NEXT-ACTIVITY PREDICTION WITH LONG-SHORT-TERM MEMORY RECURRENT NETWORKS

BATCH CONSTRUCTION STRATEGIES FOR TRACES AND A COMPARISON OF NETWORK ARCHITECTURES

VORHERSAGE DER NÄCHSTEN PROZESSAKTIVITÄT MITTELS NEURONALER NETZE

FELIX WOLFF

MASTER'S THESIS

JANUARY 25, 2019

SUPERVISED BY

Prof. Dr. Mathias Weske
Dr. Luise Pufahl
Dr. Haojin Yang

BUSINESS PROCESS TECHNOLOGY GROUP

January 25, 2019

Felix Wolff: *Next-Activity Prediction With Long-Short-Term Recurrent Networks,* Batch Construction Strategies For Traces And A Comparison of Network Architectures, © January 25, 2019

Prediction is very difficult,
especially if it's about the future

— Niels Bohr

ABSTRACT

Manual labor processes have seen a lot of automation in the past, but knowledge-intensive processes still proceed in a highly manual fashion. Systems to assist knowledge workers have been asked for in recent literature reviews. Such assistance systems need to anticipate the development of a case to offer assistance in the right circumstances, which requires the capability to foresee the next activity in a process. Predicting the next activity in a running process is a young discipline called Predictive Process Monitoring, which we contribute to by evaluating neural network prediction models for this task.

In this work, we connect next-activity prediction to sequence prediction. Thanks to this connection, we can adopt a prediction model from a natural language processing competition where the next word in a sentence was to be predicted. Additionally, we augment this prediction model with a different feature set to produce a second approach. To have a direct comparison, we reimplement two published prediction models for business processes. Because several strategies are common to create batches from sequential data of variable length, we include a comparison of these strategies in our evaluation. To make model training easier and facilitate the inclusion of the strategies into the training process, we also propose a model training framework. The framework allowed easy training of the four models with four different batch creation strategies on eight datasets.

In the evaluation, we note a connection between process complexity and prediction accuracy. Furthermore, we realize that one batching strategy delivers the most promising results, and should be explored further. Finally, we compare our accuracies with numbers from three recent next-activity prediction approaches on two different real-life process event logs. All of our four models outperform these numbers.

GERMAN ABSTRACT

Wissensarbeit läuft größtenteils manuell ab. Assistenzsysteme mit unterstützender Funktion für Wissensarbeiter werden daher häufig gefordert. Diese Systeme müssen die Entwicklung eines Prozesses vorhersehen können, um effektiv zu sein. Dazu zählt auch, die unmittelbar nächste Prozessaktivität zu antizipieren. Just diese Fähigkeit versuchen wir mittels neuronaler Netze zu verbessern.

Wir verknüpfen die Vorhersage von Prozessaktivitäten mit der Vorhersage von Sequenzen, und können so ein auf Sätzen erprobtes neuronales Netzwerk anpassen. Zusätzlich erstellen wir eine zweite Version des Netzes, die eine angepasste Form von Daten verarbeitet. Weiterhin reproduzieren wir zwei bereits veröffentlichte Netzwerke zur Vorhersage der nächsten Prozessaktivität, um eine Vergleichsbasis zu schaffen. Während der Implementierung fielen verschiedene Strategien auf, aus sequentiellen Daten verschiedener Länge Batches zum Training neuronaler Netzwerke zu schaffen. Wir möchten zum Verständnis dieser Strategien beitragen, und beziehen ein Vergleich in die Arbeit ein. Um das Trainieren der Netzwerke mit den verschiedenen Strategien zu vereinfachen, präsentieren wir im Laufe der Arbeit ein Trainings-Framework. Mittels dieses Frameworks können wir vier verschiedene Netzwerke auf Basis vier verschiedener Strategien auf acht verschiedenen Datensätzen trainieren.

In der Auswertung stellen wir einen Zusammenhang zwischen Komplexität und Genauigkeit der Vorhersagen fest. Des Weiteren stellt sich eine der Batch-Erstellungsstrategien als besonders gut heraus, und sollte erweitert werden. Abschließend vergleichen wir unsere Ergebnisse mit drei Arbeiten auf Basis zwei Prozess Logs. Auf beiden Logs erreichen alle unserer vier Modelle eine deutlich höhere Genauigkeit.

ACKNOWLEDGEMENTS

Before becoming all technical, I would like to express my sincere gratitude to the many people who were involved in the writing of this thesis and helped me get to the finish line of my master's studies.

I was given a lot of freedom to choose the course of my research and my topic, and I am thrilled that I could pursue it this way: A bit more than a year ago, I was seriously contemplating about quitting my studies at HPI for a university that would have allowed me to learn more about machine learning. Thanks to this thesis, I could not only remain at HPI, but also pursue my learning goal, and learn a lot about the structure and wording of a scientific paper along the way. My supervisors Luise Pufahl and Haojin Yang were always there to assist me with literature work and technical advice.

A range of people helped me in small but essential ways: Christian Bartz helped me choose a suitable deep-learning framework. The authors Stefan Schönig and Richard Jasinski provided help in reverse-engineering the comparison models and were also helpful in offering advice. Willi Gierke helped me with some fundamental questions on LSTM. Bernhard Rabe and Tobias Papke kept the FutureSOC running while I was doing my experiments.

Furthermore, I am happy to have had several pairs of watchful eyes proofread this document: Erick Godínez, Thomas Goerttler, Marvin Bornstein, Stephan Detje, Georg Berecz, and Luise Pufahl.

Finally, I would like to express deep thankfulness to the following parties:

Stephan Lessmann, for his *Business Analytics and Data Science* lecture, that sparked in me the interest to pursue this topic. Without it, I would not have found an interest in analytics.

Hasso Plattner, for founding the institute that has made so great of an impact on my life, it could not be summed up on a single page. I am very thankful to him, and I hope that I get the chance to express my gratitude personally one day.

Important people stood in the background since my start at HPI in 2012 and provided advice, infrastructure, warmth, and every student's favorite: free money. Dear Caroline, Stefan, and Adrian Wolff, dear Gerda Mikus - thank you for sharing life with me the way you do.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SOURCE CODES

ACRONYMS

ANN Artificial Neural Network

API Application Programming Interface

BPIC Business Process Intelligence Competition

DFA Deterministic Finite Automaton

EVM Reimplementation of the approach by Evermann et al. [23]

LSTM Long Short-Term Memory

MAPE Mean Average Percent Error

NLP Natural Language Processing

PFS Bipartite approach that uses subsequence-encoding features

SCH Reimplementation of the approach used by Schönig et al. [64]

SGD Stochastic Gradient Descent

SPiCe Sequence Prediction Challenge

SP2 Bipartite approach that uses SP-2 features

SVR Support Vector Regression

SVM Support Vector Machine

UML Unified Modeling Language

XES eXtensible Event Stream

1 INTRODUCTION

I choose a lazy person to do a hard job,
because a lazy person will find an easy way to do it.

— Bill Gates

Since the dawn of time, humanity has always tried to find ways to make work easier. With the start of the industrial revolution, this was achieved by constructing machines and other helpful contraptions which automated a task. In more recent years, a guiding procedure - a process - was installed to streamline the workflow where complete automation was not possible. With the rise of the credo "what gets measured, gets managed", key performance indicators (KPIs) were installed into manual labor processes to optimize process efficiency further [18].

However, processes and KPIs fell short of penetrating and improving the realm of knowledge work, since the foundational mechanics of it differ strongly from manual labor. Peter F. Drucker, a renowned management educator, realized this in 1999: "The most important contribution of management in the 20th century was to increase manual worker productivity fifty-fold. The most important contribution of management in the 21st century will be to increase knowledge worker productivity — hopefully by the same percentage" [20].

In the spirit of Drucker's statement and by leveraging the technical developments of the years since his statement, we apply deep learning on business process execution history to predict the next activity in a running business process. We focus on deep learning methods because they have been shown to outperform others in this use case [71]. How this is a step toward process automation in knowledge work, will be explained further in Section 1.1. We highlight the resulting contribution to current research in Section 1.2. The chapter ends in Section 1.3 with a description of the thesis outline.

1.1 MOTIVATION

Automation of work is a phenomenon that has occurred in the past with manual labor and is spreading into other types of work today, namely knowledge work. As mentioned before, manual labor and knowledge work differ:

The course of manual labor is determined by physical laws, is often very structured, and thus offers great potential for simple automa-

tion - like work at an assembly line. Knowledge work, on the other hand, requires workers to "think for a living" and is strongly shaped by the individuality of the thoughts and habits of each knowledge worker [20]. As each worker has a different knowledge background and uses information differently, this type of work is very flexible, and often no process is executed exactly the same [38]. Popular examples of knowledge work are insurance claims, handled by several employees. Each claim requires a different course of action since the information contained in each case differs.

The following 20 years since Drucker's mission statement saw the analog tools of knowledge workers evolve into a plethora of digital systems and applications. These help make their decisions more informed and faster. Furthermore, these systems also track work progress, resulting in a large number of logs which document the course that work on a running process has taken. Running processes are also referred to as cases.

These logs are a valuable source of information as they can reveal best practices that knowledge workers use in certain situations. An assistance system could work with this data and recommend such best practices in specific circumstances. These knowledge worker assistance systems have been called for in recent literature reviews by Hauder et al. [35] and Francescomarino et al. [26], but have only been implemented prototypically until now. One of the challenges that lie in the way of creating assistance systems is the task of anticipating the development of a case and foreseeing which part comes next. Once then a machine understands the *what* and is only missing the *how*, the ability to forecast process developments can be interpreted as a step toward process automation.

Forecasting the course of a running process can be useful for workers doing either manual labor or knowledge work. Questions of time and outcome dominate manual labor because the course of work is clear - natural interests in a world of distributed supply chains and just-in-time production [45]. The course of the work itself is of importance in the case of unstructured knowledge work, which is often unclear and depends on the information handled inside a case [25]. If a case were to progress in an unwanted fashion, knowledge about this development would present workers with an opportunity to intervene.

By regarding the execution history of an ongoing case, the prediction of its next activity is possible - illustrated in Figure 1. This prediction could then be processed further by an assistance system to propose an intervention if a case takes an unwanted course, for ex-

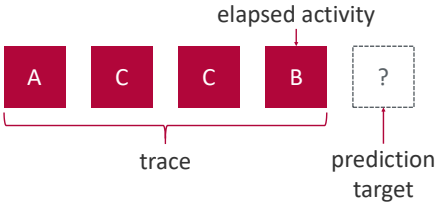


Figure 1: The next activity of a case is predicted from the sequence of elapsed activities in its trace

ample. The application of Predictive Analytics on business processes enables this type of prediction. It is relatively new and is commonly referred to as predictive process monitoring.

