Outlier classification of traces in terms of business process models

Avdis, D.

Award date: 2017

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
Outlier Classification of traces

In terms of

Business Process Models

Master Thesis

Dimitrios Avdis

Main Supervisor:

dr. Dirk Fahland (Eindhoven University of Technology)

Examination committee:

dr. Dirk Fahland

dr. Anna Wilbik

dr.ir.Joaquin Vanschoren

Eindhoven, July 2017
Abstract

This master thesis focuses on the solution of two main goals. First to conduct a literature study about how the branch of machine learning (neural networks) can successfully deal with some of the documented process mining challenges as they are presented in the process mining manifesto (Van Der Aalst, 2011). This literature study is performed based on knowledge I have gained by studying the literature related with what type of problems neural networks can successfully deal with and by applying this knowledge in the presented challenges of process mining. This is done by classifying each of the process mining challenges into what type of problem it is considered to be and then by presenting how neural networks can solve it. The second goal is to solve in practice the process mining challenge of outlier classification in terms of a process model that is discovered from a real life event log. For this reason the keras\(^1\) API (backed up by the python library Theano\(^2\)) is used in order to initially create and then train NN models. These models are trained using labeled datasets of outlier traces that are derived from the initial event log. Once trained, they are able to identify outlier traces in previously unseen instances.

\(^{1}\) https://keras.io/
\(^{2}\) http://deeplearning.net/software/theano/
Preface

This report concludes the results of my graduation project, regarding the “Business Information Systems” M.Sc. program at the department of “Computer Science & Mathematics” in Eindhoven University of Technology. In completing this graduate project, I have been fortunate to receive help, support and encouragement from many people and I would like to acknowledge them for their cooperation.
First of all, I would like to thank my graduation supervisor dr. Dirk Fahland for guiding me through, not only on this project, but during my whole studies in the Netherlands. The collaboration during the lectures and my master project has been a pleasant and a great learning experience for me.
In addition, I would like to thank all my friends and classmates for the help and the positive energy I received during my first years in the Netherlands. Being far away from home is difficult, but you made me overcome all the difficulties.
In conclusion, I would like to thank my parents and two brothers for their unconditional love and support, for all of those 28 years of my life. All the care they have provided me over the years is the greatest gift that anyone has ever given me. Last but not least, I would like to say thanks to all the supporting stuff of the Tue library, since this is the place that I spent most of my student life in Eindhoven.

Dimitris Avdis,
Eindhoven,
July 2017
# Contents

Abstract ........................................................................................................................................ iii
Preface .......................................................................................................................................... v
Contents ......................................................................................................................................... vii
List of Figures ................................................................................................................................ xi
List of Tables .................................................................................................................................. xiii
Abbreviations .............................................................................................................................. xv

## Chapter 1 Introduction .................................................................................................................. 1

1.1. Current state of the art .............................................................................................................. 1
1.2. Problem statement .................................................................................................................... 2
1.3. Methodology .......................................................................................................................... 2
1.4. Thesis Contribution .................................................................................................................. 5
1.5. Thesis Outline .......................................................................................................................... 5

## Chapter 2 Background Information ............................................................................................. 7

2.1. Process Mining (PM) .............................................................................................................. 7
  2.1.1. Definition .......................................................................................................................... 7
  2.1.2. Significance ...................................................................................................................... 8
  2.1.3. Challenges ...................................................................................................................... 9
  2.1.4. Business Process Model .................................................................................................. 9
  2.1.5. Event Log ........................................................................................................................ 10
2.2. Artificial Neural Networks (ANNs) ....................................................................................... 10
  2.2.1. Definition ...................................................................................................................... 10
2.3. Keras ...................................................................................................................................... 14
  2.3.1. Definition ...................................................................................................................... 14
Chapter 3  Deep learning in Process Mining.......................................................................................... 16

3.1.  Challenge 1: Finding, Merging, and Cleaning Event Data.......................................................... 16

3.1.1.  Merging distributed data ............................................................................................................ 16

3.1.2.  Turning event data from “object centric” to “process centric”.................................................. 17

3.1.3.  Dealing with incomplete event data ............................................................................................ 18

3.1.4.  Outlier classification in terms of a process model ..................................................................... 20

3.1.5.  Dealing with event logs containing events at different levels of granularity ............................ 22

3.1.6.  Merging of event data with contextual data .............................................................................. 23

3.2.  Challenge 2: Dealing with Complex Event Logs Having Diverse Characteristics ....................... 23

3.3.  Challenge 3: Creating representative benchmarks ......................................................................... 26

3.4.  Challenge 4: Dealing with concept drift ....................................................................................... 27

3.5.  Challenge 5: Improving the representational bias used for process discovery .............................. 29

3.6.  Challenge 6: Balancing between quality criteria such as fitness, simplicity, precision and Generalization 32

3.7.  Challenge 7: Cross-organizational mining ..................................................................................... 35

3.8.  Challenge 8: Providing operational support .................................................................................. 36

3.9.  Challenge 9: Combining process mining with other types of analysis .......................................... 41

Chapter 4  Outlier Classification of traces discovered from a Process Model’s event log ......................... 43

4.1.  Given event log................................................................................................................................. 44

4.2.  Log encoding into tabular data and labeling of traces .................................................................... 44

4.3.  Construction of deep feed forward NN .......................................................................................... 45

......................................................................................................................................................... 46

4.4.  Training of the constructed NN based on the created dataset ....................................................... 46

4.5.  Testing of the classification accuracy of the trained NN ............................................................... 46

4.6.  Compare initial classification with the one the NN performed ...................................................... 47

Chapter 5  Experiment Setup of “Sepsis case” ...................................................................................... 49

5.1.  Description of the used event log................................................................................................. 49

5.2.  Mapping model activities into binary convertible variables ............................................................ 51

5.3.  Neural network construction and training (Using manual verification dataset) ............................ 53

5.4.  Neural network construction and training (Using 10-fold cross validation dataset) ....................... 60

5.5.  Experiment Setup of “Road traffic management system case” ..................................................... 63

Chapter 6  Conclusions .......................................................................................................................... 73
6.1. Academic................................................................................................................................. 73
6.2. Limitations.................................................................................................................................. 74
6.3. Future work.................................................................................................................................. 74

Bibliography......................................................................................................................................... 75

Appendices ........................................................................................................................................ 77
Appendix A.......................................................................................................................................... 77
Appendix B.......................................................................................................................................... 81
Appendix C.......................................................................................................................................... 83
Appendix D.......................................................................................................................................... 85
Appendix E.......................................................................................................................................... 91
Appendix F.......................................................................................................................................... 96
Appendix G.......................................................................................................................................... 98
Appendix H.......................................................................................................................................... 103
List of Figures

Figure 1-1 The literature review process (Alexander, 2012) .................................................................3
Figure 1-2 CRISP-DM methodology (Hipp, 2000) ..................................................................................4
Figure 2-1 Essential components of a neuron (Sinha, 2010) .................................................................11
Figure 2-2 Simple artificial neuron (Kawaguchi, 2000) ........................................................................11
Figure 2-3 Example of a simple neural network (Gurney, 2007) ..........................................................12
Figure 2-4 Explanation of bias (Varma, 2016) ......................................................................................14
Figure 3-1 NN architectures with single task layers (a), with shared multitask layer (b) and with n+m layers of which n are shared (c) (Tax, 2016). ........................................................................19
Figure 3-2 Dealing with incomplete event data using NNs ....................................................................20
Figure 3-3 Dealing with complex event logs with NNs ..........................................................................25
Figure 3-4 Dealing with concept drift with the use of NNs ..................................................................29
Figure 3-5 Representational bias improvement with the use of NNs .....................................................31
Figure 3-6 Balancing between process modeling quality criteria using NNs ........................................34
Figure 3-7 Providing operational support for detection tasks using NNs .............................................38
Figure 3-8 Providing operational support for prediction tasks using NNs ............................................39
Figure 3-9 Providing operational support for recommendation tasks using NNs .................................40
Figure 4-1 Process for performing outlier classification ..........................................................................46
Figure 4-2 Outlier classification of process model traces using NNs ....................................................48
Figure 5-1 Process model discovered from the given event log with inductive visual miner ................50
Figure 5-2 Prom output format .............................................................................................................51
Figure 5-3 Sorting events by case id ......................................................................................................52
Figure 5-4 Vector of numbers given as input to the neural network (before normalization) ..................52
Figure 5-5 NN input values before and after normalization ...............................................................54
Figure 5-6 Model outline .......................................................................................................................55
Figure 5-7 Model summary ...................................................................................................................56
Figure 5-8 Plotting the values of model accuracy and loss ....................................................................58
Figure 5-9 Sample of traces misclassified by the NN ............................................................................59
Figure 5-10 Percentage of correct classifications ..................................................................................60
Figure 5-11 Classification accuracy for each fold ................................................................................63
Figure 0-10 Process model discovered from the given event log with inductive visual miner ..........64
Figure 0-11 Prom output format

Figure 0-12 Sorting events by case id

Figure 0-13 Vector of numbers given as input to the neural network (before normalization)

Figure 0-14 Visualization of the neural network structure

Figure 0-15 Model outline

Figure 0-16 Plotting of accuracy and loss values

Figure 0-17 Classification accuracy for each fold

Figure 0-1 Improving usability for non-experts using NNs

Figure 0-2 Improving understandability for non-experts using NNs

Figure 0-3 Table 6.1: Classification accuracy results

Figure 0-4 Model 3, derived from the initially discovered model

Figure 0-5 Model 4, derived from the initially discovered model

Figure 0-6 Model 5, derived from the initially discovered model

Figure 0-7 Model 6, derived from the initially discovered model

Figure 0-8 Model 7, derived from the initially discovered model

Figure 0-9 Model 8, derived from the initially discovered model

Figure 0-18 Back propagation neural network after training (Yoshimi, 2008)

Figure 0-19 Back propagation neural network before training (Yoshimi, 2008)

Figure 0-20 Fluctuation of the error values (Yoshimi, 2008)

Figure 0-21 Fully connected feed forward neural network (Nielsen, 2017)

Figure 0-22 Typical CNN architecture (Music Transcription with Convolutional Neural Networks)

Figure 0-23 Detailed CNN architecture (Music Transcription with Convolutional Neural Networks)

Figure 0-24 Typical RNN architecture (Olah, 2015)
List of Tables

Table 4-1 Initial set of traces........................................................................................................44
Table 4-2 Translation of an event log to vector of numbers..........................................................45
Table 4-3 Outlier discovery performed by the NN ........................................................................47
Table 4-4 Discovered outliers in the Initial log ............................................................................47
Table 5-1 Mapping of process events into letters ......................................................................51
Table 5-2 Classification accuracy for each one of the 10 folds ...................................................63
Table 0-12 Mapping of process events into letters ..................................................................64
Table 0-13 Classification accuracy for each one of the 10 folds ................................................71
Table 0-14 Classification results of models derived from model 1 ............................................72
Table 0-1 – Supervised machine learning major algorithms ......................................................77
Table 0-2 - Unsupervised machine learning major algorithms ..................................................78
Table 0-3 - Reinforcement machine learning major algorithms .................................................79
Table 0-4 - Implementation references ......................................................................................80
Table 0-5 Classification accuracy results of model 2 ..................................................................92
Table 0-6 Classification accuracy results of model 3 ..................................................................92
Table 0-7 Classification accuracy results of model 4 ..................................................................93
Table 0-8 Classification accuracy results of model 5 ..................................................................94
Table 0-9 Classification accuracy results of model 6 ..................................................................94
Table 0-10 Classification accuracy results of model 7 ..................................................................95
Table 0-11 Classification accuracy results of model 8 .................................................................95
Table 0-15 - Mapping input to output .........................................................................................101
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviations</th>
<th>Meaning</th>
<th>First Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM</td>
<td>Process Mining</td>
<td>p.1</td>
</tr>
<tr>
<td>NN</td>
<td>Neural Networks</td>
<td>p.6</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
<td>p.5</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
<td>p.95</td>
</tr>
<tr>
<td>BPM</td>
<td>Business Process Model</td>
<td>p.2</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi-Layer Perceptron</td>
<td>p.101</td>
</tr>
<tr>
<td>YAWL</td>
<td>Yet Another Workflow Language</td>
<td>p.30</td>
</tr>
<tr>
<td>BPMN</td>
<td>Business Process Model and Notation</td>
<td>p.29</td>
</tr>
<tr>
<td>BPEL</td>
<td>Business Process Execution Language</td>
<td>p.30</td>
</tr>
</tbody>
</table>
Chapter 1 Introduction

In this chapter, the presentation of the content of this thesis project is presented. Current situation is being described, research questions are defined, the methodology followed in order to answer them is included and the contribution of this project is highlighted. Along with those information, an outline of the whole report is added.

1.1. Current state of the art

Process mining is a relatively young research discipline that sits between computational intelligence and data mining on the one hand, and process modeling and analysis on the other hand. The idea of process mining is to discover, monitor and improve real processes (Van Der Aalst, 2011).

It is an important tool for modern organizations that need to manage non-trivial operational processes. On the one hand, there is an incredible growth of event data. On the other hand, processes and information need to be aligned perfectly in order to meet requirements related to compliance, efficiency, and customer service. Despite the applicability of process mining there are still important challenges, listed in the PM manifesto (Van Der Aalst, 2011) that need to be addressed. These illustrate that process mining is an emerging discipline.

A possible way to deal with all of these challenges could be with the use of deep learning models, since when trained, such models can successfully deal with highly complex problems. When having to deal with highly complex input patterns, deep neural networks can outperform other classifiers and provide accurate classification results. Deep learning refers to artificial neural networks that are composed of many layers. It's a growing trend in machine learning due to some favorable results in applications where the target function is very complex and the datasets are large.

One of these challenges in specific is the outlier classification challenge regarding the traces included within a business process model. Outliers are extreme events which when present in the event log of a process model and can significantly degrade its performance, or even cause more severe problems such as deadlocks.

When referring to Business Process Analysis, we refer to the activity of representing processes of an enterprise, so that the current process may be analyzed or improved. Process models can encounter a number of different problems that can lead to unwanted situations such as low performance, deadlocks, high execution time, etc.
1.2. Problem statement

Since process mining is still an emerging discipline, there are several challenges related to it that have not yet been dealt with. Within the process mining manifesto, eleven of them are being listed and described in detail (Van Der Aalst, 2011).

Although deep learning is considered to be a growing trend in terms of solving machine learning problems, its applicability in process mining has not been examined yet. In other words, there exists no concrete overview of which (if any) process mining problems can be tackled by using some of the already available deep learning techniques.

One of those challenges is the classification of outlier traces that are present within process models. When referring to outliers, we refer to exceptional behavior, such as noise, present in the event log data of a process model. Outliers can indicate faulty data, erroneous procedures, or areas where a certain theory might not be valid. In terms of process modeling, outliers refer to activities which are rarely or even never executed.

Knowing what outliers are and what kind of problems they can cause to a process model makes us want to find a way to get rid of them. This can be done by first analyzing the main flow of our process model, afterwards analyze the outliers and this way find patterns present in them that enable us to effectively detect them and remove them in a later step. This way the functionality and performance of our process models will be improved.

Research Questions overview

In this project the aim was two-fold. First of all, to investigate already existing literature, in order to discover whether some of the process mining challenges (as listed in the P.M. manifesto) can be solved with the use of deep learning. The second was to try to solve the outlier classification challenge, using real life event log data and deep learning.

Research questions

1. Which of the process mining challenges as listed in the PM manifesto (Aalst W. M., 2011) can be solved with the use of deep learning NN models and in what way?

2. Can the outlier classification challenge in terms of traces present in BPMs be solved by deploying deep learning NN models and using real life event log data as an input? In this case how accurate the outlier classification task actually is?

1.3. Methodology

The two main research questions tackled in this thesis project (mentioned in section 1.2) are resolved each one in a different manner. This means that this report is split into two parts, each of which is dedicated to the answer of one of the two aforementioned research questions.

In order to address the first one, an exploratory literature review took place. Aim of this review was to examine whether neural networks can handle some of the process mining challenges and in what way. The main literature sources used during this review were papers found with the use of Google scholar and online books. The Keywords that were used throughout the duration of the literature review and in order to find related
material include: “process mining challenges neural networks”, “process mining challenges deep learning”. Using those keywords 40 relevant papers were discovered, out of which 9 were actually considered directly relevant to the content of this review and were therefore used for its completion. Apart from the scientific papers, relevant information for use in the review was also found in two online books, both of which were found by applying the same keywords in Google scholar.

The first step of the review, which can also be seen in Figure 1-1, included the selection of the topic of “Process mining challenges that can be solved by neural networks”. After it, literature was searched and papers that were found got filtered. Accordingly the filtered papers were analyzed and interpreted and the final step included writing the findings of the whole review.

As far as the second research question is concerned, related to the outlier classification task in BPMs, the CRISP-DM (Cross Industry Standard Process for Data Mining) methodology as shown in Figure 1-2 (Wirth & Hipp, 2000) was used. CRISP-DM is a data mining methodology which provides a complete blueprint for conducting a data mining project. The lifecycle of a data mining project according to this methodology can be split into six phases:

1. **Business understanding** focuses on understanding project objectives from a business perspective, converting this knowledge into a data mining problem definition, and then developing a preliminary plan designed to achieve the objectives. In this phase, the outlier classification challenge was thoroughly studied and understood by me and possible ways that could lead to its solution were investigated by studying relevant literature. After this study, I concluded that this challenge could be successfully tackled with the use of NNs and deep learning and a solution plan was created.
2. **Data understanding** starts with initializing the collection of the data. Then, the data miner proceeds to increase familiarity with the data, to identify data quality problems, to discover initial insights into the data, or to detect interesting subsets to form hypotheses about hidden information. In my case,
the event logs that contain all of the data that is about to be checked for the presence of outliers were initially picked from an online repository (4TU) and then investigated by me in order to become familiar with it. I also decided about which data present in the event logs would be useful for my project in order to filter out useless data on the next step.

3. **Data preparation** covers all activities to construct the final data set or the data that will be fed into the modeling tool(s) from the initial raw data. Tasks include table, record, and attribute selection, as well as transformation and cleaning of data. The provided event logs were filtered and only useful data remained in them after the filtering process. Also, events belonging to the same trace were merged, using a process that is described in detail in section 5.2.

4. **Modeling** includes the selection of the modeling technique, the generation of test design, the creation of models, and the assessment of models. Since every modeling technique has specific requirements on the form of the data, it might be necessary to step back to the data preparation phase. Here the selection of deep feed forward NNs took place, which performed the trace classification task. Understanding the type of information those networks can deal with, made me jump back to the data preparation phase, in order to modify the provided datasets into a compatible format with the ones that neural networks use.

5. **Evaluation** is necessary to make sure the business objectives as defined in the first phase are met. Before proceeding to final deployment of the model, it is important to more thoroughly evaluate the model, and review the steps executed to construct the model, to be certain it properly achieves the initial objectives. A key objective is to determine if there is some important business issue that has not been sufficiently considered. At the end of this phase, a decision on the use of the data mining results should be reached.

6. **Deployment** means that the knowledge gained is structured and presented in a way that it can be used. Depending on the requirements, the deployment phase can be as simple as generating a report or as complex as implementing a repeatable data mining process. For this project, both a report is generated and a data mining process is implemented.
1.4. Thesis Contribution

In this thesis project a literature review takes place, aiming at discovering whether some (if any) of the process mining challenges can be successfully dealt with, using deep learning. For each of those challenges, the first step included in the review is the problem summarization. In this part all known aspects related to it are examined and presented. After this step, a problem classification takes place, where each described challenge is classified as into which larger class of challenges it belongs to. Accordingly, and based on the classification performed in the previous step, literature is provided in order to prove any claims made by me, related to the solution of each challenge. For those challenges that a solution is applicable, a presentation of it (based on literature) is included.

Apart from the literature review, this project focuses on the practical solution of one of those eleven challenges (outlier classification of traces in terms of a process model), using real life event data and deep learning. The aim is to prove that by constructing deep NN models, the outlier classification process can be successfully dealt with. This can be done by creating a tool that will be capable of classifying the traces present in the event log of real life process models as outliers or not. The application of this tool should be possible in any given process model (the tool should be generic).

By knowing where the outliers are present in an event log provides us with the capability of removing them in a later stage, so that the resulting process models in every case will be sound. This tool is going to be based on deep learning, using Artificial Neural Networks (ANNs) and more specifically the python libraries Theano and Keras which are used for creating and training deep ANNs. A deep ANN should be trained in such a way, so when provided with an event log of a process model it will be capable of classifying outlier traces in it.

1.5. Thesis Outline

As mentioned in section 1.2., within this project, the work load is split into two main parts. The former focuses on how deep learning can be used in order to solve some of the process mining challenges as they are listed in the process mining manifesto. The later one focuses on the practical solution of the outlier classification challenge, using real life event logs. Both parts are presented in the following chapters:
<table>
<thead>
<tr>
<th>Chapter 2</th>
<th>Background Information</th>
<th>Provide background knowledge related to both PM, NNs so that the reader can understand the content of the following chapters of this report</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 3</td>
<td>Deep leaning in PM</td>
<td>Exploration of which PM challenges (as listed in the PM manifesto) can be tackled with the use of deep learning models and in what way</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Outlier Classification of traces discovered from a Process Model’s event log</td>
<td>Presentation of how the outlier classification challenge in terms of the traces present in a BPM can be performed</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Experiment Setup</td>
<td>Application of the outlier classification approach described in chapter 4 in the real life event logs of “Sepsis Cases” and “Road traffic management process”³</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Conclusions</td>
<td>Presentation of the conclusions derived from this study, its limitations and proposal for future work.</td>
</tr>
</tbody>
</table>

³ [https://data.4tu.nl/repository/collection:event_logs_real](https://data.4tu.nl/repository/collection:event_logs_real)
Chapter 2 Background Information

This section provides some background information related with all the essential components that I used in order to complete this project. Namely, the information presented here is related to machine learning, process mining and artificial neural networks and can be considered as an introduction to techniques and terminology that I use in this project.

2.1. Process Mining (PM)

This section provides an introduction to the concept of process mining along with a summary of all the challenges that are related to it and have not yet been dealt with.

2.1.1. Definition

Process mining is considered to be a relatively new research discipline that lies between data mining on the one end and process modeling on the other. The reasons that contribute to the growing of the interest in process mining are that first, more and more events are being recorded, thus providing in depth information about the history of processes and second, there is a need to improve and support business processes in competitive and rapidly changing environments. Process mining could also be considered the automated construction of simulation models, model extension, repair, case prediction and history based recommendations (Van Der Aalst, 2011). Process mining is a process management technique that makes the analysis of business processes, based on event logs, possible. While process mining is being performed, specialized data mining algorithms are being applied to event logs, so as to identify trends, patterns and further details contained in them and recorded by an information system. Its aim is to improve the efficiency and understanding of processes. Automated business process discovery, is another terminology that could describe process mining (Gartner, 2015). Process mining has three goals, namely, to discover monitor and improve real processes. This can be performed by the extraction of knowledge and information from event logs that are widely available in today’s information systems. When looking at process mining from a distant perspective, three classes of it can be distinguished (process discovery, conformance checking and process enhancement). The splitting of process mining in the aforementioned three classes is based on whether a prior model exists and in the case that it does, in what way this model is being used during process mining.

---

4 http://www.processmining.org
It is considered to be a discipline for analyzing event data in the context of processes and consists of three types of problems, namely:

**Process discovery**: In the beginning of this step, no prior models exist. Having as a starting point an event log, a new model can be either constructed or discovered, based on low-level events. A representative example can be considered the use of the alpha algorithm in order to construct a petri-net out of an event log (Aalst W. v., 2004). There are multiple already established methods that can automatically construct process models, when given as an input an event log (Agrawal, 1998). As years progress, process mining is also undergoing changes and as a result today the research in this field has started to take into account more perspectives, e.g. data, time, etc. A representative example can be considered a method developed by Aalst, W. van der, & Dongen, B., which can be used to construct a social network (Aalst W. v., 2005).

**Conformance checking**: It is used when an a priori model exists. It is performed by comparing the existing model with the event log derived from the process and by checking for deviations between the log and the model. By detecting deviations, the model can be enriched. An example is the extension of a process model with performance data. In this case the a priori process model can be used to depict all the potential bottlenecks. Mentioning that, it should also be noted that model enhancement is not just about making the model better, but it also includes adding more information about the process. This may refer to performance, but also deviations or repair.

**Model enhancement**: In this step, the a priori model is extended with the target not being to check its conformance to a given event log, but rather to improve it. As an example can be considered the extension of a process model with data regarding its performance.

