders. Detecting anomalies in hybrid business process logs. In ACM SIGAPP Applied Computing Review 19.2, pages 18–30,2019* and *Stephen Pauwels and Toon Calders. An anomaly detection technique for business processes based on extended dynamic bayesian networks. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (ACM SAC '19),pages 494–501, 2019.*

6.1 Introduction

In the context of business processes, the detection of anomalous behavior is an important problem. Therefore, we introduce an anomaly detection system that can find deviating cases. This is done by learning a model which reflects the normal behavior of a system. This is done by taking both the categorical and numerical attributes into account, together with the different relations between the attributes. Attributes in an event log influence each other within and between different events in a single case, therefore giving useful extra insights in the log. We also exploit different possible types of relations between attributes. Our model uses more information than existing techniques within process mining [109]. Besides missing or wrongly ordered activities, there can be constraints that enforce that two activities must be performed by the same person or that a person needs to have a certain role to perform a certain action. Our model captures these relations too.

Diagrams, like BPMN [71], are a great tool for human understanding of a business process. For applications such as anomaly detection, BPMN models are, however, insufficiently powerful as they cannot easily express joint probability distributions over multiple attributes; they focus on a single perspective (i.e. the resource-activity perspective). Therefore, to take advantage of all possible relations between attributes in an event log, we create a model based on Dynamic Bayesian Networks (DBNs) [89]. DBNs are extensions of Bayesian Networks that can incorporate the sequential nature of business processes. DBNs link events to their predecessors to find relations between these events rather than only relations within one event.

In this chapter, we identify and alleviate some important shortcomings of DBNs when it comes to modeling the allowable sequences in a log:

- DBNs are not able to handle unseen values in a way appropriate for business process logs.
- A value that always occurs together with another value describes a common structure in event logs. We can model these relations in a DBN but only implicit, which may lead to less effective structures.
- We take the duration between attributes into account and are thus able to detect anomalous execution times
- Our model is able to handle numerical attributes which may depend on other numerical or categorical attributes.

The structure of this chapter is as follows. In the next section, we give an overview of the related work. In Section 6.3 we extend our DBN model for describing normal behavior in event logs and show how to discover anomalies in event logs. The learning of the structure of the model is described in Section 6.4. We evaluate our new method in Section 6.5.

6.2 Related Work

Different techniques have been proposed to solve this problem, both in the anomaly detection field [7, 26, 126] and in the process mining field [11, 13, 74, 87]. Some of

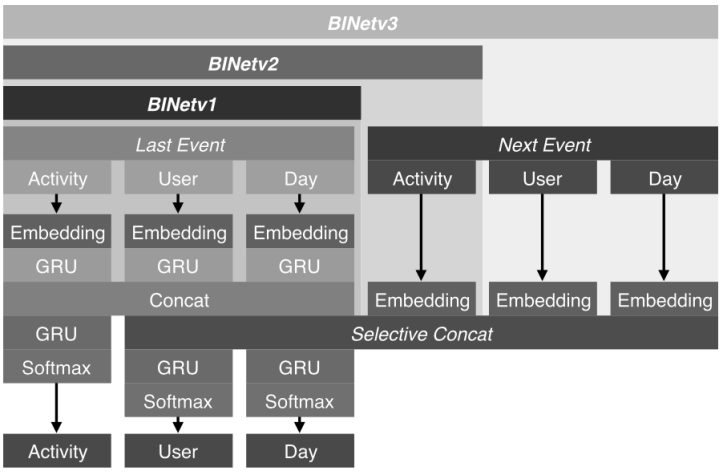


Figure 6.1: Overview of the network architectures used by Nolle et al. [74]. This figure gives an overview of the various networks that were proposed over the years by Nolle et al. Every new iteration of the network uses more available data to calculate the anomaly score.

these techniques use signatures of known anomalies that can occur in the system. These systems cannot recognize new types of anomalies and are too limited for our purpose. We are interested in model-based anomaly detection techniques, such as Markov Chains that represent the normal behavior of a system.

A first type of algorithms works on a database of univariate sequences; i.e., they only take the activity perspective into account. Bezerra et al. [11] investigate the detection of anomalies in an event log using existing Process Mining algorithms to build a model of the process. Then they use conformance checking to detect deviating traces of activities.

Other algorithms work on databases consisting of multivariate sequences which do not necessarily originate from a business process. Bertens [7] uses MDL to identify multivariate patterns that help to detect and describe anomalies. A code table consisting of mappings between encodings and frequently occurring patterns is first generated by their algorithm called DITTO [8]. An anomaly score is then defined by dividing the length of the encoded sequence given the code table by the length of the sequence.

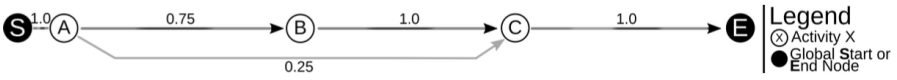
Nolle et al. [74] propose anomaly detection methods based on neural networks in business process event logs. They explicitly split the log between the control flow and the data perspective. These two perspectives are then used as inputs for their neural network that predicts both perspectives for the next event. The use of neural networks makes it possible to reduce the impact of noise in the dataset, whereas other methods need a training dataset without anomalies as a reference set. The major downside of this method is that it can only handle a few attributes in the data perspective and that it is not able to cope with unseen values (both in the activity and data perspective). An overview of the architectures proposed by Nolle et al. can be found in Figure 6.1.

Table 6.1: Summary of Related Work in comparison with our proposed method.

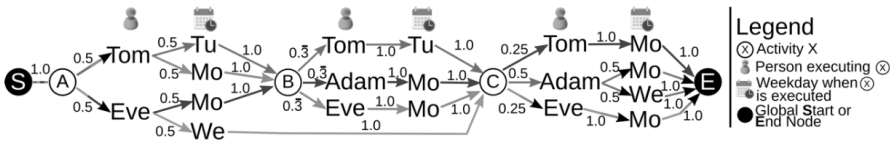
	Multi-Dim	Time	Method
Our method	✓	✓	DBN
Bezerra [11]			Process Mining
Nolle [73]	✓		Neural Networks
Bertens [7]	✓		MDL
Bohmer [13]	✓		Probabilistic Model
Rogge-Solti [87]		✓	Parzen-Windows

Bohmer et al. [13] introduce a graph-based probabilistic model that gives a score to all events in the event logs. First, a *Basic Likelihood Graph* is constructed where all activities are nodes and there is an edge between activities that follow each other. An edge indicates the possibility that given the previous activity, a certain activity happens next. The edge is labeled with the probability for this next activity. Next, this graph is extended by adding context attributes such as *resource* and *weekday* between two activities that correspond to the resource that performed the previous action on a particular weekday. Using this graph it is possible to compute a baseline score given the occurrence of a particular activity. This baseline score is compared to the actual score given to a case by the model. To score a case, Bohmer et al. use the data in the graph with the corresponding probabilities. Besides the data present in the graph, the model is also able to deal with new values. The use of extra attributes is not described in detail, but their model can be extended in a straightforward way to other attributes as well. Figure 6.2 gives an example model learned by Bohmer et al.

In the field of detecting anomalies in business processes, not much work has been done on exploiting the temporal perspective. Rogge-Solti et al. [87] propose a method for detecting temporal anomalies in Business Processes. They use a Bayesian model that can be inferred from the Petri net representation of a business process. Their main idea for finding outliers in the duration of activities is based on a hypothesis test, in which they determine the probability that a particular observation



(a) Basic Likelihood Graph



(b) Extended Likelihood graph

Figure 6.2: Example of the models learned by Bohmer et al. [13].

Table 6.2: CPT based on the example from Table 2.1 until time equals 17.

URole	*UName*	$P(UName \mid URole)$
employee	Joe	0.6
employee	Marc	0.4
manager	Linda	1
sales	Bob	1

x was taken from a learned distribution. Their method is based on a method that uses Parzen-Windows for network intrusion detection [128]. The biggest drawbacks of this method are the runtime needed when trying to analyze large event logs and the fact that it cannot cope with new activities that occur in the log.

A summary of the different techniques is found in Table 6.1.

6.3 Extended Dynamic Bayesian Networks

In this section, we extend Dynamic Bayesian Networks to create a model which is more flexible and powerful when dealing with event logs. To do so, we discuss the different types of relations that together make up the joint distribution expressed by our extended Dynamic Bayesian Network.

6.3.1 Relations Between Attributes

DBNs model the joint probability over a set of random variables as described in Section 2.2.4. In our extended model, we use two different types of dependencies for describing the behavior in an event log. The first type is the standard type in Bayesian Networks: Conditional Probability. We compute this for relations between both discrete and numerical attributes. To extend this type of relation to the numerical attributes we make use of Kernel Density Estimations. A new type of relation we introduce is the Functional Dependency, which forms a more strict relationship between attributes.

Conditional Dependencies Between Discrete Attributes

A DBN models the joint probability distribution over several random variables (attributes) by making use of the conditional dependencies between the attributes. These conditional dependencies are represented using Conditional Probability Tables (CPTs). An example of such a CPT is given in Table 6.2.

Definition 7. *A Conditional Probability Table* $CPT(X \mid Y)$ *is a table where each row contains the conditional probability for a value of attribute* X *given a combination of values of attributes* Y*. There is one row for each combination of values.*

Conditional Dependencies Between Numerical Attributes

When we want to extend our model to numerical attributes we can no longer use the Conditional as defined above, as the number of possible values is infinite. Thanks to

the decomposability of the total probability we can introduce a new way of scoring the conditional probabilities for numerical attributes without having to change how we score the categorical attributes. The model, however, has to take the type (categorical or numerical) of the attributes into account.

Often numerical methods assume that the data always follows the same distribution (e.g. a Gaussian distribution), however, in real-life applications this is often not the case. Therefore, we propose to use Kernel Density Estimation (KDE) [22] which learns a distribution without making assumptions about the data.

Definition 8. Kernel Density Estimation *is a non-parametric way for estimating the probability density function f of a random variable based on a finite set of samples. Suppose we have samples $(x_0, x_1, \ldots, x_n)$, we define the kernel density estimator $\hat{f}$ as:*

$$\hat{f} = \frac{1}{nh} \sum_{i=0}^{n} K\Big(\frac{x - x_i}{h}\Big) \tag{6.1}$$

with K a non-negative function, the kernel*, and h* the bandwidth *that acts as a smoothing parameter.*

We use a Gaussian kernel and a bandwidth α. We denote it as $KDE(x; X, \alpha)$, where x is the value we want to score, X is the set of values used to build the distribution and α is the bandwidth of the kernels used. The α parameter is determined during the learning phase, as we explain in Section 6.4. An example Kernel Density Estimation for a given set of data points can be found in Figure 6.3.

One of the advantages of KDE is that it can also be used for multidimensional distributions. As such we can use them to get the conditional probability for a numerical attribute, given its numerical parents, we have:

$$P(X|Pa(X)) = \frac{P(X, Pa(X))}{P(Pa(X))} \tag{6.2}$$

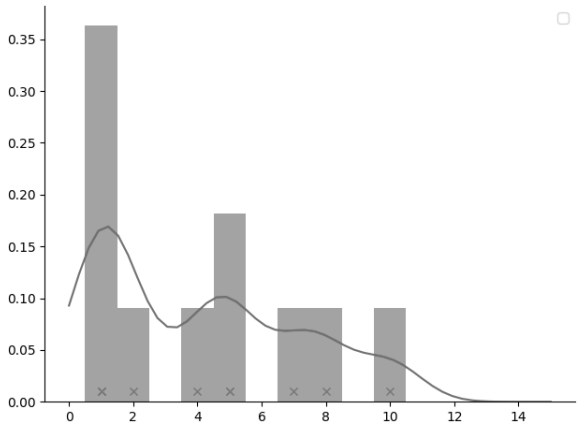


Figure 6.3: Example of a KDE estimation given the datapoints [1,1,1,1,2,4,5,5,7,8,10]. The crosses indicate the datapoints, the bars indicate the histogram and the curve indicates the estimated density.

where both $P(X, Pa(X))$ and $P(Pa(X))$ can be expressed by KDEs.

Besides numerical parents, a numerical attribute also can have categorical parents. Given N a set of numerical parents and C a set of categorical parents, if we want the conditional probability for X given both N and C we have:

$$P(X|N,C) = \frac{P(X,N,C)}{P(N,C)} \tag{6.3}$$

Without changing the correctness we can add $P(C)$ to both the nominator and denominator. We then have:

$$\frac{P(X,N,C)}{P(N,C)} = \frac{P(X,N,C)}{P(C)} * \frac{P(C)}{P(N,C)} = \frac{P(X,N|C)}{P(N|C)} \tag{6.4}$$

$P(X \cap N)$ and $P(N)$ can be calculated using KDE. The conditioning on C can be done by partitioning the training data according to C and training different KDEs for every partition. Next, we also train a general version of the KDEs without having a condition on C in case we encounter an unseen combination of values for the categorical parents. For attribute X with numerical parents $Y_1, \ldots, Y_l$ and categorical parents $Y_{l+1}, \ldots, Y_m$ we then have:

$$P(x|y_1, y_2, \ldots, y_m) = \frac{KDE\big((x, y_1, \ldots, y_l); (X \times Y_1 \times \ldots \times Y_l)\big|_{Y_{l+1},\ldots,Y_m}, \alpha\big)}{KDE\big((y_1, \ldots, y_l); (Y_1 \times \ldots \times Y_l)\big|_{Y_{l+1},\ldots,Y_m}, \alpha\big)} \tag{6.5}$$

When an attribute has no numerical parents, the KDE in the denominator is set to 1.

We use the notation $P(X|Pa(X))$ when referring to the conditional probability for both the categorical and numerical attributes, although the way of calculating them is different.

Functional Dependencies

The following example indicates the limitations of using only conditional probabilities for describing event logs:

Example 14. *Consider the situation where every User has a particular Role and certain activities can only be executed by certain roles. The attribute Activity depends on attributes User and Role in this example. When building a single CPT we have to add a row for every possible combination of values for User and Role, resulting in a large table with all probabilities equal to 1. Also, when a new user is added to the system, all combinations with this user would have to be added to the CPT.*

This observation certainly is not new, several proposals exist to deal with large CPTs [10, 64, 31] and new values [25]. In this chapter, we choose *Functional Dependencies* (FDs) to deal with these problems. We explain the reasons for this choice after the formal definition of functional dependencies.

Definition 9. *Given a log L, a Functional Dependency $A \rightarrow B$ holds in L if for all events e, $f \in L$ holds that if $e.A = f.A \neq None$, then $e.B = f.B$ for attributes A and B.*

A functional dependency between attributes X and Y can be represented by a function $FD_{X\rightarrow Y} : a\_dom(X) \rightarrow a\_dom(Y)$, $FD_{X\rightarrow Y}(x) = y$, with x and y the respective values for attributes X and Y. The set $a\_dom(A)$ is defined as follows:

Definition 10. *Let L be a log over $\mathscr{A}$ and $\{A_{i_1}, \ldots, A_{i_k}\} \subseteq \mathscr{A}(L)$. We define the active domain $a\_dom(A_{i_1}, \ldots, A_{i_k}) = \{(e.a_{i_1}, \ldots, e.a_{i_k}) \mid \exists C \in L : e \in C\}$ as the set containing all values that occur in the log for the given attributes.*

Example 15. *In the log in Table 2.1, UID and URole describe a Functional Dependency. Every value of UID maps to a single value of URole. A particular value in URole can however occur together with multiple values of UID. We have the following mappings in our log:*

$$\{001 \mapsto \textit{employee},\ 002 \mapsto \textit{manager},$$
$$003 \mapsto \textit{employee},\ 004 \mapsto \textit{sales}\}$$

A first benefit of using FDs is that they are well-studied and several highly efficient methods for listing all (approximate) functional dependencies exist [125]. They also ensure a more easy learning phase for the CPTs as some edges are already added by the FDs and should not be examined again by the Bayesian network learning algorithm. Another benefit is the compactness of the model. Every FD is kept in a separate table, making it also possible to give a better, more detailed explanation of which particular FD has been violated.

Besides FDs we could also choose to use Decision Trees (DTs) [10], Association Rules (ARs) [64] or probabilistic modeling languages (like ProbLog [31]). These first two approaches have the disadvantage that they work on value-level. ProbLog, however, does allow for expressing functional dependencies, thanks to its use of variables. The advantage of FDs as compared to ProbLog is that ProbLog is a general-purpose probabilistic modeling language, and learning ProbLog programs is a harder task than learning FDs. An interesting avenue for future work is to mine, next to functional dependencies, other specialized patterns and use ProbLog as a language to express all patterns together and use its powerful inference mechanism to exploit them. FDs allow for enforcing constraints on unseen values, unlike ARs and DTs. Indeed, suppose that we discover the functional dependency *UID* $\rightarrow$ *UName*. Such a rule would allow for spotting the inconsistency of two events with the same *UID* but different *UName*, even if the values were never observed before.

6.3.2 Unseen Categorical Values

Event logs often encounter values that have not been seen before (e.g. a new customer or employee). These unseen values do not indicate a possible anomaly in the data as long as they do not break any of the already known dependencies in the log. When using the CPTs and FDs as introduced in the previous section we would simply assign 0 to these values. Smoothing can be used to overcome this problem but may be inappropriate for attributes that allow for new values to appear frequently. We want a way of indicating which attributes are more likely to encounter unseen values. To do so, we use a known technique used in the area of Probabilistic Databases (PDBs) as presented by Ceylan et al. [25]. They return an interval of probabilities for a given query that contains known facts (seen values) and unknown

facts (unseen values). Instead of calculating a different probability for every unseen value, we calculate the probability of encountering a new value in the log for every attribute. When a new value is encountered we multiply the score by this probability, when a known value is encountered we multiply by 1 minus the probability. We use the same principle for unseen combinations of parent values for the categorical Condition Dependencies.

Example 16. *The attribute URole will never take a new value as these are fixed within the organization, therefore we get a low probability for new values. The attribute UName can contain new values when a new user is added to the system, thus the probability for a new value must be high.*

6.3.3 Extending the Dynamic Bayesian Networks

Combining all the introduced elements, we extend the definition of a DBN as follows:

Definition 11. *An extended DBN (eDBN) with memory k over attributes $\mathscr{A}$ is a tuple:*

$$(G, CPT, CPK, FDT, new\_value, new\_relation, violation) \text{ where:}$$

- $G(V,E)$ *is a directed acyclic graph with* $V = \mathscr{A}_k \cup \ldots \cup \mathscr{A}_1 \cup \mathscr{A}$ *where* $\mathscr{A}_i = \{A_i | A \in \mathscr{A}\}$, *and* $E \subseteq V \times \mathscr{A}$. $\mathscr{A}_i$ *represents the attributes of the i th event before the current event. E expresses dependencies of the attributes of the current event on the other attributes in its context.*
- *CPT the Conditional Probability Table describing* $P(A|Pa(A))$ *for each* $A \in \mathscr{A}^c$
- *CPK the used kernels for describing* $P(A|Pa(A))$ *for each* $A \in \mathscr{A}^n$
- *FDT the different tables describing all Functional Dependencies between attributes*
- $new\_value(A)$ *is a function* $\mathscr{A} \to [0,1]$ *representing the probability of encountering an unseen value*
- $new\_relation(A)$ *is a function* $\mathscr{A}^c \to [0,1]$ *representing the probability of encountering an unseen combination of parent values for the conditional dependency of A.*
- $violation(X,Y)$ *is a function* $\mathscr{A}^c \times \mathscr{A}^c \to [0,1]$ *representing the probability that* $FD_{X \to Y}$ *is broken.*

Figure 6.4 shows the structure of a possible eDBN based on our example. The structure is determined by the different conditional and functional dependencies found in the event log.