We realized that the current research in this young domain could be improved in three areas. These will henceforth be referred to as improvement areas:

Area 1 NLP-Influence: The history of a case is essentially a sequence, but little inspiration was taken from natural language processing (NLP), where sequence prediction is common [59, 66].

Area 2 Reproducibility: Divergent approaches to training the prediction models and low technical depth of the publications make it hard to reproduce the published results. Furthermore, some prediction approaches focus on whole event streams without specific regard to a case [23, 64].

Area 3 Comparability: Published approaches often use different datasets, making them hard to compare.

1.2 CONTRIBUTION

In this work, we make a contribution to ongoing research by applying neural networks to process execution logs to predict the next activity. The three improvement areas guided the work.

We address Area 1 by reusing a successful sequence prediction approach from the NLP domain and applying it on business process log data. We also adapt it to incorporate learnings from a paper on feature engineering for predictive process monitoring [48]. We compare the two models to reimplementations of two published approaches. Doing so, we ensure the direct comparability mentioned in Improvement Area 3. The models were reverse-engineered from the following publications:

1. *A Deep Learning Approach for Predicting Process Behaviour at Runtime* by Evermann et al. [23]

2. *Deep Learning Process Prediction with Discrete and Continuous Data Features* by Schönig et al. [64].

During reverse engineering, we uncovered widely divergent understandings of batch construction for neural networks from sequential data. To contribute to a better understanding in this area, we include a comparison of four different batching strategies into the evaluation.

As we remarked the comparability of current publications in Improvement Area 3, we include a training framework in this thesis. It allowed us to train four different models with four different batching strategies on eight prepared datasets with ease. It is extensible for other researchers to facilitate comparisons in future works.

We use the datasets from the Business Process Intelligence Competition (BPIC) years 2011, 2012, and 2015, where the latter consists of five datasets. The trace characteristics in each dataset differ. From these traits and the different accuracies, we infer a suggestion on which batching strategy could be preferable to use on which flavor of process log. The additional use of the HelpDesk log allows for a comparison to published next-activity prediction approaches.

We evaluate the trained models using three criteria: First, total accuracy. Second, we test for accuracy stability across the whole execution of the process, which we believe is vital for putting trust into the model [15, 25]. Third, we investigate training time requirements.

From the three evaluation criteria, we reason about the effectiveness of certain model-batching-strategy combinations.

1.3 THESIS OUTLINE

Chapter 2 delivers the necessary background on Process Science and Predictive Analytics. Furthermore, it provides information on Predictive Model Development as well as details on the inner workings of the used type of neural network.

Chapter 3 gives an overview of current approaches to the prediction task at hand, highlighting the achievements and technicalities of each publication. Furthermore, it presents works on the problem of sequence prediction in the field of natural language processing (NLP). For each publication used as comparison, implementation details are discussed.

In Chapter 4, we show how sequences relate to business processes. We then go on to adopt a winning approach from an NLP sequence prediction competition for predictive process monitoring.

In Chapter 5, we describe the various possible batch construction strategies that we perceived during the implementation of the predictive models and the exchange with Jörg Evermann and Stephan Schönig. We expand this technical issue into a simple training framework that makes it possible to compare all models and batching strategies side-by-side on all datasets. The framework permits future re-

searchers to train sequence prediction models faster and more consistent.

We bring both of the contributions mentioned above together in Chapter 6. This chapter presents the methodology we followed to obtain our results. It includes a description of the process logs, why we chose them, and how we prepared them for model training. Furthermore, we include a description of our technology landscape.

In Chapter 7 we present the results obtained from our experiments, and the insights gained. As described in the previous section, we not only focus on total model accuracy, but also the stability of the predictions as a case progresses and more history becomes available.

The thesis ends in Chapter 8, where we summarize the findings and the accomplishments of this thesis. Also, we give pointers with which to extend this work and carry predictive process monitoring forward.

2

BACKGROUND

To predict the next activity in a running case, we bring together the domains of process science, predictive modeling and deep learning. This background chapter gives comprehensive insights into the parts of each domain that are used and referenced throughout this thesis. The chapter approaches them in a top-down fashion, starting with the high-level perspective from process science and how process mining is distinguished from predictive process monitoring in Section 2.1. The section ends with information on how the process logs usually come into existence and which requirements these logs need to fulfill to be used in a prediction scenario. With the data origin explained, Section 2.2 introduces a process along which predictive model development typically takes place. Section 2.3 gives an in-depth look at neural networks and the components relevant to this thesis. Long short-term memory is also explained in detail. Finally, Section 2.4 introduces the peculiarities encountered when dealing with sequential data and introduces popular approaches for feature engineering.

2.1 PROCESS SCIENCE

Business processes are a collection of structured tasks which serve a particular business goal if ordered in a specific sequence [79]. To facilitate a common understanding, processes are visualized with modeling languages [54]. Most languages, for example the Business Process Modeling Notation (BPMN) [6], model the process flow in steps, commonly referred to as *activities* - shown in Figure 2. An instance of a process is referred to as *case*.

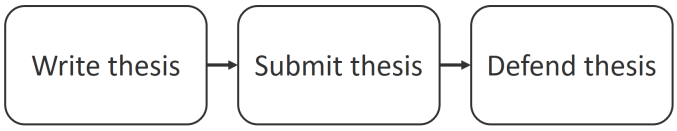


Figure 2: The boxes and their description are used in BPMN to describe an activity, the arrows inbetween denote the control flow

Process science, as loosely defined by van der Aalst [3], refers to the "broader discipline that combines knowledge from information technology and knowledge from management sciences to improve and run operational processes".

With the help of workflow management systems (WFMS), it is possible to embed and enforce business processes in an organization. When an activity in a process is executed, it goes through different

stages. These stages are captured in the activity lifecycle in Figure 3. WFMS log state transitions related to each activity as events. The types of captured events are specific to the modeling language and the execution environment of the WFMS.

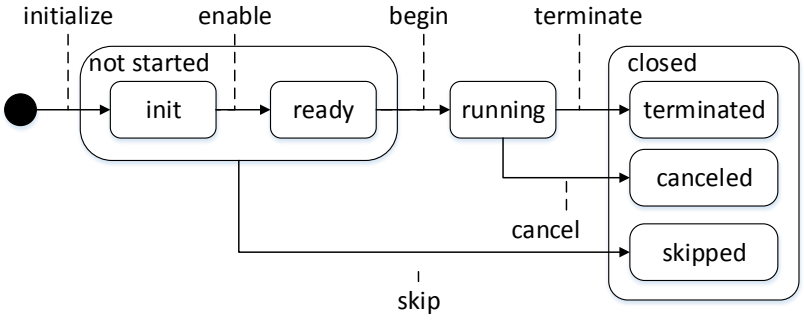


Figure 3: The activity lifecycle

The event logs give rise to the disciplines of process mining and predictive process monitoring. These disciplines revolve around analyzing the information contained in those logs [3]. Because of the paramount importance of logs for this work, we begin this chapter a definition of the properties and structure of a log in Section 2.1.1. The following Section 2.1.2 and Section 2.1.3 contrast process mining and predictive process monitoring.

2.1.1 *Process Logs*

A process log tracks the execution history of a process and can be understood in a hierarchical manner. Each log of a process consists of cases, which represent instances of a process. A case itself consists of a trace of events, i.e., a sequence that only belongs to a single case. Each event can have attributes attached [3].

Process event logs are typically made available via XES files. XES is an abbreviation for the *eXtensible Event Stream* standard [31]. An excerpt from a case extracted from the BPIC12 XES file [10] is displayed in Figure 4. In this particular excerpt, a variety of state transitions are tracked, as evidenced by the values in the `lifecycle:transition`

lifecycle:transition	concept:name	time:timestamp	org:resource
COMPLETE	A_SUBMITTED	2011-10-01 00:38:44.546000+02:00	112
COMPLETE	A_PARTLYSUBMITTED	2011-10-01 00:38:44.880000+02:00	112
COMPLETE	A_PREACCEPTED	2011-10-01 00:39:37.906000+02:00	112
SCHEDULE	W_Completeren aanvraag	2011-10-01 00:39:38.875000+02:00	112
START	W_Completeren aanvraag	2011-10-01 11:36:46.437000+02:00	NaN

Figure 4: An excerpt from a BPIC12 case

column.

Formally, a log L is defined as follows [3]. It is made up of cases $c \in \mathscr{C}$, where $\mathscr{C}$ is the case universe. Similar to events, cases have attributes, e.g., a name. The operator $\#_{attr}(c)$ returns the value of the attribute $attr$ on a case c. The function $attribute\_names$ lists the names of all attributes that an event possesses. A mandatory attribute of any case is its trace $\#_{trace}(c)$, which is especially relevant for our work. Each trace represents a finite sequence of events $e \in \mathscr{E}$, where $\mathscr{E}$ is the set of all events.

$$\#_{trace}(c) = \langle e_1, e_2, e_3 \cdots e_n \rangle$$

The events are ordered by timestamp. In turn, each event e has certain attributes associated with it. It is assumed that for a specific process, the type, number, and name of these attributes do not change. An example for common attributes are the activity name $\#_{activity}(e)$ or the timestamp $\#_{timestamp}(e)$ at which the event occurred.

Applied to the example in Figure 4, $e = \#_{trace}(c)[4]$ would get the fourth event in the trace, and $\#_{concept:name}(e)$ would produce the activity name `W_Completeren aanvraag`. Calling $attribute\_names(e)$ would yield `lifecycle:transition`, `concept:name`, `time:timestamp`, and `org:resource`.

These operators enable working with log data, which is at the core of process mining. We introduce this domain in the next subsection.

2.1.2 *Process Mining*

Process mining covers the three steps of process model discovery, conformance checking and process model enhancement [3] - all based on process event logs. It is worth noting that the logs can be enriched with data from systems other than WFMS. The three steps are enabled by techniques from Data science, making it possible to understand process mining as a link between data science and process science. This is also visualized in Figure 5.

A major problem when discovering process models is posed by spaghetti and lasagna process models [3]. These overly complex and unreadable models are mined from process logs that contain highly variable execution traces. Process mining mainly revolves around process models and the data-driven discovery and optimization based on historical data. For this reason, a possibility to act on case developments in real-time without model-imposed restrictions is desirable. Predictive process monitoring provides this possibility.