The techniques implemented in process mining are used when no formal description of the process can be obtained by other approaches, or when there is doubt about the quality of existing documentation. As an example can be considered the application of the methodology used in process mining to the audit trails of a workflow management system, transaction logs of an enterprise resource planning system (ERP), or electronic patient records in a hospital. All of them can result in models describing processes, organizations and products (Krichmer, 2013). The analysis of event logs can also be utilized in order to compare event logs with prior models in order to understand whether or not observations conform to a descriptive or prescriptive model.

### 2.1.2. Significance

The steps that take place in business process engineering (business management strategy, pioneered in the 90s) are similar to those followed by process mining. Process mining though, goes beyond those steps by providing feedback for business process modeling (Aalst W. M., 2011).

- **Process analysis**: Filtering and ordering of log files in order to gain better insights of process operations.
- **Process design**: Additional support to this step can be provided from process monitoring.
- **Process enactment**: Process mining results are being used based on logging in order to trigger further process operations.
2.1.3. Challenges
Despite the fact that Process mining has considerably wide applicability, there are still some challenges related to it that need to be taken care of. Since it is still an emerging and not established discipline, it needs to overcome these challenges successfully. A possibly effective method to deal with these is with the use of ANNs and deep learning. In the next section the ability of neural networks to confront some of these challenges will be studied. Here, all 11 of the process mining challenges are being listed (Van Der Aalst, 2011).

1. Finding Merging and Cleaning Event Data
   - Data may be distributed over a variety of sources. This information needs to be merged.
   - Event data are often “object centric”, rather than “process centric”.
   - Event data may be incomplete. Events do not explicitly point to process instances. Often it is possible to derive this information, but this may take considerable efforts.
   - An event log may contain outliers.
   - Logs may contain events at different levels of granularity.
   - Events occur in a particular context.
2. Dealing with Complex Event Logs Having Diverse Characteristics.
3. Creating Representative Benchmarks.
4. Dealing with Concept Drift.
5. Improving the Representational Bias used for Process Discovery.
6. Balancing between quality criteria such as Fitness, Simplicity, Precision and Generalization.
   - A model with good fitness allows for most of the behavior seen in the event log.
   - The simplest model that can explain the behavior seen in the log is the best model.
   - A model is precise if it does not allow for “too much” behavior.
   - A model should generalize and not restrict behavior to just the examples seen in the log.
7. Cross-Organizational Mining.
   - Detect the moment that a case deviates from the predefined process.
   - Build predictive models based on historical data.
   - Build recommender system based on predictions that propose particular actions to reduce costs or shorten the flow time.
9. Combining Process Mining with other types of Analysis.
10. Improving Usability for Non-Experts.
11. Improving Understandability for non-Experts.

2.1.4. Business Process Model
Business process modeling can be considered as the activity of representing processes within an enterprise, so that the current process can be analyzed and even improved. This activity is being performed by business analysts that have expertise in the modeling procedure, or specialized knowledge of the process that is being modeled. Another way to derive a process model is directly from event logs with the use of process mining tools\(^5\). Such an example is the a-algorithm\(^6\) that is a computer algorithm widely used in path finding and graph

\(^5\) http://www.promtools.org/doku.php
traversal. Business objectives may include increasing process speed and/or reduce processing time, in order to increase the quality of the overall model and reduce possible associated costs at the same time. Management decision to invest in the practice of business process modeling entails the need to document requirements included in an information technology project. With advances in software design, business process models can be put into practice and their vision is becoming fully executable and capable of undergoing simulations

2.1.5. Event Log

In terms of process modeling, an event log can be considered to be a basic resource that helps provide information about events usage and other metrics. An event log stores these data for retrieval by process engineers or automated systems in order to help them manage various aspects such as security, performance and transparency. In other words, an event log is a basic “log book” that is analyzed and monitored for higher level process mining. It is capable of capturing many different types of information. For example it can capture all logon sessions to a system, along with account lockouts, failed password attempts, etc. It can also record different types of application events, such as application errors, closures or other related events.

Event logs can be often used by event management tools in terms of process mining, such as Prom, that provides higher level of analysis on the contents of an event log, to help process engineers construct, analyze, or determine what is wrong within a process model.

2.2. Artificial Neural Networks (ANNs)

This section provides an introduction to the concept of neural networks, including learning algorithms, structure styles and a practical example of how a feed forward neural network using back propagation of the error performs an association between input and output values.

2.2.1. Definition

One of the various techniques that can be used for implementation of machine learning tasks is artificial neural networks. By the term “Artificial neural network”, someone refers to the interconnections that are located between neurons among the different layers of a system. They can be considered as a computational approach that is based on a large collection of neurons trying to imitate the way a biological brain operates and mimic its problem solving ability.

A neural network can be considered to be an interconnected assembly of simple processing elements, otherwise called units or nodes, whose functionality is loosely based on the animal neuron. The ability of the network to process information is stored within the interunit connection strengths, or weights which is obtained by a process of adaptation to, or learning from, a set of training patterns (Gurney, An Introduction to Neural Networks, 1997).

6 http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
8 https://www.techopedia.com/definition/25410/event-log-networking
In terms of human brain functionality, it can be said that it consists of an estimated 100 billion nerve cells (or neurons), an example of which can be seen in Figure 2-1.

![Figure 2-1 Essential components of a neuron (Sinha, 2010)](image)

Neurons communicate with the use of electrical signals that are short spikes in the voltage of the cell wall. The interneuron connections are intervened by synapses (electrochemical junctions), which are located on branches of the cell, referred to as dendrites. Each neuron may typically receive several thousands of connections from other neurons and is in this way constantly receiving a large number of incoming signals that will eventually reach the cell body. Here they are integrated together (or summed) in some way and if the resulting signal exceeds some threshold value, then the neuron will “fire” in response. This will then be transmitted to other neurons via the axon. In order to determine whether an impulse should be produced or not, some incoming signals produce an inhibitory effect and tend to prevent firing, while some others are excitatory and promote impulse generation. The distinctive ability to process of each neuron is then supposed to reside in the type (excitatory or inhibitory) and strength of its synaptic connections with other neurons. This type of architecture and processing style is incorporated in artificial neural networks and due to the emphasis that is placed on the importance of the interneuron connections this type of system is referred to as being connectionist (Gurney, An Introduction to Neural Networks, 1997). The artificial equivalents of biological neurons can be considered to be the nodes or units in the preliminary definition and an example is shown in Figure 2-2. Synapses are modeled by a single number (or weight) in such a way so that each input is multiplied by a weight (the weight of the synapse) before being sent to the equivalent of the cell body. In this figure the weighted signals are summed together, by an arithmetic addition to supply a node activation.

![Figure 2-2 Simple artificial neuron (Kawaguchi, 2000)](image)
The connection in between the nodes can be done with feed forward or feedback arcs and contains an activation function. This activation function can be a summation function that combines the values of all its inputs together. Alternatively it can follow what is called “threshold logic unit” or “TLU”, when the value is compared with a threshold. In the case that the activation exceeds the threshold, the unit produces a high value output, “1” by convention, otherwise it outputs “0”. The most commonly used activation functions include:

- Softmax
- Relu
- Tanh
- Sigmoid

The size of signals is represented by the width of the corresponding arrows, weights are depicted by multiplication symbols in circles and their values are supposed to be proportional to the symbol’s size and only positive weights are being used. The TLU is considered to be the simplest model of an artificial neuron. When referring to the term “network”, a system of artificial neurons is meant. This can range from something simple, as a simple node, or to a large connection of nodes in which each one is connected to every other node of the net. A type of such a network is shown in Figure2-3. Each node is represented by a circle, but weights are implicit on all connections. The nodes are arranged in a layered structure in which each signal originates from an input and passes throughout two nodes before reaching an output, beyond which it is no longer transformed. Each artificial neural network can be split into three types of layers. The first one is the layer with the input neurons, which has the functionality to send data through the synapses to the next layer of the network. This layer is called “hidden layer” and can be consisted itself of an arbitrary number of layers. The final layer of the system is the output layer, which contains the output neurons. The number of different layers in a network may vary and each one could consist out of a number of neurons. Each neuron can be connected with many others that belong either in the same layer or to other ones. The depicted feed forward structure of Figure 2-3 is only a possible one among several other available alternatives and is used in order to classify an input pattern into one of several alternative classes(two in this case)according to the resulting pattern of outputs (Gurney, An Introduction to Neural Networks, 1997).
These are the basic structural elements and the way they operate. Emphasis is placed on learning from experience (training). In real neurons, the synaptic strengths, could under certain conditions, be modified so that the behavior of each neuron can change or adapt to its stimulus input. In terms of artificial neurons, the equivalent of this is the modification of the weight values. In terms of information processing, there are no computer programs here and therefore the knowledge the network has is supposed to be stored in its weights, which evolve during a process of adaptation to stimulus from a set of pattern examples.

Each particular recipe for changing the weights in the network constitutes a learning rule. Weights in the network can be changed in an iterative procedure. After feeding an input pattern to the input layer, the network computes (based on its current weights), an output. Using a so-called learning rule, the output is evaluated (for example by comparing to an expected output) and a particular strategy is applied to adjust the weights based on the evaluation. When the required weight updates have been made, another pattern is presented, the output is presented with the target and new changes are made. This sequence of events is repeated iteratively many times, until the network’s behavior converges so that its response to each pattern is close to the corresponding target. The training process however is not always successful, meaning that there are cases where a correct match between network inputs and outputs is not always achieved. The process as a whole, including any ordering of pattern presentation, criteria for terminating the process, constitutes the training algorithm (Gurney, An Introduction to Neural Networks, 1997).

In one training example, called supervised learning, used in combination with nets, an input pattern is presented to the net and its response is then compared with a target output. In a letter recognition example a pixel image of a letter “A” could be the input of the network and the output could be compared with the classification code for A. The difference between the two patterns of output then determines how the weights are being altered. If after training the network is presented with a pattern it hasn’t seen before, but if it has learned the underlying structure of the problem domain, then it should classify the unseen pattern correctly and it will be described as one that generalizes well. If it does not have this property, it is little more than a classification lookup table for the training set and is of little practical use. This being said, it is noted that good generalization is one of the key properties of neural networks (Gurney, An Introduction to Neural Networks, 1997).

Artificial neural networks in combination with a learning algorithm used to train them are considered to be self-learning systems and are thus trained rather than explicitly programmed. They perform considerably well in areas where the solution is impossible to be expressed with the use of a programming language.

In order to fully define an artificial neural network, three types of parameters need to be defined:

1. The first one has to do with the structure of the network and includes the number of layers that the network consists of, along with the number of nodes that each layer includes. Along with the structure of the network, the activation function that makes the conversion of a neuron’s weighted input into its output activation also needs to be defined. The activation function of a node defines the output of that node, given an input or a set of inputs. Nonlinear activation functions allow networks to compute nontrivial problems using only a small number of nodes.
2. The interconnection pattern among the different layers of neurons, or put it differently the way that the nodes that consist the different layers of the network are connected between them.
3. The learning process in order to update the weights of the interconnections. It is a method that improves the overall performance of the network and in most cases it is applied repeatedly over it. The way it is performed is by updating the weights and the levels of bias of a network, when it is simulated in a specific environment as seen in Figure 2-4. A bias unit is an “extra” neuron added to each pre-output layer that stores the value of 1. Bias units aren’t connected to any previous layer and in this sense don’t represent a true “activity”. As shown in Figure 2-4, the bias units are characterized by the text “+1”. A bias unit is just appended to the start/end of the input and each hidden layer, and isn’t influenced by the values of the previous layer. In other words, these neurons don’t have any incoming connections. The reason that bias units exist, is that bias units still have outgoing connections and they can contribute to the output of the ANN. The learning process is a factor that decides how fast and with what accuracy the network can be developed.

As stated in the above section, there are three main models of machine learning, namely: unsupervised, supervised and reinforcement learning. The choice of which one is going to be used in every scenario, depends on the process that is about to be developed.

![Figure 2-4 Explanation of bias (Varma, 2016)](image)

2.3. Keras

In this section an introduction to the python library named keras is presented. Keras is used for the development, customization and execution of neural network models.

2.3.1. Definition

Keras is a high-level neural networks library, written in Python and is capable of running on top of either TensorFlow or Theano. It was developed with a focus on making fast experimentation possible. The ability to go from idea to result with the least possible delay is of key importance to doing good research. Keras allows for easy and fast prototyping, through total modularity, minimalism and extensibility. It has the ability to support
convolutional and recurrent networks, as well as combinations of both of them. It allows arbitrary connectivity schemes, including multi-input and multi-output training and is able of running in both CPU and GPU\textsuperscript{9}.

By the term modularity, a sequence or a graph of standalone, fully configurable modules that can be plugged together with as little restrictions as possible is meant. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularizations schemes can all be considered standalone modules that when combined can create new models.

With the term minimalism, the fact that each module should be kept short and simple is meant. Every part of code should be transparent upon first reading.

Extensibility has the meaning that new modules can be added in a very simple way. The ability to create easily new modules enables expressiveness at a very high level, making keras suitable for advanced research.

\textsuperscript{9} https://keras.io/
Chapter 3 Deep learning in Process Mining

In this section a presentation of all the eleven process mining challenges takes place along with the way that neural networks can contribute or not to their solution. The way that this exploratory literature review is organized, is by first classifying all of the listed challenges into what type of problem they are. After defining the problem type, reasoning based on literature is provided for why NNs can successfully deal with it. For those challenges that related literature was not found, reasoning is provided to back up my claims. Regarding those challenges that neural networks are not able to provide a solution, only a discussion of the corresponding challenge takes place and a justification of why NNs can’t deal with it.

Research question: Which of the process mining challenges as listed in the PM manifesto (Aalst W. M., 2011) can be solved with the use of deep learning NN models and in what way?

Research methodology: Research books, articles, journals, reports.

Research Gap: This literature review attempts to examine whether neural networks can deal with some of the process mining challenges.

Keywords used: Process Mining; Neural Networks; Deep Learning.

3.1. Challenge 1: Finding, Merging, and Cleaning Event Data

The first challenge has to deal with the process of finding, merging and cleaning event data and can be split into six sub-challenges. In the following section a description of how these sub-challenges can be solved with the use of neural networks is being presented.

3.1.1. Merging distributed data

The first sub challenge of the first process mining challenge includes the procedure of merging data that are scattered over a variety of sources.

Challenge summarization

Data can be scattered over a variety of sources and this leads to the need of merging this information together. This problem can be proved to be slightly more complicated, since different identifiers are used in the different
data sources. One such example could be a system that uses name and birthdate to identify a user, while another uses the person’s social security number. A way should be found in order to be able to merge information that belongs in the same entity, despite the presence of different identifiers used in the different data sources (Van Der Aalst, 2011).

Analysis of problem
This problem can be considered as a merging data challenge, since information belonging to the same entity but scattered over a variety of different sources, should be merged together. It can’t be solved with the use of neural networks, since it requires merging distributed information that corresponds to the same entity even when using different identifiers.

Approach for addressing the challenge
This problem can’t be considered as a classification, clustering or prediction challenge (which can be dealt with the use of NNs) thus giving more value to the argument that neural networks can’t deal with it. In order to solve this challenge, a number of rules should be created which when applied, will lead to merging data that originate from different sources. Those rules can be created with the use of dynamic programing.

3.1.2. Turning event data from “object centric” to “process centric”
The second sub challenge of the first process mining challenge includes turning event data from object centric to process centric.

Challenge summarization
Event data can often be “object centric”, rather than “process centric”, meaning that focus is put on objects instead of processes. An example could be considered individual products that may have RFID tags and recorded events refer to these tags. But in order to be able to monitor a particular customer order, such “object centric” events need to be merged and preprocessed. This means that objects that belong to a customer’s order, but contain different ID tags should, be merged together in order to compose the whole order to this customer. This actually implies that a way shall be found that turns event data from “object centric” to “process centric” (Van Der Aalst, 2011).

Analysis of problem
The process of turning event data from “object centric” to “process centric” implies that a way should be found that incorporates together objects that belong to the same process but are currently not merged. In this sense it can be considered as a merging of data challenge (requires collection of data from different sources).

Approach for addressing the challenge
In order to complete this task a data identification procedure should take place that first identifies which objects belong in the same process and at a later step incorporates them to it. This task can be performed with the use of programming and not with deep learning.
3.1.3. Dealing with incomplete event data

The third sub challenge of the first process mining challenge includes dealing with incomplete event data.

**Challenge summarization**

Event data can be incomplete in many cases. A regular problem is that events do not explicitly point to process instances. When referring to the term “process instance”, we refer to a set of process activities or certain types of processes, always in accordance to an event log. In many cases it is possible to derive this information, but this could take considerable efforts, since some type of cross-validation check should be performed between the event log (where all the process instances are listed) and the process model that is derived from this log. Furthermore, time information can be missing from some events. Timestamps might need to be inserted in order to still be able to use the available timing information. A way should be found in order to somehow deal with the missing event data (Aalst W. M., 2011).

**Analysis of problem**

The problem of missing event data from a log is considered to be a prediction challenge with regards to the next event present in a specific log. This stands in the sense that these predictions of the missing log data could eliminate the missing log values, thus complementing the initial log.

**Approach for addressing the challenge**

Neural networks are capable of dealing with this kind of prediction challenges (Tax, 2016) and are therefore capable of offering a solution to the “incomplete event data” challenge, by complementing the missing log values. The solution of this challenge using neural networks consists of six steps as seen in Figure 3-1.

**Step 1 - Initial event log**

The whole task of dealing with incomplete data starts with a given event log, that is derived from a business process model. Alternatively, a log related to a process model could be provided and in this case the process model should be discovered. This log may consist of a number of different traces and each trace of a number of different events.

**Step 2 - Log encoding into feature vectors**

In this step the explanation of how a training set can be created for the NN constructed in the next step is presented. The goal is to predict the next activity in a case and its timestamp and this can be done by learning an activity prediction function and a time prediction function. Each trace within the event log is turned into a feature vector, which is later going to be used as input to the constructed LSTM NN (training set). The feature vectors are constructed initially with a number of features that represent the type of activity of an event using one-hot encoding. One-hot encoding assigns the value “1” to a specific feature and “0” to all the rest. Three time-based features are also added to the one-hot encoding feature vector. The first time based feature of a given event is the time between the previous event in the trace and the current event. This feature enables the LSTM to learn the dependencies between the time differences at different points in the process. The second time feature is the time that the current event is taking place and the third one is the time between the current event and the end of the process. One extra element to the one hot encoding vector is added, which has the value 1 in the case that the current event of the trace is not the last one within the trace. Knowing the timestamp of the current event, makes it possible to calculate the timestamp of the following one. The “Adam”
learning algorithm is used to optimize the weights in the network, such that the cross-entropy between the ground truth one-hot encoding of the next event and the predicted one-hot encoding of the next event are minimized (Tax, 2016).

**Step 3 – Construct LSTM neural network**
This step includes the construction of an NN that may contain a number of LSTM layers and will perform the task of completing incomplete event data. This LSTM NN can have different architectures for modeling the different activity prediction and time prediction function as seen in Figure 3-1. (Tax, 2016).

![Figure 3-1 NN architectures with single task layers (a), with shared multitask layer (b) and with n+m layers of which n are shared (c) (Tax, 2016).](image)

**Step 4 – Training of LSTM NN by traversing the event log**
The training is performed by traversing each trace of the log from beginning to end and by making predictions for the next to be executed activity and its timestamp, with the feedback loop of the LSTM NN working as a feedback loop. Modeling the next activity prediction function and time prediction function with LSTMs can be done using three different architectures. Two separate models can be trained (one for the activity prediction function and one for the time prediction function), both using the same input features in each time step. Alternatively, both the time prediction function and activity prediction function can be learned jointly in a single LSTM model that generates two outputs in a multi-task learning setting. Finally a hybrid option between the two aforementioned architectures can be used. This can be done, by creating an architecture that has a number of shared LSTM layers for both tasks, followed by a number of layers that specialize in either prediction of the next activity, or prediction of the time until the next event (Tax, 2016).

**Step 5 – Testing of the LSTM NN on the part of the log that some events and their timestamps are missing**
The testing of the NN is performed in the parts of each trace that contain the missing values. The NN makes predictions about possible continuations of each partial trace and about the timestamp of each missing event.
Step 6 - LSTM NN will output several possible continuations of the partial trace
The activity prediction function outputs the probability distribution of various possible continuations of the partial trace (which includes parts of the log with the missing values) (Tax, 2016).

Step 7 – Most likely continuation picked.
The most likely continuation is going to be picked in order to fill in the missing log values (Tax, 2016). This means that the most possible continuation of every specific trace is going to be compared with the trace containing the missing values itself and this way the missing values are about to be complemented based on the LSTM trace prediction (Tax, 2016). In Figure 3-2, the solution of this challenge can be schematically seen.

3.1.4. Outlier classification in terms of a process model
The fourth sub challenge of the first process mining challenge includes the outlier detection procedure, the solution of which (with the use of neural networks) in a theoretical level is presented below in short. The detailed description of how the outlier classification works in terms of a business process model is explained in detail in chapter 4.

Challenge summarization
Outliers can be seen as unusual behavior and can also be referred as noise. When seen in the context of
process modeling, they can be considered as anomalous data points that can significantly degrade the outcome of an otherwise reasonable model identification procedure (Pearson, 2002). Put it otherwise, outliers can be seen as data points in a data set that are inconsistent with our expectations based on the volume of the available data. They occur frequently in practice and can have serious consequences. The problem of outliers is in close relation with the one of missing data. One reason that can lead to this conclusion is that outliers can arise from inappropriate treatment of missing data. Furthermore, one way to treat outliers is rejection, thus creating missing data values (since outliers are removed from the event log, resulting in a log of reduced size).

In terms of process modeling, outliers can refer to:

- a whole trace of a process
- a single event that is part of a trace
- a particular attribute that belongs to an event

**Analysis of problem**
The procedure of identifying all of the above types of outliers is considered to be a binary classification problem, since for every trace/event/attribute of an event log, an indication should be provided to the user in order to indicate the presence or absence of outliers in a process model’s event log.

NNs have the ability to be trained given enough training data, the ability to generalize and this ability that they possess makes them capable of solving a wide variety of machine learning problems, including the outlier classification one. Here the solution of this challenge is presented in theory, referring to the classification of outlier traces within a process model.

**Approach for addressing the challenge**
Neural network’s ability to generalize means that when trained, they can perform tasks (in this case the outlier classification task) with the same accuracy as the one they achieved during the training phase on the training dataset. So a trained neural network, when provided in its input with an event log belonging to a process model will be able to classify its traces/events(attributes as being outliers or not. The solution of this challenge using neural networks consists of three steps, which when implemented sequentially can provide the user with an indication about the existence or not of outliers in a given process model’s event log.

**Step 1 - Translation of the event log in compatible format with the neural network**
The first step includes the translation of the information present in the event log into a format that can be used by an artificial neural network for its training. The type of information present in an event log can’t be given directly as an input to a neural network, since neural networks accept only a specific type of information (vector of numbers) and the event log may contain information in string format. This means that some form of encoding to the elements of the event log should be applied in order to turn

---

them into vectors of numbers. Vectors of numbers can be used for training and testing by the neural network that will be constructed in the second step of this process.\footnote{\url{http://datascience.stackexchange.com/questions/869/neural-network-parse-string-data}}\footnote{\url{http://stackoverflow.com/questions/4674623/why-do-we-have-to-normalize-the-input-for-an-artificial-neuralnetwork}}

**Step 2 - Neural network training and testing process**
This step includes the procedure of actually constructing a neural network that will be able to learn how to classify each element of the event log as being an outlier or not. Choices regarding the whole structure of the network as well as choices regarding its training parameters should be explicit and justified. After the construction of the network is complete, the training phase takes place.

**Step 3 - Translation of the output of the neural network in compatible format with the event log**
After the training is complete and the accuracy of the classification task has been measured, the outliers that exist in a specific event log each time will be determined, with accuracy as high as the accuracy of the classification task is.

### 3.1.5. Dealing with event logs containing events at different levels of granularity
The fourth sub challenge of the first process mining challenge deals with event logs that contain events at different levels of granularity.

**Challenge summarization**
There is a possibility that some logs could contain events at different levels of granularity. This means that some events present in a log might be way more important than some others present in the same log. An example is a hypothetical event log of a hospital information system, describing all of the possible treatments at this hospital. Events in this log can refer to simple blood tests, simple surgical procedures, or complex ones. Furthermore, when referring to timestamps, they could also have different levels of granularity, that can range from seconds precision (28-9-2011:h11m28s32) to a general date information (28-9-2011). An approach should be developed for the process analyst, in order to be able to deal with all those types of logs (Aalst W. M., 2011).