The joint Distribution of an eDBN

An eDBN with memory k represents a joint distribution over sequences as follows:

$$P(\langle e_1, \ldots, e_m \rangle) = \prod_{e \in \langle e_1, \ldots, e_m \rangle} P(e | \mathscr{C}_k(e)) \tag{6.6}$$

$$= \prod_{e \in \langle e_1, \ldots, e_m \rangle} \prod_{A \in \mathscr{A}} P(e.A | \mathscr{C}_k(e)|_{Pa(A)}) \tag{6.7}$$

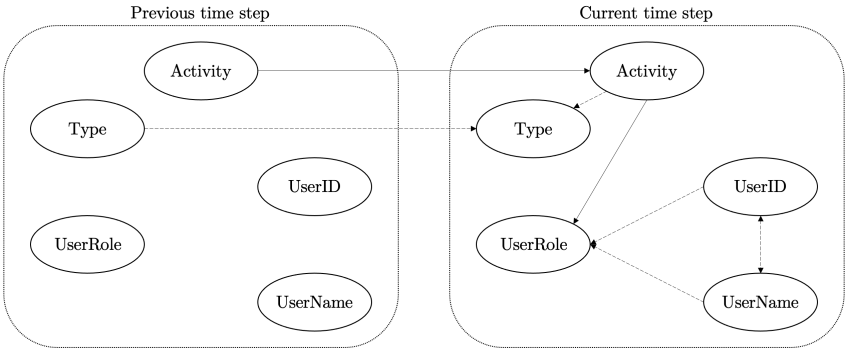


Figure 6.4: eDBN with conditional (full) and functional dependencies (dotted).

The probability for a single attribute in an event consists of three different factors:

- The first factor checks for new values and is defined as:

$$value_A(x) = \begin{cases} 1 - new\_value(A) & \text{if } x \in a\_dom(A) \\ new\_value(A) & \text{otherwise} \end{cases} \tag{6.8}$$

- The second is the probability for the conditional dependencies:

$$Relation(x|Pa(X)) = \begin{cases} new\_relation(Pa(X)) & \text{if new combination} \\ & \text{of parent values.} \\ (1 - new\_relation(Pa(X)) * & \\ \qquad CPT(x|Pa(X)) & \text{otherwise} \end{cases} \tag{6.9}$$

  With *CPT* either the discrete or the numerical version.

- The last factor is the probability for a Functional Dependency and is expressed as follows:

$$FDM_{X,Y}(y|x) = \begin{cases} 1 - violation(X,Y) & \text{if } FD_{X\to Y}(x) = y \text{ or } x \notin a\_dom(X) \\ violation(X,Y) & \text{otherwise} \end{cases} \tag{6.10}$$

We can now calculate the probability for a single attribute as follows:

$$P(e.A|\,\mathscr{C}_k(e)|_{Pa(A)}) = value_A(e.A) * Relation(e.A|Pa(A)) * \prod_{(X,A)\in FDR} FDM_{X\to A}(e.A|\mathscr{C}_k(e).X))$$

Example 17. *The probability for an event e in the model given in Figure 6.4 is equal to:*

$$\begin{aligned}
&value(Activity)\,Relation(Activity|Activity_1)\,value(Type)\\
&* FDM(Type|Activity_1)\,FDM(Type|Activity)\\
&* FDM(Type|Type_1)\,value(UID)\,FDM(UID|UName)\\
&* value(UName)\,FDM(UName|UID)\,value(URole)\\
&* Relation(URole|Activity)\,FDM(URole|UID)\\
&* FDM(URole|UName)
\end{aligned}$$

The value for the attribute UserRole for the event with ID 2 is:

$$\begin{aligned}
&value(URole)\,Relation(URole|Activity)\\
&\quad * FDM(URole|UID)\,FDM(URole|UName)\\
&= (1-0.2) * (1-0.4) * 1 * (1-0) * (1-0) = 0.48
\end{aligned}$$

6.3.4 Detecting Anomalies

To find anomalous sequences of events we use a score-based approach. The score is obtained by calculating the probability for a case $\langle e_1, \ldots, e_s \rangle$ given a model M. We normalize the result using the s-th root, with s the number of events in the case. This normalization makes sure that longer cases are not penalized.

$$Score(\langle e_1, \ldots, e_s \rangle) = \sqrt[s]{P(\langle e_1, \ldots, e_s \rangle)} \tag{6.11}$$

Sequences with a high score thus have a high probability of occurring and are more likely to represent normal behavior, whereas low scores indicate a higher chance of being an anomaly. Our method returns an ordered list of cases, sorted by their anomaly scores. The idea is that a user can only handle the first k anomalies detected. By returning a sorted list of potential anomalies, we make sure that the anomalies of which we are more certain are being investigated by the user. Because we can score any sequence of events using the k-context, we do not have to wait for a complete case before we can score it. The model can thus be used to detect anomalies in ongoing cases.

6.4 Learning the Model

We build our model using a reference dataset. Ideally, this dataset only contains normal executions of the process(es). But we show that our learning algorithm also produces a reliable model when the reference dataset contains a small amount of noise or anomalies. Recall that the timing aspect is incorporated by using the k-contextlog, so we can assume that we are working with a multivariate dataset. We only use the specific sequential aspect for determining the allowed causal relations between attributes, that is $Y \notin Pa(X)$ with Y later in time than X. All dependencies

present in our model should indicate a causality relation; events in the present cannot influence events in the past. Therefore edges that do not represent a causal relation are blacklisted. This blacklist is created by adding all edges that do not end in the *current* time step.

The complete algorithm can be found in Algorithm 1. Note that we use the notation $\mathcal{A}_0$ to indicate the attributes in the current time step, and $\mathcal{A}_*$ to denote the attributes in all available time steps. We explain the important steps in more detail.

Algorithm 1 Algorithm for learning the structure and parameters of eDBNs

Input: $k-log, \mathcal{A}, FDThreshold$
Output: $eDBN-model$
1: $FDR = \{ X \rightarrow Y : X \in \mathcal{A}_*^c, Y \in \mathcal{A}_0^c, U(X|Y) > FDThreshold \}$
2: $FD = \{ X \rightarrow Y : X, Y \in V, U(X|Y) > FDThreshold \}$
3: $blacklist = \{ X \rightarrow Y : X \in \mathcal{A}_i, Y \in \mathcal{A}_j \ with \ i \geq j > 0 \}$
4: $whitelist = FDR$
5: $G(V,E) = LearnBayesianNetwork(\mathcal{A}, FDR, blacklist)$
6: $FD = ConstructFDFunctions(FDR)$
7: $CPT = ConstructCPTables(E \setminus FDR)$
8: $CPK = ConstructCPKernels(E)$
9: $NV = \{ X \mapsto \frac{|a\_dom(X)|}{|L|} : \forall X \in \mathcal{A}^c \}$
10: $NR = \{ X \mapsto \frac{|a\_dom(Pa(X))|}{|L|} : \forall X \in \mathcal{A}^c \}$
11: $VIOL = \{ X \times Y \mapsto \frac{|\{ e \in L : FDR_{X \rightarrow Y}(e.X) \neq e.Y \}|}{|L|} : \forall (X,Y) \in FDR \}$
12: **return** $eDBN(G(V, E \setminus FD), FDR, CPT, CPK, FD, NV, NR, VIOL)$

In the first step, the algorithm searches for Functional Dependencies among the categorical attributes. In order to discover them, the Uncertainty Coefficient [83] is applied to all allowed combinations of attributes in the k-contextlog. The uncertainty coefficient is defined as follows for the categorical attributes X and Y:

$$U(X|Y) = \frac{I(X;Y)}{H(X)} \ ,$$

with $H(X)$ the *entropy* [95] of X and I(X;Y) the *Mutual Information* [28] given as:

$$I(X;Y) = \sum_{y \in a\_dom(Y)} \sum_{x \in a\_dom(X)} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$
$$H(X) = -\sum_{x \in a\_dom(X)} p(x) \log(p(x))$$

The Uncertainty Coefficient is the normalized form of Mutual Information. It gives information about how much the values of an attribute depend on another attribute. We use it to determine what attributes are related to each other and how much they relate to each other. The measure ranges from 0 (no correlation between the two attributes) to 1 (full determination of X by Y, thus indicating the existence of a Functional Dependency) [120]. If $U(X|Y) > FDthreshold$, we assume that the FD $Y \rightarrow X$ holds. This threshold has to be chosen according to the amount of noise in the data. A higher threshold means a more strict Functional Dependency is used that is less able to cope with noise.

To deal with unseen values, unknown combination of parent values, and broken functional dependencies for an attribute $A \in \mathscr{A}_c$ we introduce *new_value*(A), *new_relation*(A) and *violation*(X, Y) as follows:

$$\begin{aligned} \textit{new\_value}(A) &: \frac{|a\_dom(A)|}{|L|}, \\ \textit{new\_relation}(A) &: \frac{|a\_dom(Pa(A))|}{|L|} \\ \textit{violation}(X,Y) &: \frac{|\{e \in L : FD_{X \to Y}(e.X) \neq e.Y\}|}{|L|} \end{aligned}$$

This choice for the unseen values reflects the main idea as proposed by Ceylan et al. [25], where they add unseen tuples to the Probabilistic Database, each with a certain probability, possibly depending on the values of other attributes. We consider all unseen values as equally likely and the probability they receive should reflect only the behavior of the attribute itself, therefore we assign to every unseen value of an attribute the probability of encountering a new value for this attribute in the database.

Example 18. *When considering the log from Table 2.1 the new_value probability for attribute Type is equal to 0.1, indicating that new values are not often encountered in the log. While attribute UID has a new_value of 0.2, indicating that this attribute does encounter more unseen values and should therefore less penalize the total score when a new value has to be scored.*

6.4.1 Learning the Bayesian Network Structure

With a standard Bayesian network learning algorithm we can discover the Conditional Dependencies present in the data. It is possible to use any learning algorithm that uses data to learn the network structure. We use a greedy algorithm that tries to optimize a model score. This score takes three aspects into account. The first aspect is the prior distribution over all different models, the second is the score for the current found network structure and the last one is a penalty for overcomplicated networks. We want a good network that is not overly complex, without the penalty for the structure we would end up with a complete graph. We can drop the prior distribution, which favors more simple models, from the score as we include this aspect in the penalty part of the score. An important property of the model score used is its decomposability. When using a score that can be decomposed into the contribution of every single attribute we can use a different way of scoring categorical and numerical attributes. A categorical attribute only has categorical parents, whereas a numerical attribute can have both categorical and numerical parents.

Because dependencies between categorical attributes are either conditional or functional we do not want the BN learning algorithm to label already found functional dependencies as conditional. Therefore, we add these edges to a whitelist that is passed on to the learning algorithm. The learning algorithm should always include the edges from the whitelist in the model. This way the FDs are taken into account for calculating the score of a network but are not added or removed by the algorithm. One possible problem with this approach is that FDs do allow for cycles, but CDs do not. Therefore, we consider a cycle of FDs as a single variable in the BN

learning algorithm. Since the FDs fully determine the values of the attributes we can do this without losing any information about the data. This way we do not introduce cycles in the whitelisted relations in the BN learning algorithm and are still able to learn dependencies to and from these cycles of FDs. An important constraint on both FDs and CDs is that nodes can only influence nodes in the current time step. Otherwise, dependencies between attributes can be accounted for twice when scoring a case.

After running the greedy algorithm, we have found the Conditional and Functional Dependencies that define the structures present in our data. We can then combine them into one single model. The next step in building the model is filling in all Conditional Probability Tables, constructing the Functional Dependency functions, and learning all the Kernel Density Estimations. As the last step, we learn the probabilities for unseen values, new parent configurations, and functional dependency violations.

Scoring the Learned Model

To score the quality of a model, we use a combination of the likelihood of the underlying data given the current model and a penalty term to discourage overly complex models. The total score of a model is the sum of the scores for all attributes.

$$Score(M) = \sum_{A \in \mathscr{A}} Score(A) - penalty(A) \tag{6.12}$$

We now describe the score for a single attribute. Where we have a different approach for the categorical and numerical attributes.

For the categorical attributes, we use the Akaike Information Criterion (AIC) [2] as the score for the current network structure which is defined as follows:

$$2 * p_c - 2 * ln(L), \tag{6.13}$$

with p_c the number of distinct parent configurations of a variable (the complexity aspect) and L equal to:

$$\sum_{x \in a\_dom(X)} freq(x) * p(x) \tag{6.14}$$

We represent the distribution for numerical attributes using Kernel Density Estimation. For a numerical attribute A with categorical parents $\mathscr{C}$ and numerical parents $\mathscr{N}$, the score is decomposed as

$$P(A|Pa(A)) = P(A|\, \mathscr{C} \cup \mathscr{N}) = \frac{P(A, \mathscr{N} |\, \mathscr{C})}{P(\mathscr{N} |\, \mathscr{C})} \tag{6.15}$$

$P(A, \mathscr{C} |\, \mathscr{C})$ and $P(\mathscr{N} |\, \mathscr{C})$ are subsequently represented by a KDE for each instantiation of the categorical attributes $\mathscr{C}$. Hence, for each $\overline{c} \in a\_dom(\mathscr{C})$, we estimate the distribution $p(A, \mathscr{N} |\, \mathscr{C} = \overline{c})$ with a KDE based on the set of numerical values $\{(a, \overline{n}) |\, \exists c \in L, \exists e \in \mathscr{C} : e.A = a, e.\mathscr{N} = \overline{n}, e.\mathscr{C} = \overline{c}\}$.

In theory this means that we can never represent $P(A, \mathscr{N} |\, \mathscr{C} = \overline{c})$ more compactly than by enumerating the complete dataset. However, it is possible to sample the found KDE and only use this sample for determining the KDE without losing much of the accuracy of the KDE. Silverman et al. [96] showed the minimum sample

size needed to obtain a good approximation given the dimensionality of the data. Therefore the complexity penalty is independent of the attribute but only depends on the dimensionality. The series of minimum sample sizes is the following:

$$[4, 19, 67, 223, 768, 2790, 10700, 43700, 187000, 842000] \tag{6.16}$$

Where the first element denotes the sample size for a distribution of 1 dimension, the second for 2 dimensions, etc. Since we have multiple KDEs when categorical parents are present, we multiply the minimum sample size with the number of categorical parents.

The overal score for attribute A in a log L is hence:

$$\sum_{c \in L} \sum_{e \in c} \log\Big(p\big(e.A|Pa(A)\big)\Big) - penalty(A), \tag{6.17}$$

where $p(e.A|Pa(A))$ is estimated using the KDE. To avoid that $e.A$ itself is used to estimate $p(e.A|Pa(A))$ we can use the leave-one-out cross-validation log-likelihood. But since we are using large datasets to train the model we can use the log-likelihood itself without sacrificing any performance

6.5 Experiments

We perform several experiments to illustrate the effectiveness of our eDBN algorithm for anomaly detection.

1. We illustrate that our algorithm can achieve high accuracy on datasets with various attributes of mixed categorical and numerical type.

2. We compare our KDE-based approach to deal with numerical attributes with the standard discretization approach, showing that KDEs slightly outperform discretization.

3. We compare our eDBN with the state-of-the-art techniques proposed by Bohmer et al. [13] and Nolle et al. [74]. We show that we clearly outperform the Likelihood Graphs proposed by Bohmer and on most datasets also performed better than the techniques introduced by Nolle et al.

For these experiments, we used a multitude of datasets, listed in table 6.3, to fully evaluate our proposed model.

A serious complication in our experimental validation is the absence of large business process logs with labeled anomalies. To alleviate these problems, we use a combination of synthetic datasets, real non-sequential datasets, and real datasets with artificial anomalies introduced. All code to reproduce the shown experiments can be found on our GitHub repository[2].

[2] https://github.com/StephenPauwels/edbn/tree/master/Anomalies

Table 6.3: Summary of datasets used for comparison.

Name	#Activities	#Cases	#Events	#Attr.	#Attr. Values
Synth	6	10,000	50,000 - 60,000	13	2 - 100
Synth-Dur	7	10,000	50,000 - 60,000	4	2 - 1,000
Breast	-	-	683	9	1 - 10
Mammo	-	-	11,183	6	numerical
Cardio	-	-	1,831	21	numerical
P2P	27	5,000	48,477	3	13 - 140
Small	41	5,000	55,058	3	13 - 141
Medium	65	5,000	39,956	3	13 - 140
Large	85	5,000	61,789	3	13 - 141
Huge	109	5,000	46,919	3	13 - 140
Gigantic	152	5,000	38774	3	13 - 141
Wide	63	5,000	39,678	3	13 - 140
BPIC15_1	398	1,199	52,217	2	23 - 398
BPIC15_2	410	832	44,354	2	11 - 410
BPIC15_3	383	1,409	59,681	2	14 - 383
BPIC15_4	356	1,053	47,293	2	10 - 356
BPIC15_5	389	1,156	59,083	2	22 - 389

6.5.1 Evaluation

In this first part of our experiments, we test the basic anomaly detection capabilities and want to test how well our model performs when more anomalies are present in the training and test data. We use the area under the Receiver Operating Characteristic curve (AUROC) to measure the quality of the predictions. We choose this method because our model returns an anomaly score for every case, which we then sort from most to least anomalous. The AUROC value gives an indication of the ROC curve itself, without having to plot graphs for every experiment. A score of 1 indicates a perfect curve, meaning no false positives and false negatives occurred during the testing phase, a score of 0.5 means the detection was just random.

We built a data generation tool that allows us to create event logs containing different (known) relations between events. To do so, we first create a model of sequential activities with depending attributes. The model is based on a process that describes the shipping of goods. Goods can have a certain value and extra insurance can be taken. Goods with extra insurance need a different workflow from goods without extra insurance. We create one model for normal execution and one model for anomalous execution, where we explicitly changed the order of events or use the wrong flow of events according to the insurance chosen. Next, we introduce extra attributes where some of these attributes depend on other attributes. The data consists of 13 attributes in total. For the anomalous cases, we added random values on random places and changed the order of activities.

We generated multiple setups with a variable number of anomalies in both

Table 6.4: Mean AUROC values for different combinations of anomalies for the Synth dataset over 10 runs.

		Test							
	% Anomalies	0.1	0.5	1.0	2.5	5.0	10.0	25.0	50.0
Train	0.0	0.96	0.95	0.93	0.93	0.94	0.94	0.94	0.94
	0.5	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
	1.0	0.87	0.91	0.90	0.90	0.90	0.90	0.90	0.90
	2.5	0.91	0.92	0.90	0.90	0.90	0.90	0.90	0.90

Table 6.5: Mean AUROC values for different combinations of anomalies for the Synth-duration dataset over 10 runs.