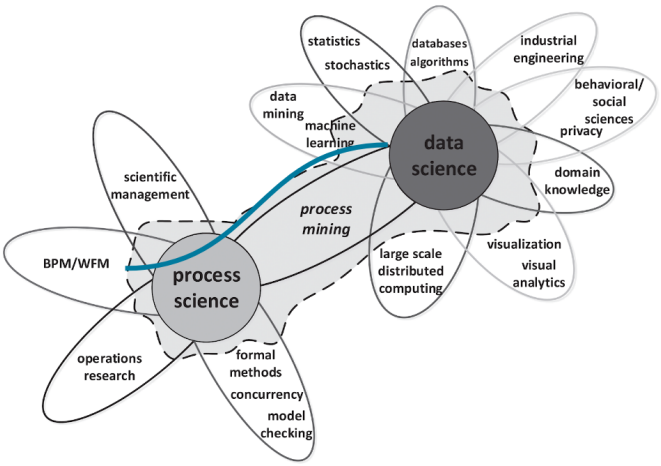


Figure 5: Process mining connects process science and data science - the blue line symbolizes the subdomains that predictive process monitoring brings together. Illustration taken from [3, p.18]

2.1.3 *Predictive Process Monitoring*

Predictive process monitoring is aimed at the use of online data to make statements about the progression of a running case [25, 64]. It is not based on process models, but predictive models and thus provides a flexibility that is not available within model-based process mining. As visualized in Figure 5, predictive process monitoring connects two sub-domains of process science and data science. This makes it a discipline within process mining [3]. With predictive process monitoring, questions such as the following can be answered:

1. Can the service level agreement still be met?
2. How long is this case still going to take?
3. What is going to be the next step in the case?

The answers to these questions can give case workers the opportunity to intervene if a case takes an unwanted course or might fail to meet requirements. Furthermore, this approach only requires sufficient amounts of logged case executions to train the predictive models. What this entails is explained in the following chapter.

2.2 PREDICTIVE MODELING

Predictive modeling is a domain that deals with creating models from learned statistical properties of data to predict outcomes [67]. These models are referred to as predictive models and are not related to process models. This section introduces a common process for predictive model generation in Section 2.2.1. As we aim to transfer a

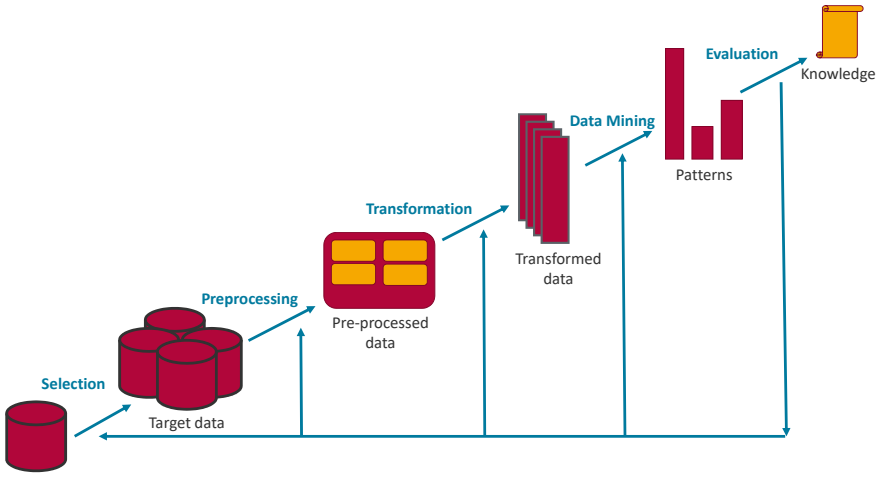


Figure 6: The process for *Knowledge Discovery in Databases*

method from NLP to Predictive process monitoring, we introduce a definition of sequences in Section 2.2.2. Then, we frame the problem in Section 2.2.3 that is fundamental to this thesis: next-element predictions in a sequence.

2.2.1 *Predictive Model Development*

In 1996 Fayyad et al. published what came to be known as the Knowledge Discovery in Databases (KDD) process [24]. Published over 20 years ago, still relevant today. It is illustrated in Figure 6 and shall be explained step by step along with the relevant technical details. The process is highly iterative, is focused intensely on the data, and jumps between every step are possible [50].

Predictive models are statistical tools that are used to make predictions. They work on *samples*, which are comprised of a number of *features* - variables that are considered useful for the prediction of the *target variable*. The target variable may either be a continuous or discrete value. In the latter case, the output is referred to as classification. Activities are discrete, and as such this thesis deals with classification. In the exemplary log in Figure 4, each row can be understood as a sample and each column as a single feature. Features are sometimes also referred to as predictor variables.

Selection

During the first step of the KDD process in Figure 6, features are chosen as predictors based on differed criteria such as their predictive

power or inter-feature correlations. Based on the latter, features can also be omitted from the original dataset.

Pre-processing

The second step of the KDD process is focused on data quality. Features may be sparse or contain data of low quality, e.g., different spellings of the same entity or typing errors. In this step, these issues are resolved, and appropriate values imputed where they are missing.

Transformation

The third step of the KDD process is aimed at making the data accessible for machine learning methods. During transformation, some features might be aggregated, normalized or encoded differently to assist the predictive model in picking up relations between variables. Common tasks are one-hot or dictionary encoding of discrete values such as strings, activity names for instance. Variable concatenations or complex aggregations might also be added as *engineered* features [50].

Data Mining

Data mining deals with the extraction of implicit, previously unknown, and potentially useful information from data. Machine learning provides a technical basis for this [3]. The goal of predicting the next activity is a classification task for which a wide array of predictive models is suitable. A selection of these are decision trees, random forests, support vector machines (SVM) or artificial neural networks (ANN) [50].

The data prepared in the previous steps are used to *train* such a model. During this training phase, the model uses accuracy metrics to assess the accuracy of its predictions and to learn the statistical properties of the data. The models adjust their internals during learning to improve accuracy. The accuracy is calculated on the target labels, i.e. the predictions that the model should produce on known data. Certain input parameters of the model are adjusted during this phase as well; an activity referred to as *hyper-parameter tuning*.

Evaluation

Typically, not all of the available data are used for training. This is to make sure that the model generalizes well, i.e., predicts as good on unseen data as on its training data. If a model is not generalizing, it is said to be *overfitting*. A model is said to be overfitting when it has learned the properties of the training data too well and thus shows poor generalization capabilities. A simple and effective method to avoid bias is to divide the available data into two sets: 75% as the

training set to train the model on, and 25% of it as the test set to verify the model's performance on unseen data [50, 51].

The log data in our use case consists of traces, a type of sequential data. We introduce sequences in the next chapter.

2.2.2 *Sequences*

This section presents a formal definition of sequences that lays the groundwork for sequence prediction. The definition is according to van der Aalst [3].

A sequence is made up of ordered, individual steps, which are referred to as items [34]. An item could represent a word, a letter, or anything that can be modeled as a sequence. Each item is an element of the set $\mathscr{I}$ of all items, and each *itemset* s is a subset of $\mathscr{I}$. A *sequence* seq is an ordered list of itemsets:

$$\begin{aligned} \mathscr{I} &= \{i_1, i_2, \cdots, i_n\} \\ s &= (x_1, x_2 \cdots x_n) \mid \forall 0 \leqslant j \leqslant n : x_j \in \mathscr{I} \\ seq &= \langle s_1, s_2 \cdots s_l \rangle \mid \forall\ 0 \leqslant j \leqslant l : s_j \subseteq \mathscr{I} \end{aligned}$$

Subsequences are coherent fragments of sequences. For subsequences, the notation $seq_{i,k}$ denotes that elements from index i through k from the original sequence are contained consecutively: $seq_{1,2} = \langle s_1, s_2 \rangle$. The relationship itself is noted as $seq_{1,2} \sqsubseteq seq$.

Under the assumption of the Markovian hypothesis that "the probability of each event depends only on the state attained in the previous event" [27], a predictive model can be trained on a database of sequences to predict the progress of an individual sequence. This is shown in the next sub-section.

2.2.3 *Sequence Prediction*

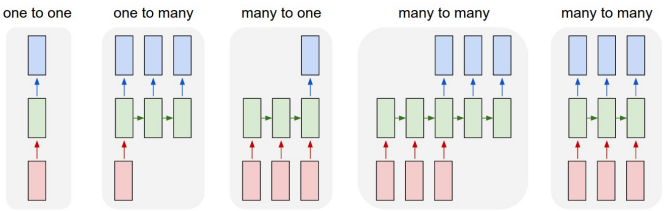


Figure 7: The different flavors of sequence prediction. Picture taken from [74]

Sequence prediction is a common task in NLP, for example in machine translation. The order of input samples is captured, and the prediction is partially based upon it. There are several flavors of it,

mostly depending on the number of inputs and outputs, as illustrated in Figure 7. In this explanation, we focus on many-to-many and many-to-one predictions.

A function $predict$ takes a sequence seq_{in} and gives out an output. The output can be either an itemset $\hat{s}$ or a sequence $\widehat{seq}$, depending on the flavor of sequence prediction. The circumflex marks the output as a prediction [50]. The following example shows a many-to-many prediction and a many-to-one prediction.

$$\begin{aligned} \widehat{seq_{out}} &= predict(seq_{in}) \\ \hat{s} &= predict(seq_{in}) \end{aligned} \tag{1}$$

Many-to-many predictions are common in the domains of machine translation. For example, the translation of the sentence *"I am writing my master's thesis"* into the German sentence *"Ich schreibe meine Masterarbeit"* can easily be mapped onto the notation previously described. With the alphabet as $\mathcal{I}$ and each itemset representing a single word, the input and target sequences can be noted as:

$$\begin{aligned} seq_{in} &= \langle (I), (am), (writing), (my), (thesis) \rangle \\ \widehat{seq_{out}} &= \langle (Ich), (schreibe), (meine), (Masterarbeit) \rangle \end{aligned}$$

Many-to-one predictions are used to generate text and even write simple novels [7, 73]. As the name suggests, they predict the next itemset in a sequence:

$$\begin{aligned} seq_{in} &= \langle (I), (am)(writing), (my) \rangle \\ \hat{s} &= \langle thesis \rangle \end{aligned}$$

In this thesis, we focus on many-to-one predictions, because we aim to predict the next activity in a running case based on its trace. How neural networks are especially suited for sequence prediction is explained in the next section.

2.3 ARTIFICIAL NEURAL NETWORKS

An artificial neural network (ANN) mimics the inner workings of a human brain in that it is made up of connected nodes referred to as neurons. These neurons are interconnected and produce an output signal upon receiving an input signal. This section first gives background on the general structure of a basic neural network in Section 2.3.1 and then highlights two enhancements for sequence prediction in Section 2.3.2. We end this section in Section 2.3.3 with insights into the architectural cues that we use during this thesis.