**Analysis of problem**
This challenge can’t be considered neither as a classification, nor as a prediction challenge regarding events or traces of events present in an event log. Furthermore, it is open to interpretation whether some events present in a log have a higher granularity comparing to some others and it also has to do with the context in which they occur. Alternatively, extra information could be provided within each log, denoting the level of granularity of each specific event, making this way the challenge of defining the different levels of granularity easier. In this sense, this challenge can be considered as a missing data challenge.

**Approach for addressing the challenge**
The problem of dealing with logs at different levels of granularity can be successfully dealt with the use of programming, where various rules regarding the relationship between different levels of data granularity can be
implemented, providing this way the system with a way to distinguish all the different levels of granularity. For these reasons neural networks aren’t able of successfully dealing with this challenge.  

### 3.1.6. Merging of event data with contextual data

The fifth sub challenge of the first process mining challenge deals with the process of merging event with contextual data.

**Challenge summarization**

Events occur in a particular context (weather, location, day, etc.) and this context may explain certain conditions related to these events. For example the case where the response time in a call center is longer than usual because of work in progress or holidays. For analysis purposes, it is better to incorporate this context in which such events occur. This is because by incorporating the context at which events occur, a more clear overview of the presented process is provided. This implies the procedure of merging event data with contextual data. Here the “curse of dimensionality” comes into play, as analysis of data becomes harder to manage when too many variables are added into them (Aalst W. M., 2011).

**Analysis of problem**

This challenge can’t be considered as a classification, clustering, or prediction challenge, but rather as a missing data problem. This is why neural networks are unable of providing a solution for it.

**Approach for addressing the challenge**

The procedure of defining the context at which events (that belong in an event log) occur includes extra information comparing to those provided in an event log. This extra information should be related with the context that each event occurs (and attached to each individual event). Put it otherwise, the solution of this challenge lies in providing extra information to the process modeler, regarding the context at which events in a log occur.

### 3.2. Challenge 2: Dealing with Complex Event Logs Having Diverse Characteristics

The second process mining challenge deals with complex event logs that have diverse characteristics.

**Challenge summarization**

There is the possibility that event logs may have very different characteristics. Some of them may be extremely large, thus making it difficult to handle them and some others are so small that not enough data is available in order to draw reliable conclusions. In some domains, large quantities of events are recorded and thus additional efforts are needed so as to improve performance and scalability. Apart from the number of events recorded, there are also other characteristics such as the average number of events per case, similarly among cases, the number of unique events and the number of unique paths.

---

13 There is one classification challenge: to change the granularity of events by aggregating several lower level events (fine granularity) into a higher level event (coarse granularity). Identifying which lower level events constitute a higher level event could be a classification problem. But in most cases the classification has only few dimensions and no NN is needed.
Due to the fact that event logs contain only sample behavior, they should not be assumed to be complete. This makes it more difficult to deal with small event logs that have a lot of variability. Some logs that contain events at a very low level of abstraction and that tend to be extremely large, have individual low-level events that are of little interest to the stakeholders. So it would be beneficial if low-level events were aggregated into high level events.

In a hypothetical example of analyzing the diagnostic and treatment processes of a particular group of patients, one may not be interested in all the individual tests that are recorded in the information system of the hospital’s laboratory. Nowadays, organizations need to use a trial and error approach to see whether an event log can be used in process mining. Therefore, tools should allow for a quick feasibility test when they are given a particular data set. Such a test should explicitly indicate potential performance problems and warn for logs that are either far from complete or too detailed (Aalst W. M., 2011).

**Analysis of problem**
The solution of this challenge requires the creation of a method (using neural networks) that will be able to classify every log provided to it, as capable of being used in process mining or not. The fact that every provided log needs to be classified depending on its usability in process mining is the reason why this is considered to be a classification challenge (either it can or can’t be used in process mining).

**Approach for addressing the challenge**
Neural networks are competent of performing classification tasks and thus able to solve this challenge (Zhang, 2000). The idea behind the solution lies in creating a pool of labeled sample logs (by judging their ability to be used in process mining or not) and by using them to train a neural network created by us. The trained neural network will be able of providing a classification for each new log given to it as input for whether it can be used or not in process mining.

**Step 1 – Create the event log**
The procedure of defining whether or not an event log can be used in PM starts with the creation of a training and test dataset for the feed forward NN constructed in the next step. This log should contain a number of different vectors and each attribute of a vector should represent a feature that can contribute to the characterization of each event log as usable or not in PM. So this first step would include the decision of which attributes can be used for the creation of training and testing dataset. The features that each vector should consist of should be related to properties of the log and can be:

- Number of unique events
- Number of unique paths
- Number of events per case
- Number of events recorded
- Extent of variability
- Number of cases
- Level of event abstraction
- Length of largest trace
- Length of smallest trace
**Step 2 – Log encoding into binary format**
This step includes the creation of the training and testing dataset for the NN, based on the features defined in the first step. It also includes the manual labeling of those traces as suitable or not for being used in process mining. The manual labeling of the traces will lead to the creation of a labeled dataset, used later on for the supervised learning of the feed forward NN.

**Step 3 – Build a feed forward neural network that will perform the classification**
The constructed neural network will consist of a number of different inputs (depending on the length of the encoded vector of numbers provided to it every time), a number of hidden layers and one output, (0/1) denoting whether it can or can’t be used in process mining. The number of inputs the created NN is about to have is in close relation with the parameters taken into account when judging whether a process model can be used in process mining that are mentioned in step 1.

**Step 4 – Training and testing the classification accuracy of the neural network**
This step includes the procedure of training and testing of the feed forward NN based using the labeled dataset constructed in step 3.

**Step 5 – Evaluation of the classification**
After some traces of the event log have been used for training and others for testing, the classification accuracy of traces (usable or not in PM) that the NN can achieve can be determined both for the training data (acc) and for the testing data (val_acc). Also the traces that were correctly classified along with those that were not can be determined. The solution of this challenge is schematically seen in Figure 3-3.

---

**Figure 3-3 Dealing with complex event logs with NNs**

1 – Create the event log using a number of different features
2 – Encoding of the traces present in the log in vectors of binary data
3 - Construct Feed Forward NN
4 – Train and test the NN by using the created dataset
5 – Identify logs that can be used in process mining and others that can’t
3.3. Challenge 3: Creating representative benchmarks

The third process mining challenge deals with the process of creating representative benchmarks.

Challenge summarization
Due to the fact that process mining is an emerging technology, good benchmarks are missing and thus need to be created. For example, there are quite a few process discovery techniques available and different vendors offer different products, but there is no agreement on the quality of these techniques. Despite the fact that there are differences in functionality and performance, it is difficult to compare all the different techniques and tools. Therefore, good benchmarks that consist of example data sets and representative quality criteria need to be developed. If we are about to break down process mining, we can claim that it mainly consists of two components: process modeling and data mining. As far as data mining is concerned, benchmarks are already available and their existence has helped tool providers and researchers to improve the performance of their techniques. When talking about process mining as a whole the procedure of creating benchmarks is more challenging, since no benchmarks related to process modeling are available. Some initial work has already been done, for example there are various metrics for measuring the quality of process mining results, such as fitness, simplicity, precision and generalization. On the one hand there is the need to create benchmarks based on real-life data sets and on the other there is a need to create synthetic datasets capturing particular characteristics, in the case those characteristics can’t be captured in real-life data sets. Those synthetic datasets help to develop process mining techniques that are oriented towards incomplete event logs, noisy event logs, or specific populations of processes. Apart from the creation of representative benchmarks, there also needs to be consensus on the criteria used to judge the quality of process mining results. Some techniques that are used in data mining, such as cross-validation techniques can be adapted in order to judge the result. In the example of k-fold checking, the event log can be split into k parts. K-1 of them can be used in order to learn a process model and conformance checking techniques can be used to judge the result with respect to the remaining part. This procedure can be repeated K times, thus providing insights into the quality of the model (Aalst W. M., 2011).

Analysis of problem
The challenge of creating representative benchmarks for process mining can be considered as a collection of data challenge, since data related to process mining should be:

- Easily available
- In a central location
- Using common definitions
- Having a statistically relevant sample size

Approach for addressing the challenge
In other words, no neural network model can be trained in order to be able to perform the benchmarking process, since it is neither a classification, nor a prediction task, that neural networks can successfully handle. Benchmarking should be a continuous procedure (and not an one-time exercise) that requires human intervention in order to be completed.
3.4. Challenge 4: Dealing with concept drift

The fourth process mining challenge deals with concept drift.

Challenge summarization

When referring to the term “concept drift”, one refers to the situation in which a process changes its characteristics as time progresses so that cases at the beginning of the event log follow a different pattern than cases at the end of the discussed event log. The inflicted changes are performed in unforeseen ways and this makes the task of conducting projections to the future not possible, since predictions become less accurate as time passes. One example could be the case where, while in the beginning of an event log two activities are concurrent, later on they become sequential. Processes might change due to a variety of reasons such as:

- Periodic/seasonal changes: For example fluctuation in the demand for one specific product around the year (such as clothes), can cause changes in a process model that models the way a company manages its inventory in all four of the different seasons of the year.
- Due to changing conditions: For example the situation that the market is getting more and more competitive can cause again some form of concept drift.

Such changes have an impact on processes and thus it is vital to detect and analyze them. One way that concept drift in a process can be discovered is by splitting the event log into smaller logs and analyzing the “footprints” of the smaller logs. Such “second order” analysis requires much more event data, but only a few processes are in steady state and thus understanding concept drift is of prime importance for the management of processes. Therefore, additional research and tool support are needed to adequately analyze concept drift (Van Der Aalst, 2011).

Analysis of problem

The challenge of dealing with concept drift can be seen as a clustering challenge, since for every event log periodical checks need to be performed in order to define whether concept drift has occurred on it or not.

Approach for addressing the challenge

Neural networks are capable of solving clustering tasks and are therefore capable of solving this challenge (Zhang, 2000). The idea that lies behind the solution of this challenge is to train a NN in an unsupervised manner (this is because a process might change at the time it is being analyzed and therefore supervised learning won’t work) so that it can provide in its output whether concept drift has occurred in a process model. In other words, the NN should be able to learn by itself when concept drift has occurred in a trace of the event log (derived from a process model) that is provided to it. The training process has to be performed in an unsupervised manner (since it is difficult to know when concept drift has occurred in a log or whether it is going to occur in the future). Traces from a different "variant" (after concept drift), go to a different cluster (if sufficiently different).

---

14 Each execution of a task leaves a distinctive trace in the data modification log, i.e. a typical set of events referring to modifications of the set of data types accessible in this task. This typical set of data types modified by an activity shall be referred to as the activity’s footprint.
Step 1 – Construct the event log
The first step includes the construction of the event log that is about to be used as input from the SOM constructed in the next step for training. This event log should consist of a number of traces and each event of a trace should correspond to a feature related with the examined process model. Each trace can be seen as a feature vector. Some possible features can include:

- Size of the whole model
- Number of unique events
- Number of unique paths
- Number of events per case
- Number of events recorded
- Extent of variability
- Level of event abstraction

Step 2 – Build a SOM that will perform the clustering
This step includes the creation of the SOM that will perform the clustering of traces (whether concept drift has occurred in them or not). SOMs are chosen here, since they can be trained in an unsupervised manner and by taking into account the input features of each training vector can provide a mapping in their output, clustering together similar traces (Kohonen, 1990). The number of inputs of the SOM is the same as the number of features that each feature vector will consist of. The output of the SOM can be viewed in a 2-d grid representation, in which all the similar traces get clustered together (Kohonen, 1990). During the construction phase of the SOM the number of neurons it will consist of also has to be defined. Each neuron has a permanent location and a variable weight. Initially every neuron’s weight vector is initialized with small random values.

Step 3 – Train a NN to identify concept drift that has occurred
This step includes the training of the constructed SOM for several epochs, so that in the end all the similar input feature vectors will be clustered together (Kohonen, 1990). During training each neuron’s weight changes, while its location remains constant. The output layer is a grid with a weight vector, corresponding to the number of features in the input. To train a specific neuron, each input vector is multiplied by a weight and then is summed to produce a single value that is set as the output vector. As the training progresses, neuron locations are changed, such that a meaningful coordinate system is formed on the map. Every neuron in the map undergoes this training and the values of each weight are changed for each neuron. As the training progresses, neuron locations are tuned, such that a meaningful coordinate system is formed on the map. The SOM thus forms a topographic map of the input patterns.

Step 4 – Use NN to check whether concept drift has occurred in a process model derived from an event log
In the end of training, similar traces will be clustered together in the (X, Y) state space and based on the clustering outcome, traces that have or haven’t undergone concept drift can be identified by looking at the state space, as they will belong in a different cluster than the rest of the traces. The solution of this challenge is schematically seen in Figure 3-4.
3.5. **Challenge 5: Improving the representational bias used for process discovery**

The fifth process mining challenge deals with improving the representational bias used for process discovery.

**Challenge summarization**

A process discovery technique has as a target to produce a model using a particular modeling language such as BPMN or petri nets. However it is important to make a separation between the visualization of the result from the representation used during the actual discovery process. The selection of a target language in a lot of cases includes several implicit assumptions:

- It limits the search space: Different modeling rules apply to each modeling language and thus the selection of one can limit the modeling options available.
- Processes that can’t be represented by the target language cannot be discovered: Again the restrictions that the modeling language includes limits the freedom of the process modeler. This actually implies that discovering a process has as a prerequisite the ability to represent it first.

The so called “representational bias” used during the discovery process should be a conscious choice, taking into account the benefits each representation can offer and should not be only driven by the preferred graphical representation (Van Der Aalst, 2011).

**Analysis of problem**

This problem can be considered as a classification challenge, since for every process that is about to be discovered, a specific representation language should be picked along with a type of structure pattern associated with this representation language.

**Approach for addressing the challenge**

The solution of this challenge with the use of neural networks relies on the observation that certain modeling languages can describe with more accuracy certain types/topologies of processes (discovered from event logs).
and should thus be preferred over others. Apart from the modeling language chosen to depict a process every time, a decision regarding the patterns chosen for the modeling task should also be made.

The representational bias used during the process discovery can be dependent each time on the type and complexity of the to-be discovered process. In order to pick the optimal representation language and the best structure pattern, a mapping should be present between type of the modeled process, ideal representational language and ideal pattern choice for each one of them. Feed forward NNs are capable of learning in a supervised manner how to map input patterns (type of modeled process in this case) with the output ones (the ideal representational language) (Zhang, 2000).

**Step 1 – Creating the event log**
The first step includes the creation of the event log used in the next step for training and testing of the feed forward NN. Each trace of the event log will consist of a number of different events and each event will correspond to a different feature. These features can lead to the characterization of the type of the modeled process each time. Some of those factors could include:

- Structure of the whole model
- Size of the whole model
- Number of unique events
- Number of unique paths
- Number of events per case
- Number of events recorded
- Extent of variability
- Level of event abstraction

The labeling of this event log can be done in a manual way, denoting this way for each trace the ideal representational language:

- 0 for BPMN (Business Process Model and Notation) notation
- 1 for BPEL (Business Process Execution Language) notation
- 2 for PetriNet notation
- 3 for YAWL (Yet Another Workflow Language) notation

**Step 2 – Log encoding into binary format**
This step includes the creation of the training and testing dataset for the NN, based on the features defined in the first step. It also includes the manual labeling of those traces as suitable or not for being used in process mining. The manual labeling of the traces will lead to the creation of a labeled dataset, used later on for the supervised learning of the feed forward NN.

**Step 3 – Construction of an NN that will perform the classification task**
Here a feed forward NN should be constructed that will perform the matching process between each log and the representation language that should be used for the process that is currently being discovered.
**Step 4 – Train and test the NN using the constructed NN**
Training and testing the constructed NN will provide with an indication of how well the NN is able to learn the mapping between type of modeled process and ideal representational language for each log.

**Step 5 – Classification evaluation**
After some logs (set of traces referring to a process model) have been used for training and others for testing, the NN’s classification accuracy (regarding the optimal representation) can be determined both for the training data (acc) and for the testing data (val_acc). Also the logs for which a correct representation was chosen and others that a wrong one was picked can be identified. *Figure 3-5* schematically depicts the solution of this challenge.

![Diagram](image)

*Figure 3-5* Representational bias improvement with the use of NNs

1 – Construct the labeled event log
2 – Encoding of the traces present in the log in vectors of binary data
3 – Construct feed forward NN
4 – Train and test NN with the labeled dataset
5 – Pick the correct representation for every event log
3.6. **Challenge 6: Balancing between quality criteria such as fitness, simplicity, precision and Generalization**

The sixth challenge is about balancing between quality criteria such as fitness, simplicity, precision and generalization.

**Challenge summarization**

Event logs are in a lot of cases incomplete and only example behavior is given. Process models allow for an infinite number of different traces and some traces may have a much lower probability than others. Those less frequent traces can be considered as noise. It is not possible to build a reasonable model for such noisy behaviors. Taking this as granted, it is not realistic to assume that every possible trace is present in the log. Furthermore, even if there are millions of traces in a log, it is almost certain that not all of the possible variations are present. Noise and incompleteness make process discovery a challenging problem. There are four competing quality dimensions:

- **Fitness:** A model with good fitness allows for most of the behavior seen in the event log and a perfect one if all traces in the log can be replayed by the model from beginning to end.
- **Simplicity:** The simplest model that can explain the behavior seen in the log is the best model.
- **Precision:** A model is precise if it does not allow for “too much” behavior. A model that is not precise is “under fitting”. Under fitting is the problem that the model overgeneralizes the example behavior in the log.
- **Generalization:** Ideally, a model should generalize and not restrict its behavior to just the examples seen in the log. A model that does not generalize is “over fitting”. Over fitting is the problem that a very specific model is generated whereas it is obvious that the log only holds example behavior (e.g. the model explains the particular sample log, but a next sample log of the same process may produce a completely different process model) (Van Der Aalst, 2011).

**Analysis of problem**

Balancing fitness, simplicity precision and generalization is challenging and this is the reason that most of the more powerful process discovery techniques provide various parameters. Improved algorithms need to be developed to better balance the four competing quality dimensions and any parameters used should be understandable by end-users (Van Der Aalst, 2011). This challenge can be seen as a Pareto optimization problem\textsuperscript{15}, in the sense that a balance needs to be achieved amongst all the four competing process model quality criteria (fitness, simplicity, precision and generalization). Neural networks are capable of solving this type of optimization problems and are therefore capable of providing a solution to this challenge (Juan Carlos Fernández Caballero, 2010).

**Approach for addressing the challenge**

The four competing objectives in this Multi-Objective Optimization Problem (MOP) which should be satisfied at the same time are:

- Maximize fitness
- Minimize simplicity

\textsuperscript{15} https://en.wikipedia.org/wiki/Multi-objective_optimization
Maximize precision
Maximize generalization

In this challenge no single solution exists that simultaneously fully satisfies each objective. In this case there exist a number of different Pareto optimal solutions, with each one satisfying each of the objectives to a different extent. A solution is called non dominated Pareto optimal if none of the objective functions\(^\text{16}\) can be improved in value without degrading some of the other objective values. In this case the objective functions are said to be conflicting. In a Pareto challenge, optimal decisions need to be taken in the presence of trade-offs between two or more conflicting objectives. Without additional subjective preference information, all of the Pareto optimal solutions are equally good. Looking at a MOP from different viewpoints proves that there exist different solution philosophies and goals when setting and solving them. After all of the Pareto optimal solutions provided with the use of a feed forward NN, the user based on his needs can pick one of those optimal solutions that balance all of the four quality criteria.

**Step 1 – Genetic algorithm outputs a set of parameters for the NN**

The whole process of defining a set of Pareto optimal solutions starts with a genetic algorithm that outputs a set of neural network parameters. Those parameters can include the number of the to-be constructed feed forward fully connected NN:

- Layers
- Nodes
- Inputs
- Outputs

Each of those set of parameters represents the construction of a different NN topology, which is about to calculate a different value for all of the four quality criteria that are about to be balanced (Juan Carlos Fernández Caballero, 2010).

**Step 2 – Construct NN and calculate values for process model quality criteria**

This set includes the calculation that the constructed NN will perform, for each set of parameters, of a different value regarding each one of the four process modeling quality criteria (Juan Carlos Fernández Caballero, 2010).

**Step 3 – Plot the values of two quality dimensions for all the set of parameters**

Out of the four quality criteria, two will be chosen every time and their values (calculated in step three from each constructed NN for each set of parameters) will be plotted in the (X, Y) state space.

---

\(^{16}\) An **objective function** is an equation to be optimized given certain constraints and with variables that need to be maximized or minimized using nonlinear programming techniques. **Nonlinear programming** is the process of solving an optimization problem defined by a system of equalities and inequalities, collectively termed constraints over a set of unknown real variables, along with an objective function to be maximized or minimized, where some of the constraints or the objective functions are nonlinear.
**Step 4 – Feed the plotted values back to the genetic algorithm**

Plotting a set of values each time will produce a point in the (X, Y) state space. The value of this point will be then fed back to the genetic algorithm and the whole process included in the aforementioned four steps will be executed recursively for a number of times until optimal values are achieved for each set of parameters (Juan Carlos Fernández Caballero, 2010). The solution of this challenge is schematically seen in Figure 3-6.

**Step 5 – Parameters are passed to a mining algorithm**

The set of parameters that are produced by the NN in its output should be passed to a mining algorithm and the algorithm will discover a model for which fitness, simplicity, precision and generalization are computed (taking into account both the constructed model and the log) and these are the values for which the Pareto front gets defined.

---

**Figure 3-6** Balancing between process modeling quality criteria using NNs
3.7. **Challenge 7: Cross-organizational mining**

The seventh process mining challenge refers to the cross-organizational mining.

**Challenge summarization**

Normally, process mining is applied within the boundaries of a single organization, however there are scenarios where the event logs of multiple organizations are available for analysis (e.g. cases where service technology, supply-chain integration and cloud computing become more widespread). There are two main settings for cross-organizational process mining:

- The first one is the collaborative setting where different organizations work together in order to handle process instances e.g. in a case where the overall process is cut into parts and is distributed over organizations that need to cooperate to successfully complete cases. Analyzing the event log within one of these organizations involved is not considered to be sufficient. In order to discover end to end processes, the event logs of different organizations need to be merged. This is not a trivial task, since events need to be correlated across organizational boundaries.

- The second one involves considering the setting where different organizations are executing the same process while sharing experiences, knowledge, or a common infrastructure. Obviously it is interesting to analyze such variations among different organizations, since these organizations can learn from each other. Service providers can improve their services and offer higher quality of services, based on the results of cross-organizational mining.

New analysis techniques need to be developed for both types of cross-organizational process mining and these techniques should also take into account privacy and security issues. Organizations might be unwilling to share information for competitive reasons or due to lack of trust. Therefore it is important to develop privacy preserving process mining techniques (Van Der Aalst, 2011).

**Analysis of problem**

The solution of this challenge comprises the solution of the two aforementioned sub challenges, namely the case where different organizations work together in order to handle process instances and the case in which different organizations execute the same process, while sharing experience and knowledge.

- As far as the case that different organizations work together in order to handle process instances is concerned, this actually implies that event logs that belong to different organizations need to be merged. This challenge can be categorized as a **merging of data problem** and therefore NNs can’t contribute to its solution. In order to solve this challenge, a number of rules should be created which when applied, will lead to merging data that originate from different sources. Those rules can be created with the use of dynamic programing\(^\text{17}\).

- In the case that different organizations execute the same process, it is implied that the variations within each company need to be analyzed to successfully deal with this challenge. This can’t be seen as a classification or prediction challenge and thus NNs are unable to provide a solution to this challenge. In fact

\(^{17}\) Dynamic programing is a method for solving a complex problem by breaking it down into a collection of simpler sub problems, solving each of those sub problems and storing their solutions (ideally using a memory based data structure).
3.8. Challenge 8: Providing operational support

The eighth process mining challenge includes the process of providing operational support.

**Challenge summarization**

In the beginning, process mining mainly aimed at analyzing historical data (events that happened in the past). Today, however, many data sources are updated in near real-time fashion and sufficient computing power is available in order to analyze events at the time they occur. Therefore, process mining should not be restricted only to off-line analysis but can also be used for online operational support. Three main operational support activities can be identified:

1. **Detect**: The exact moment that a case shows a deviation from the predefined process, this deviation can be detected and subsequently the system can generate an alert. In an ideal situation, those notifications should be created immediately so that things can be influenced in an online fashion (the exact moment they occur).