		Test							
	% Anomalies	0.1	0.5	1.0	2.5	5.0	10.0	25.0	50.0
Train	0.0	0.93	0.94	0.93	0.93	0.92	0.93	0.93	0.93
	0.5	0.91	0.92	0.91	0.91	0.92	0.92	0.92	0.92
	1.0	0.88	0.92	0.92	0.91	0.91	0.91	0.90	0.91
	2.5	0.90	0.92	0.91	0.90	0.90	0.90	0.91	0.91

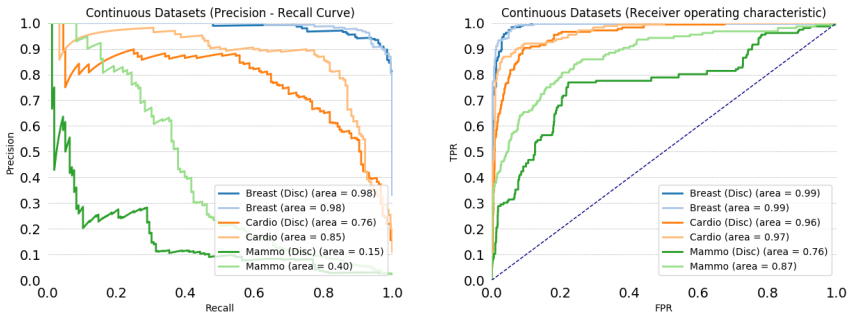
training and test data. We added anomalies in our training data to check and show that our approach does not require a flawless event log as training data but can deal with a small amount of unexpected behavior in the data. To minimize the impact of the random generation of the data we run every test 10 times and report the mean AUROC value of all runs. The AUROC-scores for different amounts of anomalies in both training and test data can be found in Table 6.4. This test shows that our algorithm can find the relations mentioned in Section 6.3, even when the training set contains a small amount of noise or anomalies.

To test the detection of duration anomalies, we use a synthetic dataset with an easy process, meaning few variations in activities. To determine the duration between two events, we use Gaussian distributions, where we use unique Gaussians for every combination of activities. Furthermore, we also added a user that performed the activity and a random attribute. The results can be found in Table 6.5. These results show that our model performs evenly well when detecting temporal anomalies as when detecting categorical anomalies in datasets containing multiple attributes.

6.5.2 Evaluate Numerical Attributes

Next, we test the performance of our KDEs in comparison with a discretization method. Since there exists no business process log containing both numerical attributes and labeled anomalies, we use three existing non-sequential datasets containing labeled anomalies. These datasets are the Breast Cancer Wisconsin (breast)[3],

[3] http://odds.cs.stonybrook.edu/breast-cancer-wisconsin-original-dataset/



(a) Precision-Recall curve

(b) ROC curve

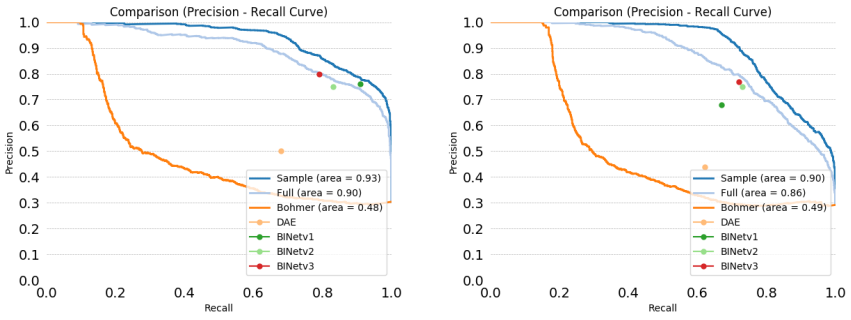
Figure 6.5: Results for the continuous datasets. Showing both results when using discretization and our numerical approach.

the Cardiotocography (cardio)[4] and Mammography (mammo)[5]. We tested these datasets both with discretization and without. The discretized version generates a model containing only categorical variables, while the original version generates a model containing numerical variables. Figure 6.5 shows the Precision-Recall curve and the ROC curve for the continuous datasets. We can see that the darker curves (corresponding to the discretized version) are always below the light curves. Meaning that the model using numerical variables performs better in detecting anomalies. The difference for the breast dataset is very small because of only a limited amount of attributes that have only integer values instead of floating-point values. Although the KDEs outperform the discretization, this experiment does show that even with a discretized approach our eDBN method also performs well.

6.5.3 Comparison

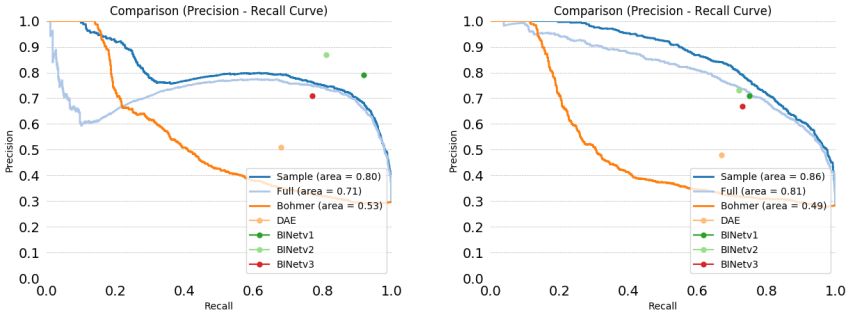
To compare with state-of-the-art methods, we have chosen to compare with the Likelihood graph [13] and neural network methods [74]. This comparison cannot be performed straightforwardly. This is because every method works differently. The Likelihood graph and neural networks all return for every case if it is an anomaly or not, while our method returns a score for every case. For the Likelihood graph, we were able to easily transform the binary classification into scores indicating how likely it is that the cases are anomalous. The Likelihood graph calculates the likelihood for the ongoing case and compares this with a baseline score to indicate if a case is an anomaly. But since the baseline score is based on the last event seen so far we can ignore this baseline because all our cases are considered done and all have the same end activity that indicates the end of the case. The lower the ongoing likelihood the more likely it is that the case contains an anomaly. Due to the way the neural networks work, and their complex way of determining if a trace is anomalous or not we were not able to transform this method. In the comparison, we thus will

[4] http://odds.cs.stonybrook.edu/cardiotocogrpahy-dataset/
[5] http://odds.cs.stonybrook.edu/mammography-dataset/



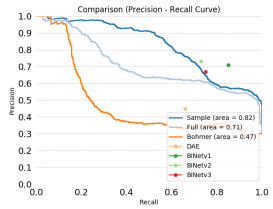
(a) Small

(b) Medium

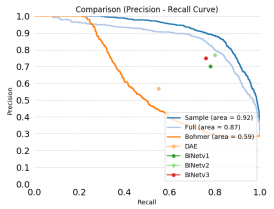


(c) Large

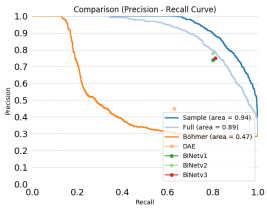
(d) Huge



(e) Gigantic



(f) Wide



(g) p2p

Figure 6.6: Comparison of precision-recall values. Sample corresponds to our method when using a different training and test set, full indicates that the entire dataset was used for training and testing.

use the Precision-Recall curve for our method and the Likelihood graph and indicate the precision-recall results of the neural networks as single points in these graphs.

Another difference between the methods is the use of training and testing datasets. The Likelihood graph needs a clean training dataset and a different testing dataset, as every case that occurs in the training dataset is assumed to be normal and will never be flagged as anomalous. The neural networks use the same dataset for both training and testing. This is mainly because they cannot cope with the occurrence of unseen values. In the previous section, we showed that our model is capable of dealing with a training dataset that contains anomalies. When we compare our method with both the Likelihood graph and the neural networks we always include two variants of the eDBN results. The first is a variant where we use a clean training set, as for the Likelihood graph. For the second variant, we use the same dataset for training and testing, as in the neural network methods.

Comparison using Synthetic Data

To prevent any bias in the data towards our model, we used the datasets generated by Nolle et al. [74]. Because the Likelihood Graph was optimized in a setting with 3 attributes (activity, resource, weekday) we use the datasets containing only these three attributes, as described in Table 6.3. The results for the 7 datasets can be found in Figure 6.6. For most datasets, we outperform the other algorithms. It also indicates that our algorithm does perform best when a clean training dataset is used, but that the performance does not diminish much when using the full dataset for training. The Likelihood Graphs score worst, indicating that they are not able to deal with more complex dependencies between the attributes and events in an event log.

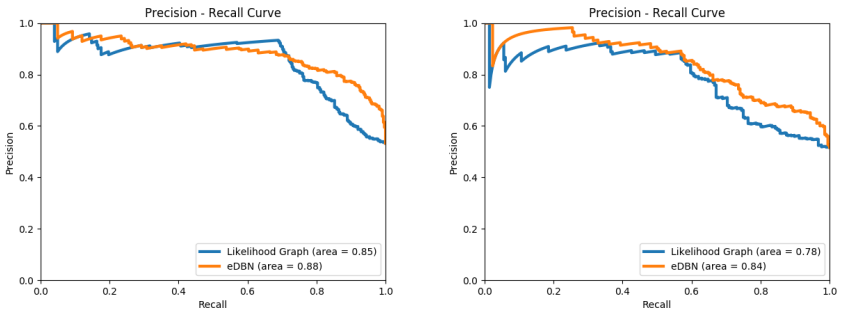
Comparison using the BPIC15 data

To compare our approach to the solution presented by Bohmer et al. we first implemented the algorithm as explained in their paper [13]. Next, we generated the data as described by Bohmer et al. starting from the reduced BPIC15 data [112]. We randomly split the original data into two equal data sets, one for training and one for testing. In the test data, we introduced anomalies according to the description by Bohmer et al. The statistics for the generated logs can be found in Table 6.6. Normal input will, however, never contain this many anomalies.

We use both the Precision-Recall-curve and the ROC-curve to compare the two approaches. The results can be found in the graphs in Figure 6.7 and 6.8 for each of

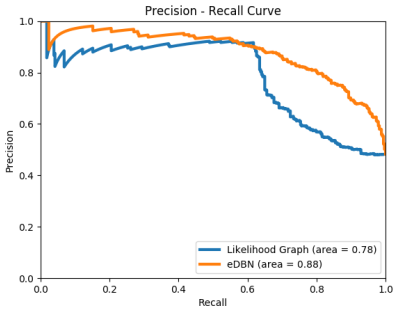
Table 6.6: Number of cases present in the different event logs.

Log	Training size	Test size	# Anomalies in Test set
BPIC15_1	589	610	291 (47.7%)
BPIC15_2	408	423	214 (50.5%)
BPIC15_3	723	686	356 (51.8%)
BPIC15_4	522	530	257 (48.4%)
BPIC15_5	595	561	283 (50.4%)

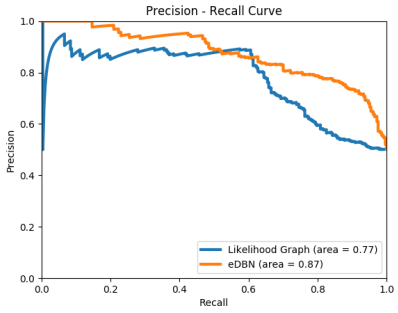


(a) BPIC15_1

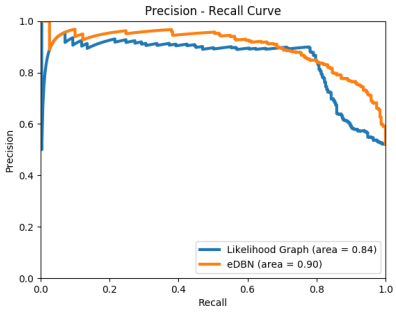
(b) BPIC15_2



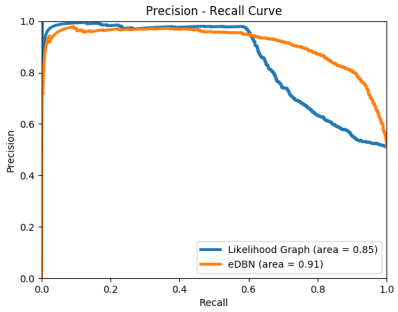
(c) BPIC15_3



(d) BPIC15_4



(e) BPIC15_5

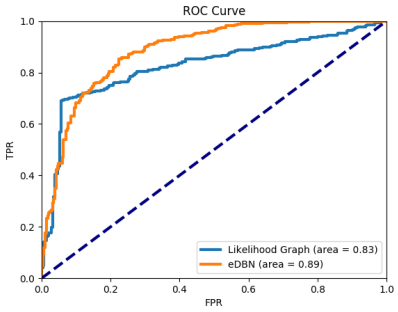


(f) Combined BPIC15

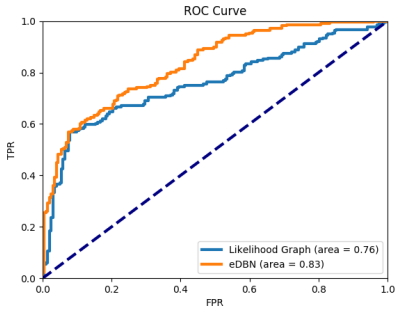
Figure 6.7: Comparison of precision/recall-graphs.

the five different municipalities. Because all five municipalities have different ways of executing the different processes, we also created one event log containing all data of all municipalities. We then introduced anomalies in the same way as we did for the other logs. This combined dataset allows us to test how well each approach can handle different processes, or different variations, in a single event log.

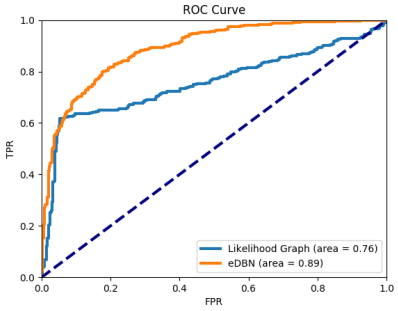
We can see that our method always outperforms the likelihood graphs using the BPIC15 data. Especially in the case where we merged all five datasets. We can



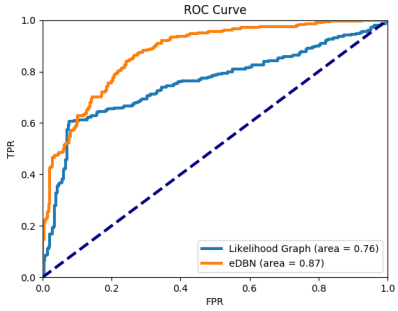
(a) BPIC15_1



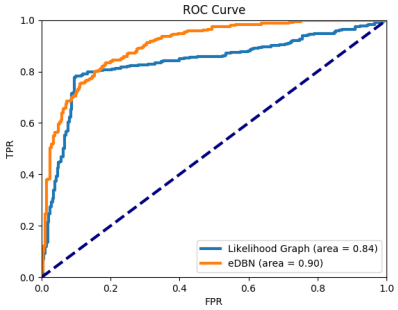
(b) BPIC15_2



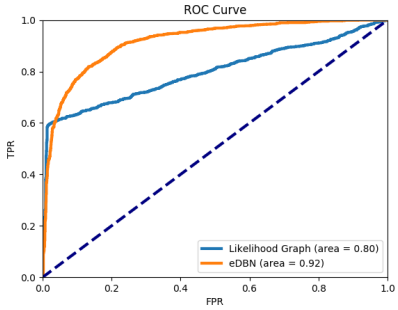
(c) BPIC15_3



(d) BPIC15_4



(e) BPIC15_5



(f) Combined BPIC15

Figure 6.8: Comparison of ROC-graphs.

Table 6.7: Overview of results for different anomaly detection techniques.

	AUC	
Method	Synth data	BPIC15 data
eDBN	1.00	0.89
eDBN without FD	0.69	0.83
Bohmer et al. [13]	1.00	0.85
FastABOD [52]	0.50	0.56
LOF [18]	0.49	0.55
SOD [53]	0.53	0.60
Feature Bagging [57]	0.75	0.57
SimpleCOP [130]	0.53	0.60
LibSVMOneClass [92]	0.51	0.46
COP [54]	0.81	0.63
DWOF [72]	0.51	0.44
OpticSOF [17]	0.53	0.46
ALOCI [77]	-	0.86
Bertens et al. [7]	-	-

conclude that our model is better in scoring the different anomalies in the data, especially when the number of processes present in the data becomes larger.

Comparison with Other Anomaly Detection Methods

We also tested our method against other anomaly detection methods (not necessarily methods that take into account the sequential or Business Process nature). We used the *k-context* format as input for all the algorithms. The best parameters were chosen after performing some experiments. We performed the experiments using the ELKI - tool [93]. Since none of these methods uses a different (clean) training dataset we used the same data for training and testing the models. The results can be seen in Table 6.7. We see that only ALOCI outperforms us on the BPIC15 data, this is because we used the same data for training and testing. Our method and the one proposed by Bohmer et al. perform best when having a clean training dataset. For some algorithms, we were not able to get results for both datasets (due to runtimes and memory usage).

6.6 Conclusion

In this chapter, we extended Dynamic Bayesian Networks using other techniques to create a new model that allows us to better and in more detail describe the structure and properties of an event log. As some attributes are very closely linked together (for example an employee name and his employee number), we extended the DBN with Functional Dependencies which can capture this behavior more closely. We changed the behavior when a Functional Dependency is broken in contrast to when

a Conditional Dependency is broken. Within an event log, some attributes can allow new values, as long as these values do not break any of the known dependencies. Therefore we alter the behavior for unseen values to take into account if a certain attribute should allow for change or not.

We showed how we use our model for both categorical and numerical data, in contrast to other state-of-the-art methods. For categorical data we use the standard conditional probability tables that we used before, to deal with numerical attributes we use kernel density estimation. To be able to deal with these KDEs we also updated the learning method for our model. Using the DBN as the basis for our model, lets us learn and use the dependencies between different attributes between different time steps. Unlike having to hard-code the used attributes in the code, our model can use a varying amount of extra attributes during the learning phase. We performed several evaluations using different, synthetic and real-life, datasets. These results showed that our model outperforms existing models. We also show that our model can deal with noise in the reference event log.

An important issue with evaluating anomaly detection methods in Business Processes is the lack of anomaly datasets. To test using existing datasets, we artificially inject anomalies in the datasets. However, these datasets may already contain (unknown) anomalies. This makes that our labels for the datasets might not be fully correct, resulting in wrong precision/recall values. To tackle this issue we can use synthetic datasets, but they might be engineered towards the methods we implemented, overfitting the data to the models. To reduce the risk of over-engineering the data towards our model, we used the data that was generated by Nolle et al. [74].

In the next chapters, we take a closer look at the explainability of the eDBN method. This visual model allows the user to inspect the model before using it, or even to make small changes to.

CHAPTER 7

Detecting and Explaining Drifts in Yearly Grant Applications

During the lifetime of business processes, changes can be made to the workflow, the required resources, required documents, ... Different cases from the same Business Process within a single event log can thus differ substantially due to these changes. We propose a method that is able to detect concept drift in multivariate logs with a various attributes. We test our approach on the BPI Challenge 2018 data consisting of applications for EU direct payment from farmers in Germany where we use it to detect Concept Drift. In contrast to other methods, our algorithm does not require the manual selection of the features used to detect drift. Our method first creates a model that captures the relations between attributes and between events of different time steps. This model is then used to score every event and case. These scores can be used to detect outlying cases and concept drift. Thanks to the decomposability of the score we can perform a detailed root-cause analysis. The work in this chapter clearly showcases the ability of DBNs to produce interpretable models and explanations of the returned results.[1]

[1] This chapter is based on *Stephen Pauwels and Toon Calders. Detecting and explaining drifts in yearly grant applications. In Business Process Intelligence Challenge (BPI Challenge'18).*

7.1 Introduction

Event logs contain valuable information concerning the execution of different business processes in an organization [108]. By applying different mining techniques on these logs we can extract the workflow, have a better understanding of the resource management, show differences between departments, ... Most process mining techniques require a stable state to perform well. Business processes, however, are prone to different kinds of changes, both expected, documented changes as well as unexpected changes that may occur. These changes can be induced by new company policies, new regulations, new activities, ... Concept Drift Detection tries to identify the possible change points (i.e. the points in time where changes to the process happen) and tries to explain what caused the drift. After detecting these points, it is possible to split the log into smaller logs that do represent the same execution of a Business Process without changes and use standard process mining techniques to further examine the different processes [97].