2.3.1 *Structural Basics*

The first and most trivial type of ANN is the feedforward neural network [63]. It is not optimally suited for sequence prediction but serves as a sufficiently complex example to explain the inner workings of an ANN, and for pointing out why long short-term memory networks are more suited for sequence prediction.

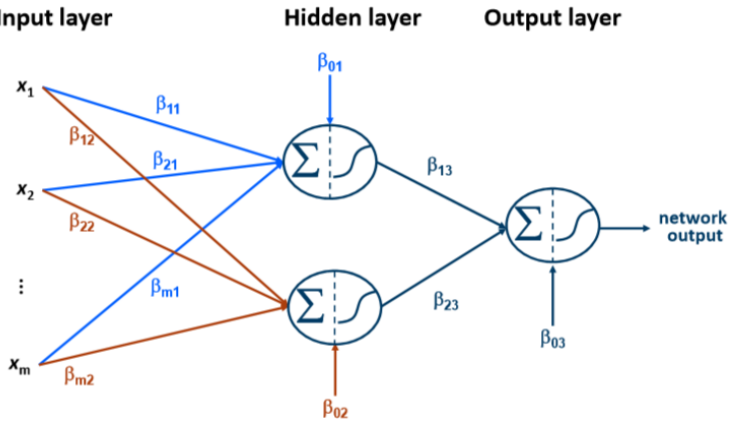


Figure 8: A feedforward ANN with a single hidden layer. Illustration taken from [51]

The nodes in any ANN are organized in layers, with each neuron connecting to every neuron in the adjacent layers, resulting in fully connected layers. The samples are fed into the input layer in the form of a vector of features. The prediction ultimately comes out of the output layer. The layers between the input and output layers are referred to as *hidden layers* and can be greater in number.

Figure 8 shows an exemplary ANN with a single hidden layer. The features x_1 to x_m are fed into the network as one vector, and their numeric values are passed along the edges to every neuron in the next layer. For each neuron, the incoming values are multiplied with the respective edge weight β and summed up, as depicted by the $\sum$ symbol in each circle. The sum is passed to a function called *activation function* - illustrated by the sigmoid curve - which controls whether the neuron emits a signal or not. The signals are passed along the output edges of the respective neurons, and the process is repeated for the next layer until the signals have reached the output layer. This process of continually feeding the outputs forward to the next layer also gives this type of hidden layer its name: feedforward layer.

The weights β are adjusted during training to increase prediction accuracy. For this purpose, the training samples are grouped into *batches*. The samples in a batch are passed into the network, and the predictions for this batch are compared to the target labels. At this point, the *loss function* calculates how far the network's predictions missed the labels. The weights are then adjusted to reduce the loss

in the next batch. When all batches have passed through the network once, an *epoch* has elapsed. Because the goal of the training algorithm is to minimize the network's loss, the weights can be adapted too well to the training set. This can happen when the network is trained for too long, making it overfit.

Overfitting an ANN during training can be avoided by calculating the network's loss both on the training set as well as on the test set. The weight adjustments are made based on the loss on the training set, but the training is stopped once the loss on the test set has ceased to improve for a certain number of epochs. This technique is referred to as early stopping.

Because feedforward layers are passing their outputs toward the output layer, and the neurons have no memory, this type of layer does not have any capacity for persisting what it has previously processed. However, this is a desirable capability for sequence predictions, as several samples can belong to a single sequence.

The next subsection shows how recurrent neural networks (RNNs) have been created with this capacity in mind, making them suited for working with sequential data. Long short-term memory builds on RNNs and enhances the capacity to remember even further.

2.3.2 *Enhancements for Sequence Prediction*

The layers in a recurrent neural network (RNN) implement a feedback loop. This loop makes information from the output of a neuron available to itself for the next step. Introducing a time dimension, RNNs are often displayed in an unrolled fashion as in Figure 9. This form of illustration also reveals that recurrent neural networks are intimately related to sequences and lists.

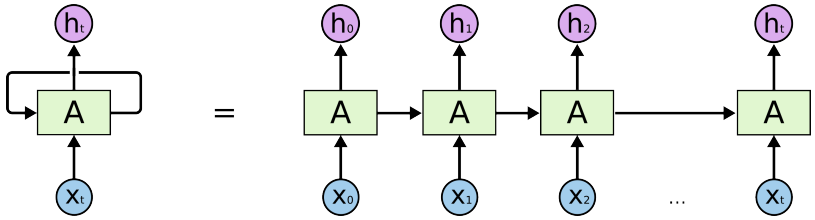


Figure 9: An unrolled neuron of an ANN, taken from [77]

A neuron receives an input x_t and outputs a value h_t. The letter t indicates the time dimension. By showing the neuron in an unrolled fashion, it becomes visible that the output of the neuron at time t becomes available to itself at time $t+1$ [77].

The introduction of the time dimension causes the perception of input data to change: It is now required to be three-dimensional. With feedforward ANNs, training data was two-dimensional and consisted of a list of independent samples. Samples were the same as a feature

vector, but for RNNs, samples need to be understood as lists of feature vectors. This makes a sample suitable to represent a sequence because its feature vectors can represent the individual timesteps.

While the loop in an RNN indeed allows recognizing short-term dependencies, the network as a whole will still underperform with long-term dependencies when the gaps between related inputs become too big. The root cause is that RNNs only loop back inputs, and lack memory to bridge those gaps. This problem has been thoroughly explored by Hochreiter et al. [39] who also proposed the long short-term memory fix presented in the following.

Long Short-Term Memory

In recent years, RNNs were applied with great success to a variety of problems: speech recognition, language modeling or translation. This success can be attributed in part to the enhancement of RNNs with long short-term memory (LSTM) cells [44, 50, 63].

Hochreiter & Schmidhuber published this enhancement in 1997 [40] which now sees broad application. Essentially, the repeating neurons of an RNN are turned into cells by equipping them with a cell state C, that provides them with the capacity to remember.

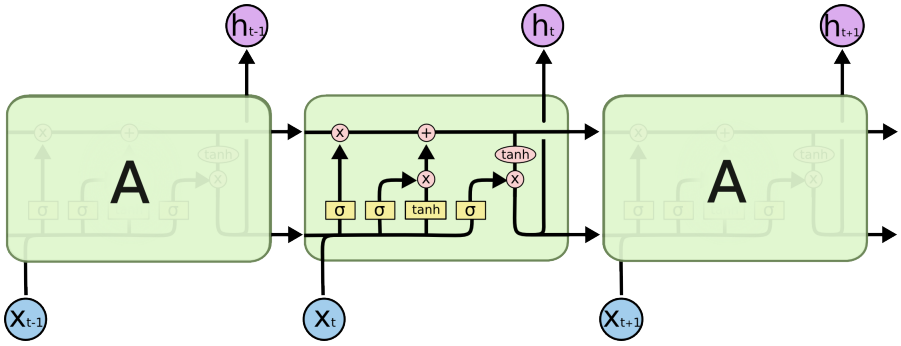


Figure 10: An unrolled LSTM cell and its architecture. Illustration taken from [77]

Figure 10 showcases the architecture of an unrolled LSTM cell. In the diagram, each line carries an entire vector. The pink circles represent pointwise operations, like vector addition. Yellow boxes symbolize operations. Merging lines denote concatenation, while a forking line denotes copying its content in both directions. The cell state C is symbolized by the horizontal upper line running from left to right. The lower line on the left represents h_{t-1}, the output of the cell in the previous step.

The σ boxes represent gates. Which parts of C_{t-1} to keep and what to forget is decided at the first gate from the left, by taking h_{t-1} and x_t into account. The following σ gate and $\tanh$ operator together create the update to C_{t-1}. Finally, on the right, the updated cell state C_t is used together with the activation function $\tanh$ to produce the

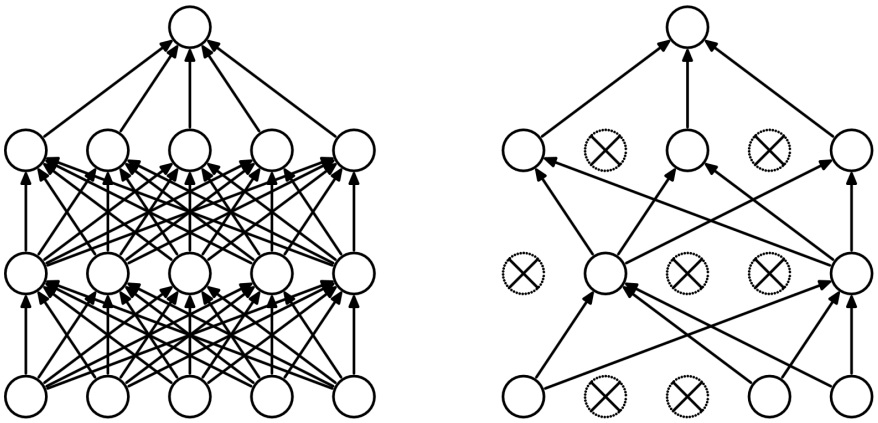


Figure 11: The effects of applying dropout. Illustration taken from [69]

output h_t. Variants of this original LSTM cell architecture exist, with most modifications made around the construction of the state C - all exhibiting similar performance [30].

The types of networks layers presented until here can be freely combined inside a network architecture. For instance, a feedforward layer can be attached to an LSTM layer. In our work, we apply several architectural cues besides the combination of different network types. These are explained in the following subsections.

2.3.3 *Architectural Cues*

A neural network architecture is seldomly comprised only out of feedforward or recurrent layers. Typically, the layers are augmented with activation functions and other types of layers to improve accuracy. In our work, we make use of Dropout layers and Embedding layers, which we present in the following. Furthermore, we employ activation functions to speed up learning and implement the classification output.

Dropout Layers

As we outlined in the beginning of this section, ANNs implement an optimization function. This optimization naturally makes the network prone to overfit. The use of dropout layers presents a simple, yet effective technique to avoid overfitting. By breaking the connection between adjacent neurons with a given probability of p, the network learns more robust features. Dropout also approximately doubles the number of iterations required to converge but decreases training for an epoch [69]. Figure 11 illustrates the effects of dropout application between two layers.

Embedding layers

Embedding layers make it possible to use word embeddings. Embeddings map inputs to a continuous vector space. The layer weights are often imported from pretrained embeddings, as is explained in depth in Section 2.4.2.

Softmax activation

The softmax activation is useful for converting the output of a network into a classification. As outlined in the beginning of this section, an activation function determines the output of a neuron.