2. **Predict**: Historical data can be used in order to build predictive models. These can be used to guide already running process instances, based on data related to the past that have been retrieved and subsequently analyzed. As an example, it is possible to predict the remaining process time of a case, when taking into account historical data.

3. **Recommend**: Based on the aforementioned predictions (by making use of historical data), recommender systems can be built that propose particular actions to reduce costs or shorten the flow time.

Applying process mining techniques in such an online setting creates additional challenges in terms of computing power and data quality. The challenge of providing operational support consists of three sub problems (detect a deviation, predict and recommend the next event) all in terms of a process model and can be dealt with, by solving each one of the aforementioned operational support activities separately (Van Der Aalst, 2011).

**Approach for addressing the challenge**

Each of the three sub-challenges that are comprised within challenge 8 can be dealt with, by using an approach for each one, depending on the type of problem each one represents. Below, a detailed problem analysis is included for each of the three sub challenges along with a solution for it.

1. **Providing operational support for the detection of cases that deviate from the predefined process**

   **Analysis of problem**
   
The process of providing operational support for cases that deviate can be seen as a prediction challenge. This stands in the sense that if a neural network could be constructed, that is able to predict (based on historical data related to the process) the next event that should be present in a process model, this could lead to the detection
of cases that deviate from the predefined process. This would require the creation of a NN model able to make predictions regarding the next activity that should be executed in a process model. NNs are capable of predicting (based on information related to the history of a case) the next event that will be executed in a process model.

Step 1 - Initial event log
The whole task of detecting cases that deviate starts with a given event log that is derived from a business process model. Alternatively, a log related to a process model could be provided and in this case the process model should be discovered. This log may consist of a number of different traces and each trace of a number of different events.

Step 2 - Log encoding into feature vectors
In this step the explanation of how a training set can be created for the NN constructed in the next step is presented. The goal is to detect a case that deviates from a predefined process and its timestamp and this can be done by learning an activity prediction function and a time prediction function. Each trace within the event log is turned into a feature vector, which is later going to be used as input to the constructed LSTM NN (training set). The feature vectors are constructed initially with a number of features that represent the type of activity of an event using one-hot encoding. One-hot encoding assigns the value “1” to a specific feature and “0” to all the rest. Three time-based features are also added to the one-hot encoding feature vector. The first time based feature of a given event is the time between the previous event in the trace and the current event. This feature enables the LSTM to learn the dependencies between the time differences at different points in the process. The second time feature is the time that the current event is taking place and the third one is the time between the current event and the end of the process. One extra element to the one hot encoding vector is added, which has the value 1 in the case that the current event of the trace is not the last one within the trace. Knowing the timestamp of the current event, makes it possible to calculate the timestamp of the following one. The “Adam” learning algorithm is used to optimize the weights in the network, such that the cross-entropy between the ground truth one-hot encoding of the next event and the predicted one-hot encoding of the next event are minimized (Tax, 2016).

Step 3 – Construct LSTM NN
This step includes the construction of an NN that may contain a number of LSTM layers and will perform the task of detecting cases that deviate from the predefined process. This LSTM NN can have different architectures for modeling the different activity prediction and time prediction function as seen in Figure 3-1. (Tax, 2016).

Step 4 – Training of LSTM NN
The training is performed by traversing each trace of the log from beginning to end and by making predictions for the next to be executed activity and its timestamp, with the feedback loop of the LSTM NN working as a feedback loop. Modeling the next activity prediction function and time prediction function with LSTMs can be done using three different architectures. Two separate models can be trained (one for the activity prediction function and one for the time prediction function), both using the same input features in each time step. Alternatively, both the time prediction function and activity prediction function can be learned jointly in a single LSTM model that generates two outputs in a multi-task learning setting. Finally a hybrid option between the two aforementioned architectures can be used. This can be done, by creating an architecture that has a number of
shared LSTM layers for both tasks, followed by a number of layers that specialize in either prediction of the next activity, or prediction of the time until the next event (Tax, 2016).

**Step 5 – Testing of LSTM NN**
The testing of the NN is performed in the parts of each trace that a case deviates from the predefined process. The NN makes predictions about possible continuations of each partial trace and about the timestamp of each deviating event.

**Step 6 - LSTM NN will output several possible continuations of the partial trace**
The activity prediction function outputs the probability distribution of various possible continuations of the partial trace (which includes deviating cases) (Tax, 2016).

**Step 7 – Most likely continuation picked.**
The most likely continuation is going to be picked and in the case that it deviates from the predefined process, it is an indication that there is a possible deviation from the predefined process and therefore a deviating case (Tax, 2016). This means that the most possible continuation of every specific trace is going to be compared with the actual continuation of the trace seen in the process model and this way possible deviations are going to be identified based on the LSTM trace predictions (Tax, 2016). The solution of this challenge is schematically seen in *Figure 3-7*. 

![Figure 3-7 Providing operational support for detection tasks using NNs](image-url)
2. Providing operational support for prediction tasks (based on observation of historical data)

Analysis of problem
As far as the prediction of the next task that is about to be executed in terms of a process model is concerned, a way of discovering it should be invented. Put it otherwise, a technique should be developed, which will be able to predict the next activity of an unfinished process model. This challenge can be considered as a prediction problem of the next to-be-executed event in an event log with missing elements. This prediction should be based on the information known about the history of the process and by making use of it. Long Short-Term Memory (LSTM) neural networks are able of predicting the next event of a running case and also the full continuation of it (Tax, 2016). What is more, they can perform this task by outperforming other existing tailor-made methods. The steps need to be followed in order to provide a solution to this challenge are similar to those followed for the solution of the deviation detection challenge (3.8.1). They can be seen in short in Figure 3-8.

---

Figure 3-8 Providing operational support for prediction tasks using NNs
3. Providing operational support for recommendation tasks

Analysis of problem

The problem of “recommendation” of the next activity that is about to be executed in an event log is in close relation to the one of “prediction”. Put it otherwise, this challenge can be considered as a prediction challenge, based on events executed in the past of the process (discovered from an event log), thus providing a recommendation for the next event that is about to be executed (events that could be executed in the future). This prediction will be based on data related to the history of the process, meaning that LSTM NNs can be applied here in a similar way as for the “prediction” task. This is because recommendation of the next activity executed in a process model can be seen as prediction of a missing activity in a process model. The steps need to be followed in order to provide a solution to this challenge is similar to those followed for the solution of the deviation detection challenge (3.8.1). They can be seen in short in Figure 3-9.

Figure 3-9 Providing operational support for recommendation tasks using NNs
3.9. Challenge 9: Combining process mining with other types of analysis

The ninth process mining challenge includes the combination of process mining with other types of analysis, such as data mining, operations management and visual analytics.

Challenge summarization

Process mining can be combined with various analysis techniques such as:

1. **Data mining**: is the analysis of often large data sets in order to find unsuspected relationships and to summarize the data in ways that make them both understandable and useful for the data owner. There are a lot of data mining tools that are used to support business decisions in specific areas (e.g. which products should be placed together in the supermarket).

2. **Operations management**: is a branch of management science heavily relying on modeling. A variety of mathematical models ranging from linear programming and project planning to queueing models, Markov chains and simulation are used. It is an area of management concerned with designing and controlling the processes of production and redesigning business operations in the productions of high quality services.

3. **Visual analytics**: It combines automated analysis with interactive visualizations for a better understanding of large and complex data sets. Visual analytics exploits the capabilities of humans to see patterns in unstructured data.

In the hypothetical example of executing a simulation, process mining techniques can be used to learn a simulation model based on historical data (data related to the past). Subsequently the simulation model can be used to provide operational support. Because of the close connection between event log and model, the model can be used to replay history and one can start simulations from the current state thus providing a “fast forward button” into the future based on live data. Data mining, operations management and visual analytics provide valuable analysis techniques. The challenge is to combine the techniques in these fields with process mining (Van Der Aalst, 2011).

Analysis of problem

Here, an analysis of how process mining can be combined with data mining, operations management and visual analytics takes place. In the end a conclusion has to be drawn for whether this combination can be performed with the use of neural networks.

1. **Data mining connection to process mining**

Process mining combines the strengths of both data mining and process modeling, by automatically creating process models based on existing IT log data. Process mining yields live models that are connected to the business and can easily be updated at any point in time. Both process mining and data mining process large amount of data that can’t be evaluated by hand. Unlike data mining, process mining focuses on the process perspective. It includes the temporal aspect and looks at a single process execution as a sequence of activities that have been performed. Most of the data mining techniques extract abstract patterns in the form of, for example, rules or decision trees. On the other hand, process mining creates complete process models and then uses them to highlight possible bottlenecks or wrong deviations. Data mining uses the so called “generalization” in order to avoid over fitting the data. So all the examples that do not match the general rule are removed. In process mining, generalization is also important in order to deal with complex processes and understand the
main process flows. However, understanding the exceptions is important to discover inefficiencies and points of improvement. In data mining, models are often trained to make predictions about future similar instances in the same space. Today’s processes are so complex, that accurate predictions are often not realistic. The gained knowledge from the discovered patterns and processes help to deal with the complexity, where the true value lies\(^\text{18}\).

2. **Operations management connection to process mining**
   Process mining can be combined with operations management since both of them heavily rely on modeling. Put it otherwise there is a connection between both of them. Operations management on the one hand is heavily dependent on modeling, since it is considered to be an area of management concerned with designing and controlling production processes. Process mining on the other hand is a combination of process modeling used in conjunction with data mining in order to extract meaningful conclusions out of the data.

3. **Visual analytics connection to process mining**
   Process mining can be combined with visual analytics. With the combination of automated process mining techniques with interactive visual analytics, it is possible to extract more insights from event data, since visual representation of process mining results make it easier for the user to draw conclusions.

**Approach for addressing the challenge**
The challenge of combining process mining with data mining, operations management and visual analytics can’t be solved with the use of neural networks. This is because, despite the fact that there is a direct connection of all of the three types of analysis with process mining, no neural network can be trained to perform this connection task. The task of combining process mining with the other three analysis techniques can’t be seen as classification, clustering, or prediction for which NNs can provide a solution, thus giving more value to the previous argument. The connection procedure can only be performed with the use of programming, or otherwise a human could combine the results of two methods together each time, in order to draw meaningful conclusions. All of the above arguments are valid for the current state of knowledge, as in the future this condition might change.

Chapter 4  Outlier Classification of traces discovered from a Process Model’s event log

In this chapter the way that the outlier classification challenge can be solved with the use of deep NNs is presented. The solution includes a schematic representation, along with some more details compared to the solution of the rest of the process mining challenges (since the solution of this challenge is going to take place in chapter 5). Put it otherwise, the steps that need to be followed for any given event log in order to perform the outlier classification task are described in detail.

Challenge summarization
Outliers can be seen as unusual behavior and can also be referred as noise. When seen in the context of process modeling, they can be considered as anomalous data points that can significantly degrade the outcome of an otherwise reasonable model identification procedure (Pearson, 2002). Put it otherwise, outliers can be seen as data points in a data set that are inconsistent with our expectations based on the volume of the available data. They occur frequently in practice and can have serious consequences. The problem of outliers is in close relation with the one of missing data. One reason that can lead to this conclusion is that outliers can arise from inappropriate treatment of missing data. Furthermore, one way to treat outliers is rejection, thus creating missing data values (since outliers are removed from the event log, resulting in a log of reduced size).

In terms of process modeling, outliers can refer to:

- a whole trace of a process
- a single event that is part of a trace
- a particular attribute that belongs to an event

The procedure of identifying all of the above types of outliers is considered to be a binary classification problem, since for every trace/event/attribute of an event log, an indication should be provided to the user in order to indicate the presence or absence of outliers in a process model’s event log (Van Der Aalst, 2011).

Analysis of problem
Neural networks have the ability to be trained given enough training data, the ability to generalize and this ability that they possess makes them capable of solving a wide variety of machine learning problems. Outlier
detection can be considered as a classification problem (in the sense that every provided trace is either an outlier or not) and neural networks are actually capable of solving this (Zhang, 2000).

**Approach for addressing the challenge**

Neural network’s ability to generalize means that when trained, they can perform tasks (in this case the outlier classification task) with the same accuracy as the one they achieved during the training phase on the training dataset. So a trained neural network, when provided in its input with an event log belonging to a process model will be able to classify its traces/events/attributes as being outliers or not.19

The solution of this challenge using neural networks consists of six steps, which when implemented sequentially can provide the user with an indication about the existence or not of outliers in a given process model’s event log.

### 4.1. Given event log

The whole outlier classification task starts with a given event log. One such log, containing the sample events “a, b, c” can be seen in Table 4-1. This log may consist of a number of different traces and each trace of a number of different events. Purpose of this task is to identify which of those traces are outliers and which not.

<table>
<thead>
<tr>
<th>Trace ID</th>
<th>Events of each trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a, b, c</td>
</tr>
<tr>
<td>1</td>
<td>b, c, a</td>
</tr>
<tr>
<td>2</td>
<td>c, b, a</td>
</tr>
<tr>
<td>3</td>
<td>a, c, b</td>
</tr>
<tr>
<td>4</td>
<td>b, a, c</td>
</tr>
<tr>
<td>5</td>
<td>c, a, b</td>
</tr>
</tbody>
</table>

Table 4-1 Initial set of traces

### 4.2. Log encoding into tabular data and labeling of traces

This step includes the translation of the information present in the event log into a format that can be used by an artificial neural network (binary format) for training. Before translation, the log has a string format, since it is derived from a process model which contains activities. The type of information present in an event log can’t be given directly as an input to a neural network, since neural networks accept only a specific type of information (vector of numbers) and the event log contains information in string format. This means that some form of translation to the elements of the event log should be applied in order to turn them into vectors of numbers.

that can actually be used for training and testing from the neural network that will be constructed in the second step of this process\textsuperscript{20,21}.

Table 4-2 Translation of an event log to vector of numbers

<table>
<thead>
<tr>
<th>Initial log</th>
<th>Translated log</th>
<th>Labeled log</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b, c</td>
<td>110001, 110010, 110011</td>
<td>1</td>
</tr>
<tr>
<td>b, c, a</td>
<td>110010, 110011, 110001</td>
<td>1</td>
</tr>
<tr>
<td>c, b, a</td>
<td>110001, 110010, 110001</td>
<td>0</td>
</tr>
<tr>
<td>a, c, b</td>
<td>110010, 110001, 110010</td>
<td>1</td>
</tr>
<tr>
<td>b, a, c</td>
<td>110001, 110011, 110011</td>
<td>0</td>
</tr>
<tr>
<td>c, a, b</td>
<td>110011, 110001, 110001</td>
<td>0</td>
</tr>
</tbody>
</table>

Each input vector represents a trace of the process model discovered from the initial log. Some traces have more activities than others, so as point of reference each time is considered the trace of the log with the most activities. Traces that have fewer activities are filled in with zero values (in order to reach the length of the longest trace). Table 4-2 depicts the whole process taking place in step 2. This step also includes the labeling of the event log traces as being outliers or not. This way, the feed forward NN constructed in the next step can be trained in a supervised manner.

4.3. **Construction of deep feed forward NN**

Feed forward neural networks are capable of dealing with binary classification challenges and classifying the provided dataset with extremely high accuracy (Goodfellow, 2016). Those types of networks can be trained in a supervised manner, using a labeled training dataset and back propagation of error, so as to be able to classify new unseen instances when trained. The nature of the given problem (binary classification) in combination with the existence of a labeled training dataset, lead me to the conclusion that a feed forward neural network would be ideal to successfully deal with the outlier classification challenge.

The number of inputs of such a network should be the same as the length of the largest trace of the dataset and one output should be sufficient to distinguish outlier traces from normal ones. The number of hidden layers should be defined after a process of experimentation in order to reach to a number that will provide the highest classification accuracy possible, but at the same time will not lead to overtraining of the network. There are various rules for defining the number of hidden layers, but since they are generic and don’t apply to every case,

\textsuperscript{20} http://datascience.stackexchange.com/questions/869/neural-network-parse-string-data
\textsuperscript{21} http://stackoverflow.com/questions/4674623/why-do-we-have-to-normalize-the-input-for-an-artificial-neural-network
the number of hidden layers is be defined here after an experimentation process. Goal of this process is to lead to a network structure that will provide the highest possible classification accuracy for all of the traces.

Later in the practical outlier classification part of this project (chapter 5) these parameters will be explicitly mentioned and their choice will be justified.

4.4. **Training of the constructed NN based on the created dataset**

After the construction of the network is complete, the training phase takes place. A labeled training dataset should be constructed, using a subset of the elements present in the event log and the rest of the elements should be used for testing, evaluating in this way the classification accuracy of the provided dataset of the constructed network model. Again in the implementation part of this project, the way that the training and test set are going to be labeled will be explicitly stated. The training and the test data set ideally should not have any elements in common, as this would compromise the accuracy of the whole classification process.

4.5. **Testing of the classification accuracy of the trained NN**

After the training phase is complete on a subset of the whole dataset (training dataset), another part of the event log (test dataset) should be provided into the same network and the prediction it is going to provide in its output should be compared to the label of each element present in the log. Based on the number of correct predictions the network has made out of all the predictions it actually conducted, can lead us to calculate the classification accuracy of the neural network in this specific dataset. In Figure 4-1 the process related to the third
fourth and fifth step of dealing with the outlier classification challenge is depicted. The above diagram is executed clock wise and its execution ends when a given data set is being classified with satisfyingly high accuracy by a trained NN. In the case that an occurring classification does not have the expected accuracy the above cycle is being executed time after time until the resulting classification provides satisfying results.

4.6. **Compare initial classification with the one the NN performed**

After the training is complete and the accuracy of the classification task has been measured, the outliers that exist in a specific event log each time will be determined, with accuracy as high as the accuracy of the classification task is. In *Table 4-3*, outliers discovered with the use of a trained neural network are marked. Each trace is being classified as being either an outlier (traces with 1 in the classification column) or not (traces with 0 in the classification column).

**Table 4-3 Outlier discovery performed by the NN**

<table>
<thead>
<tr>
<th>Binary encoded traces</th>
<th>Manual classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100001, 1100010, 1100011</td>
<td>1</td>
</tr>
<tr>
<td>1100010, 1100011, 1100001</td>
<td>1</td>
</tr>
<tr>
<td>1100011, 1100010, 1100001</td>
<td>0</td>
</tr>
<tr>
<td>1100001, 1100011, 1100010</td>
<td>1</td>
</tr>
<tr>
<td>1100010, 1100001, 1100011</td>
<td>0</td>
</tr>
<tr>
<td>1100011, 1100001, 1100010</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary encoded traces</th>
<th>NN classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100001, 1100010, 1100011</td>
<td>1</td>
</tr>
<tr>
<td>1100010, 1100011, 1100001</td>
<td>1</td>
</tr>
<tr>
<td>1100011, 1100010, 1100001</td>
<td>0</td>
</tr>
<tr>
<td>1100001, 1100011, 1100010</td>
<td>0</td>
</tr>
<tr>
<td>1100010, 1100001, 1100011</td>
<td>0</td>
</tr>
<tr>
<td>1100011, 1100001, 1100010</td>
<td>0</td>
</tr>
</tbody>
</table>

Manually classified Log  
Log classified by Neural network

At the end of the above six steps, the traces that where classified correctly by the trained NN along with those that where misclassified can be defined.

**Table 4-4 Discovered outliers in the Initial log**

<table>
<thead>
<tr>
<th>Encoded traces</th>
<th>Manually labeled</th>
<th>Mistakes in NN classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100001, 1100010, 1100011</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1100010, 1100011, 1100001</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1100011, 1100010, 1100001</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1100001, 1100011, 1100010</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1100010, 1100001, 1100011</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1100011, 1100001, 1100010</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initial traces</th>
<th>Manually labeled</th>
<th>Mistakes in NN classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b, c</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b, c, a</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c, b, a</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a, c, b</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b, a, c</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c, a, b</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Identify mistakes in NN classification  
Initial traces misclassified
In Figure 4-2 seen below, all of the six steps included in the solution of the outlier classification challenge can be schematically seen.

1 - Event log

2 - Encode log and manually label its traces as outliers or not

3 - Construct feed forward NN

4 - Train NN using part of the labeled event log

5 - Test the trained NN on the rest of the traces of the event log

6 - Evaluate classification accuracy

Figure 4-2 Outlier classification of process model traces using NNs
Chapter 5  Experiment Setup of “Sepsis case”

Here all the experiments performed in order to verify the accuracy of the outlier classification method are being described in detail, along with the context (experiment setup) within which they are performed.

5.1. Description of the used event log

Here the detailed description of how outlier classification with the use of neural networks can work in a real life event log is described. The classification of the outlier traces in this chapter is performed in the “Sepsis case” event log (4TU).

5.1.1. Event Log used for outlier detection

The real-life event log that is being used here in order to perform the outlier classification task is one that has to do with the process of sepsis cases from a hospital. It consists out of 1050 traces (each trace is linked to a different sepsis case) and of 15000 discrete events. One case represents the pathway through the hospital. Every trace consists of a number of events that ranges from 3 to 185 and the average number of events per trace is 14.49.

5.1.2. Process model derived from the event log

The outlier classification procedure can be implemented in a process model derived from the given event log and not in the event log itself. For this reason from the given log a model should be discovered, accordingly all the different traces should be identified (both normal and outliers) and a way should be found to classify the outlier traces. With the use of the “inductive visual miner” algorithm, a model is discovered from the provided event log. This model allows for most of the traces that are present in the log, depending on its level of fitness. The discovered model can be slightly modified so that some more or less events of a trace that are present in the log are not included in the model. These slightly modified models (compared to the one discovered by the inductive visual miner) can be created using YASPER\(^{22}\), which is a process modeling software. The created model is in .pnml format, which can be directly imported to ProM so as to be checked for conformance with the given event log. After conformance analysis has been performed between the created process model and the event log it has been discovered from it can be seen that some events belonging to a trace are only present in the log and not in the model. Those events are considered to be outliers and all the rest as normal ones. The goal is to

\(^{22}\) http://www.yasper.org/
find a model that when trained will be capable of classifying all of those traces that contain events which are only present in the log and n
5.2. Mapping model activities into binary convertible variables

Here the way that event log data are turned into tabular data is described.

Log encoding
The first step of turning event data into tabular data includes turning each activity present in the event log into a character following mapping seen in Table 5-1.

Table 5-1 Mapping of process events into letters

<table>
<thead>
<tr>
<th>Event name</th>
<th>Letter</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER Registration</td>
<td>A</td>
</tr>
<tr>
<td>ER Triage</td>
<td>B</td>
</tr>
<tr>
<td>IV Liquid</td>
<td>C</td>
</tr>
<tr>
<td>ER Sepsis Triage</td>
<td>D</td>
</tr>
<tr>
<td>CRP</td>
<td>E</td>
</tr>
<tr>
<td>Admission IC</td>
<td>F</td>
</tr>
<tr>
<td>IV antibiotics</td>
<td>G</td>
</tr>
<tr>
<td>Leucocytes</td>
<td>H</td>
</tr>
<tr>
<td>Lactic Acid</td>
<td>I</td>
</tr>
<tr>
<td>Admission NC</td>
<td>J</td>
</tr>
<tr>
<td>Release B</td>
<td>K</td>
</tr>
<tr>
<td>Release A</td>
<td>L</td>
</tr>
<tr>
<td>Return ER</td>
<td>M</td>
</tr>
<tr>
<td>Release E</td>
<td>N</td>
</tr>
<tr>
<td>Release C</td>
<td>O</td>
</tr>
<tr>
<td>Release D</td>
<td>P</td>
</tr>
</tbody>
</table>

This is done, because since only vectors of numbers can be provided to a neural network as inputs, a way should be found in order to turn each trace of the process model into a vector of numbers. Each input vector represents a trace of the process model discovered from the initial log. Some traces have more activities than others, so as point of reference each time is considered the trace of the log with the most activities. Traces that have fewer activities are filled in with zero values (in order to reach the length of the longest trace). So by mapping the components of each trace (activities) into letters and then by turning those letters into numbers (with the use of the built-in python function ord()), the desired vectors of numbers that will be fed into the neural network can be constructed.