The model introduced in Chapter 6 can score cases to test for anomalies within an event log. In this chapter, we show that the same model can be used to detect drifts in an event log. Rather than providing a fully automated solution, we offer visual aids that people can use to analyze data. Data that is properly visualized can easily be investigated by people with often better results than both fully automated systems or fully manual systems [48].

We use the BPI Challenge 2018 dataset [110] to show the capabilities of our drift detection algorithm. The dataset consists of applications for EU direct payments from farmers in Germany. Due to changes in regulations, the process changes every year making it a suitable dataset for testing our method.

Our goal is to provide visual aids that allow us to easily detect drifts and help in the investigation of the main causes of these drifts. These visual aids are designed to be easily understood by both business experts and technical people.

This chapter is structured as follows: we describe the related work within the field of concept drift detection. The dataset itself is described in Section 7.3 together with the few preprocessing steps we have made. In Section 7.4 we explain the model used for scoring the events and we introduce a general workflow that uses these scores to determine concept drift. In Section 7.5 we show the results and analysis of the BPIC 2018 data, showing that our method does provide useful tools for detecting and explaining concept drift.

7.2 Related Work

Bose et al. [14, 15] propose a concept drift detection method that uses statistical tests over feature vectors. These feature vectors are based on the order in which activities occur in the workflow. They record for example how many activities always, sometimes, or never occur after a given activity. Other measures can be used to add extra features that may incorporate the other attributes in the log. The method does require manual selection of the exact features to be used in the statistical tests. They also show that concept drift can occur in all perspectives (control-flow, data, resource) and that different types of drift exist, each of which may require a different approach to deal with it. A disadvantage of this technique is, however, that drifts are

only detected after they occurred, leading to a delay in the ability to update existing models.

To detect drifts, Bose et al. use a sliding window approach. This window is divided into two sub-windows which we want to compare to each other to determine if drift has occurred between the two windows. The characterization of changes between windows is done using a statistical test, such as the Kolmogorov-Smirnov test, the Mann-Whitney U test, or the Hotelling T^2 test.

Maaradji et al. [65] propose an automated method for detecting drifts. They focus only on the activities within the process and transform the cases in the log into partial order runs. These runs summarize the different workflows possible for the process. To detect drifts, they search for statistically significant changes in the distribution of the most recent runs.

Accorsi et al. [1] propose to use clustering of cases to detect drifts in the process. The clustering is done by using the number of intermediate activities between activities within different windows.

Carmona et al. [24] create an abstract representation of the process in the form of a polyhedron, which is afterward used to test how well the representation includes the cases from the log. When the cases differ from the representation a drift is detected. To find other drifts the entire method needs to be repeated. This method can work in an online environment where we can check for drifts during the execution of the process, whereas other methods require an event log containing completed cases.

7.3 Dataset

The dataset [110] consists of applications for EU agricultural grants from the last three years. It contains both attributes that were known when the application was submitted and attributes that contain the outcome of the application after the case finished. We call the former type of attributes the *basic attributes* and the latter type the *derived attributes*.

The basic attributes contain information about the activity, the applicant, his parcels, the document used, etc, whereas the derived attributes contain whether or not the grant was assigned, possible penalties on the requested amount, and the actual amount granted.

Unless otherwise stated, we only use the basic attributes for building and testing the model, making it possible to create a model that can be used to test cases while they are still going on and detect drift as soon as possible.

The data consists of 2,514,266 events grouped in 43,809 cases each representing a single application. The shortest case contains only 24 events, the longest 2,973 and on average there are 57 events per case.

Example 19. *A typical snippet of an example case can be found in Table 7.1, which is taken from the BPIC dataset. Some attributes remain the same within a case (i.e.* case, year *and* department*) while others may have different values.*

Table 7.1: Example case from the event log.

case	activity	doctype	subprocess	success	year	dept.
8b99873a6136cfa6	mail income	Payment application	Application	true	2015	e7
8b99873a6136cfa6	mail valid	Payment application	Application	true	2015	e7
8b99873a6136cfa6	mail valid	Entitlement application	Main	true	2015	e7
8b99873a6136cfa6	initialize	Parcel document	Main	true	2015	e7
8b99873a6136cfa6	initialize	Control summary	Main	true	2015	e7
8b99873a6136cfa6	begin editing	Control summary	Main	true	2015	e7
8b99873a6136cfa6	finish editing	Control summary	Main	true	2015	e7
⋮	⋮	⋮	⋮	⋮	⋮	⋮
8b99873a6136cfa6	begin payment	Payment application	Application	true	2015	e7
8b99873a6136cfa6	finish payment	Payment application	Application	true	2015	e7

7.4 Concept Drift Detection

In this section, we first explain the model we use for scoring the events and then introduce a workflow for detecting drifts and finding differences in the data.

7.4.1 The Model

Our eDBN model is based on Dynamic Bayesian Networks (DBNs) [89], an extension of the well-known Bayesian Networks. These networks are capable of capturing Conditional Dependencies between various variables both within one single time step and over different time steps. As we have indicated in Chapter 6, a DBN contains useful elements but lacks some expressiveness to be useful in the context of Business Process logs. We introduced a second type of dependency, the Functional Dependency (FD), that describes mappings of values from one attribute to another.

Example 20. *An example structure based on the dataset for an eDBN is given in Figure 7.1. In the dataset, every applicant has the same value for* area *and* young_farmer *within a single case. Therefore there exist a Functional Dependency from* applicant *to both* area *and* young_farmer *in the current time step. And since these values do not change between events in a case we also link each of these attributes from the previous time step to the current time step.*

The attribute activity *in the previous time step has a Conditional Dependency to* activity *in the current time step because the current activity depends on the previous. Branches in the workflow may occur so multiple activities can follow a certain activity, explaining why this is a CD and not an FD.*

After finding the different dependencies within the model we have to learn the parameters to be able to score events. These parameters consist of all the Conditional Probability Tables, the Functional Dependency Mappings, and the probability of encountering new values or new relations. We add these new values *and* new relations *measures to the model to be able to handle new (unseen) values and relations. Dynamic Bayesian Networks return a probability of 0 when encountering a value that was not present in the training dataset, but event logs do allow the occurrence of new values (a new employee entering the company, . . .). Using these measures we can take this into account without severely penalizing the score for an event in case of a justified new value.*

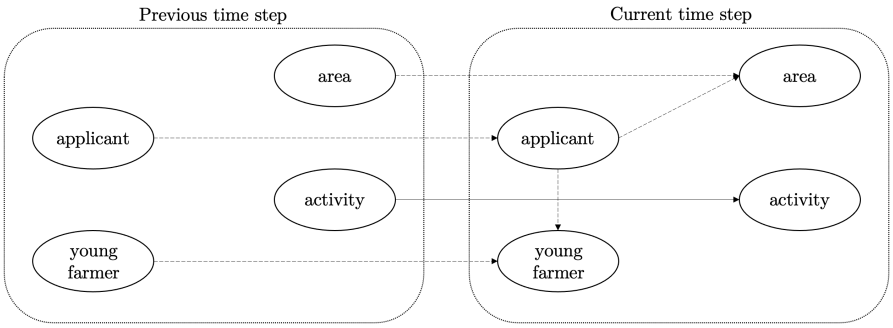


Figure 7.1: eDBN example with conditional (full) and functional dependencies (dashed) based on the dataset.

Example 21. *To learn all the parameters for the model in Figure 7.1 we start by learning the CPT for the relation activity$_1$ → activity. An example CPT is shown in Table 7.2. For the different FDs, we store the mappings that are present between values of the attributes in the training data. The FDT of the relation applicant → young farmer will link all applicants to the fact whether they are young farmers or not. To determine the new_values and new_relations rate, we calculate for every attribute in the current time step the probability of encountering a new value or relation.*

The score for an event e given the model can be calculated by:

$$Score(e) = \prod_{A \in \mathscr{A} \cup \{Activity\}} P(e.A|Pa(A))$$

with

$$\begin{aligned} P(e.A|Pa(A)) = {} & value_A(e.A) \\ & * CPT(e.A|Pa(A)) \\ & * FDT(e.A) \end{aligned}$$

Where $value_A$ indicates the score for the value itself (does it occur in the training set or is it a new value), *CPT* returns the probability for a particular value given the values of the attributes it depends on, and *FDT* returns a score based on whether or not the Functional Dependencies were broken or not.

Table 7.2: CPT for relation between $activity_1$ and *activity*.

$activity_1$	*activity*	$P(activity \mid activity_1)$
mail income	mail valid	1.0
initialize	initialize	0.6
	begin editing	0.3
	performed	0.1
begin editing	finish editing	0.7
	calculate	0.3

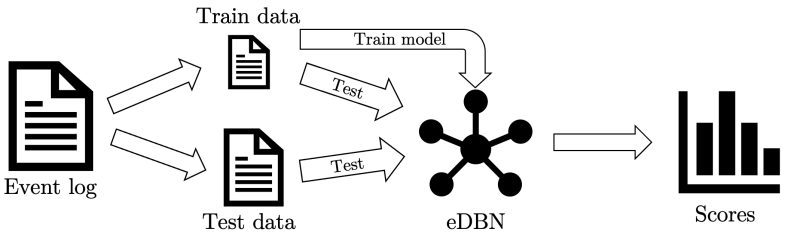


Figure 7.2: Workflow used to score the cases in the log.

The score for an event is built from the contributions of every attribute equally. This helps us to split the score into partial scores where we get the contribution of every attribute by itself. Furthermore, the partial score itself can also be decomposed into its components giving an even better and more detailed way of determining the root cause of changes and differences.

Thanks to the decomposability of the score assigned to an event we created an expressive model that can easily explain how the score was computed. Furthermore, thanks to how we composed the score, equal events and cases get approximately the same score making it easy to group cases that show the same behavior and making it possible to detect drifts and differences.

7.4.2 Detecting Drifts

First, we select the cases in the log we want to use as the reference set, we call these cases the *training dataset*. This training dataset forms the baseline where we will compare all other cases to, which we refer to as the *test dataset*. When detecting concept drift we always select the first n cases as our training dataset and compare them with the other cases. Cases that occur after a drift are supposed to receive a different score than the ones before the drift. The basic workflow of building the model and calculating the scores is given in Figure 7.2. We can also use this workflow with different kinds of training data (e.g. only selecting events that satisfy a certain condition) to find different types of drifts or differences.

To be able to compare cases we assign a score to every case based on the model we created using the training data. This score is calculated as the mean score for every event in the case. We use this score in contrast to the n-th root of the product of all event scores (where n is the number of events in the case) used in Chapter 6 as we are only looking to characterize a case using this score. When we would use the original formula to compute the score for a case, a single violation of an FD can result in the total score being 0. When looking for anomalies this is an important property as these FDs are supposed not to be broken. Also, it would be impossible to find out if only one attribute had a partial score of 0 or multiple. By using the mean score we ensure that we can see this difference.

Definition 12. *The score for a case is computed as:* $Score_{case}(c) = \frac{\sum_{e \in c} Score(e)}{|c|}$. *With* $|c|$ *equal to the number of events in a case. When there is no confusion possible between the score of an event and a case we omit the subscript.*

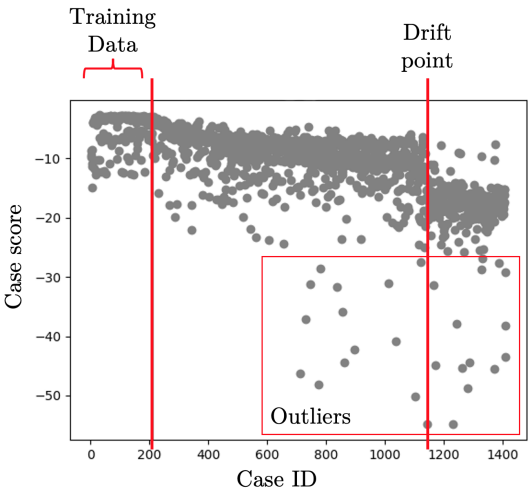


Figure 7.3: Example of scores for different cases picked from a random dataset. The first vertical red line indicates the difference between the cases used for training and testing, the second red line indicates the drift.

After we have calculated the scores for every case we can sort them according to timestamp. Cases that represent the same Business Process (without changes in the workflow or attributes) are supposed to have scores closely related to each other. After a drift occurred the scores for the cases change due to the changes that were made to the process itself. The distribution of scores after such a change point is different than before the drift.

Case-score plot

The first visual aid we introduce is the *case-score plot* which can be seen in Figure 7.3. The score of every case is plotted on this graph, giving us a first idea of the different variations present in the log. We use a logarithmic scale for plotting the different scores because the scores of the cases are very small as they are based on probabilities. The logarithmic scale helps us better visualize these differences. The x-axis in this plot is time, mostly indicated by the incremental number of the case in the log. This plot helps us to look at how the scores are distributed over time. A first phenomenon that we can observe is that the overall distribution of scores changes after a certain point in time, these changes represent drifts in the business process. A second interesting observation that can be made is that of outlying cases that deviate from the main distribution of scores. We can then decide to investigate these outlying cases in more detail, as checking all cases is often not feasible.

Drift plot

To better visualize the changes in the scores we use the two-sample Kolmogorov-Smirnov statistical test for checking the hypotheses that the distributions in both samples are similar. The test returns a p-value between 0 and 1, where 1 stands for

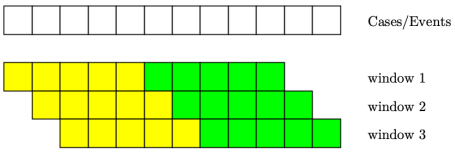


Figure 7.4: Sliding window of size 10 with the *reference window* (yellow) and *test window* (green).

accepting the hypothesis that both distributions are equal and 0 rejecting it. To find the drift points in the entire log we use a sliding window technique. The first half of the window is used as the *reference window* and the second half as the *testing window*, as can be seen in Figure 7.4. We apply the two-sample Kolmogorov-Smirnov test on both halves of the window to find possible drift points. The size of the window has to be manually set. A smaller window is more sensitive for finding the smaller drifts but is also more likely to return false positives and a larger window is less sensitive but might return more false negatives. Since we have all the scores calculated and they do not depend on the window size, it is doable to check some different window sizes and compare the plots they return. We can plot all the resulting p-values from the statistical test to visually see drift points as shown in Figure 7.5. Again we make use of a logarithmic scale since the differences are otherwise not visible. We refer to this type of plot as the *drift plot*.

Attribute-density plot

To find the root cause for the drifts we found, we take a look at the partial scores of the events, instead of the total event scores themselves. We accumulate the scores according to the attributes. We can plot all the partial case scores in an *attribute-density plot* as can been seen in Figure 7.6, as before we use a logarithmic scale.

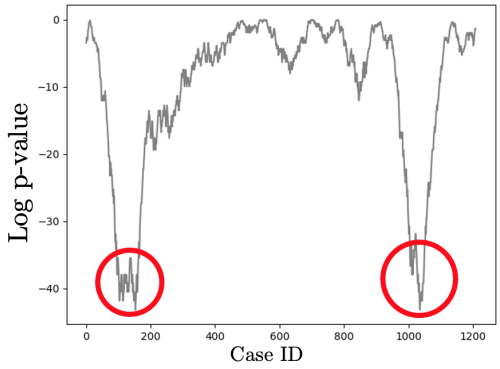


Figure 7.5: Plotting all the p-values obtained by the Kolmogorov-Smirnov test. We can see two dips in this plot. The first dip indicates the change point between the training data and test data. The second dip indicates the real drift point.

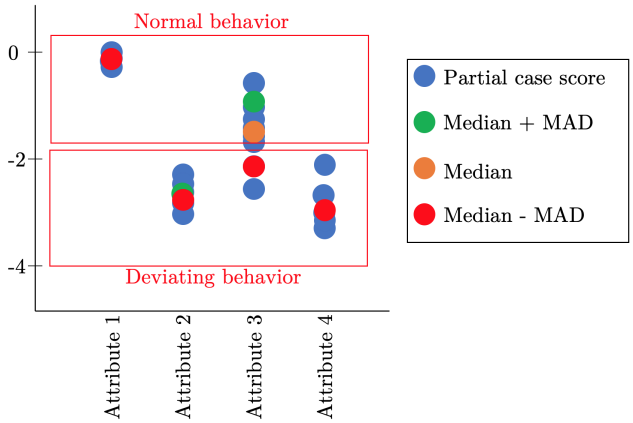


Figure 7.6: Example attribute-density plot for 4 attributes.

We add the median and Median Absolute Deviation (MAD) [59] in order to get a better insight into the density and distribution of scores for a single attribute. In contrast to the other types of plots, these plots characterize the data in terms of the different attributes. To be able to use these plots for comparing different segments of a log we first select a reference segment which is used to learn the model used for calculating the partial event scores of all segments. Every segment can be visualized in a different attribute-density plot for comparison. Any technique can be used to help users find differences between these plots. In the remainder of this chapter, we plotted all the median values on the same plot to determine differences.

Towards a general workflow

Figure 7.7 shows the relations of the three different plots we introduced. The case-score plot and drift plot are used to divide the log into different segments where each segment contains cases with similar behavior. This is done using the temporal aspect of the log and our sliding window technique. To better understand the differences and equalities among the segments we first take a reference segment (or a subset of a segment) that is used to train the model. This model represents the expected cases in the event log based on the reference segment. It can then be applied to every other segment (including the reference segment) to get the partial scores for the cases in that segment. These partial scores can be plotted on an attribute-density plot, making it possible to find the root cause for the difference between the segments. This general workflow for determining the root cause is shown in Figure 7.8. All of the analyses performed in Section 7.5 are based on this general workflow.

7.5 Analyzing the Data

In this section, we analyze the BPI Challenge 2018 data to show the effectiveness of our proposed workflow. And we show that it can find interesting insights in the given data.

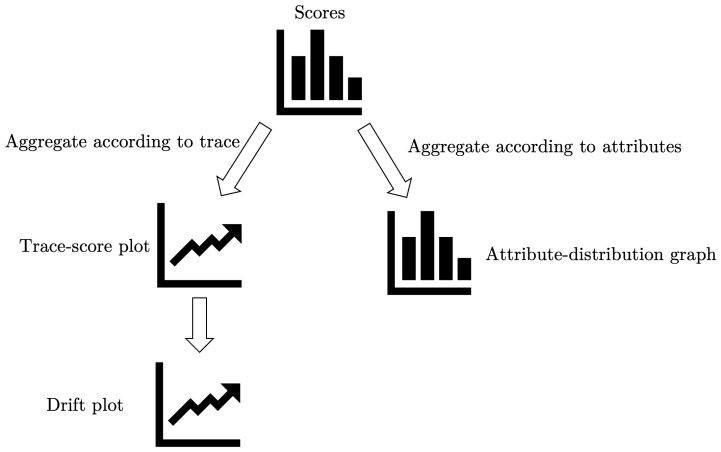


Figure 7.7: Workflow of building the different types of plots and showing how they relate to each other.

7.5.1 Preprocessing of the Data

Although our method does not require any preprocessing, we perform a little bit of preprocessing to avoid overly complex models without any added value over more simple models. We perform the following steps:

1. Since our implementation works with CSV-files we first use the Prom Tool [114] to convert the given XES-files.

2. We filter out all the *derived attributes* as we are only interested in the attributes that really depend on the activity and not the outcome of the process.

3. Next we filter out the attributes *eventid*, *event_identify_id* and *identity_id* since their respective values are unique for every event and would thus form Functional Dependencies with every other attribute in the event.