The softmax function normalizes the output y_i of a neuron i to a value between 0 and 1:

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

The sum of all output values of all neurons in a softmax-activated layer always equals 1. A is understood as the placeholder of a single class, and its output is interpreted as the probability of this class. The neuron with the highest probability reveals the prediction of the network.

Rectified Linear Unit Activation

The rectified linear unit (ReLU) function has become a popular choice for activating hidden layers [28]. It is defined as follows:

$$\mathrm{ReLU}(y_i) = y_i^+ = max(0, y_i)$$

They are biologically plausible, giving a one-sided output. Also, they activate sparsely. In a randomly initialized network, only about half of all hidden units are activated. Combined with the simple calculation of the function, the activation function makes for efficient computation [49].

2.4 DATA PREPARATION AND FEATURE ENGINEERING

Data preparation and feature engineering are the tasks in predictive model development that demand the most effort and have the greatest impact on prediction accuracy [50]. To assist predictive models during training, most features need to be reformatted. This requirement comes with an implication that can also harm model performance, which is explained in Section 2.4.1. Then, relevant methods used for categorical variables are presented in Section 2.4.2. Section 2.4.3 points out methods with which to make sequential inputs of various length conform with the fixed-width input requirement of predictive models. In the case of ANNs, this requirement can easily

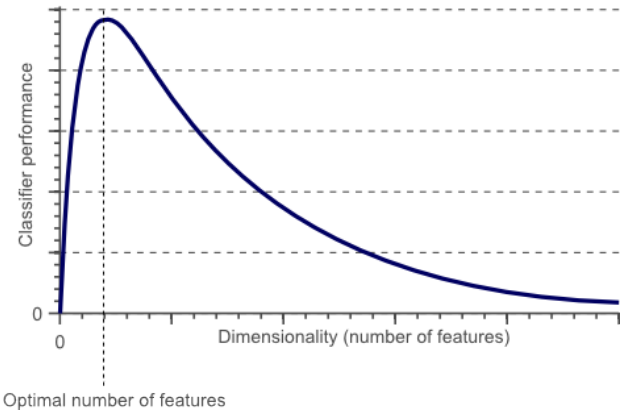


Figure 12: Adding features to the feature space does not improve model performance indefinitely

be explained by the fixed number of units on the input layer. The methods are demonstrated in the example of the following two sequences:

$$seq1 = \langle(\mathrm{Arthur}), (\mathrm{is}), (\mathrm{the}), (\mathrm{king})\rangle$$
$$seq2 = \langle(\mathrm{Maria}), (\mathrm{wants}), (\mathrm{to}), (\mathrm{be}), (\mathrm{the}), (\mathrm{queen})\rangle$$

2.4.1 *The Curse of Dimensionality*

The curse of dimensionality describes a phenomenon that arises when using more and more features [3] with a constant number of samples negatively impacts accuracy on test data. It appears across various domains such as combinatorics, sampling optimization, and machine learning, and is especially prominent when features are engineered.

The increase in dimensionality caused by the number of features increases the size of the solution space. A greater solution space eventually causes problems for machine learning algorithms which require statistic significance, making the algorithm less accurate. To counter this effect, more training data would be necessary. Thus, increasing the number of features for training a predictive model only improves accuracy to a certain extent when the number of samples is constant. This relationship is illustrated in Figure 12.

2.4.2 *Feature Engineering for Categorical Features*

While ordinal features are most often normalized, categorical features can be encoded in a variety of ways. Categorical features could be strings like *red* and *black*, which have no numerical format and may or may not be relative to each other. Where one-hot and dictionary encodings are traditional approaches for these features, word embeddings represent a relatively new and promising approach.

One-hot Encoding

One-hot encoding is often used for features which have a small to medium-sized number of unique values. Each feature is expanded to an n column representation, where n represents the number all possible values, also referred to as the alphabet. The columns hold boolean flags, of which only one is ever true in any given row. This type of encoding is also sometimes referred to as "dummy encoding" [55]. Thus, the first word of $seq1$ would be encoded as in Table 1. As the alphabet becomes larger, n increases and the feature vector becomes more sparse. Sparse is a word used to describe data which mostly contains zeros. This characteristic can easily cause symptoms of the curse of dimensionality to appear, which is why it is not used for very large n.

Maria	Arthur	is	wants	to	king	be	the	queen
0	1	0	0	0	0	0	0	0

Table 1: One-hot encoding of "Arthur" from alphabet of $seq1$ and $seq2$

Dictionary Encoding

Widely used in compression, within in-memory database systems [57] for example, dictionary encoding is also used to encode categorical values. Since predictive models only process numerical values, dictionary encoding is used to create a look-up table and map every value of a feature to a numeric code, like in Table 2.

In contrast to one-hot encoding, this dictionary encoding does not result in wide and sparse inputs, making it suitable for features with a large number of distinct features. However, it has one important ramification: It imposes order on features which previously might not have had one. In our example, the condition $Maria > Arthur$ holds after encoding. The dictionary-encoded sequences would look like this:

$$seq1 = \langle 1\ 2\ 7\ 5 \rangle$$
$$seq2 = \langle 0\ 3\ 4\ 6\ 7\ 8 \rangle$$

Word	Maria	Arthur	is	wants	to	king	be	the	queen
Code	0	1	2	3	4	5	6	7	8

Table 2: The dictionary built from the $seq1$ and $seq2$

Word Embedding

Because dictionary encodings impose order and one-hot encodings lead to sparse data, word embeddings are perceived as a viable alternative for encoding categorical variables. Word embeddings are rooted in NLP, where applications deal with very large alphabets.

A word embedding is the output of a neural network with a single hidden layer that has been trained on a large, dictionary-encoded text corpus. It detects how words relate to each other and puts out highly dimensional vectors for each word [43]. These represent any word property that the model has picked up. In the following example from [53], the network was able to detect gender properties and social semantics of words:

$$\vec{v}_{\mathrm{king}} - \vec{v}_{\mathrm{man}} + \vec{v}_{\mathrm{women}} = \vec{v}_{\mathrm{queen}}$$

A word embedding thus allows clustering of words by specific properties. Made famous by Google, word embeddings have motivated numerous publications [5, 29]. Word embeddings are often available pre-trained on extensive text databases, like Wikipedia. To use such a pre-trained embedding, the weights from it can be loaded into the respective layer before training the network.

2.4.3 *Feature Engineering for Sequences*

Predictive model inputs take in feature vectors of fixed size. As shown, sequences can be of arbitrary length, and thus these two requirements collide. To bring variable length inputs into a usable format for predictive models, several methods of formatting have been invented. As some of their names suggest, these come from the domain of natural language processing (NLP), but can easily be transferred to the problem at hand.

Sliding Window

The sliding window format is widespread in the areas of NLP and Time Series Forecasting. The sequences are divided into chunks of width c. Chunking results in several input tuples for every sequence and creates the desired tabular format for feeding into the model - illustrated in Table 3.

Bag-of-words

The bag-of-words (BOW) encoding produces l input tuples, where l is the length of the encoded sequence. Similarly to one-hot encoding, the alphabet needs to be known. One arrives at the encoding by marking the occurrence of each item in the alphabet for each subsequence $seq_{1,k}$. k is incremented for every sample from 1 to l, making

Word 1	Word 2
Arthur	is
is	the
the	king
Maria	wants
wants	to
to	be
be	the
the	queen

Table 3: Windows created from seq1 and seq2 with $c = 2$

Sequence	Maria	Arthur	is	wants	to	king	be	the	queen
$seq1_{1,1}$	0	1	0	0	0	0	0	0	0
$seq1_{1,2}$	0	1	1	0	0	0	0	0	0
$seq1_{1,3}$	0	1	1	0	0	0	0	1	0
$seq1_{1,4}$	0	1	1	0	0	1	0	1	0
$seq2_{1,1}$	1	0	0	0	0	0	0	0	0
$seq2_{1,2}$	1	0	0	1	0	0	0	0	0
$seq2_{1,3}$	1	0	0	1	1	0	0	0	0
$seq2_{1,4}$	1	0	0	1	1	0	1	0	0
$seq2_{1,5}$	1	0	0	1	1	0	1	1	0
$seq2_{1,5}$	1	0	0	1	1	0	1	1	1

Table 4: Bag-of-words encoding for seq1 and seq2

the feature vector fill up with truth values as k increases. As Table 4 evidences, this type of encoding results in sparse data, too.

n-*grams*

The n-gram approach is very popular in computational linguistics, biology and data compression and is effectively an $(n-1)$-order Markov model, with the most popular choices for n being 1, 2 and 3. These models would be called *unigram*, *bigram* and *trigram* models.

Similar to BOW, an n-gram model counts occurrences. Different from BOW, it tracks the occurrence of subsequences of length n. Suppose a set LS holds all possible values of a single feature. From LS, a feature set FS is constructed with every item being a permutation of length n. A feature in this set would be referred to as *gram*, and can also be understood as a possible subsequence.

$$FS = S(LS, n)$$

While n-grams can be powerful features to use, their high dimensionality causes high computational loads and sparse data. With the small feature set of 9 words in the given example, a feature vector using tri-grams would already be $\frac{|FS|!}{(|FS|-n)!} = 504$ elements wide.

Subsequence Mining

One of the main weaknesses of n-grams is the restriction to a single n and that "one may need to examine a combinatorially explosive number of possible subsequence patterns" [56]. For this reason, approaches have been developed that combine the benefit of n-grams with more flexibility by avoiding the limitation to n. An algorithm sifts through input sequences and detects sequential patterns of any length. These sequences are then encoded, resulting in line items similar to the ones in Table 4 The respective boolean entry denotes that a specific subsequence has occurred in the sequence. Examples for these algorithms are GSP [68], FreeSpan [34] and PrefixSpan [56].

For every mined subsequence ss, the *support* metric $supp(ss)$ can be calculated. This metric indicates in how many complete sequences, ss occurs as a subsequence. Sometimes, shorter subsequences are part of longer subsequences, like ss1 is contained in ss2:

$$s1 = \langle a, d \rangle$$
$$s2 = \langle a, d, c \rangle$$

If the condition $supp(ss1) > supp(ss2)$ holds true, then ss1 is called a *closed* subsequence as it can not be expanded without shrinking its support.

3 RELATED WORK

This chapter on related literature presents selected publications from Predictive process monitoring as well as from natural language processing. Publications related to Predictive process monitoring are presented in Section 3.1. We end the chapter in a detailed presentation of the works reimplemented for comparison. Similarly, we present in Section 3.2 other sequence prediction approaches from NLP and end with in-depth information on the publication which we adapted for the use case at hand.