Data labeling
Event data exported from prom are in .csv format and for each event they include the name of the event, its case id and the where this event is present (event log, process model, or at both):

<table>
<thead>
<tr>
<th>case id</th>
<th>activity</th>
<th>move type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKA</td>
<td>A</td>
<td>sync</td>
</tr>
<tr>
<td>AKA</td>
<td>B</td>
<td>sync</td>
</tr>
<tr>
<td>AKA</td>
<td>tr6</td>
<td>invisible</td>
</tr>
<tr>
<td>AKA</td>
<td>tr7</td>
<td>invisible</td>
</tr>
</tbody>
</table>

Figure 5-2 Prom output format
The first step of data manipulation includes sorting events by case id:

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>['Async', 'Bsync', 'Dsync', 'Hsync', 'Isync', 'Esync', 'Csync', 'Gsync']</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AAA</td>
<td>['Async', 'Bsync', 'Dsync', 'Csync', 'Hsync', 'Isync', 'Esync', 'Gsync', 'Jsync', 'Lsync', 'Msync']</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AB</td>
<td>['Async', 'Bsync', 'Dsync', 'Esync', 'Isync', 'Hsync', 'Csync', 'Gsync']</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

By a convention made by me (in order to find a way to separate outlier data from the rest), event log data that are present in the initial event log but not in the discovered process model are considered to be outlier events, and the traces that they are present in as outlier traces. After those outlier traces are identified, they are automatically assigned with a label of one (1) in the column that is present next to the last event of each trace, and those that are not outlier traces get a label of zero (0). This way, the constructed dataset containing all of the 1050 traces of the log, which are sorted based on their case id becomes labeled and can be used by the neural network constructed in the next step for training and testing.

In order to improve the outlier classification accuracy of the constructed neural network and take it as high as possible, the data present on the 2d array that is used for training and testing (Figure 5-4) should be normalized before they are fed into the neural network.

For this reason, the scikit-learn machine learning library of python was used to perform the standardization of the constructed dataset using the StandardScaler class. Instead of performing the standardization procedure on the entire dataset, it is beneficial to train the standardization procedure on the training data with a pass of a cross validation run and to use the trained standardization instance to prepare the unseen test fold. This makes standardization a step in model preparation in the cross validation process and prevents the algorithm of having knowledge of unseen data during evaluation. This can be achieved by using a Pipeline class. The pipeline can be considered as a wrapper that executes one or more models within a pass of the cross validation procedure. The pipeline is defined with the StandardScaler followed by the neural network model.

The complete code written in python, which is responsible for converting the exported in .csv format data into labeled tabular data, that can be used by the neural network constructed in chapter 5.6 for its training and testing is available in Appendix F.
5.3. Neural network construction and training (Using manual verification dataset)

Here the description of the construction of a neural network that will perform the classification of the dataset created in chapter 5.2, using a manual verification dataset (66% for training and 33% for testing) takes place. The construction consists of the steps mentioned below.

**Step 1 – Load data**

In this part, all the python libraries about to be used are imported. Whenever working with machine learning algorithms that use a stochastic process (e.g. random numbers), it is a good idea to initialize the random number generator with a fixed seed value. This is so that when running the same code again and again the same result will be provided. The random number generator can be initialized with any seed, and here the number “7” is chosen.

StandardScaler() can be used to standardize features by removing the mean and scaling to unit variance.

Now the training and testing dataset can be loaded. This is done with the use of the NumPy function loadtxt(). There are twenty input variables and one output variable (the last column). Once loaded, the dataset is split into input variables (X) and the output class variable (Y). Since the features of every input vector to the neural network is either a number between 65 and 80 either zero this can contribute to non-optimal classification accuracy. For this reason, with the use of the function fit_transform(X) which performs input data normalization higher accuracy results can be achieved.

```python
from keras.models import Sequential
from keras.layers import Dense
from sklearn.cross_validation import train_test_split
import matplotlib.pyplot as plt
import numpy
from sklearn.preprocessing import StandardScaler
import timeit
from keras.callbacks import ModelCheckpoint
from keras.optimizers import SGD
```

**Catalog 5-1 Load libraries**

Here, data is being split into training and testing parts and the input data vector is being normalized. The normalization is being performed with the use of StandardScaler() and is performed for improving the classification accuracy. Keras allows to manually specify the dataset to use for validation during training. Here, the train_test_split() function from the Python scikit-learn machine learning library is being used to separate the data used into a training and test dataset. I use 67% for training and the remaining 33% of the data for validation.

```python
# simulation timing
def start = timeit.default_timer()
# fix random seed for reproducibility
seed = 7
```

---

Catalog 5-2 Data manipulation

Below the values before and after the data normalization can be seen. Accumulating the values of the input vector around zero leads to higher classification accuracy.

```python
numpy.random.seed(seed)
dataset = numpy.loadtxt("ex4.csv", delimiter="","")
# split into input (X) and output (Y) variables
X = dataset[:,0:185]
scaler = StandardScaler()
X = scaler.fit_transform(X)
Y = dataset[:,185]
# split into 67% for train and 33% for test
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=seed)
```

Step 2 – Define model

The model about to perform the classification of traces is being defined here. It consists of two hidden layers (150 and 100 neurons respectively), one input layer of 185 and one output of one neuron. Models in Keras are defined as a sequence of layers. A sequential model is being created by adding layers one at a time until a network topology is being formed. The first thing to get right is to ensure the input layer has the right number of inputs. This can be specified when creating the first layer with the `input_dim` argument and setting it to 20 for the 20 input variables. The number of layers can be defined with the use of heuristics\(^\text{24}\), but the best network structure is found through a process of trial and experimentation. Generally, a network large enough is needed to capture the structure of the problem.

\(^{24}\) See section 2.3.3
Here, a fully connected network with four layers will be used. Fully connected layers are defined using the `Dense` class. The number of neurons in the layer can be specified as the first argument, the initialization method as the second argument as `init` and specify the activation function using the `activation` argument. The network weights are being initialized to a small random number generated from a uniform distribution (uniform), in this case between 0 and 0.05, since that is the default uniform weight initialization in Keras. The rectifier (relu) activation function will be used for the first two layers and the sigmoid in the output layer. Sigmoid and tanh activation functions could also be preferred for all layers, however better performance is being achieved using the rectifier activation function. The sigmoid activation function on the output layer ensures that the network output is between 0 and 1 and easy to map to either a probability of class 1 or snap to a hard classification of either class with a default threshold of 0.5. To complete the whole network, each layer should be added on top of the other, forming this way the whole structure. The first hidden layer has 150 neurons and expects 185 input variables. The second hidden layer has 100 neurons and the finally the output layer has 1 neuron to predict the class (outlier trace or not).

```python
# create model
model = Sequential()
model.add(Dense(185, input_dim=185, init='uniform', activation='relu'))
model.add(Dense(150, init='uniform', activation='relu'))
model.add(Dense(100, init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='sigmoid'))
```

Catalog 5-3 Model definition

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
<th>Connected to</th>
</tr>
</thead>
<tbody>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 185)</td>
<td>34410</td>
<td>dense_input_1[0][0]</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>(None, 150)</td>
<td>27900</td>
<td>dense_1[0][0]</td>
</tr>
<tr>
<td>dense_3 (Dense)</td>
<td>(None, 100)</td>
<td>15100</td>
<td>dense_2[0][0]</td>
</tr>
<tr>
<td>dense_4 (Dense)</td>
<td>(None, 1)</td>
<td>101</td>
<td>dense_3[0][0]</td>
</tr>
<tr>
<td><strong>Total params:</strong></td>
<td></td>
<td>77511</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5-6 Model outline
Step 3 – Compile model and create checkpoint

Now that the model is defined, it can be compiled. Compiling the model uses the efficient numerical library (the so-called backend) Theano. The backend automatically chooses the best way to represent the network for training and making predictions to run on the provided hardware. When compiling, some additional properties must be specified, which are required when training the network. Training the network means finding the best set of weights to make predictions for this problem.

The loss function used to evaluate a set of weights should be specified, along with the optimizer used to search through different weights for the network and any other optional metrics that we would like to collect and report during training. In this case, logarithmic loss will be used, which for a binary classification problem is defined in Keras as `binary_crossentropy`. I will also use the efficient gradient descent algorithm `sgd`. Finally, since this challenge is a classification problem, the classification accuracy will be collected and reported as the metric.

```
# Compile model
sgd = SGD(lr=0.1, momentum=0.9, decay=0.0, nesterov=False)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
```

A checkpoint is created in order to monitor the “val_acc” values, so as to determine the optimal number of epochs that is about to lead to the highest possible classification accuracy (referring to testing data) for the given model configuration. In this case, the optimal number of epochs is 25.

```
# checkpoint
filepath="weights-improvement-{epoch:02d}-{val_acc:.2f}.hdf5"
cHECKPOINT = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]
```

Figure 5-7 Model summary
Step 4 – Fit model
The model has been defined and compiled and now it is time to execute it on the data derived from the provided event log. The model can be trained or fitted on the loaded data by calling the `fit()` function on the model. The training process will run for a fixed number of iterations through the dataset called epochs, that will be specified using the `nb_epoch` argument. The number of instances that are evaluated before a weight is updated in the network can be set using the `batch_size` argument. For this problem the simulation of the network will run for 25 epochs and the batch size used will be 200.

```python
# Fit the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), nb_epoch=26, batch_size=200, callbacks=callbacks_list, verbose=0)
epoch_loss = history.history['loss']
print(history.history.keys())
```

Catalog 5-6 Fit model

Step 5 – Evaluate model
The evaluation part includes all the metrics based on which the model evaluation will take place. Here all of the simulation metrics regarding both the training and testing of the model (accuracy, validation accuracy, loss, validation loss) are being recorded in the history variable, so as to be available to monitor the fluctuation of these values and draw conclusions.

Since the network has been trained on the entire dataset its performance on the same dataset can be evaluated. This will provide an indication of how well the constructed network can classify the provided traces to it. Since the dataset has been split into training and testing parts, both of them are about to be evaluated with the use of the `evaluate()` function. The arguments passed in this function are the same input and output used to train the model. This will create a prediction for each input and output pair and collect scores, including the accuracy and the average loss.

```python
# evaluate the model
scores = model.evaluate(X, Y)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
stop = timeit.default_timer()
print "Simulation lasted for: ", stop - start , "seconds"
```

Catalog 5-7 Evaluate model

Step 6 – Plot statistics
Here the monitoring of accuracy and loss values are being recorded and below plotted.

```python
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')
plt.show()
```

Catalog 5-8 Plot accuracy
The accuracy of a model is determined after the model parameters are learned and no learning process is taking place. Then the test samples are fed to the model and the number of mistakes the model makes is recorded, after comparing the network classification output to the correct target values for each trace. After this, the percentage of misclassified event logs is calculated. In this case the 91.8% of the test traces where correctly classified.

As far as the loss and validation loss values are concerned, the lower they are, the better the model is (unless it has over-fitted to the training data). A model is over fitting the training data when it is very complex, meaning that it has a large number of layers containing many neurons. The number of connections in such a model is enormous (reaching millions) and as a result overfitting occurs. When the accuracy of an overfitting model is evaluated on new data it performs really poor, or put it otherwise, such a model doesn’t generalize well. The neural network model designed here, performs classification of new traces with extremely high accuracy (91.8%) and furthermore it doesn’t contain a large number of layers (4) or neurons (250). These indications lead me to the conclusion that this model does not over fit the training dataset.

The loss values are calculated on both the training and validation set and they provide an indication of how well the model is doing for those two sets. The loss values are not represented in percentage as opposed to the accuracy values and they are the summation of the errors made for each example in training or validation sets. In the case of the classification performed by the aforementioned neural network, the loss is in the form of negative log-likelihood. The main objective here is to minimize the loss function’s value with respect to the
model’s parameters by changing the weight vector values through different optimization methods such as back propagation. A certain loss value implies how well or bad a certain model is behaving after each iteration of optimization. Ideally, loss values should be reduced after each or several iterations.

The “model accuracy” metric, can be considered as the number of traces correctly classified, divided by the total amount of traces. It is given by the formula:

\[ ACC = \frac{TP+TN}{TP+FN+TN+FP} \]

- True positives (TP), which are the traced that are positives and are classified as positives.
- False positives (FP), which are the traces that are negatives and are classified as positives.
- False negatives (FN), which are the traces that are positives and are classified as negatives.
- True negatives (TN), which are the traces that are negatives and are classified as negatives.

These values come from a confusion matrix\(^{25}\). The confusion matrix is an \( mxm \), where \( m \) is the number of classes to be predicted. For binary classification problems, the number of classes is 2, thus the confusion matrix will have 2 rows and two columns.

Using Catalog 5-10 and by looking for non-zero entries in vector \( y \), traces that where misclassified by the NN can be defined both in the training and testing part of the event log. The \( \text{model.predict} (X\_train) \) is compared with \( Y\_train \) and \( \text{model.predict} (X\_test) \) with \( Y\_test \). The predict method is called on the test and training data and its output is compared with the existing labels for each trace.

The number of wrong NN classifications is approximated to 86 out of all 1050 traces. This has as a result a percentage of 91.8% correct classifications made by the NN, which is also the classification accuracy metric outputted as a simulation statistic. The reason that I use \( \text{numpy.round} \) is that the activation function that I use in the NN output is sigmoid. This means that values that are closer to zero are classified as zero and those closer to one are classified as one.

\[ \text{Catalog 5-10 NN misclassified traces} \]

Using Catalog 5-10 and by looking for non-zero entries in vector \( y \), traces that where misclassified by the NN can be defined both in the training and testing part of the event log. The \( \text{model.predict} (X\_train) \) is compared with \( Y\_train \) and \( \text{model.predict} (X\_test) \) with \( Y\_test \). The predict method is called on the test and training data and its output is compared with the existing labels for each trace.

The number of wrong NN classifications is approximated to 86 out of all 1050 traces. This has as a result a percentage of 91.8% correct classifications made by the NN, which is also the classification accuracy metric outputted as a simulation statistic. The reason that I use \( \text{numpy.round} \) is that the activation function that I use in the NN output is sigmoid. This means that values that are closer to zero are classified as zero and those closer to one are classified as one.

\[ \text{Catalog 5-10 NN misclassified traces} \]

Using Catalog 5-10 and by looking for non-zero entries in vector \( y \), traces that where misclassified by the NN can be defined both in the training and testing part of the event log. The \( \text{model.predict} (X\_train) \) is compared with \( Y\_train \) and \( \text{model.predict} (X\_test) \) with \( Y\_test \). The predict method is called on the test and training data and its output is compared with the existing labels for each trace.

The number of wrong NN classifications is approximated to 86 out of all 1050 traces. This has as a result a percentage of 91.8% correct classifications made by the NN, which is also the classification accuracy metric outputted as a simulation statistic. The reason that I use \( \text{numpy.round} \) is that the activation function that I use in the NN output is sigmoid. This means that values that are closer to zero are classified as zero and those closer to one are classified as one.
In Figure 5-10 the number of correct and incorrect classifications is presented as a percentage, plotted in a graph.

![Classification accuracy chart

Figure 5-10 Percentage of correct classifications](image)

5.4. Neural network construction and training (Using 10-fold cross validation dataset)

Here the description of the construction of a neural network that will perform the classification of the dataset created in chapter 5.2, using 10-fold cross validation takes place. The construction consists of the steps mentioned below.

**Step 1 - Load Data**

First of all, all of the classes and functions needed are being importer and the random number generator is initialized in order to make sure that the simulation results can be reproduced when this code is executed again. To use keras models with Scikit-learn the KerasClassifier wrapper is being used. This class accepts as an input a function that creates a neural network model, and returns the neural network model. Other arguments that it also accept include the number of epochs and the batch size and those will be passed along to the call to fit().

```python
#binary classification of sepsis case dataset
from keras.models import Sequential
from keras.layers import Dense
import numpy
from sklearn.preprocessing import StandardScaler
import timeit
from sklearn.cross_validation import StratifiedKFold
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.pipeline import Pipeline
from sklearn.cross_validation import cross_val_score
```
from keras.layers import Dropout
from keras.constraints import maxnorm
from keras.optimizers import SGD

# simulation timing
start = timeit.default_timer()
# fix random seed for reproducible results
seed = 7
numpy.random.seed(seed)

Catalog 5-11 Load libraries and seed random number generator

Now the dataset can be loaded using the numpy python package and split it into 185 input variables (X) and one output variable (Y).

# load dataset
dataset = numpy.loadtxt("ex4.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:185]
Y = dataset[:,185]

Catalog 5-12 Load and split dataset

Step 2 – Define model
Now the neural network model can be created using the python deep learning library, Keras. The machine learning Scikit-learn library of python is used to evaluate the model using k-fold cross validation. This will provide with an estimation for the classification accuracy of the model. The model defined here has 3 fully connected layers with 185, 150 and 100 nodes respectively. Weights are initialized using a small Gaussian random number and the Rectifier activation function is being used. The output layer contains a single neuron that is able to make predictions and uses the sigmoid activation function so as to produce a probability output in the range of 0 to 1. The logarithmic loss function binary_crossentropy is used during training, which is the preferred loss function used for binary classification problems. The stochastic gradient descent (sgd) is being used as an optimization algorithm and accuracy metrics will be collected once the model is trained. The learning rate is set to 1 and momentum to 0.9 (Nitish Srivastava, 2014). Dropout is applied between the input and the hidden layer and also between the hidden layers. Its rate is set at 20%, meaning that one out of every five inputs will be excluded from each update cycle. Also, a constraint is imposed on the weights of every hidden layer, making this way sure that the maximum norm of the weights will not exceed a value of 3 (Nitish Srivastava, 2014). This is achieved by setting the argument W_constraint on the dense class when defining the layers.

# NN model
def create_NN():
    # create model
    model = Sequential()
    model.add(Dropout(0.2, input_shape=(185,)))
    model.add(Dense(185, input_dim=185, init='uniform', activation='relu', W_constraint=maxnorm(3)))
    model.add(Dropout(0.2))
    model.add(Dense(150, init='uniform', activation='relu', W_constraint=maxnorm(3)))
    model.add(Dropout(0.2))
model.add(Dense(100, init='uniform', activation='relu',
  W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(1, init='uniform', activation='sigmoid'))

# Compile model
sgd = SGD(lr=0.1, momentum=0.9, decay=0.0, nesterov=False)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
return model

---

Catalog 5-13 Create NN model

**Step 3 – Evaluate model**

This step includes the evaluation of the model with the use of stratified cross validation within the scikit-learn framework. The number of training epochs is being passed to the KerasClassifier. Verbose is turned off keeping in mind that the model will be create 10 times while the 10-fold cross validation is being performed and setting its value to 0 will make the algorithm run an order of magnitude faster.

In order to improve the classification accuracy of the results, standardization of the input data to the neural network is being performed. This means the the data is rescaled such that the mean value for each attribute is 0 and the standard deviation is 1. This way the Gaussian distribution is preserved while the central tendency of each attribute is being normalized. StandardScaler() class is being used to perform the standardization of my dataset.

Instead of performing standardization on the entire dataset, it is preferred to train the standardization procedure on the training data within the pass of a cross validation run and use the trained standardization instance to prepare the unseen test fold. In this way, the standardization is being made a step in model preparation in the cross validation process and prevents the algorithm of having knowledge of unseen data during evaluation, knowledge that might be passed from the data preparation scheme.

This can be achieved in scikit-learn using a Pipeline class. The pipeline can be seen as a wrapper that executes one or more models within the pass of the cross validation procedure. The pipeline is defined here with the StandardScaler followed by the neural network model.

```python
# evaluate NN model
numpy.random.seed(seed)
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasClassifier(build_fn=create_NN, nb_epoch=70,
batch_size=200, verbose=0)))
 pipeline = Pipeline(estimators)
kfold = StratifiedKFold(Y, n_splits=10, shuffle=True, random_state=seed)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("classification accuracy: %.2f%% (%.2f%%)" % (results.mean()*100,
results.std()*100))
for i in results:
  print i*100, "%"
stop = timeit.default_timer()
print "Simulation lasted for: ", stop - start, "seconds"
```

Catalog 5-14 Evaluate model
Running the above defined network model outputs:

Table 5-2 Classification accuracy for each one of the 10 folds

<table>
<thead>
<tr>
<th>Folds</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>92.452</td>
</tr>
<tr>
<td>2</td>
<td>92.452</td>
</tr>
<tr>
<td>3</td>
<td>91.428</td>
</tr>
<tr>
<td>4</td>
<td>91.428</td>
</tr>
<tr>
<td>5</td>
<td>86.666</td>
</tr>
<tr>
<td>6</td>
<td>91.428</td>
</tr>
<tr>
<td>7</td>
<td>89.523</td>
</tr>
<tr>
<td>8</td>
<td>91.428</td>
</tr>
<tr>
<td>9</td>
<td>90.384</td>
</tr>
<tr>
<td>10</td>
<td>89.423</td>
</tr>
</tbody>
</table>

Overall classification accuracy: 90.66%
Standard deviation: 1.66%
Simulation duration 49 sec

5.5. Experiment Setup of “Road traffic management system case”
Here all the experiments performed in order to verify the accuracy of the outlier classification method are being described in detail, along with the context (experiment setup) within which they are performed. Included in this chapter are also the simulation results.

Description of the used event log
Here the detailed description of how outlier classification with the use of neural networks can work in a real life event log is described. Initially, the procedure will be explained in detail for one possible process model that can be derived from the given event log. After the detailed explanation for how this methodology can be applied in one process model, the results of applying the same methodology in other process models derived from the same event log will be presented by summarizing the classification results.

5.5.1. Event Log used for outlier detection
The real life event log that is being used here in order to perform the outlier classification task is one that has to do with the process of managing traffic fines (4TU). It consists out of 150,370 traces and of 561,470 discrete events. Every trace consists of a number of events that ranges from 2 to 20 and the average number of events per trace is 3,734.

5.5.2. Process model derived from the event log (Model 1)
The outlier detection procedure can be implemented in a process model derived from the given event log and not in the event log itself. For this reason from the given log a model should be discovered, accordingly all the
different traces should be identified (both normal and outliers) and a way should be found to automatically detect the outlier traces. With the use of the “inductive visual miner” algorithm, a model can be discovered from the provided event log. This model should allow for most of the traces that are present in the log, depending on its level of fitness. The discovered model can be slightly modified so that some more or less events of a trace that are present in the log are not included in the model. These slightly modified models (compared to the one discovered by the inductive visual miner) can be created using YASPER\(^{26}\), which is a process modeling software. The created model is in .pnml format, which can be directly imported to ProM so as to be checked for conformance with the given event log. After conformance analysis has been performed between the created process model and the event log it has been discovered from it can be seen that some events belonging to a trace are only present in the log and not in the model. Those events are considered to be outliers and all the rest as normal ones. The goal is to find a model that when trained will be capable of automatically detecting all of those traces that contain events which are only present in the log and not in the model.

Mapping model activities into binary convertible variables
Here the way that event log data are turned into tabular data is described.

Log encoding
The first step of turning event data into tabular data includes turning each activity present in the event log into a character following the below present mapping.

<table>
<thead>
<tr>
<th>Event name</th>
<th>Letter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create fine</td>
<td>A</td>
</tr>
<tr>
<td>Send Fine</td>
<td>B</td>
</tr>
<tr>
<td>Payment</td>
<td>C</td>
</tr>
<tr>
<td>Insert fine Notification</td>
<td>D</td>
</tr>
<tr>
<td>Send Appeal to Prefecture</td>
<td>E</td>
</tr>
<tr>
<td>Insert Date Appeal to Prefecture</td>
<td>F</td>
</tr>
<tr>
<td>Add penalty</td>
<td>G</td>
</tr>
</tbody>
</table>

\(^{26}\) http://www.yasper.org/
<table>
<thead>
<tr>
<th>Receive Result Appeal from Prefecture</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appeal to Judge</td>
<td>I</td>
</tr>
<tr>
<td>Notify Result Appeal to Offender</td>
<td>J</td>
</tr>
<tr>
<td>Send for Credit Collection</td>
<td>K</td>
</tr>
</tbody>
</table>

This is done, because since only vectors of numbers can be provided to a neural network as inputs, a way should be found in order to turn each trace of the process model into a vector of numbers. Each input vector represents a trace of the process model discovered from the initial log. Some traces have more activities than others, so as point of reference each time is considered the trace of the log with the most activities. Traces that have fewer activities are filled in with zero values (in order to reach the length of the longest trace). So by mapping the components of each trace (activities) into letters and then by turning those letters into numbers (with the use of the built-in python function `ord()`), the desired vectors of numbers that will be fed into the neural network can be constructed.

**Data labeling**

Event data exported from prom are in .csv format and for each event they include the name of the event, its case id and the where this event is present (event log, process model, or at both):

```
case id;activity;move type
A11;A;sync
A11;B;sync
A11;tr4;invalible
...
```

*Figure 5-13 Prom output format*

The first step of data manipulation includes sorting events by case id:

```
A1     ['Async', 'Bsync']
A100   ['Async', 'Bsync', 'Dsync', 'Gsync', 'Ksync']
A10000 ['Async', 'Bsync', 'Dsync', 'Gsync', 'Ksync']
A10001 ['Async', 'Bsync', 'Dsync', 'Psync', 'Gsync', 'Ksync']
...
```

*Figure 5-14 Sorting events by case id*

By a convention made by me (in order to find a way to separate outlier data from the rest), event log data that are present in the initial event log but not in the discovered process model are considered to be outlier events, and the traces that they are present in as outlier traces. After those outlier traces are identified, they are automatically assigned with a label of one (1) in the column that is present next to the last event of each trace, and those that are not outlier traces get a label of zero (0). This way, the constructed dataset containing all of the 150.370 traces of the log, which are sorted based on their case id becomes labeled and can be used by the neural network constructed in the next step for training and testing.