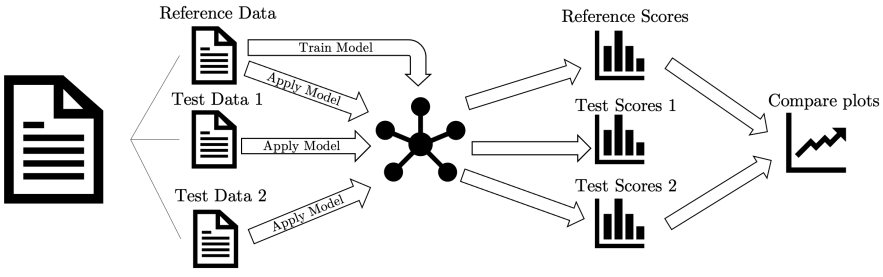


Figure 7.8: General workflow for describing changes between segments of an event log.

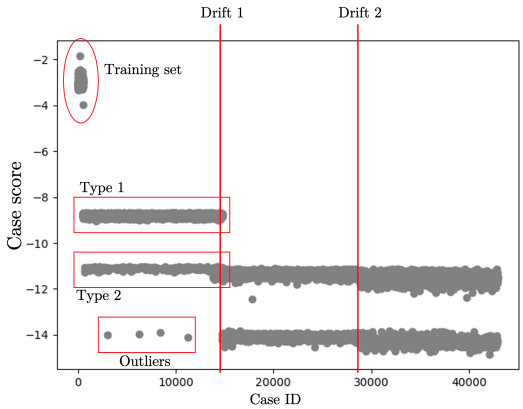


Figure 7.9: Annotated case-score plot.

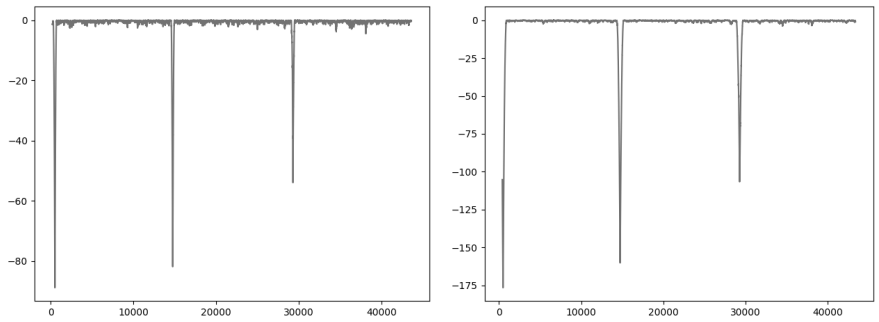
4. Because we are primarily looking to test our model for concept drift detection, we also omit the *year* attribute as the process changes every year and we do not want our model to be biased towards this attribute.

All of these preprocessing steps were repeated for every experiment conducted on the data unless explicitly stated otherwise. Important to note is that our preprocessing steps did not alter the nature of the log or any of the processes in the log.

7.5.2 Finding Drift Points

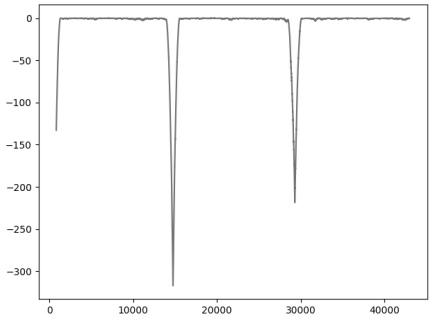
To look for any drift points in the data we first learn a model according to the log and use this to test all the cases in the following way:

1. As a first step we read the data and perform the preprocessing step.
2. To learn our model we use the first 30,000 events from the log as the training dataset. We use the same dataset for training the model structure as for training all the parameters.
3. For testing we use all the preprocessed data, including the training set.
4. We calculate the score for every event in the log given the model.
5. We calculate the mean score for every case. This is the *case score* that we use to further visualize the drifts in the data.
6. As the first visual step we plot all the scores from the different cases in a case-score plot, according to the timestamp of the first event of every case. This is shown in Figure 7.9.
7. Looking at this plot we can already get some insights about the data:
    - The first cluster of cases indicates the cases used for training the model (we can ignore this cluster).



(a) Windows size of 400

(b) Windows size of 800



(c) Windows size of 1600

Figure 7.10: Drift plots containing the p-values for different window sizes.

- We can see a difference in scores around case 14,000. Which is a clear indication for a drift point.
- A second possible drift point can be seen around case 29,000, using the drift log in the next step will make it possible to determine if this is a drift point.
- The cases cluster together in two distinct groups. Indicating that the data has mainly two types of cases, each with its own characteristics.
- We can see 4 outlying cases before the first drift point.

These insights are visually highlighted in Figure 7.9 and further explored in the next subsections.

8. Next we perform the Kolmogorov-Smirnov test with a sliding window over the scores. The resulting drift plots for different window sizes can be found in Figure 7.10. These plots clearly show the 2 (3 including the training data) drifts occurring in the data, confirming our initial thoughts.

We show in the next subsections that our model is able to explore our findings even further in order to give an explanation for the observed behavior.

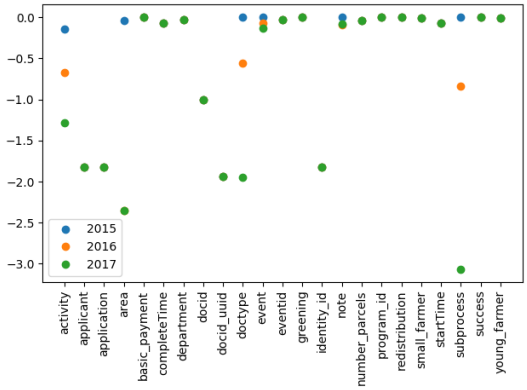


Figure 7.11: Plot showing all the median attribute values for every year.

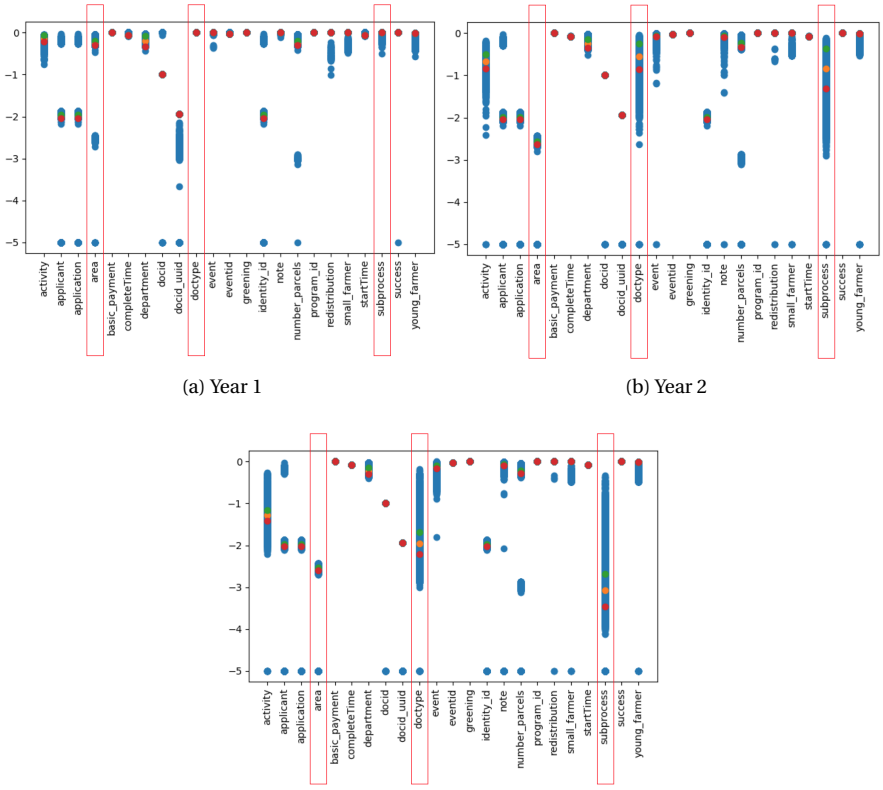
7.5.3 Inspecting the Different Attributes

To find the root cause of drifts in the log we use the attribute-density plot to visually represent the characteristics of a process. To do so we perform the following steps:

1. Learn the model as before.
2. Get the partial scores of every attribute from every event and combine these for the different attributes (in contrast to accumulating the total score for an event as we did before).
3. Using these partial scores we create multiple attribute-density plots, one plot for every period between drifts. These plots can be used to compare the different segments with each other to determine the root cause for the drifts.
4. Figure 7.12 shows the attribute-density plots for the three different years.
5. To better compare these plots we plotted all the median values in one plot. This plot can be found in Figure 7.11.

Using the plots in Figure 7.12 and 7.11 we can see three big changes between the first year, and the other two years.

- The first difference is found for the attribute *area*, this can be explained by the fact that farmers acquire new land or sell parts of their lands. It is also possible that the used technique for discretizing influences this attribute, as the different bins can change over the years.
- The following two changes are probably closely related to each other. Starting in the second year, a new document type was used for the applications, this explains the large differences for this attribute between the different years. And it is normal that due to the changes to the documents, the subprocesses also change.



(a) Year 1

(b) Year 2

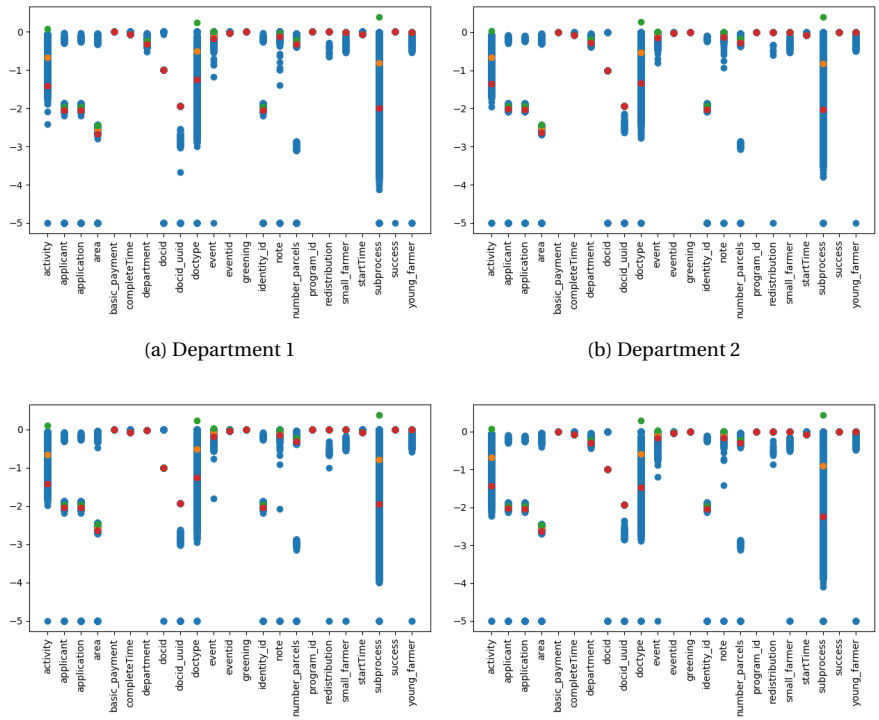
(c) Year 3

Figure 7.12: The attribute-density plots for the different years, where we indicated the most changing attributes.

- Between years 2 and 3 we have the largest difference in the subprocess attribute but less in the other attributes, indicating only some changes in the different subprocesses have occurred.

We have thus found two drift points: one at the beginning of each new year. The difference between years 1 and 2 was large as there is a significant change in the area but most importantly a new document type was used that had its consequences on the used subprocesses. It is also possible that this new document type influenced the way the area had to be filled in. Between years 2 and 3 the difference is less large, this was already visible in the case-score plot. The most important change between these years was the way the subprocesses were used.

To find a more detailed analysis of why this attribute changed that much, other tools and methods can be used to give more detailed explanations and insights.



(a) Department 1

(b) Department 2

(c) Department 3

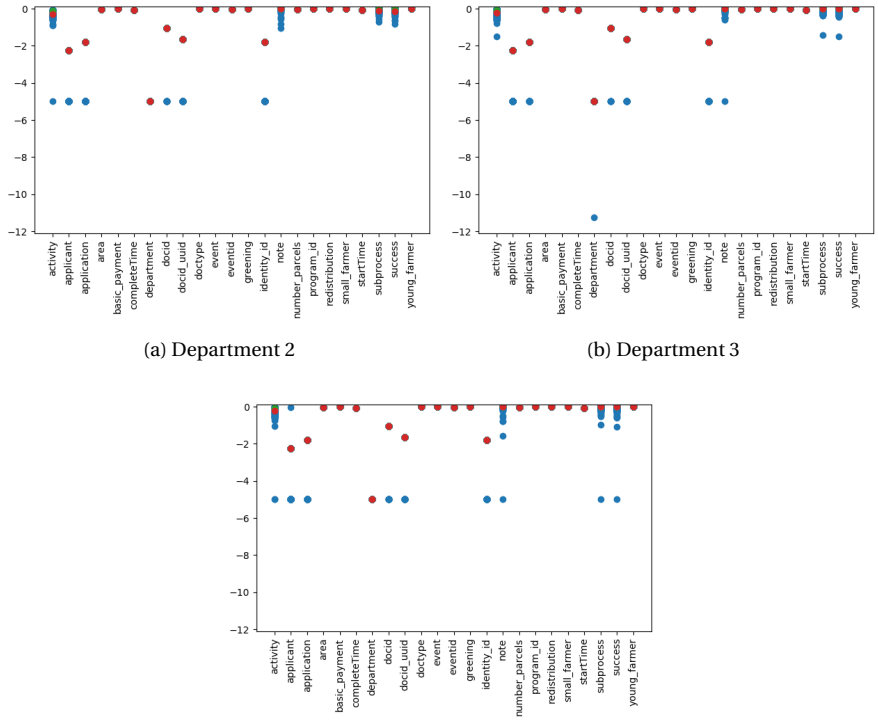
(d) Department 4

Figure 7.13: Attribute-density plots for the different departments. When using the first 30,000 events as training data.

7.5.4 Comparing Departments with Each Other

The method we used in the previous sections can be generalized to look for differences between groups of cases. We now take a look at possible differences between departments. As a first experiment, we use the same training data as before and plot an attribute-density plot for every single department. These plots can be found in Figure 7.13. When comparing these departments with each other we cannot see a clear difference in the scores for the attributes.

To better test differences between departments, we train our model with events from department 1 and use the other departments as separate test segments. This approach can be generalized for testing any kind of difference. We take a reference dataset for training and use the resulting model for testing other segments of cases. As before we show the attribute-density plots created for every department in Figure 7.14. We were looking for differences in activities and subprocesses, but when comparing the plots we cannot see a substantial difference between any attribute.



(a) Department 2

(b) Department 3

(c) Department 4

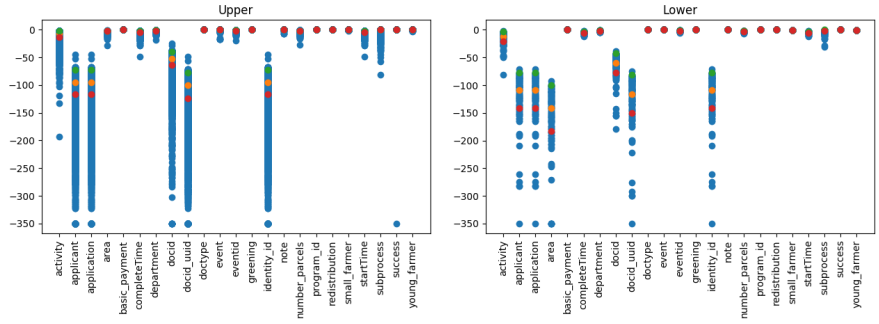
Figure 7.14: Attribute-density plots after training with department 1.

7.5.5 Difference Between Types of Cases

In the case-score plot, we saw that within every year we have two types of cases. In this section, we take a closer look by decomposing the score even further. We only investigate cases that occur in the first year, as otherwise other drifts and differences might interfere. The first step is to divide the cases into two parts: we have the first type (score between -8 en -10) and the second type (score between -10 and -12). These two types are also highlighted in Figure 7.9. To find a possible explanation for this difference we create an attribute-density plot for both types of cases. These plots can be found in Figure 7.15.

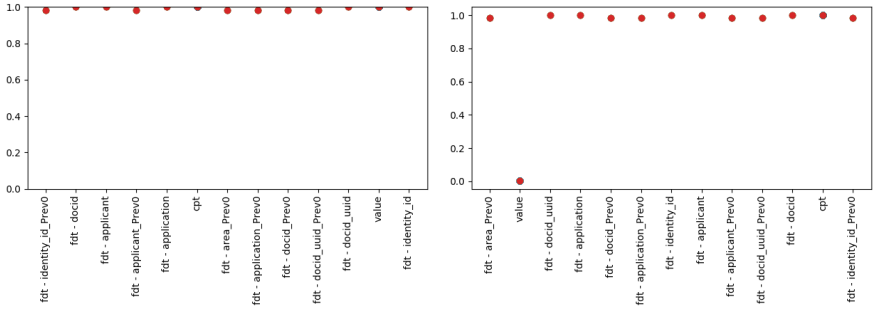
When comparing these two plots we can see that there is a big difference in the scores for the *area* attribute, which causes the split in two clusters. The score for the attribute *area* is further decomposable in different partial scores: one score is related to the value of the attribute itself. If it is already encountered in the data it returns 1, otherwise, it returns the probability of encountering a new value for that attribute based on the training data. The next partial score we got from the Conditional Probability Table, but since the attribute has no Conditional Dependencies this returns 1. Last we get a score for every single Functional Dependency. The results of all these scores can be found in Figure 7.16 for both clusters.

The comparison between these two plots shows that the difference between the



(a) Cases of type 1 (b) Cases of type 2

Figure 7.15: Attribute-density plot for both types of cases



(a) Cases of type 1 (b) Cases of type 2

Figure 7.16: Detailed attribute-density plot for the attribute area for both types of cases.

two types is caused by the fact that more unseen values for the area are encountered in cases from type 2. Thanks to these plots we have a clear insight into which aspect of which attribute causes these deviations. Now we only have to analyze the area attribute in more detail without having to check all attributes present in the log. This effect is, again, due to the way the attribute area was discretized. In the first year, we have 148 different values for the area attribute. Of these 148 values, only 2 of these values did not appear in the training data and causes the effect of the two types of cases. Since it is a result of how the data was initially preprocessed we cannot make any conclusions about the real process.

7.5.6 Investigating Anomalous Events in the First Year

In the case-score plot, we saw that four dots were not part of any of the two clusters. We use the attribute-density plot to quickly look for any significant differences between the median found in the attribute-density plot for the first year (Figure 7.12) and the scores obtained for these four cases (Figure 7.17). When comparing these

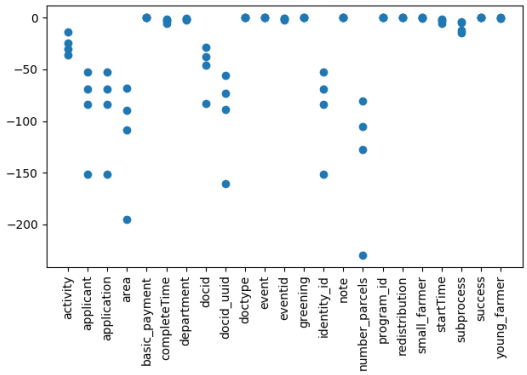


Figure 7.17: Attribute-density plot for the outlier cases

cases to the expected scores we see that especially the *area* and *number_parcels* attributes have scores that are unexpected when comparing them to the attribute-density plot for the first year. When we investigate these findings we see that these four cases are the only cases in the first year with both the area and number_parcels equal to 0, thus indeed indicating outlying cases.