3.1 PREDICTIVE PROCESS MONITORING

We identified three types of prediction targets in current publications on Predictive process monitoring: constraint predictions, case-specific next-event predictions, and general next-event predictions without regard to a specific case. These three types will serve as subdivisions to this section.

Constraint Predictions

A constraint could be the delivery of a product within a given timeframe, or the total runtime of a case [25, 79].

Polato et al. make use of event attributes in their work for improving the prediction of the remaining time of business process instances [58]. During feature engineering, their training data is enriched with information about possible other activities. With support vector regression (SVR) from the WEKA toolkit [78] and default parameters, they reach 6% and 9% mean absolute percentage error (MAPE) on two non-public datasets of 5000 and 1500 traces. While the authors criticize the use of non-public datasets, they also do not publish theirs nor their source code, making the results hard to compare.

Metzger et al. predict constraint violation of a case by comparing fundamentally different prediction models and combining them into an ensemble. The ensemble is tested with different voting mechanisms, e.g., majority voting or recall-orientation. They argue that predictions that are either late and precise or early and incorrect are equally worthless for interventions. Because of this, they argue that predictions should stabilize as early as possible. They establish that

after progressing 60% to 85% into the case, predictions become stable and precise.

Three approaches are used to predict constraint violations: constraint satisfaction, Quality-of-Service (QoS) violation checks and machine learning. The two former models use rules to predict a process outcome while the latter is a trained neural network. For it, Metzger et al. use the ANN implementation from the WEKA machine learning toolkit with a single hidden layer.

The authors can formulate rules for the two other models because the process at hand is very rigid and well defined: The Cargo 2000 process is a standard proposed by the International Air Transport Association (IATA) [52]. Two-thirds of their non-public dataset of 3942 traces and 56082 events were used for training and the remainder for testing.

While model ensembles can be expected to bring small accuracy improvements, they incur a lot of work and are typically only found in research [51]. Furthermore, we expect that the use an RNN on this simple process would have yielded a higher accuracy than the stated 0.7.

Francescomarino et al. organized predictive models in a clustered fashion [25] to predict predicate fulfillment. Having clustered the training data, one model was trained on each cluster. Then, to obtain a prediction, the cluster for a new data item needed to be found, and the corresponding model selected. Furthermore, the authors varied the probability threshold for accepting a prediction and measured how it affected the point in process progress at which the predictions become stable. They refer to this characteristic as *earliness*, similar to stability in Metzger's work [52]. Their approach, implemented as a ProM plugin called *Predictive Process Monitoring*, uses k-means or DBSCAN to cluster the data and decision trees or random forests to make the prediction. It was tested on the BPIC11 [9] dataset and obtained accuracies of up to 0.9.

Clustering improves tree-based models because it makes the training data more homogeneous. By using embedding layers, an ANN can cluster the data by itself, making a manual clustering unnecessary.

Case-specific Next-Event Predictions

Next-event predictions specific to a case allow answering the question "which activity comes next in this special case?" - the question that we answer with deep learning.

Huber [42] gives an example of how a next-step prediction and recommendation system for caseworkers might look like. The system

is prototypically implemented within CoCaMa, a case management application. Its predictions are produced as follows: After gathering the training data from various sources via an extract, transform and load (ETL) process, several *Next Models* are constructed. There are four Next Models which use different data: timestamps, deadlines, decisions, and goals. The predictions from these individual models are combined by the recommender via weights to produce a recommendation. Huber uses decision trees from the WEKA [78] machine learning toolkit to implement them. The system has been evaluated with 25 hand-made traces [42], which explains why high accuracies could not be obtained.

Next-event prediction without any machine learning is done by Böhmer et al. with a method rooted in heuristic analysis [15]. The authors argue that the amount of trust that users put into ANN predictions is limited due to the fact that ANNs are not only computationally expensive but that their black-box nature makes their predictions and inner workings very hard to comprehend.

Because potentially large organizational changes could be made based on the predictions, the authors stress the importance of the second- and third-best results. They go on to explain the aspects which motivated specific prediction results. Böhmer et al. approximate trace similarities with the Damerau-Levenshtein method and a custom cost function. Using this method, they filter historical traces to produce a set of similar executions and go on to mine probability distributions from this set. The most likely behavior is used as the final next-event prediction [15]. Additionally, they predict the timestamp of its execution. They evaluate their approach on the BPIC12 datase [10] as well as a Helpdesk event log [37]. On both datasets, they achieve an accuracy of 0.77 for the next-event predictions.

Understanding the reasons for a prediction is essential. Forgoing black-box models could mean missing out on potentially higher accuracies [71] for the sake of understandability. Research has been done to avoid this trade-off: With techniques such as LIME, predictions made by black-box models can be explained [61]. Nonetheless, the computational power requirements are undeniably higher.

Klinkmüller et al. enrich their training dataset with features that encode subsequence occurrence to predict the next event in a trace based on its history. Doing so on a synthetic dataset, they are able to increase the accuracy of random forests in comparison to a baseline that is not using the engineered subsequence features. Furthermore, the authors find that training models on complete traces is preferable to the widespread practice of trace truncation [48]. These findings inspired us to explore subsequence features in our evaluation, and

question model training strategies that truncate traces.

Francescomarino et al. also use LSTM networks to predict the next events of a case, and engineer features which indicate the presence of a loop [19]. To enhance the predictions further, they apply a compliance-checking logic on top of the predictions which leverages a priori knowledge to rule out forbidden next events. The network is trained with windowed samples, and the following accuracies on these six datasets have been obtained:

- EnvLog [22]: 0.07
- HelpDesk [37]: 0.816
- BPIC11 [9]: 0.276
- BPIC12 [10]: 0.408
- BPIC13 [11]: 0.516
- BPIC17 [13]: 0.439

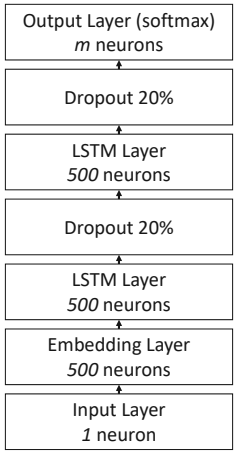
Tax et al. worked on the prediction of the next activity and its timestamp in a running case. He did so with LSTM neural networks. Both prediction targets were predicted with a single network with two output layers. They compared various architectures in terms of the number of shared and split LSTM layers. Sharing a single layer turned out to contribute to the best results on both the HelpDesk [37] and BPIC12 [10] datasets. The obtained accuracies were 0.712 and 0.760, respectively [72].

General Next-Event Predictions

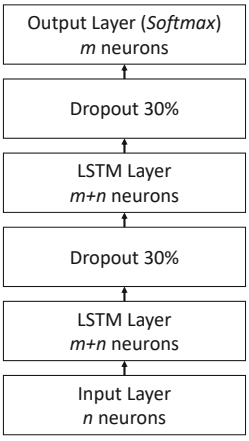
General next-event predictions predict the next event in a stream of events, without specifying which case this event belongs to. Here, Evermann et al. [23] and Schönig et al. [64] made contributions, which also build upon each other.

Evermann et al. remark the lack of research on next event predictions and have successfully demonstrated the applicability of LSTM neural networks in the context of predicting the next event in a stream of events. Using a neural network implemented with Tensorflow, precisions between 0.600 to 0.900 on the BPIC12 and BPIC13 datasets are achieved. Evermann et al. want their work to be understood as a "demonstration of the applicability of the approach and the potential for future work." The authors highlight that their work lacks the use of data attributes during model training [23].

Schönig et al.picked up on the last point and demonstrated on the BPIC dataset from 2017 that using data attributes complementary to



(a) Architecture used by Evermann et al. [23].



(b) Architecture used by Schönig et al. [64].

Figure 13: An overview of the reverse-engineered networks that are used as a comparison in this thesis. n denotes the dimensionality of the input vector, and m the number of output classes, including the end-of-sequence marker.

the event names does increase prediction accuracy. The authors implemented their solution with Keras and trained it with stratified 5-fold cross-validation [64]. The data was pre-processed with one-hot encoding for categorical variables and min-max-normalization for continuous features. Also, the work demonstrates that an increasing number of included data attributes can improve accuracy [64, p.5]. Their approach reaches accuracies around 0.90, depending on its configuration.

We take the works of Evermann et al. and Schönig et al. and reimplement them as direct comparisons to our adapted approaches as they were tested on BPIC data, their source code was made readily available, and their authors were helpful in ensuring the correctness of our understanding. Furthermore, their approaches were entirely based on neural-networks with little additional pre- or post-processing. In close collaboration with the two authors, their neural networks were reverse-engineered and also the architectures in Figure 13 were confirmed to be correct. Both model the problem as a multi-classification problem.

Clearly, Evermann's architecture left an inspiring impression with Schönig, who decided to remove the Embedding layer and adapt unit counts [64]. In our conversation, Schönig argued that the Embedding was not needed, as the number of unique events was a lot lower than that of words in a text, for which Embeddings were originally developed. Where Evermann et al. used stochastic gradient descent (SGD) with a manual adaption of learning rate decay to 0.75 after the 25th epoch, Schönig et al. use RMSprop with default values to optimize

the network's loss. Furthermore, Schönig et al. fed the training data into their networks in a windowed fashion, contrary to Klinkmüller's suggestion that this might produce unstable results [48].

3.2 SEQUENCE PREDICTION

Sequence prediction deals with predicting the next element or next sequence from a given input, as explained in Section 2.2.3. It is rooted in NLP, and relevant publications are discussed in this section.

Kokkinos et al. leveraged a tree-structured neural network to classify sentiments in sentences [59]. In a detailed comparison with other works, their bipartite tree approach yields the highest accuracy on the Stanford Sentiment Treebank dataset. While the proposition of tree-structured Gated Recurrent Units (GRUs, a variant of LSTM cells) and their use of a technique called attention form the main contribution of this work, the authors make use of a bipartite network architecture and word embeddings as well. This gives further appeal to the idea of adapting the approach of Shibata et al., presented in the following.

In 2016, the International Conference in Grammatical Inference 2016 (ICGI) held a competition called SPiCE "about guessing the next element in a sequence" [65]. It entailed making next-element predictions on twelve different datasets on which to make predictions for the next word. Most entries submitted to this competition made use of RNNs with LSTM cells to do so.