*Figure 5-15 Vector of numbers given as input to the neural network (before normalization)*
In order to improve the outlier classification accuracy of the constructed neural network and take it as high as possible, the data present on the 2d array that is used for training and testing (Figure 0-13) should be normalized before they are fed into the neural network.

For this reason the StandardScaler () method is being imported from the sklearn.preprocessing module. In particular the fit_transform() function is used from the StandardScaler () method. This function joins the steps of the two distinct functions fit () and transform () and is used for the initial fitting of parameters on the training set (supposedly x), but it also returns a transformed x’ . Internally, it just calls first fit () and then transform () on the same data.

Every sklearn’s transform’s fit () just calculates the parameters of mean and deviation (µ and σ in the case of StandardScaler () ) and saves them as an internal objects state. Afterwards, its transform () method can be called to apply the transformation to a particular set.

The complete code written in python, which is responsible for converting the exported in .csv format data into labeled tabular data, that can be used by the neural network constructed in chapter 5.3. for its training and testing is available in Appendix B.

Neural network construction and training (Using a manual verification dataset)

Here a detailed description of the structure of the whole neural network will take place. The problem of outlier classification of traces in term of a process model can be seen as a binary classification challenge in the sense that each one of the 150.370 traces that are present in the whole provided event log need to be characterized as being an outlier trace or not. Feed forward neural networks are capable of dealing with this type of challenges and classifying the provided dataset with extremely high accuracy. The creation of the feed forward neural network that will perform the classification task consists of five steps:

Step 1 - Load Data

Load libraries and fix random seed generator.

```python
from keras.models import Sequential
from keras.layers import Dense
from sklearn.cross_validation import train_test_split
import numpy
import matplotlib.pyplot as plt

# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

Catalog 0-1 Load libraries and seed random number generator

Load the dataset created in 6.2.

```python
# load the training and testing dataset of the network
data = numpy.loadtxt("full4.csv", delimiter="",)
# split into input (X) and output (Y) variables
X = data[:,0:20]
Y = data[:,20]
```

Catalog 0-2 Load the dataset using Numpy
Split the traces included in the whole dataset, into ones used for training and testing.

```python
# split into 67% for train and 33% for test
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=seed)
```

Catalog 0-3 Split the dataset into training and testing

Step 2 - Define Model

Model definition

```python
# create model
model = Sequential()
model.add(Dense(10, input_dim=20, init='uniform', activation='relu'))
model.add(Dense(10, init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='sigmoid'))
```

Catalog 0-4 Define the neural network model in keras

Below the network structure is presented schematically.

![Visualization of the neural network structure](image)

**Figure 5-16 Visualization of the neural network structure**

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
<th>Connected to</th>
</tr>
</thead>
<tbody>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 20)</td>
<td>420</td>
<td>dense_input_1[0][0]</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>(None, 10)</td>
<td>210</td>
<td>dense_1[0][0]</td>
</tr>
<tr>
<td>dense_3 (Dense)</td>
<td>(None, 1)</td>
<td>11</td>
<td>dense_2[0][0]</td>
</tr>
</tbody>
</table>

Total params: 641

**Figure 5-17 Model outline**
Step 3 - Compile Model

Compile the created model.

```python
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# checkpoint
filepath="weights-improvement-{epoch:02d}-{val_acc:.2f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]
```

Catalog 0-5 Compile the neural network model

Step 4 - Fit Model

Model is being fitted.

```python
# Fit the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), nb_epoch=117, batch_size=200, verbose = 0)
epoch_loss = history.history["loss"]
```

Catalog 0-6 Fit the neural network model to the dataset

Step 5 - Simulation Metrics and Model Evaluation

Execute simulations to measure the classification accuracy of the created model.

```python
# store simulation metrics
epoch_loss = history.history["loss"]
print(history.history)

# evaluate the model
scores = model.evaluate(X, Y)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

Catalog 0-7 Evaluate the neural network model on the dataset

The simulation is performed on my laptop running on the CPU (Intel Core i5-4200 CPU @ 2.50 GHz) with a Theano backend and lasted for about 54 seconds.

```python
import timeit

# simulation timing
start = timeit.default_timer()
stop = timeit.default_timer()
print("Simulation lasted for: ", stop - start, " seconds")
```

Catalog 0-8 Calculate the duration of the neural network simulation

Collecting all of the simulation results, regarding all of the 117 epochs for the training set (accuracy, loss) and for the test set (validation accuracy, validation loss) can lead to the construction of the below 2 graphs which depict how well the designed neural network model is able to classify the provided traces to it regarding both the training and testing dataset. In other words it is an indication of how accurately the outlier classification regarding all the traces of the given process model can be performed. Even without data normalization, extremely high trace classification accuracy was achieved.
Neural network construction and training (Using a 10-fold cross validation dataset)

Here the description of the construction of a neural network that will perform the classification of the dataset created in chapter 6.2, using 10-fold cross validation takes place. The construction consists of the steps mentioned below.
Step -1
Loading of libraries and initiation of seed for reproducibility.

```python
from keras.models import Sequential
from keras.layers import Dense
import numpy
from sklearn.preprocessing import StandardScaler
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.cross_validation import StratifiedKFold
from sklearn.cross_validation import cross_val_score
from sklearn.pipeline import Pipeline
from keras.optimizers import SGD
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

Catalog 0-11 Load libraries

Splitting of the dataset into inputs and outputs.

```python
# load data
dataset = numpy.loadtxt("full4.csv", delimiter="",)
# split into input (X) and output (Y) variables
X = dataset[:,0:20]
Y = dataset[:,20]
# create model
```

Catalog 0-12 Load and split data

Step-2 Define model
Neural network model construction.

```python
def create_baseline():
    # create model
    model = Sequential()
    model.add(Dense(10, input_dim=20, init='uniform', activation='relu'))
    model.add(Dense(5, init='uniform', activation='relu'))
    model.add(Dense(1, init='uniform', activation='sigmoid'))
    # Compile model
    sgd = SGD(lr=0.1, momentum=0.9, decay=0.0, nesterov=False)
    model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
    return model
```

Catalog 0-13 Create NN model

Step-3 Evaluate model
Classification evaluation of the created model.

```python
# evaluate baseline model with standardized dataset
numpy.random.seed(seed)
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasClassifier(build_fn=create_baseline, nb_epoch=15,
batch_size=200, verbose=0)))
pipeline = Pipeline(estimators)
```
kfold = StratifiedKFold(Y, n_folds=10, shuffle=True, random_state=seed)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Standardized: %.2f%% (%.2f%%)" % (results.mean() * 100, results.std() * 100))
for i in results:
    print i*100, "%"

Catalog 0-14 Evaluate NN model

Running the above defined network model outputs:

Table 5-4 Classification accuracy for each one of the 10 folds

<table>
<thead>
<tr>
<th>Folds</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>96.256</td>
</tr>
<tr>
<td>2</td>
<td>95.910</td>
</tr>
<tr>
<td>3</td>
<td>96.289</td>
</tr>
<tr>
<td>4</td>
<td>95.557</td>
</tr>
<tr>
<td>5</td>
<td>96.023</td>
</tr>
<tr>
<td>6</td>
<td>96.142</td>
</tr>
<tr>
<td>7</td>
<td>95.604</td>
</tr>
<tr>
<td>8</td>
<td>96.422</td>
</tr>
<tr>
<td>9</td>
<td>96.741</td>
</tr>
<tr>
<td>10</td>
<td>96.295</td>
</tr>
</tbody>
</table>

Overall classification accuracy: 96.12%
Standard deviation: 0.35%
Simulation duration 76 sec

Figure 5-19 Classification accuracy for each fold

Different classifications
In order to add more value to the results of the outlier classification method that is developed in this thesis, its functionality should be tested on different variations of the initially discovered model. This is why its classification accuracy is being tested on the models 2-9 seen in Appendix E.

The discovered model with the use of inductive visual miner (seen in Figure 0-10, or otherwise model 1) that is initially used for conformance checking with the provided event log (for the outlier classification task performed in 6.3 and 6.4) is only one out of many that can be discovered from the same event log. Other possible discovered models may contain a different number of activities, even being executed in a different order.

This can lead to the discovery of different traces within the newly discovered process models, which has as a result the need to perform the outlier classification task again for each one of them. For every case, the neural network used in order to perform the outlier classification might need to be modified in structure, when comparing to the one used for the classification of the traces of model 1.

This section includes variations of the initially discovered from the event log process model. For all of them, neural network models where constructed in order to check the classification accuracy of the outlier classification process.
In *Table 0-14*, the simulation results for each one of the 7 alternative models derived from the initial log are summarized.

**Table 5-5 Classification results of models derived from model 1**

<table>
<thead>
<tr>
<th>Model</th>
<th>No. of epochs</th>
<th>Accuracy</th>
<th>Validation Acc.</th>
<th>Loss</th>
<th>Validation Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>77</td>
<td>0.998</td>
<td>0.996</td>
<td>0.003</td>
<td>0.003</td>
</tr>
<tr>
<td>3</td>
<td>118</td>
<td>0.999</td>
<td>0.999</td>
<td>0.004</td>
<td>0.007</td>
</tr>
<tr>
<td>4</td>
<td>103</td>
<td>0.999</td>
<td>0.999</td>
<td>0.004</td>
<td>0.007</td>
</tr>
<tr>
<td>5</td>
<td>83</td>
<td>0.999</td>
<td>0.999</td>
<td>0.0008</td>
<td>0.001</td>
</tr>
<tr>
<td>6</td>
<td>111</td>
<td>0.999</td>
<td>0.999</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>7</td>
<td>123</td>
<td>0.998</td>
<td>0.999</td>
<td>0.004</td>
<td>0.003</td>
</tr>
<tr>
<td>8</td>
<td>52</td>
<td>0.999</td>
<td>0.999</td>
<td>0.0008</td>
<td>0.0007</td>
</tr>
</tbody>
</table>
Chapter 6  Conclusions

Here the conclusions to which I have reached after completing this project are being presented. Along with them, suggestions for future work (as a continuation of this project) are also included.

6.1.  Academic

The aim of this project was twofold as it was stated in the abstract part of this report. The two aspects that I had to deal with, included initially a literature study aiming to indicate whether all or some of the process mining challenges listed in the process mining manifesto (Van Der Aalst, 2011) could be successfully dealt with using the problem solving ability of neural networks. The second aspect of the problem this project aimed at solving was the one of outlier classification in terms of business process models discovered from real life event logs.

After the completion of this seven month thesis project it can be concluded that some solutions/suggestions of how to deal with them were found for the two challenges that it included.

As far as the literature review concerning whether some of the process mining challenges can be solved with the use of neural networks and how, a detailed presentation of all those challenges that are solvable by neural networks is been given along with what steps should be followed by the user in order to lead him to the solution. For all the other challenges that after studying the related literature, I could not find a way with which neural networks can contribute to their solution, concrete reasoning is being presented to back up my claims.

As far as the challenge of dealing with outlier classification in terms of a process model that has been derived from a real life event log is concerned, I have proved that it can be achieved using neural networks. In particular, this can be done by constructing feed forward NNs which (after training) will be able to perform the classification of traces task in previously unseen instances. Adjusting the parameters of such a network (such as the number of nodes in its hidden layers, or the activation functions of the nodes, etc.) can be decided after an experimentation process that will aim at achieving the highest possible classification accuracy. The constructed NN is able to accept encoded traces of an event log in its input layer and can provide with a classification (as outlier or not) for every trace included in it.
6.2. Limitations

The main limitations related to this project are related to the practical solution of the “outlier classification” task and these include:

- The NNs used to classify outliers, are trained in a supervised manner, meaning that they learn by example. For this reason labeled datasets need to be created which are later being used by the NNs for training.
- The outlier classification process, takes as granted that the process models discovered from the provided event logs that are used for the classification process are in a steady state and are not changing during the passage of times. However it is extremely common for a process model to change at the same time that it is being analyzed. In this case, different methods should be deployed in order to perform outlier classification of traces tasks.

6.3. Future work

Due to the fact that this project contains a literature part, (my claims are backed up by literature findings and my suggestions are not implemented in practice) it means that there is plenty of space for implementing the literature findings using real life data. This means that all of the suggested solutions including ways that neural networks can solve the listed process mining challenges in the process mining manifesto can be put into practical use and tested for their validity.

As far as the outlier classification part is concerned, there can also be suggestions for further study. First of all, the applicability of LSTM NNs should be tested on solving outlier classification challenges, since there exists literature that suggests their capability in dealing with classification tasks. Furthermore, there are cases where a process model changes at the same time it is being analyzed or checked for outlier events. In those cases, maybe different types of neural networks would need to be applied in order to accurately perform the outlier classification task. Finally, the above outlier classification implementation refers to whole traces. However even a single event could be an outlier or even an attribute of an event. The capability of neural networks to identify also those types of outliers should be examined and put into practical implementation.
Bibliography

4TU. (sd). 4TU.Center for Research Data. Opgehaald van https://data.4tu.nl/repository/collection:event_logs_real


Juan Carlos Fernández Caballero. (2010). Sensitivity Versus Accuracy in Multiclass Problems Using Memetic Pareto Evolutionary Neural Networks. IEEE TRANSACTIONS ON NEURAL NETWORKS.

Kim, Y. (2014). *Convolutional Neural Networks for Sentence Classification*.


Varma, R. (2016, August). Applying Neural Networks to Natural Language Processing Tasks.


Appendices

Appendix A
Here a list of the major algorithms that are used in all three of the machine learning techniques (supervised, unsupervised and reinforcement learning) is being presented. Their use in NNs is specifically highlighted as long as some further information including their functionality, their use, a representative example for each one and possible already existing implementations in some programming languages along with references.

Table 0-1 – Supervised machine learning major algorithms

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Used for</th>
<th>Concepts</th>
<th>Implementation</th>
<th>Example</th>
<th>Use in NNs</th>
<th>Form of input and output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Regression</td>
<td>Qualitative response/discrete choice models</td>
<td>Measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution. The dependent variable is a characteristic or quantity that is measured using the independent variable</td>
<td>Python, Java, Matlab, R</td>
<td>Measuring the success rate of marketing campaigns</td>
<td>Logistic regression is useful when working with a dataset where the classes are more or less linearly separable. NNs are somewhat related to logistic regression. Logistic regression can be considered as a 1-layer neural network.</td>
<td>Set of variables as an input and binary values as an output. e.g.: Input: (0,0),(0,1),(1,0),(1,1) Output: (0 or 1)</td>
</tr>
<tr>
<td>Back propagation NN</td>
<td>It is used to find a way to train a multi-layered NN, such that it can learn the appropriate internal representations to allow it to learn any arbitrary mapping of input to output</td>
<td>Common method for training ANN, used in conjunction with an optimization method such as gradient descent. The method calculated the gradient of a loss function with respect to all the weights in the network. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function</td>
<td>Python, Matlab, Java</td>
<td>Convert image to text</td>
<td>The goal of Back propagation algorithm is to find a way to train a multi-layered NN such that it can learn the appropriate internal representations to allow it to learn any arbitrary mapping of input to output. The method calculates the gradient of a loss function with respect to all the weights in the network. The gradient is fed into the optimization method which in turn uses it to update the weights in an attempt to minimize the loss function. Back propagation requires a known desirable output for each input value in order to calculate the loss function gradient.</td>
<td>Boolean input and output</td>
</tr>
<tr>
<td>k-Nearest Neighbor (k-NN)</td>
<td>Classification and Regression</td>
<td>In both cases, the input consists of the k closest examples in the feature space. In classification, an object is classified by a majority votes of its neighbors, with the object being assigned to the class most common</td>
<td>Java, Matlab, R, C</td>
<td>Detecting multiple outliers (more examples)</td>
<td>Using a NN for solving a problem where k-NN would suffice is worth avoiding. This is because NNs are more powerful in general</td>
<td>Not applied</td>
</tr>
</tbody>
</table>
### Table 0-2 - Unsupervised machine learning major algorithms

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Used for</th>
<th>Concepts</th>
<th>Implementation</th>
<th>Example</th>
<th>Use in NNs</th>
<th>Form of input and output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical Clustering</td>
<td>Building a hierarchy of clusters</td>
<td>It can be splitted into Agglomerative (Bottom up) and Divisive (Top down); Merges and splits are determined in a greedy manner (local optimal choice at each stage). Its results are presented in a dendrogram</td>
<td>Java, C#</td>
<td>Create a dendrogram out of a provided list of element-values</td>
<td>NNs can perform the tasks of hierarchical clustering in a higher level, offering superior accuracy.</td>
<td>Hierarchical clustering can handle numerical data.</td>
</tr>
<tr>
<td>K-means</td>
<td>Clustering</td>
<td>Simple clustering algorithm, that works really well with large data sets</td>
<td>C++, Matlab</td>
<td>Classify elements placed in 1-D and</td>
<td>K-means clustering can be performed with</td>
<td>K-means clustering can handle</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>Text categorization</td>
<td>Simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. All naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable</td>
<td>R, Java, Matlab</td>
<td>Classify a mail as spam, classify a news article about technology, politics, or sports</td>
<td>NNs can handle correlations between input variables, while Naive Bayes assumes all input variables are independent. So NNs can provide a solution to a naive Bayes problem more efficiently.</td>
<td>Naive Bayes can process categorical and numerical data.</td>
</tr>
<tr>
<td>Decision Tree learning</td>
<td>Visually represent decisions and decision making</td>
<td>Uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. It is one of the predictive modeling approaches used in statistics, data mining and ML. Tree models where the target variable can take a finite set of values are called classification trees. Classification trees where the target value can take continuous values are called regression trees</td>
<td>C#, R, Java, Python</td>
<td>Decision tree for mapping a specific person to a political party based on a number of different criteria (e.g. his opinion on a number of different issues)</td>
<td>Both methods can model data that has nonlinear relationships between variables and both can handle interactions between variables. Some differences include: - Decision trees are faster and more interpretable - NNs can model more arbitrary functions and might be more accurate</td>
<td>Decision trees can handle both numerical and categorical data. NNs can handle numerical data</td>
</tr>
<tr>
<td>Support Vector Machines</td>
<td>Classification and regression analysis</td>
<td>Given a set of training examples, each marked to belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other</td>
<td>Java, Python, Matlab</td>
<td>Boston housing problem: prices must be predicted from socio-economic and environmental factors, such as crime rate, age of property</td>
<td>When comparing to NNs the solution to an SVM is unique and it has a simple geometric representation. Also comparing to NNs are less prone to over fitting.</td>
<td>SVM uses continuous variables. Can handle categorical data also, but encoding is required</td>
</tr>
</tbody>
</table>

### Notes

- **Naive Bayes Problem**: A Naive Bayes problem can be solved more efficiently. The solution to a naïve Bayes problem can provide a better prediction of the value of any other feature, given the class variable. However, Naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

- **Difference between Naive Bayes and Neural Networks (NNs)**: While Naive Bayes assumes all input variables are independent, NNs can handle correlations between input variables. Naive Bayes can provide a solution to a Naive Bayes problem more efficiently.

- **Decision Trees vs. Support Vector Machines (SVMs)**: Decision trees can handle both numerical and categorical data. SVMs use continuous variables and are less prone to overfitting.

- **Hierarchical Clustering**: Hierarchical clustering can handle numerical data and is unique in its ability to handle large datasets and offer superior accuracy.

- **K-means Clustering**: K-means clustering can handle large datasets and is simple to implement.

- **Support Vector Machines (SVMs)**: SVMs use continuous variables and are less prone to overfitting. SVMs can handle both numerical and categorical data.

- **Naive Bayes vs. SVMs**: Both methods can model data that has nonlinear relationships between variables, but SVMs are faster and more interpretable.

- **Decision Trees**: Decision trees are faster and more interpretable than SVMs but can handle both numerical and categorical data.

- **Naive Bayes vs. NNs**: Both Naive Bayes and NNs can handle both numerical and categorical data. However, NNs can model more arbitrary functions and might be more accurate.

- **Implementation**: Various programming languages are used for implementation, including Java, Python, and Matlab.

- **Example**: Naive Bayes can be used to classify a news article about technology, politics, or sports.

- **Use in NNs**: SVMs are used in NNs to solve tasks of hierarchical clustering in a higher level, offering superior accuracy.

- **Form of Input and Output**: SVMs can handle continuous variables and are less prone to overfitting. Decision trees can handle both numerical and categorical data.

- **Table 0-2**: This table summarizes the major algorithms used for unsupervised machine learning, including their usage, implementation, and performance characteristics.
Self Organizing Maps (SOM) are data visualization techniques that reduce the dimensions of data (through the use of self-organizing neural networks) and display similarities. SOMs (actually a type of ANN) can perform clustering. SOM clustering can handle Numerical data.

Apriori Algorithm is an Item set mining and association rule learning over transactional databases method. It identifies the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. Frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database.

Gaussian Mixture model is a Clustering and Visualization method that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. It classifies elements placed in the 2-D state space.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Used for</th>
<th>Concepts</th>
<th>Implementation</th>
<th>Example</th>
<th>Use in NNs</th>
<th>Form of input and output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-Learning (QV-Learning has a similar functionality)</td>
<td>To find the optimal next action when the pool of possible actions is extremely large. In any other occasion dynamic programing can be applied. It is an off-policy learner (learns the value of the optimal policy independently of the agent’s actions)</td>
<td>It works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter.</td>
<td>Python, Matlab, Excel implementations</td>
<td>World-grid navigation</td>
<td>NNs can be trained in order to implement the q-learning algorithm.</td>
<td>State and next state</td>
</tr>
<tr>
<td>Sarsa (State Action Reward State Action)</td>
<td>To find the optimal policy, taking into account the exploration inherent in the policy. It is an on-policy learner (learns the value of the policy being carried out by the agent, including the exploration steps)</td>
<td>An agent initially at state's', does an action 'a', receives reward 'r' and ends up in state 's' from which it decided to do action 'a'.</td>
<td>Java</td>
<td></td>
<td>same as q-learning</td>
<td>same as q-learning</td>
</tr>
<tr>
<td>AC (ACLA has a similar functionality)</td>
<td>To find the optimal next action when the number of possible next actions is extremely large. It is an on-policy algorithm like Sarsa.</td>
<td>It keeps track of two functions: a Critic that evaluates states and an Actor that maps states to a preference value for each action. After each action selection, the critic evaluates the new state to determine whether things have gone better or worse than expected.</td>
<td>No implementation available</td>
<td>Navigation of a robot in an unmapped environment</td>
<td>same as q-learning</td>
<td>same as q-learning</td>
</tr>
<tr>
<td>Algorithms</td>
<td>Implementation References</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Matlab: <a href="https://ml.mathworks.com/help/stats/mnrfit.html">https://ml.mathworks.com/help/stats/mnrfit.html</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• R: <a href="http://www.ats.ucla.edu/stat/r/descriptives/logit.htm">http://www.ats.ucla.edu/stat/r/descriptives/logit.htm</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Matlab: <a href="http://www.cs.umbc.edu/~dpatt03/nn/backprop.html">http://www.cs.umbc.edu/~dpatt03/nn/backprop.html</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k-Nearest Neighbor (k-NN)</td>
<td>• Java: <a href="http://www.programcreek.com/2013/01/use-k-nearest-neighbors-knn-classifier-in-java/">http://www.programcreek.com/2013/01/use-k-nearest-neighbors-knn-classifier-in-java/</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Matlab: <a href="https://ml.mathworks.com/help/stats/knnsearch.html">https://ml.mathworks.com/help/stats/knnsearch.html</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• R: <a href="https://www.analyticsvidhya.com/blog/2015/08/learning-concept-knn-algorithms-programming/">https://www.analyticsvidhya.com/blog/2015/08/learning-concept-knn-algorithms-programming/</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case-Based</td>
<td>No implementation available</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decision Tree learning</td>
<td>• C#: <a href="http://csouza.com/2012/01/04/decision-trees-in-c/">http://csouza.com/2012/01/04/decision-trees-in-c/</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• R: <a href="http://www.statmethods.net/advstats/cart.html">http://www.statmethods.net/advstats/cart.html</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>• R: <a href="https://rfg2late.wordpress.com/2015/11/06/a-gentle-introduction-to-naive-bayes-classification-using-r/">https://rfg2late.wordpress.com/2015/11/06/a-gentle-introduction-to-naive-bayes-classification-using-r/</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Java: <a href="https://github.com/datumbox/NaiveBayesClassifier/blob/master/src/com/datumbox/op">https://github.com/datumbox/NaiveBayesClassifier/blob/master/src/com/datumbox/op</a> ensource/classifiers/NaiveBayes.java</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hierarchical Clustering</td>
<td>• Matlab: <a href="https://ml.mathworks.com/help/stats/kmeans.html">https://ml.mathworks.com/help/stats/kmeans.html</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-means</td>
<td>• C++: <a href="https://github.com/michaelchughes/KMeansRex">https://github.com/michaelchughes/KMeansRex</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Matlab: <a href="https://ml.mathworks.com/help/stats/kmeans.html">https://ml.mathworks.com/help/stats/kmeans.html</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaussian Mixture model</td>
<td>• C++: <a href="http://www.milpack.org/docs/mil-ack-gt/doxygen.php?doc=classmilpack_1_1gmm_1_1GMM.html">http://www.milpack.org/docs/mil-ack-gt/doxygen.php?doc=classmilpack_1_1gmm_1_1GMM.html</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Matlab: <a href="https://ml.mathworks.com/help/stats/cluster-using-gaussian-mixture-models.html">https://ml.mathworks.com/help/stats/cluster-using-gaussian-mixture-models.html</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apriori Algorithm</td>
<td>• Java: <a href="https://pist.github.io/monopurus/7157717">https://pist.github.io/monopurus/7157717</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• C++: <a href="http://ankurml/implementation-of-apriori-algorithm-in-c/">http://ankurml/implementation-of-apriori-algorithm-in-c/</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• R: <a href="https://www.r-bloggers.com/implementing-apriori-algorithm-in-r/">https://www.r-bloggers.com/implementing-apriori-algorithm-in-r/</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q-Learning</td>
<td>• Weka: <a href="http://www.kiet.edu/KIETIJICI/Vol1issue1/lecture7.pdf">http://www.kiet.edu/KIETIJICI/Vol1issue1/lecture7.pdf</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Hadoop: <a href="https://github.com/gallilee/hadoop-apriori">https://github.com/gallilee/hadoop-apriori</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Matlab: <a href="https://github.com/mak02/q-learning-algorithm-implementation-in-MATLAB">https://github.com/mak02/q-learning-algorithm-implementation-in-MATLAB</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Excel: <a href="http://people.revoledu.com/kardi/AutoR/ReinforcementLearningQ-Learning-Excel.htm">http://people.revoledu.com/kardi/AutoR/ReinforcementLearningQ-Learning-Excel.htm</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AC</td>
<td>No implementation available</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix B
Here the code that is turns the Prom output csv file into tabular data that can be provided to the constructed neural network for training is presented.