7.6 Conclusion

We showed that we can use the explainability of our eDBN to detect and analyze Concept Drift in event logs. We also showed that by changing the reference dataset, we can analyze other differences within an event log such as the differences in behavior between departments. The way the eDBN model scores events and cases is appropriate for generating a more in-depth analysis of why certain cases do deviate. The analysis of the data proves the effectiveness of our method on a real-life dataset where we were able to split the data into the three different years present in the log by only using our drift points. In contrast to existing work, our model looks further than only the activity perspective. By including the data perspective, we can formulate other conclusions on occurring drifts or other changes within a process.

In this chapter we were interested in answering two questions of the BPI Challenge 2018:

1. How can one characterize these differences as a particular instantiation of concept drift?

2. How can one characterize the differences between departments and is there indeed a relation?

To answer this first question we used our case-score plot and drift plot to first detect the drift itself. We showed that our model did manage to accurately find the drift points in the data. To characterize these drifts we further investigated the segments in the log between these drifts. Using the attribute-density plot we

detected that the document type and subprocesses are the main causes for the drift between the years.

The approach of comparing different years with each other can be generalized to solve the second question. We used the first department for training our model and tested the other departments against this model. Besides the normal deviations in attributes representing IDs, we did not note any difference between the departments.

As the last analysis, we analyzed individual, outlying, cases, and small clusters of cases. Again we used the attribute-density plots to find differences between either two clusters or between some events and the expected scores. This method proved to be useful as we could successfully explain certain differences in the given event log.

The applications mentioned in this chapter are a starting point and give a clear picture of the explanation capabilities of extended Dynamic Bayesian Networks. The way our analysis is structured should give insight into the logical steps one can follow to explore multivariate event logs. We implemented the techniques used in this chapter in a single tool, showing the ease and usefulness of the different visual aids when analyzing event logs. This tool, called ACD2, is presented in Chapter 8.

CHAPTER 8

ACD2: Interactively Explore Event Logs

ACD2 is a tool for detecting anomalies and concept drift in Business process logs. In contrast to many other existing algorithms and tools, ACD2 does not require manual parameters to be set. ACD2 is based on extended Dynamic Bayesian Networks. These models are constructed automatically using machine learning but can be checked and changed by the user afterwards. This model can then be used for scoring cases. Our tool visually represents these scores in different ways, making it easy for a user to investigate the data and to get useful insights.[1]

[1] This chapter is based on *Stephen Pauwels and Toon Calders. ACD2: A Tool to interactively Explore Business Process Logs. In BPM (PhD/Demos), pages 129–133, 2019.*

8.1 Introduction

ACD2 is a fully interactive, easy-to-use tool based on our winning submission for the BPI 2018 Challenge introduced in the previous chapter. Thanks to the positive feedback we received, we have further elaborated the ideas presented in the report and incorporated them into this tool. The tool provides an intuitive way to test for anomalies or concept drift in event logs. The tool aims at both Business Process experts and domain-specific experts with no knowledge about the underlying algorithms.

An extended Dynamic Bayesian Network (eDBN) captures the different relations between attributes in a log. This model learns the normal behavior found in the log. An eDBN represents a joint probability distribution and can therefore be used for scoring new events and cases (the case score is the accumulation of event scores), the score indicates how much a case is compliant with a reference event log. An important property of this score is its decomposability; we can easily get the score contribution of individual attributes. In the remainder of this chapter, we refer to this score when talking about the score of an attribute, event, or case. For more info on eDBNs, we refer to Chapter 6.

The main workflow of the tool is to first use an event log to learn the eDBN model structure. The user can then inspect the learned model and make changes to the structure. When the user is happy with the current model, she can use it to test a log for deviations, anomalies, or drifts given a reference log containing normal behavior.

Performing both the training and testing phases can be a computationally heavy task. Therefore we have created a client-server-based application, where all heavy computations happen on the server. Because the tool is fully web-based it can also be used on tablets and smartphones, increasing its usability for non-experts.

8.2 Functionalities

ACD2 consist out of the following main parts:

1. Loading datasets
2. Learning the model of the EDBN
3. Inspecting and updating the Model
4. Testing a dataset
5. Anomaly Detection

8.2.1 Loading the Data

The first step is to upload the datasets we want to use for both learning and testing the model. After uploading, the user can select which attributes she wants to include in the dataset and indicate which attribute corresponds to the case identifier, the time attribute, and an (optional) label, indicating if the event is anomalous or not. After uploading, a background task starts to preprocess and store the data on the server for easy access later. The card on the dataset page indicates when preprocessing is done.

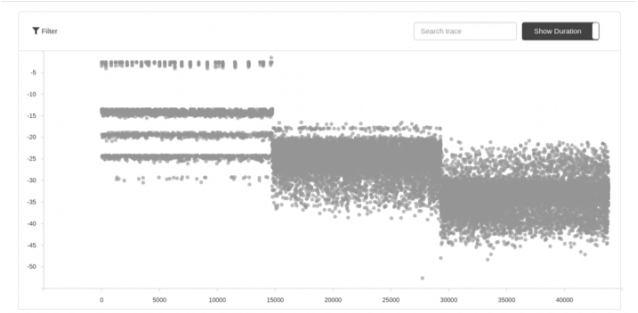


Figure 8.1: Case graph with time (or ID) on the x-axis and the score on the y-axis.

8.2.2 Learning the Model

In this phase, all relations between attributes that are present in a given dataset are learned. We have chosen to use an asynchronous process because learning the model is computationally demanding and can take some time. When learned, the model gets saved on the server and can be updated by the user or used for testing.

8.2.3 Inspecting and Updating the Model

Once a model is fully learned it can be inspected by the user, who can make changes according to her own insights into the data. Every time the user makes a change the difference in quality between the updated model and the original model is calculated and displayed. We use the Akaike Information Criterion [2] for determining the quality of a model. This metric takes the log-likelihood of the reference data given the model and the complexity of the model into account.

8.2.4 Testing a Dataset

Now we can test a new log against the learned model. At this point the model only contains the structure of the eDBN, therefore we first need to further train the different parameters in the model. We thus select a training dataset and a test dataset. An important constraint on these datasets is that they all need to have the same attributes as the original dataset used for learning the structure to be compatible. After submitting, the model is further trained and the scores are calculated for the test dataset. When all results are computed a user can start analyzing the results.

The result page consists of three main parts. The middle graph plots all cases in the test log according to their timestamp (or ID) and score, an example graph is shown in Figure 8.1. This graph forms the basis of any analysis. It can be used to analyze global behavior, to find clusters of cases (cases with similar properties or behavior), or to inspect individual cases. A low score for a case (or event) means that they do deviate more from the normal behavior, learned from the training dataset. When clicking on a single case, a new table appears below the case graph showing all events in that particular case. Events that deviate more than one standard deviation from the mean score of events are highlighted in red to indicate a possible inconsistency.

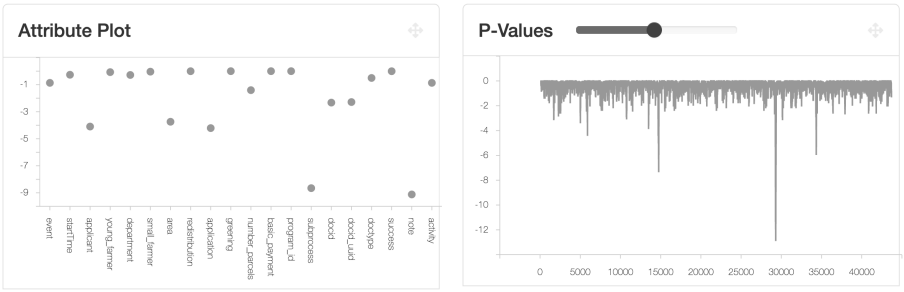


Figure 8.2: The Attribute plot and P-values used for explaining and detecting drifts.

Above the case graph, we show the attribute plot. For every attribute in the data, it shows the mean value of the score for this attribute in the test set. And again, the lower the score, the more the scores for this attribute deviate from the training dataset. To compare a subset of the data to the entire dataset, we can set a filter for creating a comparison dataset. Using this we can easily see in which attribute(s) a drift occurs.

Drifts in the data can be detected by looking at the case scores. After a drift the distribution of these scores changes. Sometimes these changes can be seen on the graph itself. To determine the drift point(s) in more detail, we use the P-values plot, which is created using the Kolmogorov-Smirnov test (KS-test) [67]. The KS-test checks if two distributions are similar or different. We compare some case scores before a possible drift point to some scores after the drift point. The lower the p-value the more different the distributions are. Examples of these graphs can be found in Figure 8.2.

8.2.5 Anomaly Detection

Besides concept drift detection, ACD2 can also be used for detecting anomalies in the test dataset. The lower a case in the case graph, the more likely it is to be an anomaly. We can also use the tool with labeled data in order to examine the quality of a certain anomaly detection method. When a user selected a label attribute, two extra graphs are added to the analysis page, namely, the Precision-Recall curve

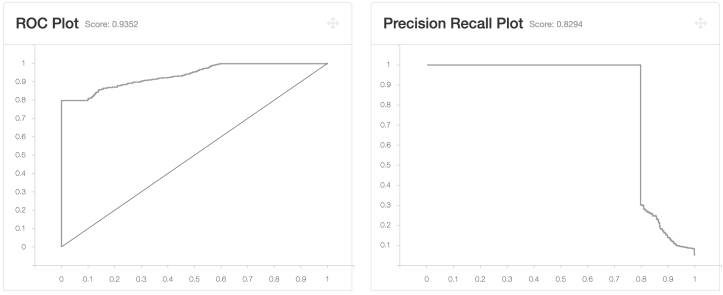


Figure 8.3: The ROC and PR curve as shown in the tool.

(PR-curve) and the Receiver Operating Characteristic curve (ROC-curve) [30], which both give an indication of the quality of the scores given to the cases. Example PR and ROC curves can be seen in Figure 8.3.

8.3 Use Case

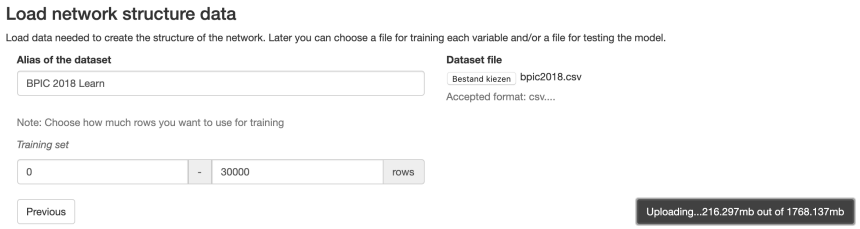
In this section, we show in detail and step-by-step how our tool can be used for analyzing the BPI 2018 Challenge data[2].

The description of the data indicates different documents and procedures that get used in the different years. During this use case, we want to test if our tool can detect these drifts and give an explanation for them.

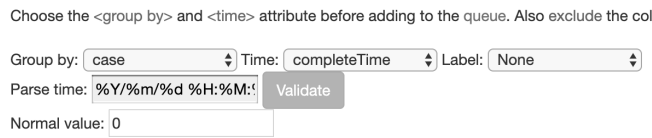
8.3.1 Preparing the Data

The first step in our analysis is to upload the dataset. At the moment our tool only allows for .csv files. Using Prom we have converted the xes-format to csv.

On the data upload page, we can give an alias to our new dataset, we call it 'BPIC 2018 Learn'. Next, we select which file from our local machine we want to upload and select the number of rows we want to use for this dataset. For our experiment, we set the training set going from row 0 to row 30,000. This is set like this because we want to train our model on the first part of the full dataset and afterward compare the rest of the data with this training set. Now we can start uploading the file and the first analysis is performed to select the attributes that are present in the data.



Next, we get an overview of all attributes. We also have to indicate some of the special attributes present in the log, like the case ID, the time, and optionally a label. We also have to give the exact time format used in the log, so the tool can correctly interpret the time. For the BPIC 2018 data, we have to use the following format: *%Y/%m/%d %H:%M:%S.%f*.



Next, we select which attributes we want to include in our dataset. The BPIC2018 data consists of two types of attributes. One type about the application and process

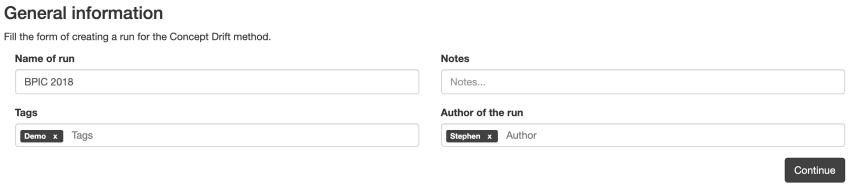
[2]https://www.win.tue.nl/bpi/doku.php?id=2018:challenge

itself, that is known when the events are executed. The other type consist of attributes that indicate the result of the entire process, we do not want these attributes to be used in our analysis. We thus only select the appropriate attributes, as shown below. We also remove the year attribute, as it is our goal to detect these by detecting the Concept Drift in the data.

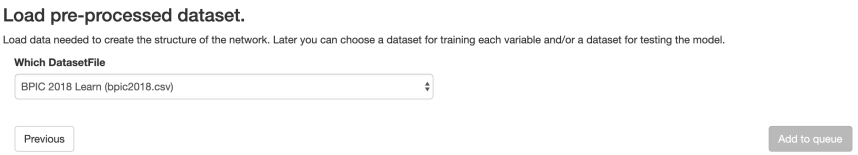
Next, we repeat these steps for the testing dataset. Which is identical to the training dataset, but uses all of the rows present in the log.

8.3.2 Learning the Model

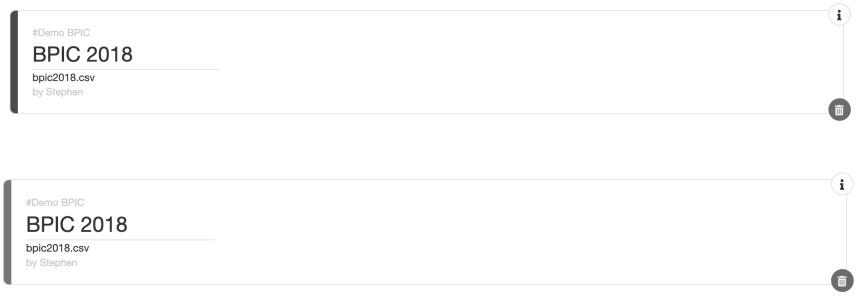
Next, we need to learn the model structure for the eDBN that will form the basis of our experiments. First, we give some information about the model.



On the next page we select the data we want to use to train the structure of our model. We select the Learn dataset we just uploaded.



The data and request are now sent to the server where the structure learning algorithm now starts. We can see in the *Experiments* tab the unfinished experiments for which the server is still computing the model structure.

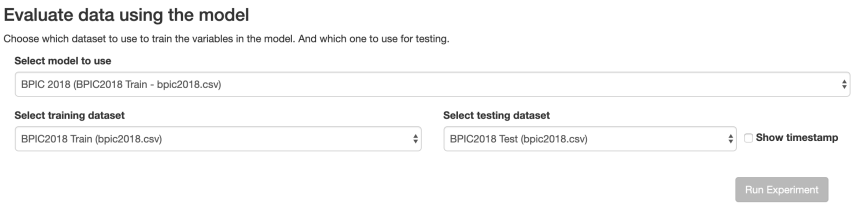


We can inspect the learned model by opening the experiment. We get the network as shown below, if needed we could manually add or remove dependencies.

If we are happy with the structure of the network we can go further to initiate an experiment.

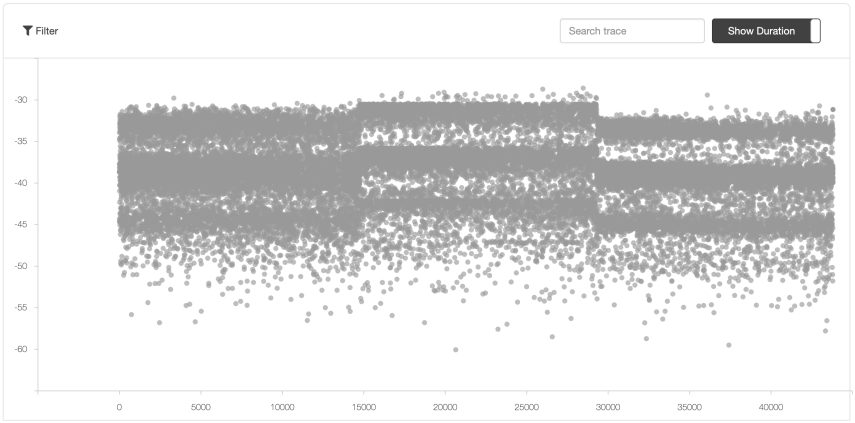
8.3.3 Performing an Experiment

We select the file we want to use for training the model parameters and a file for testing the trained model afterward. We choose the learning and testing version of the datasets we have uploaded. We also deselect the *Show Timestamp* checkbox, as it is our goal to find the differences between the years using the attributes that contain no direct timing information. When everything is entered correctly we can continue.

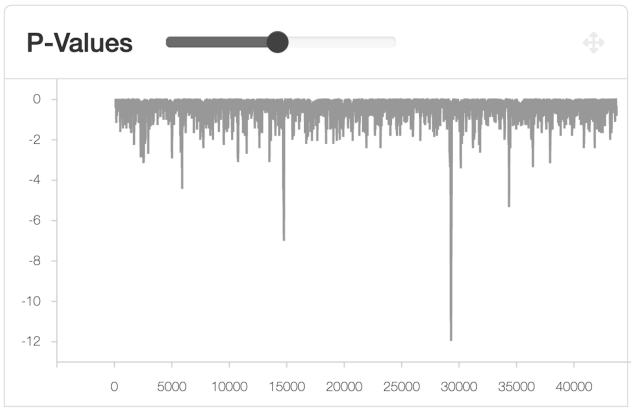


When the model has finished training and all events are scored we can start analyzing the results.

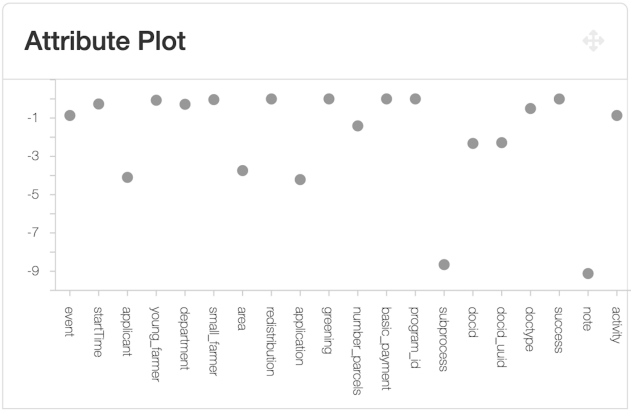
The first step in our analysis is to take a closer look at the trace graph. In this graph, we can look at possible clusters or outlying traces. Below we get a detailed view of the graph.