The winning submission by Shibata et al. uses a bipartite network architecture, training separate layers on different features of the same sentence. The results of these separate layers are then merged in the middle hidden layers to produce a single output [66], as Figure 14 shows. Architectural similarities with Schönig et al. and Evermann et al. are especially prominent in the left part of Shibata's model. The most recent word is fed into an Embedding layer and passes through two LSTM layers on the left. On the right side, a vector of SP-2 features is fed into an Embedding layer with a ReLU activation function. Then, both intermediate representations are concatenated and passed through a ReLU-activated layer. The result is passed through a 50% Dropout layer, to finally produce the prediction of the next word from a Softmax-activated output layer.

While one-half of the layers is trained on the most recent word of the sentence, the other half is trained on SP-2 features, which encode the prefix of that word. As this prefix can be of any length, Shibata et al. propose a binary bag-of-words encoding, representing the states of an SP-2 automaton. SP-k languages are used to describe certain long-term dependencies through forbidden subsequences. For example, if

$\langle a, b \rangle$ is forbidden, then no b may ever occur after a. According to Heinz, who assisted Shibata, deterministic finite automatons (DFA) can also be used to characterize SP-k languages, if the DFA states encode those subsequences of size $k-1$ present in the previous prefixes [36]. To make this concept more tangible, Table 5 illustrates a small example. The authors argue that LSTMs are not completely understood yet and it has not been proven that they are fully capable of recognizing sequences, which is why these features should assist the network. To prove their point, they compare the SP-k bipartite architecture with a basic one that is nearly identical to the one by Evermann et al. in Figure 13a. The performance differences between the two models are acknowledged as generally "slight," while the basic architecture performs "significantly worse on three problems."

			SP-2 vector				
t	$seq_{t,t}$	$seq_{0,t}$	[a	b	c	d	e]
0	a	a	[1	0	0	0	0]
1	d	ad	[1	0	0	1	0]
2	a	ada	[1	0	0	1	0]
3	c	adac	[1	0	1	1	0]
4	d	adacd	[1	0	1	1	0]

Table 5: Traces encoded with SP-2. As t progresses, more and more single-item subsequences ($k-1=1$) are marked as occurred. The alphabet is $I = \{a, b, c, d, e\}$. This example is taken from Shibata et al. [66].

From the literature review, we were inspired to adapt Shibata's approach to Predictive process monitoring. We learned about the different understandings of next-event predictions, and how features that encode subsequences can help model learning. We address this in the next chapter. Also, we noted the contradiction between the findings of Klinkmüller et al. about training models on the complete history and the fact that several other authors use windowing. How we deal with this fact is described in the chapter after that.

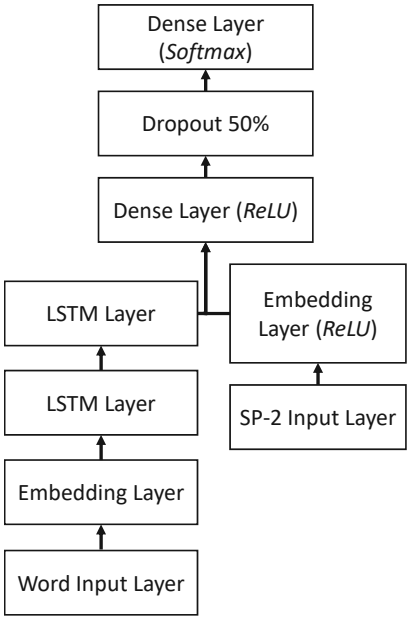


Figure 14: Network architecture used by Shibata et al.

4

SEQUENCE PREDICTION WITH TRACES

Previous works have demonstrated the practicality of LSTM neural networks for predictive process monitoring. However, only the use of embedding layers was inspired by natural language processing (NLP) until now [23]. We believe that more knowledge could be transferred because both domains revolve around sequential data. In this chapter, we outline how we take inspiration from sequence prediction in NLP, and thus contribute to Improvement Area 1 (NLP-influence) from Section 1.1.

We establish the connection between traces and sequences by connecting their definitions in Section 4.1. This provides the underpinning for introducing two NLP-inspired approaches for predicting the next activity in a case. These approaches are presented in Section 4.2 and Section 4.3.

4.1 UNDERSTANDING TRACES AS SEQUENCES

The definition for sequences presented in Section 2.2.2 and the definition of traces in Section 2.1.1 will be connected in this section to make it clear how events can be understood as itemsets.

Events are ordered in traces, and each case c contains its execution history in its trace attribute $\#_{trace}(c)$. Similarly, a sequence seq is defined to contain itemsets in an ordered fashion:

$$seq = \langle s_1, s_2 \cdots s_l \rangle \mid \forall\ 1 \leqslant j \leqslant l : s_j \subseteq \mathscr{I}$$
$$\#_{trace}(c) = \langle e_1, e_2, e_3 \cdots e_n \rangle$$

An itemset $s = (i_1, i_2 \cdots i_n)$ consists of items $i \in \mathscr{I}$, and is defined as $s \subseteq \mathscr{I}$. An event e is made up of attributes that are accessed via the # operator. It can be said that both itemsets and events act as containers of information. Therefore, we connect the definitions of the contained information in a first step. The set of items $\mathscr{I}$ is then defined as the set of all attributes of all events in all cases in a given log L:

$$\mathscr{I} = \{\#_a(e) \mid c \in L \wedge e \in \#_{trace}(c) \wedge a \in attribute\_names(e)\}$$

The definition of itemsets $s \subseteq \mathscr{I}$ does not place any restrictions on the content of s, so an itemset could consist only out of timestamps. To prevent such cases, we introduce a schema on the itemsets in a

second step. It is introduced as a direct mapping from an event e to an itemset s:

$$s = (i_1, i_2 \cdots i_n) \mid i_k = \#_{attribute\_names_k(e)}(e), 1 \leqslant k \leqslant n$$

The definition places each item i_k on a specific place in the itemset, depending on the index k in the attribute list $attribute\_names(e)_k$. The original condition $s \subseteq \mathscr{I}$ hold true. This definition enables sequence prediction on traces, and also provides the tabular format required for machine learning.

In Section 2.2.3, a many-to-one prediction is defined to target the next itemset $\widehat{s_{k+1}}$ of any sequence $seq_{1,k}$. In the use case at hand, this would entail the prediction of all data attributes in $\widehat{s_{k+1}}$. Since we specifically target the name of the next activity, we adjust the definition of the prediction function to target a single item $\hat{i}_j$. The index j corresponds to the column index of the target variable inside the itemset s_{k+1}, i.e., the index of the activity name in our case.

$$predict(seq_{1,k}) = \hat{i}_j$$

With the prediction target at hand, we move on to present the network architectures for executing the prediction.

4.2 ADAPTING A COMPETITION SUBMISSION

Shibata et al.'s bipartite network architecture has shown outstanding performance in the Sequence Prediction challenge (SPiCe) [65]. In the last section we have shown that a case trace can also be understood as a sequence, and thus we adapt Shibata's approach for use in the business process domain.

Figure 15 displays the adapted network architecture from Shibata et al. It is referred to as SP2 from here on.

We copied the bipartite structure and made small changes to dimensions and layers. The sequential feature vectors, i.e., all event attributes, are fed into the left input layer. n indicates the dimensionality of the event feature vector, which contains the encoded activity name, timestamp and other data attributes of the respective step. The sequential data passes through two LSTM layers interleaved with dropout layers to prevent overfitting. Hidden layers contain $n + m$ neurons, making their size dependent on the input and output dimensions. This cue was taken from Schönig et al. [64]. The LSTM layer output is concatenated with the output of the processed SP-2 features for each timestep, which were fed in on the right side of the tree and preprocessed by a feedforward layer. The concatenated vectors are passed through a ReLU-activated feedforward layer, and finally to the output layer. A Softmax activation function is applied to the output layer, putting out the next activity in one-hot encoded form

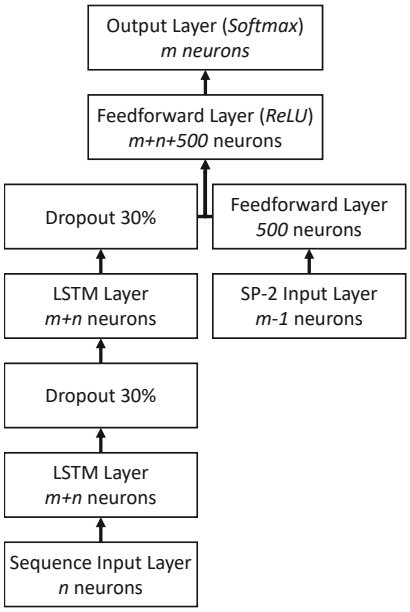


Figure 15: The SP2 network architecture

through m neurons. This models the problem as a multi-classification problem.

In contrast to the original model, we completely removed the embedding layers. Above the SP-2 feature input layer, the embedding layer was replaced with a feedforward layer. We argue that embedding layers are not necessary for this context, because the number of activity names in a process is small compared to the number of words usually processed with embedding layers in NLP scenarios.

Another difference is the dimensionality $n + m$ of the hidden layers. In contrast to Evermann et al. and Shibata et al., we do not fix it or make it smaller than the output unit count m. This follows general advice not to introduce bottlenecks in the hidden layers by using fewer units than required in the output layer [1, 70].

Shibata et al. engineered their SP-2 features from prediction targets [66]. Therefore, we produce the SP-2 features from the sequence of activity names. An exemplary SP-2 feature vector from activity names is depicted in Table 6. It is based on a trace from a case c, which contains numeric activity names.

			SP-2 vector								
t	$seq_{t,t}$	$seq_{0,t}$	[1	2	3	4	5	6	7	8	9]
0	1	1	[1	0	0	0	0	0	0	0	0]
1	8	18	[1	0	0	0	0	0	0	1	0]
2	6	186	[1	0	0	0	0	1	0	1	0]
3	8	1868	[1	0	0	0	0	1	0	1	0]
4	6	18686	[1	0	0	0	0	1	0	1	0]

Table 6: SP-2 features created from an activity trace $seq = \#_{trace}(c) = \langle(1),(8),(6),(8),(6)\rangle$

4.3 ENCODING SUBSEQUENCE OCCURRENCE

Klinkmüller et al. compared different feature representations and found that features that encode sub-trace occurrence can help models cover a broader variety of relationships [48]. While they found this to be true for random forests, we want to investigate the applicability of such features with LSTM neural networks. Francescomarino et al. tried this in their work and achieved discouraging accuracies around 0.50 [19]. We are convinced that another network architecture could make a difference.

SP-2 features already encode history and subsequence encodings do the same on a higher level of abstraction. Therefore, we take the SP2 model architecture and inject different features in place of the SP-2 features. The respective architecture is shown in Figure 16.