```python
import csv

with open("full.csv", "rb") as f:  # open file
    reader = csv.reader(f, delimiter="", )  # read content
    my_list = list(reader)  # put content in my_list

with open("full2.csv", "w") as f:
    wr = csv.writer(f, dialect='excel', lineterminator='
')
    my_list = [[item.split(';') for item in lst] for lst in my_list]
    for i in my_list:
        wr.writerows(i)  # unfold list

Catalog Appendix B1: Sort events by trace id

```
if any (t.endswith(prefix2) for t in value):
    value = ['A' if x == "Async" or x=="Create finelog" else x for x in value]
value = ['B' if x == "Bsync" or x=="Send Finelog" else x for x in value]
value = ['C' if x == "Csync" or x=="Paymentlog" else x for x in value]
value = ['D' if x == "Dsync" or x=="Insert fine notificationlog" else x for x in value]
    value = ['E' if x == "Esync" or x=="Send Appeal to Prefecturelog" else x for x in value]
value = ['F' if x == "Fsync" or x=="Insert Date Appeal to Prefecturelog" else x for x in value]
value = ['G' if x == "Gsync" or x=="Add penaltylog" else x for x in value]
value = ['H' if x == "Hsync" or x=="Receive Result Appeal from Prefecturelog" else x for x in value]
value = ['I' if x == "Isync" or x=="Appeal to Judgelog" else x for x in value]
value = ['J' if x == "Jsync" or x=="Notify Result Appeal to Offenderlog" else x for x in value]
value = ['K' if x == "Ksync" or x=="Send for credit collectionlog" else x for x in value]

value = [ord(i) for i in value] + [0]*(max_len - len(value))
#max_len = max(len(value), max_len)
cr.writerow(value+[1])
#print len(value)
#print value
#print max_len
else:
    value = ['A' if x == "Async" else x for x in value]
value = ['B' if x == "Bsync" else x for x in value]
value = ['C' if x == "Csync" else x for x in value]
value = ['D' if x == "Dsync" else x for x in value]
value = ['E' if x == "Esync" else x for x in value]
value = ['F' if x == "Fsync" else x for x in value]
value = ['G' if x == "Gsync" else x for x in value]
value = ['H' if x == "Hsync" else x for x in value]
value = ['I' if x == "Isync" else x for x in value]
value = ['J' if x == "Jsync" else x for x in value]
value = ['K' if x == "Ksync" else x for x in value]

#max_len = max(len(value), max_len)
value = [ord(i) for i in value] + [0]*(max_len - len(value))
#print max_len
#print value
cr.writerow(value+[0])

Catalog Appendix B2: Convert event traces into normalized vectors of numbers
Appendix C

Here, the outlier classification of the event log traces (derived from the process model (seen in Figure 0-10) using a 10-fold cross validation neural network is presented. The manipulation of the full.csv file that is extracted from Prom is the same as the one described in Appendix G and the only thing that changes is the way of evaluating the classification accuracy on the test dataset.

Another way of evaluating the classification accuracy of the event log data is by using $k$-fold cross validation. It provides an estimate of the performance of a model on previously unseen data. This is done by splitting the training dataset into $k$ subsets and by training models on all subsets except the one which is used for evaluating model performance on the validation dataset. The process is repeated until all of the $k$ subsets are given an opportunity to be the validation set. The performance measure is then averaged across all models that are created.

Cross validation is often not used for evaluating deep learning models since it has a greater computational cost, comparing for example with the manual verification neural network. For example $k$-fold cross validation is often used with 5 or 10 folds and here I use it with 10 folds. As such, 10 models must be constructed and evaluated, adding more computational time to the evaluation of a model. So, when the problem is small enough and there are available sufficient computer resources, $k$-fold cross validation can give a less biased estimate of the performance of the model.

In the example below I use the StratifiedKFold class from the scikit-learn Python machine learning library to split up the training dataset into 10 folds. The folds are stratified, meaning that the algorithm attempts to balance the number of instances of each class in each fold. Here 10 models are created and evaluated using the 10 splits of the data and all of the scores are collected. The verbose output for each epoch is turned off by passing verbose=0 to the fit() and evaluate() functions on the model. The performance is printed for each model and it is stored. The average and standard deviation of the model performance is then printed at the end of the run to provide an estimate of model accuracy.

```python
from keras.models import Sequential
from keras.layers import Dense
from sklearn.cross_validation import StratifiedKFold
import numpy
from sklearn.preprocessing import StandardScaler
import timeit

# simulation timing
start = timeit.default_timer()
# fix random seed for Reproducibility
seed = 7
numpy.random.seed(seed)

dataset = numpy.loadtxt("full4.csv", delimiter="",
# split into input (X) and output (Y) variables
X = dataset[:,0:20]
scaler = StandardScaler()
X = scaler.fit_transform(X)
Y = dataset[:,20]

# define 10-fold cross validation test harness
```
```python
kfold = StratifiedKFold(y=Y, n_folds=10, shuffle=True, random_state=seed)
cvscores = []
for i, (train, test) in enumerate(kfold):
    # create model
    model = Sequential()
    model.add(Dense(10, input_dim=20, init='uniform', activation='relu'))
    model.add(Dense(10, init='uniform', activation='relu'))
    model.add(Dense(1, init='uniform', activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    # Fit the model
    model.fit(X[train], Y[train], nb_epoch=70, batch_size=200, verbose=0)
    # evaluate the model
    scores = model.evaluate(X[test], Y[test], verbose=0)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1] * 100))
    cvscores.append(scores[1] * 100)
    stop = timeit.default_timer()
    print("%lf%% (+/- %lf%%)" % (numpy.mean(cvscores), numpy.std(cvscores)))
print("Simulation lasted for: ", (stop - start) / 60, "minutes")
```

Catalog Appendix C1: Evaluate a neural network using scikit-learn

Running this code will take approximately 5:31 minutes and the following output will be produced:

```
acc: 99.83%
acc: 96.52%
acc: 98.43%
acc: 99.79%
acc: 99.81%
acc: 99.87%
acc: 96.10%
acc: 99.87%
acc: 99.93%
acc: 99.90%
99.00% (+/- 1.42%)
Simulation lasted for: 5.31248123835 minutes
```

Catalog Appendix C2: Output of Evaluating A Neural Network Using scikit-learn.

The model had to be re-created for each loop to then fit and evaluate it with the data for the fold.
Appendix D
Here the presentation of how the last two process mining challenges can be solved with the use of neural networks is presented

Challenge 10: Improving usability for non-experts
The tenth process mining challenge includes the improvement of usability for non-experts.

Challenge summarization
Amongst the goals of process mining is the creation of process models that are used on a daily basis by process analysts in order to draw conclusions. These process models are considered to be “living”, are regularly used and are not static models that end up useless in some archive. With the use of new event data, emerging behavior that was not visible before can be discovered. The established link that is present between event data and process models can lead to the projection of the current state and recent activities onto up-to-date models. This way, end users can interact with the results of process mining on a day to day basis. The challenge can be considered two-folded:

1. **Create user friendly interfaces**: On the one hand, it includes hiding all the sophisticated process mining algorithms behind user-friendly interfaces. This actually implies the creation of such interfaces that will enable even a non-expert user to interact with the results of process mining and explore all the benefits of these interactions.

2. **Automatic setting of parameters**: On the other hand, these interfaces should be capable of automatically setting parameters and suggesting suitable types of analysis to the user, based on the information they have regarding his knowledge background or position within an enterprise (Van Der Aalst, 2011).

Analysis of problem
Here, an analysis of how usability of process mining can be improved for non-experts with the use of neural networks takes place. This implies that it should be examined whether both sub challenges (creating user friendly interfaces to hide PM algorithms and automatically set parameters for PM) of the 10th process mining challenges can be solved using deep learning.

1. **Creation of user friendly interfaces solution**
   As far as the creation of user friendly interfaces is considered, neural networks can’t contribute to this task, since it can’t be considered as a machine learning task in any way. Rather is can be seen as an user-friendly interface creation task. These interfaces can be created with various programming tools such as “fore UI” or “LivePipe UI”, but not with the use of artificial intelligence. Aim of such interfaces would be to hide all the sophisticated process mining algorithms behind them, thus making the execution of process mining possible even for amateur users.

2. **Automatically setting of parameters and suggesting suitable type of analysis challenge solution**
   As far as the ability of these interfaces to automatically set parameters and suggest suitable types of analysis to the user is concerned, neural networks can offer a solution to this challenge. This can be done by offering a way to identify all the possible users that use the interfaces that were created in the previous step and offer a personalized output of the results for each user that uses it. The challenge of identifying all possible users that
use the application and offer a specific representation to each one of them can be considered as a clustering problem. This stands if we take into account that all the users of one cluster should get the same representation of the results. Neural networks are able of solving clustering problems, by learning (after training) in a supervised manner how to distinguish the elements of each cluster amongst them. For the solution of this challenge, four steps should be followed:

**Approach for addressing the challenge**

**Step 1 – Use feed forward NNs in order to fill a created database with all user information**

The process of filling a database containing data of all the users that use a process mining application can be performed with the use of neural networks. This is because the construction of this database can be seen as a clustering challenge, with which neural networks can deal with. Each user is clustered based on his knowledge background, as being capable of correctly interpreting a specific output (derived from a process mining application) or not. In the end, a suitable representation should be made available for each one. Using a NN for filling the database with all user info and picture requires first training one in order to be able to identify all of the users based on their image (retrieved from a camera) and later on the system can learn about each one’s preferences regarding the application output.

- **Encode each user’s image into binary format.** This way the image can be provided as an input to the input layer of a NN that will be constructed in the next step.
- **Train a feed forward NN to be able to perform the clustering task.** The training process will be performed in a supervised manner, since for every user it will be known to which cluster he belongs, making it this way possible to create a labeled training dataset. In its input layer, an encoded event log will be provided to it and it will output the predefined class that each user belongs to.
- **Test the clustering accuracy of the NN.** In this step, the clustering accuracy of the trained NN will be measured, in order to be able to know the accuracy of the clustered results.
- **Apply clustering using the trained NN for every new user of the application.** Here the trained NN will be provided with the encoded event log in order to cluster all of the users.

**Step 2 – Application user identification**

The second step includes the identification of each user that uses the already created interfaces. All the different users that use a process mining application should be identified and split into categories (clusters), based on their knowledge background, profession, etc. For each category created, different choice of parameters and suitable types of analysis will be available. With the use of convolutional neural networks (Lawrence, 1997) face recognition tasks can be performed, providing in this way each user with a representation he can interpret. The facial image of each user that uses a process mining application can be retrieved with the use of a camera and using the database created in step 1 it can be mapped with a representation of the output that can be understood by the user of the application each time.

As an input to this kind of application the database consisting of photos that belong to each one of the application’s users, created in the previous step is required to be provided as an input. Each photo belonging to each user should be linked within the database to the knowledge background of him and thus to a preferred output representation of the process mining result.
Step 3 – Creation of software patch compatible with all process mining applications

There are a number of different applications that can perform process mining tasks, such as ProM, QPR, my-invenio, etc. For this reason a software patch that can be installed on top of each one of those applications needs to be created. This means that the database that is created in step 1 along with the convolutional neural network that will do the matching task between each user and his preferred output representation format will actually compose the patch. Purpose of this patch will be to perform the task of user-preferred output matching.

Step 4 - Personalized presentation of the Process mining output

After the patch has been created and installed to each process mining application and the user-output matching has been performed, the selected output each time for each user will be presented to him. The preferred representation should be automatically picked (for each background of knowledge of the users) by the application. This way the system (with the use of a camera) will be able to decide what output will provide to each user, based on the registered information in the database created in step 1.

As an extra functionality, the patch could allow the user to switch between all the available representations of the output format of the process mining application, so as to provide him with extra output choices.

Figure 0-1 Improving usability for non-experts using NNs
Challenge 11: Improving understandability for non-experts
The eleventh process mining challenge includes the improvement of understandability for non-experts.

Challenge summarization
The challenge for improving the understandability of process mining results for those that are not experts in this field can be split in two main sub challenges:

1. **The first one includes the interpretation of the results**: Even in the case that it is easy to generate some process mining results, it does not mean that those results are actually useful and easy to be interpreted by a user. The user might have problems in understanding the provided results or might be tempted to reach incorrect conclusions. Put it otherwise, the user might have problems with correctly interpreting the provided results that are derived from whichever process mining application.

   Discovered models may focus on different perspectives (control-flow, data-flow, time, resources, etc.) and show these at different levels of precision. Each member within an enterprise may wish to view the same results, but from a different perspective. For example a manager may want to see an informal process model that focuses on cost parameters, whereas a process analyst may want to see a detailed process model focusing on deviations from the normal flow.

   Furthermore, different stakeholders may want to view each process at different levels of granularity:
   - Strategic, in which decisions have long-term effects and are based on aggregate event data over a longer period.
   - Tactical, where decisions have medium-term effects and are mainly based on recent data.
   - Operational, where decisions have immediate effects and are related to event data related to running cases.

   All the above analysis indicates that there should be a personalized representation of the output of a process mining application for every user. This actually implies that some sort of user identification process should take place in order to make it possible to distinguish all the different users between them and subsequently present to each one a result that he can easily understand and correctly interpret.

2. **The second one is about trustworthiness of the results**: The trustworthiness of the results should always be clearly indicated. In some cases the data available might be insufficient in order to justify particular conclusions. For example, the existing process discovery techniques typically do not warn for low fitness or for overfitting. They always depict a model, even when it is clear that there is too little data to justify any conclusions (Van Der Aalst, 2011).

Analysis of problem
Here, an analysis of how process mining results can be presented to the non-experts in a form that they can understand using neural networks takes place. This implies that it should be examined whether both subchallenges (interpretation and trustworthiness of PM results) of the 11th process mining challenges can be solved using deep learning.
1. Interpretation of results challenge solution

As far as the interpretation of results challenge is concerned, neural networks could provide a solution to this first sub-challenge of the eleventh process mining challenge by offering a way to identify all the possible users that use a process mining application and offer a personalized output of the results for each and every one of them. Put it otherwise the interpretation of results challenge can be considered as a clustering problem (by making the hypothesis that every group of individuals that get one specific output representation belong to the same cluster) and thus neural networks can contribute to its solution.

Step 1 – Use feed forward NNs in order to fill a created database with all user information

The process of filling a database containing data of all the users that use a process mining application can be performed with the use of neural networks. This is because the construction of this database can be seen as a clustering challenge, with which neural networks can deal with. Each user is clustered based on his knowledge background, as being capable of correctly interpreting a specific output (derived from a process mining application) or not. In the end, a suitable representation should be made available for each one. Using a NN for filling the database with all user info and picture requires first training one in order to be able to identify all of the users based on their image (retrieved from a camera) and later on the system can learn about each one’s preferences regarding the application output.

- **Encode each user’s image into binary format.** This way the image can be provided as an input to the input layer of a NN that will be constructed in the next step.
- **Train a feed forward NN to be able to perform the clustering task.** The training process will be performed in a supervised manner, since for every user it will be known to which cluster he belongs, making it this way possible to create a labeled training dataset. In its input layer, an encoded event log will be provided to it and it will output the predefined class that each user belongs to.
- **Test the clustering accuracy of the NN.** In this step, the clustering accuracy of the trained NN will be measured, in order to be able to know the accuracy of the clustered results.
- **Apply clustering using the trained NN for every new user of the application.** Here the trained NN will be provided with the encoded event log in order to cluster all of the users.

Step 2 – Application user identification

The second step includes the identification of each user that uses the already created interfaces. All the different users that use a process mining application should be identified and split into categories (clusters), based on their knowledge background, profession, etc. For each category created, different choice of parameters and suitable types of analysis will be available. With the use of convolutional neural networks (Lawrence, 1997) face recognition tasks can be performed, providing in this way each user with a representation he can interpret. The facial image of each user that uses a process mining application can be retrieved with the use of a camera and using the database created in step 1 it can be mapped with a representation of the output that can be understood by the user of the application each time.

As an input to this kind of application the database consisting of photos that belong to each one of the application’s users, created in the previous step is required to be provided as an input. Each photo belonging to
each user should be linked within the database to the knowledge background of him and thus to a preferred output representation of the process mining result.

**Step 3 – Creation of software patch compatible with all process mining applications**
There are a number of different applications that can perform process mining tasks, such as ProM, QPR, my-invenio, etc. For this reason a software patch that can be installed on top of each one of those applications needs to be created. This means that the database that is created in step 1 along with the convolutional neural network that will do the matching task between each user and his preferred output representation format will actually compose the patch. Purpose of this patch will be to perform the task of user-preferred output matching.

**Step 4 - Personalized presentation of the Process mining output**
After the patch has been created and installed to each process mining application and the user - output matching has been performed, the selected output each time for each user will be presented to him. The preferred representation should be automatically picked (for each background of knowledge of the employees within each specific enterprise) by the application. This way the system (with the use of a camera) will be able to decide what output will provide to each user, based on the registered information in the database created in step 1.

As an extra functionality, the patch should allow the user to switch between all the available representations of the output format of the process mining application, so as to provide the user with extra choices.

---

**Figure 0-2** Improving understandability for non-experts using NNs
2. Trustworthiness of the results challenge

Analysis of problem

As far as the trustworthiness of the results is concerned, neural networks can’t contribute to the solution of this challenge, since it is not possible to train a neural network in order to be able to identify, whether a specific result is trustworthy or not.

This is because the criteria leading to trustworthiness or not of a result are implicit, dependent to interpretation of the user, something that a neural network can’t be trained to distinguish and thus human judgment needs to be present and taken into account in every occasion.

Appendix E

Variations of the discovered from the event log process model used for outlier detection

Here the presentation of the variations of the initial model of the “road traffic management”, the classification accuracy results of which are summarized in Table 0-14 case are presented in more detail.

Alternative discovered process models are shown below along with the implementation of conformance checking between each distinct discovered model and the initial provided event log. After conformance checking has been done, the exported from ProM (.csv) file can be given as an input to a feedforward neural network that will perform the outlier classification task.

Model 2

The second model that is discovered from the initial event log is seen in Figure 0-3. After applying to the .csv file (derived from ProM after conformance checking has been applied to this process model and the initial event log) a feedforward neural network with 2 hidden layers (12, 12 neurons respectively for each hidden layer) and executing it for 70 epochs I get the results seen in Table 0-5.
**Explanation of the overall approach**

As outlier traces are defined those that contain events that are present in the initial given event log, but not in the model that is discovered from it using the inductive visual miner algorithm.

**Table 0-5 Classification accuracy results of model 2**

<table>
<thead>
<tr>
<th>epoch</th>
<th>Accuracy</th>
<th>Validation accuracy</th>
<th>Loss</th>
<th>Validation loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/70</td>
<td>0.9551748432942237</td>
<td>0.96219494753933432</td>
<td>0.23671804665485296</td>
<td>0.070594642996402751</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>70/70</td>
<td>0.99856075256119559</td>
<td>0.99893194789804607</td>
<td>0.0038070931916954838</td>
<td>0.0037105354078496638</td>
</tr>
</tbody>
</table>

acc: 99.89%

The above simulation results suggest that the detection of the outlier traces was successful, since the overall outlier classification approach had an accuracy of almost 99%. This suggest that the classification results are trustworthy.

**Model 3**

The third model that is discovered from the initial event log is seen in *Figure 0-4*. After applying to the .csv file (derived from ProM after conformance checking has been applied to this process model and the initial event log) a feedforward neural network with 2 hidden layers (12, 12 neurons respectively for each hidden layer) and executing it for 70 epochs I get the results seen in *Table 0-6*.

![Figure 0-4 Model 3, derived from the initially discovered model](image-url)

**Table 0-6 Classification accuracy results of model 3**

<table>
<thead>
<tr>
<th>epoch</th>
<th>Accuracy</th>
<th>Validation accuracy</th>
<th>Loss</th>
<th>Validation loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/70</td>
<td>0.94473284275146951</td>
<td>0.94762508789847222</td>
<td>0.99899240367740194</td>
<td>0.17504203956879519</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>70/70</td>
<td>0.99919600660315067</td>
<td>0.99899240367740194</td>
<td>0.0046191038561415173</td>
<td>0.0077752228207141078</td>
</tr>
</tbody>
</table>

acc: 99.92%
Model 4
The fourth model that is discovered from the initial event log is seen in Figure 0-5. After applying to the .csv file (derived from ProM after conformance checking has been applied to this process model and the initial event log) a feedforward neural network with 2 hidden layers (10, 12 neurons respectively for each hidden layer) and executing it for 70 epochs I get the results seen in Table 0-7.

![Figure 0-5 Model 4, derived from the initially discovered model](image)

Table 0-7 Classification accuracy results of model 4

<table>
<thead>
<tr>
<th>epoch</th>
<th>Accuracy</th>
<th>Validation accuracy</th>
<th>Loss</th>
<th>Validation loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/70</td>
<td>0.97630699109635688</td>
<td>0.99296697766826547</td>
<td>0.18713461689843883</td>
<td>0.033178785848322338</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>70/70</td>
<td>0.99919600660315067</td>
<td>0.99899240367740194</td>
<td>0.0046191038561415173</td>
<td>0.0077752228207141078</td>
</tr>
</tbody>
</table>

acc: 99.92%

Model 5
The fifth model that is discovered from the initial event log is seen in Figure 0-6. After applying to the .csv file (derived from ProM after conformance checking has been applied to this process model and the initial event log) a feedforward neural network with 2 hidden layers (12, 10 neurons respectively for each hidden layer) and executing it for 70 epochs I get the results seen in Table 0-8.

![Figure 0-6 Model 5, derived from the initially discovered model](image)
Table 0-8 Classification accuracy results of model 5

<table>
<thead>
<tr>
<th>epoch</th>
<th>Accuracy</th>
<th>Validation accuracy</th>
<th>Loss</th>
<th>Validation loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/70</td>
<td>0.9735078988437084</td>
<td>0.99399472591731552</td>
<td>0.19101628836831941</td>
<td>0.030553247325582622</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>70/70</td>
<td>0.99978163142307797</td>
<td>0.9996756917676864</td>
<td>0.00084648498442517299</td>
<td>0.0010214697216295646</td>
</tr>
</tbody>
</table>

acc: 99.97%

Model 6

The sixth model that is discovered from the initial event log is seen in Figure 0-7. After applying to the .csv file (derived from ProM after conformance checking has been applied to this process model and the initial event log) a feedforward neural network with 2 hidden layers (10, 8 neurons respectively for each hidden layer) and executing it for 70 epochs I get the results seen in Table 0-9.

Table 0-9 Classification accuracy results of model 6

<table>
<thead>
<tr>
<th>epoch</th>
<th>Accuracy</th>
<th>Validation accuracy</th>
<th>Loss</th>
<th>Validation loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/70</td>
<td>0.96850526738178877</td>
<td>0.99385366243215179</td>
<td>0.2297601660762874</td>
<td>0.030557341729620442</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>70/70</td>
<td>0.99969229882342803</td>
<td>0.99969772110322053</td>
<td>0.0010173992297552192</td>
<td>0.0011187874775036122</td>
</tr>
</tbody>
</table>

acc: 99.98%

Model 7

The seventh model that is discovered from the initial event log is seen in Figure 0-8. After applying to the .csv file
(derived from ProM after conformance checking has been applied to this process model and the initial event log) a feedforward neural network with 2 hidden layers (14, 10 neurons respectively for each hidden layer) and executing it for 70 epochs I get the results seen in Table 0-10.

<table>
<thead>
<tr>
<th>epoch</th>
<th>Accuracy</th>
<th>Validation accuracy</th>
<th>Loss</th>
<th>Validation loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/70</td>
<td>0.95448004343880577</td>
<td>0.98984342857333785</td>
<td>0.18745018019241716</td>
<td>0.021195027847053582</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>70/70</td>
<td>0.99879897279320606</td>
<td>0.99903270753030582</td>
<td>0.0041045296592247477</td>
<td>0.0032842055074734037</td>
</tr>
</tbody>
</table>

acc: 99.89%

Model 8
The eighth model that is discovered from the initial event log is seen in Figure 0-9. After applying to the .csv file (derived from ProM after conformance checking has been applied to this process model and the initial event log) a feedforward neural network with 2 hidden layers (9, 12 neurons respectively for each hidden layer) and executing it for 70 epochs I get the results seen in Table 0-11.

<table>
<thead>
<tr>
<th>epoch</th>
<th>Accuracy</th>
<th>Validation accuracy</th>
<th>Loss</th>
<th>Validation loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/70</td>
<td>0.93762593686646334</td>
<td>0.99685629947349397</td>
<td>0.1932736237036507</td>
<td>0.011994635473196288</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>70/70</td>
<td>0.99978163142307797</td>
<td>0.99979848073548039</td>
<td>0.00082563744070014353</td>
<td>0.00075734738343107358</td>
</tr>
</tbody>
</table>

acc: 99.98%

Figure 0-9 Model 8, derived from the initially discovered model
Appendix F
Here the code that is turns the Prom output csv file into tabular data that can be provided to the constructed neural network for training is presented.