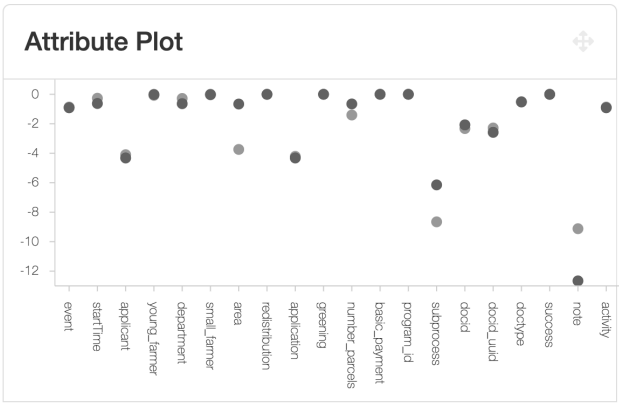
Visually we can already distinguish three regions with a clear difference in the distributions of the scores. When looking at the p-value graph we can confirm our suspicions.



Next, we can start investigating what attributes do cause these differences between the regions. To do so, we use the Attribute Plot. Below we can see the mean score for every attribute over the entire test dataset.



After applying a filter based on the found drift points we get the following graph:



Which shows results in the line with our initial findings presented at the BPI 2018 Challenge. All used datasets are available in the demo tool which can be found online at `http://adrem.uantwerpen.be/conceptdrift`.

8.4 Related Work

The ProM tool[3] contains different conformance checking and concept drift detection algorithms, often based on computing alignments. Almost all available algorithms only take the control-flow (and sometimes resource) perspective into account [38]. Our method overcomes this shortcoming by allowing any number of extra attributes to be used. The ProM tool, however powerful and extensive, is less suitable for non-experts in the Business Process domain as these methods often require lots of parameters to be set and a fair understanding of the algorithms used to get the best results out of it. ACD2 tries to hide away all these details for the user, making the tool more easy and intuitive to use.

[3] http://www.promtools.org

8.5 Conclusion

We showed our functional Anomaly and Concept Drift detection tool ACD2 that is based on work done for the BPI 2018 Challenge. The main goal for this tool is to allow for visual analysis of Business Process event logs. One of the ideas, therefore, is that people with only limited knowledge about Business Processes and the underlying algorithms should be able to interact with the tool and use it to detect and analyze Concept Drift or Anomalies in a given event log. This is one of the main reasons for making it a web-based application with easy wizard-like steps to perform.

Our method uses the extended Dynamic Bayesian Networks, making it an expressive and intuitive way of capturing the normal behavior found in event logs. The model can be represented in a visual way, allowing for user intervention after the model has been learned. This is in contrast to most other techniques that do not allow for direct user input.

We used the data from the BPI 2018 Challenge for an illustrative example. This data consists of more than 2 million events and 43,809 cases, indicating that our tool is able to handle larger datasets, thanks to our client-server architecture.

The tool is available at `http://adrem.uantwerpen.be/conceptdrift`.

CHAPTER 9

Conclusions and Outlook

9.1 Conclusions

In this thesis, we looked at how to use established machine learning approaches in the field of business process management, and more specifically process monitoring. We choose to investigate the usability of Dynamic Bayesian Networks for these tasks. DBNs showed to be useful in the context of business processes as they can correctly describe an event log by learning the relations present in the data. Unlike some of the other models, DBNs are also able to learn relations between different perspectives of an event log.

In **Chapter 3** we showed that existing DBNs can be used with only a small modification for next activity prediction. We put a constraint on what relations can be discovered to focus on predicting the next event. We only allowed relations that correspond with a causal relationship. Next, we added a way of dealing with unseen values and unknown combinations of values within the different relations. In our experiments, we showed that this model performs at par with existing state-of-the-art techniques. However, our proposed method can learn its model much faster, without the need for specialized (and expensive) hardware. Even though predicting the suffix worked well on some datasets, our model best performs for next activity prediction. More research is needed to improve the model to better predict the entire suffix of a running case.

We showed in **Chapter 4** that our proposed model is suitable for rapid learn-and-test cycles. This is useful in situations where we want to keep our model as up-to-date as possible. Existing techniques all use a static model and static way of testing the performance of their models. These models need to be retrained regularly to remain up-to-date with the actual process data. This has to be done at fixed times, potentially wasting valuable resources when no change occurs in the data. We

looked at techniques to easily improve existing methods so that they can perform better in a changing environment. Our proposed techniques show improvement in both accuracy and runtime for different existing next activity prediction algorithms based on neural networks. We introduced a simple neural network model that reaches the same fast learn-and-test cycles as our DBN model and trains orders of magnitude faster than other, existing neural network methods.

Predicting the next activity is sometimes being seen as one of the easiest tasks within business processes as every event log can be used to evaluate the method. The ease with which datasets can be selected and used for evaluating next activity prediction has led to a large number of new publications in recent years. All of which contain seemingly valid comparisons with existing methods. However, some aspects of the data and how it can be different when using different preprocessing steps, are seldom being spoken of in these papers. In **Chapter 5** we identified some of these data-related choices that everybody has to make when performing evaluations. After examining the state-of-the-art methods we concluded that no two comparisons are the same and that accuracy numbers are often copied between papers. It is imperative, for the future, that these issues, on how good evaluations should be performed, are solved.

More and more complex models are being proposed in the literature. We showed in our extensive evaluation, however, that our far less complex methods perform on par or sometimes even outperform these complex models. This raises the question of what the benefit of all these complex models, which sometimes only differ by a small detail, is for the field of next-activity predictions and predictions in general. All these methods increase the prediction accuracy by only a marginal amount but increase the complexity and runtime needed by a large amount, making them harder to run on standard hardware with only limited knowledge about the model itself. An important aspect of the eDBN method is the lack of hyperparameters, where typical Neural networks have a plethora of hyperparameters that can have a significant impact on the final result.

In **Chapter 6** we extended our model to be able to learn the normal behavior within an event log and use it for identifying anomalies. It is in this field that next activity prediction methods have significant added value. We added functional dependencies and a mechanism that can deal with new values in the data to our model. Existing methods identify unknown values as anomalies, but in some cases, these values are not necessarily anomalous. Our model takes into account the rate at which new values occur within the train data while testing. The experiments show that our method is capable of taking both categorical and numerical attributes into account and outperforms other methods while being faster to train. We performed a wide variety of experiments showing the effectiveness of our model on different types of data.

In **Chapter 7 and 8** we showed how our model can also be used for concept drift detection. Thanks to the underlying DBN model we can give a detailed description of why a certain drift has happened, or why certain events or cases are anomalous. These out-of-the-box explanations form one of the more important features of our entire model, as both users and regulators (start to) demand explanations from automated systems. Besides the use for concept drift detection, we can also use the same methods to explain predictions for the next activity. We can incorporate the concept drift detection with the techniques proposed in **Chapter 4**. This removes

the need for a separate concept drift detection method.

9.2 Outlook

After working on this thesis, we see some interesting avenues for future work. In **Chapter 6** we showed how we can extend the model introduced in **Chapter 3**. An interesting topic would be to see how this new, extended, model can be used to solve our initial problem of making predictions. It would be interesting to see how we can create a single unified model which only has to be trained once and can then be used for the different tasks. Furthermore, given the similarities between the three monitoring tasks described in this thesis, it is useful to look at the usability of other existing algorithms, which were designed for a single task, and look at how we can extend them to also be able to perform the other tasks. Having to train only a single monitoring model, which can be used for a plethora of tasks is very beneficial and can reduce the cost for hardware, computing time, and maintenance drastically.

Despite lots of recent publications on the topic of predicting in business processes, we show in **Chapter 5** that the performed evaluations are often flawed or incomplete. Ketyko et al. [49] and Weytjens and De Weerdt [119] confirm these observations and propose their solutions to the problem. There is a growing consensus within the field that a better, more robust, and uniform way of performing evaluations should be implemented and used. As a first step, comparison experiments must all use the same datasets and perform the same preprocessing on the data. We showed that different ways of preprocessing the data can lead to a difference in overall results.

When looking at the performances of state-of-the-art models, we see that most of them perform equally. It would be useful to take a closer look at when activities get predicted correctly and when they are predicted wrong. The question we should ask is: *do the different models predict the same events correctly or is there a significant variation?* Only by knowing what types of activities/cases we can already predict correctly, we can improve the existing methods in a meaningful way.

Our experiments show that our model, in its current form, is less suitable for predicting the entire suffix of a running case. A new two-step prediction approach, however, could improve the performance. As we showed that predicting only the next activity achieves high accuracy and precision results, we could first predict the next activity and, using this predicted activity, further predict the remaining attributes of the event. This could increase the quality of the predictions of the extra attributes and thus in their turn improve the quality for the next activity. Also, it is interesting to look deeper into how to use the model to predict the remaining duration of an event or a case.

Another interesting thing to look at is to use our predictive model for prescriptive monitoring, where the model returns the best action that ensures a positive outcome. We could do this by incorporating the outcome in our learned model, making the model learn to differentiate between cases with a positive and a negative outcome. This approach could both be applied to our DBN and to existing neural networks. However, the integration with the DBN model is more natural as this would simply add an attribute to the events that our model can use to better predict the activity. Some of the deep learning methods use a more complex way of preprocessing and

encoding the data that makes this addition much harder to implement, as it would change the entire embedding used in the network.

In **Chapter 7** we showed that we can use the decomposability of the anomaly score to detect and explain drifts between different years. We could generalize this usage by looking at how we can use the obtained scores for analyzing how a typical instance of our process is performing. As all attributes are handled equally, we would not be limited by only a certain interesting perspective but could use the perspective given by any attribute that is present in the event log.

As a final result, one could extend the functionality of our ACD2 tool, to showcase the power and ease-of-use of this one-model-fits-all approach that our model can deliver.

Bibliography

[1] Rafael Accorsi and Thomas Stocker. Discovering workflow changes with time-based trace clustering. In *International Symposium on Data-Driven Process Discovery and Analysis*, pages 154–168. Springer, 2011.

[2] Hirotugu Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

[3] Barbro Back, Teija Laitinen, and Kaisa Sere. Neural networks and genetic algorithms for bankruptcy predictions. *Expert systems with applications*, 11 (4):407–413, 1996.

[4] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Convolutional sequence modeling revisited. 2018.

[5] Saba Bashir, Usman Qamar, and M Younus Javed. An ensemble based decision support framework for intelligent heart disease diagnosis. In *International conference on information society (i-Society 2014)*, pages 259–264. IEEE, 2014.

[6] Jörg Becker, Dominic Breuker, Patrick Delfmann, and Martin Matzner. Designing and implementing a framework for event-based predictive modelling of business processes. *Enterprise modelling and information systems architectures-EMISA 2014*, 2014.

[7] Roel Bertens. *Insight Information: from Abstract to Anomaly*. Universiteit Utrecht, 2017.

[8] Roel Bertens, Jilles Vreeken, and Arno Siebes. Keeping it short and simple: Summarising complex event sequences with multivariate patterns. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 735–744. ACM, 2016.

[9] Alessandro Berti. Improving process mining prediction results in processes that change over time. *Data Analytics*, 2016:49, 2016.

[10] Alina Beygelzimer, John Langford, Yuri Lifshits, Gregory Sorkin, and Alex Strehl. Conditional probability tree estimation analysis and algorithms. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 51–58. AUAI Press, 2009.

[11] Fábio Bezerra, Jacques Wainer, and Wil MP van der Aalst. Anomaly detection using process mining. In *Enterprise, business-process and information systems modeling*, pages 149–161. Springer, 2009.

[12] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*, pages 443–448. SIAM, 2007.

[13] Kristof Böhmer and Stefanie Rinderle-Ma. Multi-perspective anomaly detection in business process execution events. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 80–98. Springer, 2016.

[14] RP Jagadeesh Chandra Bose, Wil MP van der Aalst, Indrė Žliobaitė, and Mykola Pechenizkiy. Handling concept drift in process mining. In *International Conference on Advanced Information Systems Engineering*, pages 391–405. Springer, 2011.

[15] RP Jagadeesh Chandra Bose, Wil MP Van Der Aalst, Indrė Žliobaitė, and Mykola Pechenizkiy. Dealing with concept drifts in process mining. *IEEE transactions on neural networks and learning systems*, 25(1):154–171, 2013.

[16] Dominic Breuker, Martin Matzner, Patrick Delfmann, and Jörg Becker. Comprehensible predictive models for business processes. *Mis Quarterly*, 40(4): 1009–1034, 2016.

[17] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Optics-of: Identifying local outliers. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 262–270. Springer, 1999.

[18] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[19] Suratna Budalakoti, Ashok N Srivastava, and Matthew Eric Otey. Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(1):101–113, 2008.

[20] Andrea Burattin and Josep Carmona. A framework for online conformance checking. In Ernest Teniente and Matthias Weidlich, editors, *Business Process Management Workshops*, pages 165–177. Springer International Publishing, 2018. ISBN 978-3-319-74030-0.

[21] Andrea Burattin, Marta Cimitile, Fabrizio M. Maggi, and Alessandro Sperduti. Online discovery of declarative process models from event streams. *IEEE Transactions on Services Computing*, 8(6):833–846, 2015. doi: 10.1109/TSC.2015.2459703.

[22] Theophilos Cacoullos. Estimation of a multivariate density. *Annals of the Institute of Statistical Mathematics*, 18(1):179–189, 1966.

[23] Manuel Camargo, Marlon Dumas, and Oscar González-Rojas. Learning accurate lstm models of business processes. In *International Conference on Business Process Management*, pages 286–302. Springer, 2019.

[24] Josep Carmona and Ricard Gavalda. Online techniques for dealing with concept drift in process mining. In *International Symposium on Intelligent Data Analysis*, pages 90–102. Springer, 2012.

[25] Ismail Ilkan Ceylan, Adnan Darwiche, and Guy Van den Broeck. Open-world probabilistic databases. In *Description Logics*, 2016.

[26] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection for discrete sequences: A survey. *IEEE TKDE*, 24(5):823–839, 2012.

[27] Li-Chiu Chi and Tseng-Chung Tang. Bankruptcy prediction: Application of logit analysis in export credit risks. *Australian Journal of Management*, 31(1): 17–27, 2006.

[28] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[29] Fred J Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[30] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[31] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, pages 2462–2467. Hyderabad, 2007.

[32] Li Deng. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, 3, 2014.

[33] Chiara Di Francescomarino, Chiara Ghidini, Fabrizio Maria Maggi, and Fredrik Milani. Predictive process monitoring methods: Which one suits me best? In *International Conference on Business Process Management*, pages 462–479. Springer, 2018.

[34] Chiara Di Francescomarino, Chiara Ghidini, Fabrizio Maria Maggi, Williams Rizzi, and Cosimo Damiano Persia. Incremental predictive process monitoring: How to deal with the variability of real environments. *arXiv preprint arXiv:1804.03967*, 2018.

[35] Nicola Di Mauro, Annalisa Appice, and Teresa MA Basile. Activity prediction of business process instances with inception cnn models. In *International Conference of the Italian Association for Artificial Intelligence*, pages 348–361. Springer, 2019.

[36] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2000.

[37] Anton Dries and Ulrich Rückert. Adaptive concept drift detection. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2(5-6):311–327, 2009.

[38] Sebastian Dunzer, Matthias Stierle, Martin Matzner, and Stephan Baier. Conformance checking: a state-of-the-art literature review. In *Proceedings of the 11th International Conference on Subject-Oriented Business Process Management*, page 4. ACM, 2019.

[39] Eleazar Eskin, Wenke Lee, and Salvatore J Stolfo. Modeling system calls for intrusion detection with dynamic window sizes. In *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01*, volume 1, pages 165–175. IEEE, 2001.

[40] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. Predicting process behaviour using deep learning. *Decision Support Systems*, 100:129–140, 2017.

[41] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014.

[42] Alexander Gepperth and Barbara Hammer. Incremental learning algorithms and applications. In *European symposium on artificial neural networks (ESANN)*, 2016.

[43] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[44] Michael Hammer. Reengineering work: don't automate, obliterate. *Harvard business review*, 68(4):104–112, 1990.

[45] Markku Hinkka, Teemu Lehto, and Keijo Heljanko. Exploiting event log event attributes in rnn based prediction. In *European Conference on Advances in Databases and Information Systems*, pages 405–416. Springer, 2019.

[46] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of computer security*, 6(3):151–180, 1998.

[47] MA Jabbar, BL Deekshatulu, and Priti Chndra. Alternating decision trees for early diagnosis of heart disease. In *International Conference on Circuits, Communication, Control and Computing*, pages 322–328. IEEE, 2014.

[48] Daniel A Keim, Florian Mansmann, Jörn Schneidewind, and Hartmut Ziegler. Challenges in visual data analysis. In *Information Visualization, 2006. IV 2006. Tenth International Conference on*, pages 9–16. IEEE, 2006.

[49] István Ketykó, Felix Mannhardt, Marwan Hassani, and Boudewijn van Dongen. What averages do not tell–predicting real life processes with sequential deep learning. *arXiv preprint arXiv:2110.10225*, 2021.

[50] Ralf Klinkenberg and Thorsten Joachims. Detecting concept drift with support vector machines. In *ICML*, pages 487–494, 2000.

[51] Wolfgang Kratsch, Jonas Manderscheid, Maximilian Röglinger, Johannes Seyfried, et al. Machine learning in business process monitoring: A comparison of deep learning and classical approaches used for outcome prediction. *Business & Information Systems Engineering: The International Journal of Wirtschaftsinformatik*, pages 1–16, 2020.

[52] Hans-Peter Kriegel, Arthur Zimek, et al. Angle-based outlier detection in high-dimensional data. In *Procs of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 444–452. ACM, 2008.

[53] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Outlier detection in axis-parallel subspaces of high dimensional data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 831–838. Springer, 2009.

[54] Hans-Peter Kriegel, Peer Kroger, Erich Schubert, and Arthur Zimek. Outlier detection in arbitrarily oriented subspaces. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 379–388. IEEE, 2012.

[55] Geetika T Lakshmanan, Davood Shamsi, Yurdaer N Doganata, Merve Unuvar, and Rania Khalaf. A markov prediction model for data-driven semi-structured business processes. *Knowledge and Information Systems*, 42(1):97–126, 2015.

[56] Yan-Nei Law and Carlo Zaniolo. An adaptive nearest neighbor classification algorithm for data streams. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 108–120. Springer, 2005.

[57] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Procs of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166. ACM, 2005.

[58] Tian-Shyug Lee, Chih-Chou Chiu, Yu-Chao Chou, and Chi-Jie Lu. Mining the customer credit using classification and regression tree and multivariate adaptive regression splines. *Computational Statistics & Data Analysis*, 50(4): 1113–1130, 2006.

[59] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766, 2013.

[60] Hui Li and Jie Sun. Majority voting combination of multiple case-based reasoning for financial distress prediction. *Expert Systems with Applications*, 36(3):4363–4373, 2009.

[61] Michael K Lim and So Young Sohn. Cluster-based dynamic scoring model. *Expert Systems with Applications*, 32(2):427–431, 2007.

[62] Li Lin, Lijie Wen, and Jianmin Wang. Mm-pred: a deep predictive model for multi-attribute event sequence. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 118–126. SIAM, 2019.

[63] Wei-Yang Lin, Ya-Han Hu, and Chih-Fong Tsai. Machine learning in financial crisis prediction: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):421–436, 2011.

[64] Jianxiong Luo and Susan M Bridges. Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection. *International Journal of Intelligent Systems*, 15(8):687–703, 2000.

[65] Abderrahmane Maaradji, Marlon Dumas, Marcello La Rosa, and Alireza Ostovar. Fast and accurate business process drift detection. In *International Conference on Business Process Management*, pages 406–422. Springer, 2015.

[66] Marco Maisenbacher and Matthias Weidlich. Handling concept drift in predictive process monitoring. *SCC*, 17:1–8, 2017.

[67] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[68] K McCormack. The development of a measure of business process orientation and its relationship to organizational performance. *Online tutorial available at http://www. prosci. com/mccormack. htm*, 514, 1999.