```
import csv

with open("ex.csv","rb") as f: # open file
    reader = csv.reader(f,delimiter="",) # read content
    my_list = list(reader) #put content in my_list

with open("ex2.csv","w") as f:
    wr = csv.writer(f, dialect='excel', lineterminator='\n')
    my_list = [[item.split(";") for item in lst] for lst in my_list]
    for i in my_list:
        wr.writerows(i) # unfold list

Catalog Appendix F1: Sort events by trace id

import csv
import collections
import sys

max_len = 185# maximum length of the largest trace

# read from prom output file
with open("ex2.csv","rb") as f:
    cr = csv.reader(f,delimiter="")
    d=collections.defaultdict(lambda : list())
    header=next(cr)
    # read title. Retrieve the next item from the iterator by calling its _next_() method.
    for r in cr:
        d[r[0]].append(r[1]+r[2])

#write trace id + trace
with open("ex3.csv","w") as f:
    cr = csv.writer(f,sys.stdout, lineterminator='\n')
    od = collections.OrderedDict(sorted(d.items()))#sort items based on dictionary key value
    for key, value in od.items():
        #print key,value
        prefix1 = ("tr")
        prefix2 = ("log")
        prefix3 = ("sync")
        for i in value[:]:
            if i.startswith(prefix1):
                value.remove(i)
```
cr.writerow([key, value])

# Print value

# Write trace with classification
with open("ex4.csv", "w") as f:
    cr = csv.writer(f, sys.stdout, lineterminator='\n')
od = collections.OrderedDict(sorted(d.items()))
# Sort items based on dictionary
key value
for key, value in od.items():
    # For k in value:
    if any(t.endswith(prefix2) for t in value):
        value = ["A" if x == "Async" or x == "ER Registrationlog" or
        x == "Amodel" else x for x in value]
        value = ["B" if x == "Bsync" or x == "ER Triagelog" else x for x in value]
        value = ["C" if x == "Csync" or x == "IV Liquidlog" else x for x in value]
        value = ["D" if x == "Dsync" or x == "ER Sepsis Triagelog" else x for x in value]
        value = ["E" if x == "Esync" or x == "CRPlog" else x for x in value]
        value = ["F" if x == "Fsync" else x for x in value]
        value = ["G" if x == "Gsync" else x for x in value]
        value = ["H" if x == "Hsync" or x == "Leucocyteslog" else x for x in value]
        value = ["I" if x == "Isync" or x == "LacticAcidlog" else x for x in value]
        value = ["J" if x == "Jsync" else x for x in value]
        value = ["K" if x == "Ksync" or x == "Release Blog" else x for x in value]
        value = ["L" if x == "Lsync" else x for x in value]
        value = ["M" if x == "Msync" or x == "Return ERlog" else x for x in value]
        value = ["N" if x == "Nsync" or x == "Release Elog" else x for x in value]
        value = ["O" if x == "Osync" or x == "Release Clog" else x for x in value]
        value = ["P" if x == "Psync" or x == "Release Dlog" else x for x in value]
        value = [ord(i) for i in value] + [0]*(max_len - len(value))
        # max_len = max(len(value)+1, max_len)
        cr.writerow(value+[1])
    # Print len(value)
    # Print value
    # Print max_len
else:
    value = ["A" if x == "Async" else x for x in value]
    value = ["B" if x == "Bsync" else x for x in value]
    value = ["C" if x == "Csync" else x for x in value]
    value = ["D" if x == "Dsync" else x for x in value]
    value = ["E" if x == "Esync" else x for x in value]
    value = ["F" if x == "Fsync" else x for x in value]
    value = ["G" if x == "Gsync" else x for x in value]
    value = ["H" if x == "Hsync" else x for x in value]
    value = ["I" if x == "Isync" else x for x in value]
value = ["J" if x == "Jsync" else x for x in value]
value = ["K" if x == "Ksync" else x for x in value]
value = ["L" if x == "Lsync" else x for x in value]
value = ["M" if x == "Msync" else x for x in value]
value = ["N" if x == "Nsync" else x for x in value]
value = ["O" if x == "Osync" else x for x in value]
value = ["P" if x == "Psync" else x for x in value]
#max_len = max(len(value), max_len)
value = [ord(i) for i in value] + [0]*(max_len - len(value))
# print max_len
# print value
cr.writerow([value+[0]])

Catalog Appendix F2: Convert event traces into normalized vectors of numbers

Appendix G
In this part, some background knowledge related to deep learning is being presented.

Machine learning (ML)
Machine learning can be considered as a field of computer science that provides computers with the ability to learn, without being explicitly programmed (Simon, 2013). There are three main learning techniques, each of which maps into a particular learning task. These are: Supervised, unsupervised and reinforcement learning (Brownlee, 2016).

Supervised learning
This type of learning is used by the majority of practical machine learning tasks. In supervised learning, given the input variables “X” and the output variable “Y”, the goal is to learn the mapping function from the input to the output “Y=f(X)”. This mapping function should be approximated so as to when new input data “X” is provided, the output variable “Y” for these data can be predicted.

The reason it is called supervised learning, is due to the fact that the process of an algorithm learning from the training dataset can be considered as a teacher that is supervising the whole learning procedure. The correct answers are already known, the algorithm makes predictions iteratively on the training data and is corrected by the teacher. At the moment that the algorithm achieves an acceptable level of performance, the learning process can stop.

Supervised learning problems can be grouped into classification and regression problems.

- A **classification** problem is when the output variable is a category, e.g. “red” or “blue”.
- A **regression** problem is when the output variable is a real value, such as “weight”.

Common types of problems that are built on top of regression and classification include time series prediction and recommendation respectively.
Some indicative examples of supervised machine learning algorithms are:

- Linear regression for regression problems.
- Random forest for classification and regression problems.
- Support vector machines for classification problems.

**Practical example**

1. A human builds a classifier based on input and output data (since the desired output is already known, there is no need to develop an algorithm that will create the classifier).
2. The classifier is trained with a training set of data.
3. That classifier is tested with a test set of data.
4. Deployment if the output is satisfactory.

Supervised learning should be used when the user knows how to classify the data, but needs the classifier to do the whole process. Put it otherwise, the user knows the desired classification for a given sample data set, but needs to apply a general classifier to all available data.

**Unsupervised Learning**

This learning pattern is described by a case that a user has only input data “X” and no corresponding output variables. Goal of the user is to model the underlying structure or distribution on the data in order to learn more about it. It is called unsupervised learning, because in contrast to what happens in supervised learning, there is no correct answer and there is no teacher. Algorithms are left on their own to discover and present notable structures in the data.

Unsupervised learning problems can be further grouped into clustering and association problems.

- A **clustering** problem is one that the user wants to discover the inherent groupings in the data, similar to grouping customers based on their purchasing behavior.
- An **association** rule learning problem is where the user wants to discover rules that describe large portions of the given data. One representative example can be considered the association in the buying behavior of customers. E.g. people that buy product X also tend to buy product Y.

Popular examples of unsupervised learning algorithms are:

- K-means for clustering problems.
- Apriori algorithm for association rule learning problems.

**Practical example**

1. A human builds an algorithm based on input data.
2. That algorithm is tested with a test set of data (in which the algorithm created the classifier).
3. If the classification performed by one developed algorithm is not useful, another algorithm that eventually leads to an alternative classification should be built.

---

27 http://stats.stackexchange.com/questions/144154/supervised-learning-unsupervised-learning-and-reinforcement-learning-workflow
4. Deployment if the classifier is satisfactory.

To be used when the user has no idea of how to classify the data and the algorithm creates a classifier for the user.

**Reinforcement Learning**

Models that use the reinforcement learning model actually rely on the reward/punishment method. This makes it possible to explore and memorize the states of an environment or the actions using a way that bears huge resemblance with the way a human brain works. Only useful information is used in order to predict the reward. Information that is not considered useful is blocked.

**Practical example**

5. A human builds an algorithm based on input data.
6. That algorithm presents a state dependent on the input data in which a user rewards or punishes the algorithm via the action the algorithm took, this continues over time.
7. That algorithm learns from the reward/punishment and updates itself.
8. It is always in production, it needs to learn real data to be able to present actions from states.

To be used when the user does not know how to classify the data and uses a reward or punishment method depending on whether the classification was correct or false respectively.

**Example of a neural network using Back propagation for training**

In *Figures 0-18, 0-19*, a neural network that uses Back propagation of error for training is visible. In *Figure 0-18*, the network is depicted without any activation and in *Figure 0-19* after it has been trained to map a set of specific inputs into one of specific outputs. It is a fully connected feed forward neural network composed of three layers: one input (layer 1), consisting of 4 nodes, one output (layer 3) consisting again of four nodes and one hidden layer (layer 2) of five nodes. Each number in a node represents the activation of this specific node. In other words it defines the output of that node, given an input, or a set of inputs. The different colors in the bottom of each arc (that have the form of small discs) depict the weights of each one. Red color in a synapse corresponds to a positive weight, while blue in a negative one. The strengths of the synapses correspond to the size of these discs. This task is actually a pattern association one, where one set of vectors (input vectors) is associated with another one (output vectors). The activation function for the nodes of the first layer is the linear one and for the nodes of the second and third layer, the logistic one. With the use of this network that has three layers (one input of four nodes, one hidden of five and one output of four) a mapping is being achieved between an input and output sequence as follows:
This neural network is being trained with the use of backward propagation of error that is used in conjunction with the gradient descent method\textsuperscript{28}. With the term “error”, the difference between current and expected output is meant. The algorithm repeats a two phase cycle that consists of backwards propagation of error and weights update. When the first input vector is being fed into the network it is propagated forward through it layer by layer until the output layer is being reached. The output of the network is then compared with the

\begin{table}
\centering
\begin{tabular}{|c|c|}
\hline
Input & Target \\
\hline
1000 & 0001 \\
0100 & 0010 \\
0010 & 0100 \\
0001 & 1000 \\
\hline
\end{tabular}
\caption{Mapping input to output}
\end{table}

\textsuperscript{28}Gradient descent is an iterative optimization algorithm. To find a local minimum of a function with the use of gradient descent, steps proportional to the negative of the gradient of the function at the current point are being taken. It is used here to find the local minimum of the error function.
desired output using a loss function\textsuperscript{29} and an error value is calculated for each of the neurons in the output layer. Then the error values are propagated backwards, starting from the output, until each neuron has an associated error value which represents its contribution to the original output. Back propagation uses these values of the error to calculate the gradient of the loss function with respect to the weights of the network. Accordingly this gradient is fed into optimization method, which in turn uses it to update the weight in order to minimize the loss function.

In Figure 0-20, the values that the error function takes during the training of the network shown in Figures 0-18 and 0-19 are visible. While in Figure 0-19 all of the weights have the same value (all discs are of the same size and color), in Figure 0-18 it can be seen that both their color and size has changed. This has occurred, since this network was trained to be able to make a certain mapping between input and output sequences. In the given example approximately 450 iterations where sufficient for the error to reach its lowest value (zero). Zero error means that the network, when provided with any one of the possible four input sequences in its input layer, can provide the corresponding target value for each one in the output layer.

Since Back propagation requires the desired output for each input value in order to calculate the loss function gradient it can be considered to be a supervised learning method.

\textsuperscript{29} A loss function is a function that maps an event of values of one or more variables onto a real number, representing a cost associated with the event.
Appendix H

In this part, some background knowledge related to NNs is being presented.

Structure

ANNs can consist of multiple layers which can lead a signal from the front layer to the layer in the back. In the case of back propagation example as seen in Figure 0-21 the forward run of the network is being used to reset the weights on the front neural units. This is sometimes done in combination with training where the correct result is known. Modern networks are considered to be a bit more free flowing in the sense that connections interact in a much more complex way. This complexity originates from the fact that modern neural networks when comparing with the classical ones have many more hidden layers.

This increased number of hidden layers leads to much more complex network structures, and therefore to many more alternative connection patterns between the nodes located in those extra hidden layers. Researches constantly discover new patterns in the construction of artificial neural networks. A modern approach is to use connections that extend a lot further and link processing layers rather than be always localized to adjacent neurons. Artificial neural networks have their base on real numbers, with the value of the core and of the axon being in the range between 0.0 and 1. Dynamic neural networks are the most advanced ones, in the sense that they can form new connections and new neural units while disabling some others. The process of forming and disabling transitions is each time based on rules.

Neural network topologies refer to the arrangement of layers in the network. For a normal feedforward neural network the topology would consist of many fully connected layers. That is, every neuron in the network is connected to every neuron in adjacent layers, as seen in Figure 0-21. They tend to be straightforward networks

---

30 https://www.researchgate.net/post/What_is_the_difference_between_the_classical_artificial_neural_network_and_the_new_deep_learning_generation
that associate inputs with outputs and don’t contain any feedback loops, meaning that the output of any layer does not affect the same layer. They are extensively used in pattern recognition and are suitable for modeling relationships between a set of predictor or input variables and one or more response or output variables. Put it otherwise, they are appropriate for any functional mapping problem where we want to know how a number of input variables affect the output variable. The multilayer feed forward neural networks, also called multi-layer perceptrons (MLPs) are the most widely studied and used neural network model in practice. They consist of multiple layers of computational units and each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply as activation function the sigmoid. The number of layers and the number of processing elements per layer are important decisions. These parameters to a feed forward back propagation topology are also the most important ones. There is no quantifiable best decision to the drawing of the layout of the network for any particular application. There are however general rules developed over time that most engineers apply to their problems:

4. As the complexity in the relationship between the input data and the desired output increases, the number of the processing elements in the hidden layer should also increase.
5. If the process being modeled is separable into multiple stages, then additional hidden layer(s) may be required. If the process is not separable into stages, then additional layers may simply enable memorization of the training set, and not a true general solution effective with other data.
6. The amount of training data available sets an upper bound for the number of processing elements in the hidden layer(s). To calculate this upper bound the number of cases in the training data set should be divided by the sum of the number of nodes in the input and output layers of the network. Then this number should be divided again with a scaling factor between five and ten. Larger scaling factors should be used for relatively noisy-free data. If too many artificial neurons are used, the training set will be memorized and in this case generalization of the data will not occur, thus making the network useless on new data sets.

However as we move to convolutional and recurrent networks, their structure becomes more complex as local connectivity becomes involved.

**Convolutional neural networks (CNNs),** as the one shown in Figure 0-22, are used when the size of a general fully connected feedforward net becomes unmanageable. They are particularly useful in signal processing tasks, such as image processing, computer vision tasks, such as handwritten text recognition natural object classification and segmentation of natural images. Finally, they are also useful in speech recognition applications. They use several layers of convolutions (convolutional layers) with nonlinear activation functions like ReLu, sigmoid, or tanh applied to the results. A convolutional layer is a layer of neurons that is connected to other layers of neurons, in which not all of its nodes are connected to its neighbor layers (it is not a fully

---

connected layer). These layers are used when the size of a fully connected NN becomes unmanageable. In a feed forward neural network each input neuron is connected to each output neuron in the next layer. This is also called a fully connected layer. In CNNs instead of this, convolutions are used over the input layer to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output (or in other words, a limited amount of neurons of the input is connected to a neuron of the output). The region of the input neurons that is connected to one neuron of the output is called “stride” of the convolutional layer. In Figure 0-23, the architecture of a convolutional network is seen, with the network containing both convolutional (1st hidden and 2nd hidden layers) and fully connected layers (3rd hidden layer).

A bias is an input that is an additional entrance to the neurons of a layer. The main function of a bias is to provide every node with a trainable constant value (in addition to the normal inputs that the node receives). The output of a convolutional layer of neurons is called “feature map” and with one input one can take several feature maps. Each layer of the network applies different filters and their results are combined. During the training phase a CNN learns automatically the values of its filters based on the task that needs to be performed. In the example of image classification a CNN can learn to detect edges from raw pixels in the first layer, then use these edges to detect simple shapes in the second layer and then use these shapes to detect higher level features, such as facial shapes in higher layers. The last layer is then used as a classifier that uses these high-level features. Puling (or downsampling) is a process that reduces the number of neurons, or the resolution of the network, as we progress through it and is most often being applied by taking the maximum (in other cases the
average value of the outputs is being used) of a number of outputs neurons of a layer and by giving it as input to the next layer. This way the number of neurons is being reduced. Pooling operations are being used when we move from one convolutional layer to another and in this way the size of convolutional layers is being reduced. It may be applied not only inside one layer, but also between layers (or between feature maps). When the number of neurons in a layer becomes small and the number of feature maps becomes also small, a fully connected neural network can be applied in order to achieve the desired output (Kim, 2014).

Recurrent neural networks (RNNs), as the one shown in Figure 0-24, can have signals traveling in both directions by introducing loops in the network. Feedback networks are powerful and can get extremely complicated. Computations that are derived from earlier input are fed back into the network, which gives them a kind of memory. They are dynamic, meaning that their state can change continuously, until they reach a point of equilibrium and remain into this equilibrium point until the input changes and a new equilibrium needs to be found. They take static inputs in time and give an output. If the same input in time is given the same output will be produced and are able to learn and remember sequences, since they are able to remember the past.

![Typical RNN architecture](olah.png)

Figure 0-16 Typical RNN architecture (Olah, 2015)

One major problem that RNNs have to solve is the long term dependencies problem, which is the problem where the network might need to connect previous information to the current task (e.g. using previous video frames might inform the understanding of the present frame). Some types of recurrent neural networks include:

- **Hopfield Network**: It is not designed to process sequences of patterns, but it requires stationary inputs and is an RNN in which all connections are symmetric.
- **Boltzmann machine**: It can be thought of as a noisy Hopfield network and is one of the first NNs to demonstrate learning of latent variables (Hidden units).
- **Simple recurrent network**: It is based on simple feedforward network, with the addition of feedback loops to it. At each time step, the input is propagated in a standard feedforward fashion and then a simple backpropagation-like learning rule is applied (this rule however is not performing proper gradient
descent). The fixed back connections result in the context units always maintaining a copy of the previous values of the hidden units (since they propagate over the connections before the learning rule is applied) (Holk Cruse, 2006).

- **Echo state network (ESN):** The ESN is a recurrent neural network with a sparsely connected random hidden layer. The weights of output neurons are the only part of the network that can change and be trained. They are good at reproducing certain time series (Jaeger, 2004).

- **Bi-directional RNN:** The use a finite sequence to predict or label each element of the sequence based on both the past and the future context of the element. This is done by adding the outputs of two RNNs: one processing of the sequence from the left to right, the other one from right to left. The combined outputs are the predictions of the teacher-given target signals (Graves A., 2005).

- **Long short-term network (LSTM):** It is an artificial neural network structure that in contrast with the traditional RNN doesn’t have the vanishing gradient problem. LSTM RNNs perform particularly well in sequence learning methods and in various applications such as language learning and handwriting recognition (Gers, 2001).

- **Hierarchical RNN:** Hierarchical RNN’s elements are connected in various ways to decompose hierarchical behavior into useful subprograms.

- **Stochastic neural networks:** They differ from typical neural networks since they introduce random variations into the network. In a probabilistic view of neural networks, such random variations can be viewed as a form of statistical sampling.

### Usability

As goal for each artificial neural network can be considered for it to be able to solve problems by imitating the way the human brain would solve them. Despite this fact, all of them are much more abstract, since it is impossible to bear a close resemblance to the complexity of the human brain. Modern artificial neural networks used to solve problems can use up to several millions of neural units and connections, which comparing with the human brain is several orders of magnitude less complex.

One drawback of artificial neural networks is that it is difficult to predict whether the training process is going to lead in success or in failure. After training some of them become capable of solving problems, while some others not. Training a neural network requires several thousands of iterations.

Neural networks can be used to solve a wide variety of tasks, such as speech recognition and computer vision, which are hard to solve using rule-based programming.

### Learning Algorithms

The process of training of a neural network actually implies the process throughout which the network can generalize well (ability to classify previously unseen patterns) when provided with new input data based on the training it had during the learning process. After the structure of a network has been defined (feed forward or recurrent and number of layers and nodes), the network is ready to be trained. In supervised learning training,

---

33 http://neuralnetworksanddeeplearning.com/chap5.html
34 http://ai-depot.com/Tutorial/RuleBased.html
the network’s calculated values for the output nodes are compared to the “correct” values of the nodes and an error term is calculated for each node. These error terms are then used to adjust the weights in the hidden layers, so that the next time the output values are going to be closer to the “correct” values. A key feature of neural networks is an iterative learning process in which data cases are provided to the network one at a time and the weights that are associated with the input values are adjusted each time. After all classes are presented, the process starts over again. During all this learning phase the network learns by adjusting the weights so as to be able to predict the correct class label of the input samples out of all the possible classes. The main algorithms that are used for training artificial neural network models include some form of gradient descent and back propagation to compute the size of the actual gradients. This can be done by taking the derivative of the cost function with respect to the network parameters and then changing those parameters in a gradient-related direction. In the beginning of this process the initial weights are randomly chosen and then the training process begins. The network processes the records of input values in the training data one at a time, using the weights and functions in the hidden layers and then compares the resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights for application to the next record to be processed. This process occurs continually as the weights are continually adjusted. During the training of a network the same set of data is processed many times as the connection weights are being refined. There are however some networks that never learn. This can be due to the fact that the input data does not contain the specific information from which the output is derived. Networks also do not converge if there is not enough data to enable complete learning. Ideally there should be enough data so that part of the data can be held back as a validation data set.