[69] Houda Mezrigui, Foued Theljani, and Kaouther Laabidi. Decision support system for medical diagnosis using a kernel-based approach. In *2017 International Conference on Control, Automation and Diagnosis (ICCAD)*, pages 303–308. IEEE, 2017.

[70] Jae H Min and Young-Chan Lee. Bankruptcy prediction using support vector machine with optimal choice of kernel function parameters. *Expert systems with applications*, 28(4):603–614, 2005.

[71] Business Process Model. Notation (bpmn) version 2.0. *OMG Specification, Object Management Group*, pages 22–31, 2011.

[72] Rana Momtaz, Nesma Mohssen, and Mohammad A Gowayyed. Dwof: A robust density-based outlier detection approach. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 517–525. Springer, 2013.

[73] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. Binet: Multivariate business process anomaly detection using deep learning. In *International Conference on Business Process Management*, pages 271–287. Springer, 2018.

[74] Timo Nolle, Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser. Binet: Multi-perspective business process anomaly classification. *Information Systems*, page 101458, 2019.

[75] Alireza Ostovar, Abderrahmane Maaradji, Marcello La Rosa, Arthur H. M. ter Hofstede, and Boudewijn F. V. van Dongen. Detecting drift from event streams of unpredictable business processes. In Isabelle Comyn-Wattiau, Katsumi Tanaka, Il-Yeol Song, Shuichiro Yamamoto, and Motoshi Saeki, editors, *Conceptual Modeling*, pages 330–346. Springer International Publishing, 2016.

[76] Kanika Pahwa and Ravinder Kumar. Prediction of heart disease using hybrid technique for selecting features. In *2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON)*, pages 500–504. IEEE, 2017.

[77] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B Gibbons, and Christos Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 315–326. IEEE, 2003.

[78] Vincenzo Pasquadibisceglie, Annalisa Appice, Giovanna Castellano, and Donato Malerba. Using convolutional neural networks for predictive process analytics. In *2019 International Conference on Process Mining (ICPM)*, pages 129–136. IEEE, 2019.

[79] Jan Peter Patist. Optimal window change detection. In *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*, pages 557–562. IEEE, 2007.

[80] Stephen Pauwels and Toon Calders. Mining multi-dimensional complex log data. *Benelearn*, 2016.

[81] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[82] Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano de Leoni. Time and activity sequence prediction of business process instances. *Computing*, 100(9):1005–1031, 2018.

[83] William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[84] Yan Qiao, XW Xin, Yang Bin, and S Ge. Anomaly intrusion detection method based on hmm. *Electronics letters*, 38(13):663–664, 2002.

[85] VV Ramalingam, Ayantan Dandapath, and M Karthik Raja. Heart disease prediction using machine learning techniques: a survey. *International Journal of Engineering & Technology*, 7(2.8):684–687, 2018.

[86] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 427–438, 2000.

[87] Andreas Rogge-Solti and Gjergji Kasneci. Temporal anomaly detection in business processes. In *International Conference on Business Process Management*, pages 234–249. Springer, 2014.

[88] Lior Rokach and Oded Maimon. Clustering methods. In *Data mining and knowledge discovery handbook*, pages 321–352. Springer, 2005.

[89] Stuart Jonathan Russell and Peter Norvig. Artificial intelligence: a modern approach (3rd edition), 2009.

[90] Ridhi Saini, Namita Bindal, and Puneet Bansal. Classification of heart diseases from ecg signals using wavelet transform and knn classifier. In *International Conference on Computing, Communication & Automation*, pages 1208–1215. IEEE, 2015.

[91] Jeffrey C Schlimmer and Richard H Granger. Incremental learning from noisy data. *Machine learning*, 1(3):317–354, 1986.

[92] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[93] Erich Schubert, Alexander Koos, Tobias Emrich, Andreas Züfle, Klaus Arthur Schmid, and Arthur Zimek. A framework for clustering uncertain data. *PVLDB*, 8(12):1976–1979, 2015.

[94] Joan Serrà Julià, Dídac Surís, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *Dy J, Krause A, editors. Proceedings of the 35th International Conference on Machine Learning (ICML 2018); 2018 Jul 10-15; Stockholmsmässan, Sweden.[Massachusetts: PMLR; 2018]. p. 4548-57*. Proceedings of Machine Learning Research, 2018.

[95] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, January 2001. ISSN 1559-1662.

[96] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[97] Minseok Song, Christian W Günther, and Wil MP Van der Aalst. Trace clustering in process mining. In *International Conference on Business Process Management*, pages 109–120. Springer, 2008.

[98] Lili Sun and Prakash P Shenoy. Using bayesian networks for bankruptcy prediction: Some methodological issues. *European Journal of Operational Research*, 180(2):738–753, 2007.

[99] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[100] Bayu Adhi Tama, Marco Comuzzi, and Jonghyeon Ko. An empirical investigation of different classifiers, encoding, and ensemble schemes for next event prediction using business process event logs. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(6):1–34, 2020.

[101] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. Predictive business process monitoring with lstm neural networks. In *International Conference on Advanced Information Systems Engineering*, pages 477–492. Springer, 2017.

[102] Niek Tax, Irene Teinemaa, and Sebastiaan J van Zelst. An interdisciplinary comparison of sequence modeling methods for next-element prediction. *Software and Systems Modeling*, 19(6):1345–1365, 2020.

[103] Farbod Taymouri, Marcello La Rosa, Sarah Erfani, Zahra Dasht Bozorgi, and Ilya Verenich. Predictive business process monitoring via generative adversarial nets: The case of next event prediction. In *International Conference on Business Process Management*, pages 237–256. Springer, 2020.

[104] Irene Teinemaa, Marlon Dumas, Marcello La Rosa, and Fabrizio Maria Maggi. Outcome-oriented predictive process monitoring: Review and benchmark. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13(2):1–57, 2019.

[105] Julian Theis and Houshang Darabi. Decay replay mining to predict next process events. *IEEE Access*, 7:119787–119803, 2019.

[106] Wil Van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE transactions on knowledge and data engineering*, 16(9):1128–1142, 2004.

[107] Wil Van Der Aalst, Arya Adriansyah, Ana Karla Alves De Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter Van Den Brand, Ronald Brandtjen, Joos Buijs, et al. Process mining manifesto. In *International conference on business process management*, pages 169–194. Springer, 2011.

[108] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, Incorporated, 1st edition, 2011. ISBN 3642193447, 9783642193446.

[109] Wil MP Van der Aalst and Ana Karla A de Medeiros. Process mining and security: Detecting anomalous process executions and checking process conformance. *Electronic Notes in Theoretical Computer Science*, 121:3–21, 2005.

[110] B. van Dongen and F. Borchert. (2018) bpi challenge 2018. eindhoven university of technology., 2018.

[111] B.F. van Dongen. (2012) bpi challenge 2012. eindhoven university of technology., 2012.

[112] B.F. van Dongen. (2015) bpi challenge 2015. eindhoven university of technology., 2015.

[113] Boudewijn van Dongen. Real-life event logs - hospital log, Mar 2011.

[114] Boudewijn F Van Dongen, Ana Karla A de Medeiros, HMW Verbeek, AJMM Weijters, and Wil MP Van Der Aalst. The prom framework: A new era in process mining tool support. In *International Conference on Application and Theory of Petri Nets*, pages 444–454. Springer, 2005.

[115] Ilya Verenich. Helpdesk, mendeley data, v1, 2016.

[116] Mark Von Rosing, Henrik Von Scheel, and August-Wilhelm Scheer. *The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM*, volume 1. Morgan Kaufmann, 2014.

[117] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE symposium on security and privacy (Cat. No. 99CB36344)*, pages 133–145. IEEE, 1999.

[118] S Weinzierl, S Zilker, J Brunk, K Revoredo, A Nguyen, M Matzner, J Becker, and B Eskofier. An empirical comparison of deep-neural-network architectures for next activity prediction using context-enriched process event logs. *arXiv preprint arXiv:2005.01194*, 2020.

[119] Hans Weytjens and Jochen De Weerdt. Creating unbiased public benchmark datasets with data leakage prevention for predictive process monitoring. *arXiv preprint arXiv:2107.01905*, 2021.

[120] JV White, Sam Steingold, and CG Fournelle. Performance metrics for group-detection algorithms. *Proceedings of Interface 2004*, 2004.

[121] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

[122] Jun Wu, Dayong Ding, Xian-Sheng Hua, and Bo Zhang. Tracking concept drifting with an online-optimized incremental learning framework. In *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pages 33–40, 2005.

[123] Wei-Wen Wu. Beyond business failure prediction. *Expert systems with applications*, 37(3):2371–2376, 2010.

[124] Kenji Yamanishi and Yuko Maruyama. Dynamic syslog mining for network failure monitoring. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 499–508, 2005.

[125] Hong Yao and Howard J Hamilton. Mining functional dependencies from data. *Data Mining and Knowledge Discovery*, 16(2):197–219, 2008.

[126] Nong Ye et al. A markov chain model of temporal behavior for anomaly detection. In *Procs of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, volume 166, page 169. West Point, NY, 2000.

[127] I-Cheng Yeh and Che-hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2):2473–2480, 2009.

[128] Dit-Yan Yeung and Calvin Chow. Parzen-window network intrusion detectors. In *Object recognition supported by user interaction for service robots*, volume 4, pages 385–388. IEEE, 2002.

[129] Paul D Yoo, Maria H Kim, and Tony Jan. Machine learning techniques and use of event information for stock market prediction: A survey and evaluation. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, volume 2, pages 835–841. IEEE, 2005.

[130] Arthur Zimek. Correlation clustering. *ACM SIGKDD Explorations Newsletter*, 11(1):53–54, 2009.

[131] Indrė Žliobaitė. Learning under concept drift: an overview. *arXiv preprint arXiv:1010.4784*, 2010.

Summary

In our highly automated world, where a lot of manufacturing processes are often performed by autonomous robots, the need for describing and checking these running processes is high. The field of Business Process Management tries to identify, formalize, analyze and improve the different processes that are present in an organization. Within this broad field, we have a subfield that takes event logs as input and learns the underlying processes. Next we can visualize these processes using diagrams that are easily readable by humans.

Modern processes are, however, getting more complicated and often require the collaboration of multiple systems (both fully automated and fully manual). Monitoring running processes manually or semi-automated becomes harder due to this added complexity. To aid the monitoring of running processes, we propose to use machine learning techniques that can learn models based on the data. Next, this model is used to validate new, unseen instances of the processes and can indicate if this process is behaving as expected or if the execution is anomalous and human intervention is needed to avoid the loss of a product or a customer that is not satisfied.

This thesis starts by introducing the use of Dynamic Bayesian Networks for predicting the next activity in an event log. We then show that starting from this model, we can expand it further to be able to detect anomalies or deviations from the original process. The big advantage of our Bayesian Network-based approach is its explainability. As the impact of a predicted anomaly might be high, people first need proof and persuasion before they will take action to correct the running process.

Contributions

- In **Chapter 3** we introduce the use of Dynamic Bayesian Networks for predicting next event(s) in Business Processes. Thanks to the conditional dependencies used by these networks we can capture the most important dependencies between the different attributes present in the event log. Thanks to these dependencies we can make accurate predictions of what the activity of the next event is going to be.
- One aspect of business processes that is often overlooked in the research of predicting next events is that these processes change over time. In **Chapter 4** we look at different strategies that can be used to update the models. We look at how to update our Bayesian-based model and also at how we can change

existing neural network methods to be able to update themselves when new data arrives. To have a lightweight and easy-to-use network for testing we created the Single-Dense Layer neural network for next activity prediction. We showed that this simple network performs at par with current state-of-the-art methods. Giving rise to the question of whether more complex is better for these types of prediction tasks.

- In **Chapter 5** we take a critical look at how authors are evaluating their next activity prediction algorithms. We show that one of the biggest problems is how different methods are compared with each other. A lot of authors copy/paste results from other papers to compare with their new methods. The problem is that often these datasets are preprocessed differently or are even different versions of the same dataset. Thus making the comparison with the previously reported numbers invalid. We investigated a number of choices that have to be made when using datasets and looked at their impact on the accuracy and precision results. We show that the differences should not be ignored and a new paradigm for evaluating prediction tasks is needed.

- We show in **Chapter 6** how to extend our DBN model with an extra type of relations (the Functional dependency) and make it describe a probability distribution that can indicate how anomalous a certain event is. Based on the score given by the extended model, the events get sorted. This makes it possible for users to prioritize events that are highly likely to contain an anomaly as it is impossible to manually fix hundreds of possible (small) anomalies.

- The anomaly score that was introduced in the previous chapter is fully decomposable into the various factors for all attributes present in the event log. In **Chapter 7** we use this aspect to first detect and then explain drifts that happen in the event log of the BPI Challenge 2018. Using our scores we can generate visual aids that help the analyst understand the processes and understand why and where the differences between cases are.

- In **Chapter 8** we introduce the tool we implemented that incorporates the Anomaly and Concept drift detection (ACD2). Using an easy-to-use web interface a user can upload its event log. The model learns the different dependencies based on this data. After completion, the user can upload a separate test file to analyze. The tool enables the user to investigate different parts of the data. The event log can also be inspected on a trace and event level. Where anomalous traces are clearly visible and even the most anomalous event are identified.

Samenvatting

In onze geautomatiseerde wereld, waarbij veel productieprocessen automatisch en vaak door robots worden uitgevoerd, is de nood naar het beschrijven en controleren van deze lopende processen hoog. Het onderzoek naar deze bedrijfsprocessen probeert de verschillende processen die aanwezig zijn in een organisatie te identificeren, te formalizeren, te analyseren en te verbeteren. Binnen dit brede veld hebben we het deelveld dat event logs als input neemt en het onderliggende process hieruit leert. Vervolgens kunnen we deze processen visualiseren door middel van diagrammen die eenvoudig begrijpbaar zijn voor gebruikers.

Moderne processes worden echter steeds complexer en vereisen vaak samenwerking tussen verschillende systemen (zowel volledig automatische als manuele systemen). Het manueel of half-automatisch monitoren van lopende processen wordt dan ook steeds moeilijker door deze toegenomen complexiteit. Om het monitoren van lopende processen te ondersteunen, stellen we in deze thesis technieken voor die gebruik maken van machinaal leren om modellen te leren op basis van data. Vervolgens kunnen we deze modellen gebruiken om nieuwe, onbekende data van processen te gaan valideren. De modellen kunnen ook gebruikt worden om aan te duiden of een bepaald process zich gedraagt zoals verwacht of dat we een abnormale uitvoering hebben en we mogelijk menselijke tussenkomst nodig hebben om ervoor te zorgen dat het product niet verloren gaat of de klant niet ontevreden wordt.

Deze thesis begint met het introduceren van Dynamische Bayesiaanse Netwerken om de volgende activiteit in een event log te voorspellen. Vervolgens laten we zien dat, startende van dit model, we dit verder kunnen uitbreiden zodanig dat we ook anomalieën of abnormaal gedrag kunnen detecteren. Het grote voordeel van deze Bayesiaanse Netwerken gebaseerde aanpak is de interpreteerbaarheid van het model. De impact door een bepaalde voorspelling kan groot zijn, en dus hebben mensen vaak eerst een geldige uitleg nodig voordat ze zullen overgaan tot actie om het proces te corrigeren.

Contributies

- In **Hoofdstuk 3** introduceren we het gebruik van Dynamische Bayesiaanse Netwerken om het volgende event in bedrijfsprocessen te voorspellen. Dankzij de conditionele afhankelijkheden die gebruikt worden door deze netwerken kunnen we verschillende relaties tussen attributen binnen een event log voor-

stellen. Deze afhankelijkheden zorgen ervoor dat we accurate voorspelllingen kunnen maken over wat de activiteit van het volgende event zal zijn.

- Een aspect van bedrijfsprocessen dat vaak over het hoofd gezien wordt bij het voorspellen van het volgende event, is hoe de processen veranderen na verloop van tijd. In **Hoofdstuk 4** bekijken we verschillende strategieën die gebruikt kunnen worden om modellen te updaten. We kijken naar hoe we ons Bayesiaanse model en andere bestaande modellen, gebasseerd op neurale netwerken, kunnen updaten zodat ze zichzelf kunnen aanpassen wanneer nieuwe data beschikbaar is. Om een eenvoudig en makkelijk te gebruiken netwerk te hebben om te testen, stellen we het Single-Dense Layer neuraal netwerk voor om de volgende activiteit te voorspellen. We tonen aan dat dit eenvoudige netwerk even goed presteert als huidige state-of-the-art methoden, wat de vraag doet rijzen of meer complexe modellen wel betere zijn voor dit soort van voorspellende taken.

- In **Hoofstuk 5** kijken we met een kritische blik naar hoe auteurs hun voorspellende algoritmen evalueren. We tonen dat de manier waarop verschillende methoden met elkaar vergeleken worden één van de grootste problemen is. Veel auteurs kopiëren en plakken de resultaten uit andere papers om deze vervolgens te vergelijken met hun nieuwe methode. Het probleem hierbij is dat auteurs deze datasets vaak anders voorbereiden of soms ook andere versies gebruiken van dezelfde dataset. Dit zorgt ervoor dat het vergelijken van gerapporteerde resultaten niet mogelijk is. We onderzoeken een aantal keuzes die gemaakt moeten worden als we datasets gebruiken en kijken naar hun impact op de accuraatheid en precisie van de resultaten. We tonen aan dat deze verschillen niet genegeerd mogen worden en dat een nieuw paradigma voor het evalueren van voorspellende taken nodig is.

- We tonen in **Hoofdstuk 6** hoe we ons DBN model kunnen uitbreiden met een extra type relatie (de Functionele Afhankelijkheid) en zorgen ervoor dat ons model een kansverdeling beschrijft die gebruikt kan worden om aan te duiden hoe onwaarschijnlijk een bepaald evenement is. Gebaseerd op de score gegeven door dit uitgebreide model kunnen we de cases sorteren volgens hoe afwijkend ze zijn. Op deze manier is het mogelijk dat gebruikers een hogere prioriteit geven aan het oplossen en bekijken van evenementen die een grotere waarschijnlijkheid hebben om een anomalie te zijn aangezien het onmogelijk is om manueel honderden (kleine) mogelijke fouten te bekijken.

- De anomalie score die we in het vorige hoofdstuk hebben geïntroduceerd kunnen we volledig uit elkaar halen in de verschillende factoren voor alle attributen die aanwezig zijn in de event log. In **Hoofdstuk 7** gebruiken we dit aspect om eerst drift in de log file van de BPI Challenge 2018 te detecteren en deze vervolgens uit te leggen. Door het gebruik van onze scores kunnen we een visuele hulp maken die analysten kan helpen om processen te begrijpen, en te verstaan waarom en waar bepaalde verschillen aanwezig zijn tussen uitvoeringen.

- In **Hoofdstuk 8** introduceren we een tool die we hebben geïmplementeerd die zowel Anomalie en Concept Drift Detectie (ACD2) bevat. Via een eenvoudig te

gebruiken web interface kan een gebruiker zijn event log inladen. Het model leert vervolgens de verschillende afhankelijkheden gebsseerd op de gegeven data. Na voltooiing kan de gebruiker nieuwe data inladen om te analyseren. De tool maakt het mogelijk voor de gebruiker om verschillende delen van de data apart te onderzoeken. De event log kan ook geïnspecteerd worden op een case- en eventniveau, waarbij abnormale uitvoeringen duidelijk zichtbaar zijn en het meest abnormale event binnen een case aangeduid is.