We will henceforth refer to both the architecture and the subsequence features as PFS. The acronym PFS originates from the word PrefixSpan, which is the name of the algorithm that we use later to mine the subsequences. The PFS feature vector has l dimensions and is fed in on the right side of the tree.

The PFS vector encodes whether common subsequences have occurred yet, as Table 7 illustrates. The illustration is based on the trace $seq = \#_{trace}(c) = \langle(1),(8),(6),(8),(6)\rangle$. From the log that seq was taken from, we mined four common subsequences: $\langle 6\rangle$, $\langle 1,6\rangle$, $\langle 8,6\rangle$, and $\langle 1,8,6\rangle$. In the example, a field inside the feature vector is set to 1 when the respective subsequence has occurred.

We presented two neural network architectures in this chapter. They incorporate learnings from a successful NLP competition submission, and a paper on feature engineering. Training these models on a multitude of datasets presented us with a challenge. We solved it by constructing a small training framework which we present in the next chapter.

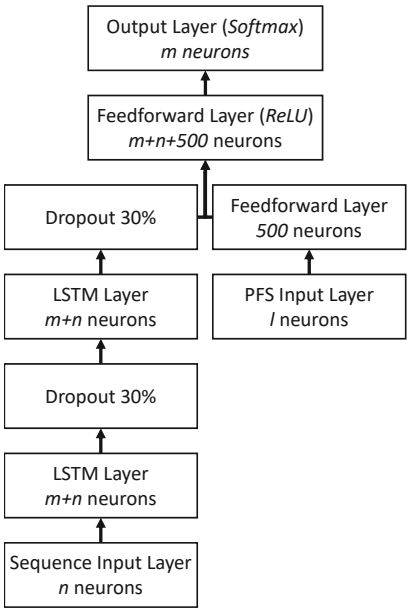


Figure 16: The PFS network architecture

			PFS vector			
t	$seq_{t,t}$	$seq_{0,t}$	$[\langle 6\rangle$	$\langle 1,6\rangle$	$\langle 8,6\rangle$	$\langle 1,8,6\rangle]$
0	1	1	[0	0	0	0]
1	8	18	[1	0	0	0]
2	6	186	[1	0	1	1]
3	8	1868	[1	0	1	1]
4	6	18686	[1	0	1	1]

Table 7: PFS features created from an activity trace

5 BATCH CONSTRUCTION STRATEGIES

During the development of the models, we realized that there are highly divergent understandings regarding the construction of batches from sequential data. We are convinced that these different understandings have a substantial impact on the reproducibility of any prediction approach. Furthermore, we needed to automate the training of the networks because we used multiple datasets. We present here a model training framework as a solution to both challenges. We share it because we believe that it enables researchers to compare the different understandings and quickly try out new model architectures. In doing so, we hope to contribute to Improvement Area 2 (Reproducibility) and Area 3 (Comparability) in Section 1.1.

We show a selection of possible batch construction strategies in Section 5.1 and the underlying technical reasons. Finally, we present the training framework in Section 5.2.

5.1 CONTRASTING DIFFERENT STRATEGIES

We implemented the SP2 and PFS models as well as the comparison models using Python and the Keras neural networks API. While doing so, a wide range of recommended possibilities to train the models on sequential data were noted in relevant publications and on online platforms.

Some recommend sliding window approaches, while others deem this against the concept of LSTM cells. Especially Klinkmüller et al. argue against it: "[...]the popular strategy of cutting traces to certain prefix lengths to learn prediction models for ongoing instances is prone to yield unreliable models" [48]. Again others argue that padding and truncating sequences to the same length makes training faster and does not affect accuracy. The batch size strongly affects convergence properties of any neural network, making it an important hyper-parameter to tune [47]. No guidelines how to sort traces into batches were found. We believe that the understanding of the merits of each batch construction strategy can be of general value to any application of sequence prediction.

Keras' implementation of LSTM is stateful *during* a batch, and resets the cell state C before the next batch [46]. The state reset is configurable, but for the sake of simplicity, it is advisable to keep related timesteps inside a single batch. Translated to traces, this means keep-

ing a trace completely inside a batch. A Keras LSTM layer requires input data to be formatted in a three-dimensional array of fixed size for every batch. Each dimension has a defined meaning as indicated by the suffix:

$$(n_{samples}, n_{timesteps}, n_{features})$$

A batch contains $n_{samples}$ samples, which correspond to traces in our case. Every sample contains $n_{timesteps}$ feature vectors. Each of these vectors has $n_{features}$ dimensions. These dimensions correspond to the size of the feature vector that is fed into the input layer. For each sample, the Keras LSTM layer cells maintain a separate state. This means that several traces can be trained on simultaneously [80]. As the batches themselves do not need to have the same dimensions, this definition opens up a range of strategies to construct batches from traces of variable length. We highlight four of them in the following paragraphs. To the best of our knowledge, these have not been compared in this context yet.

THE INDIVIDUAL STRATEGY First and easiest, batches can be constructed from a single sample. This fulfills the requirement of constant dimensions since only a single sample is used. Figure 17 illustrates this batching strategy.

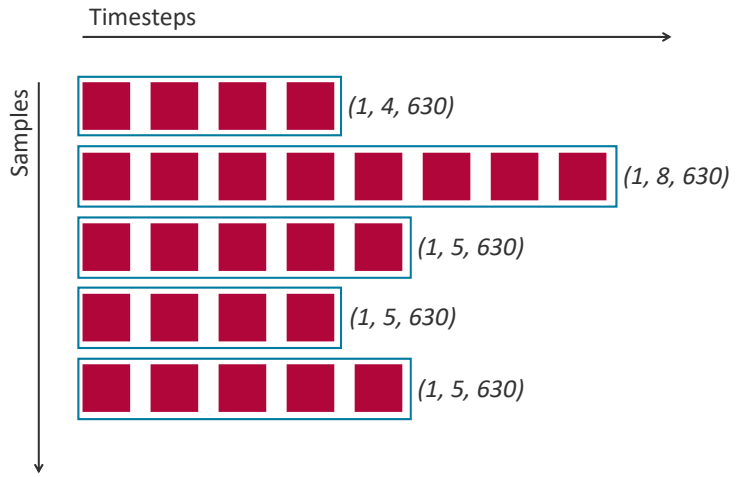


Figure 17: Batch construction from a single sample. A batch is boxed in a blue rectangle, and its dimensions are written beside it.

THE GROUPING STRATEGY Second, *some* traces can be trained inside a single batch. This is possible if all traces inside a batch have the same number of timesteps, i.e., have the same length. Figure 18 illustrates how a grouping could look like. The figure also illustrates

that the number of samples and timesteps may vary between batches. A look at Figure 19 reveals a power-law distribution of trace lengths in BPIC11. The grouping strategy could bias the model toward the long tail of the distribution, as longer and fewer traces are put into batches of their own.

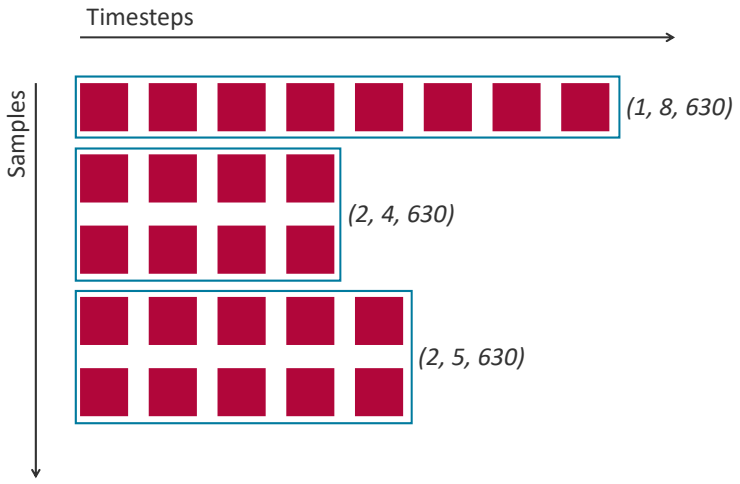


Figure 18: Batch construction by grouping samples. A batch is boxed in a blue rectangle, and its dimensions are written beside it.

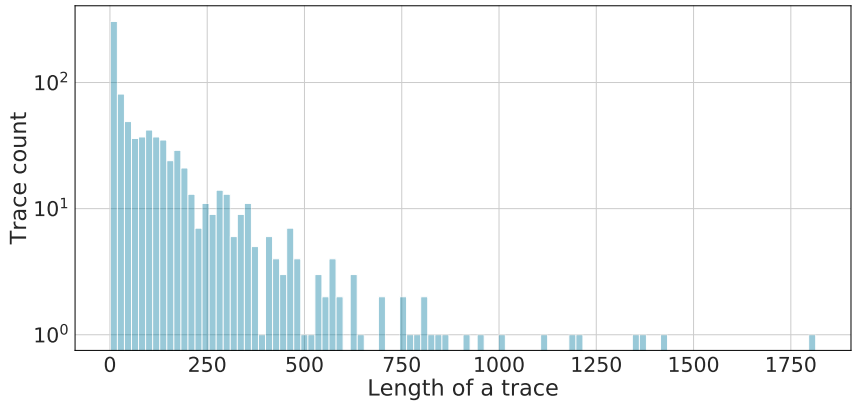


Figure 19: Distribution of trace lengths in BPIC11. Logarithmic scale to highlight extremely long outliers.

THE PADDING STRATEGY there is the possibility of padding the number of timesteps in a sequence to the same length. The padded values are then filtered out using a Masking layer during training [46]. The padding length is determined by the maximum trace length and may incur a significant memory overhead. This is especially the case for large outlier values like in BPIC11.

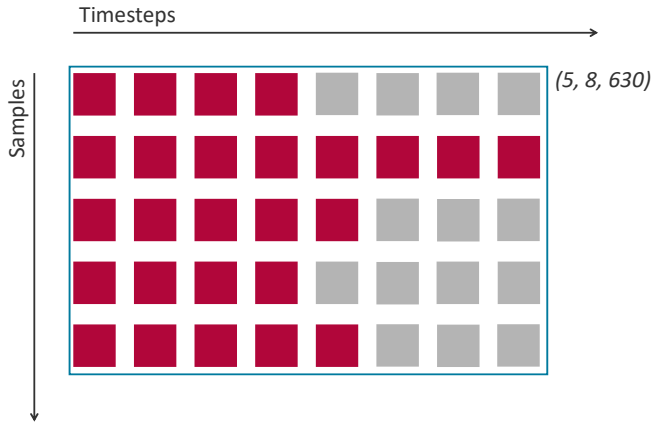


Figure 20: Batch construction from padded samples. Padding values are depicted by gray boxes.

THE WINDOWING STRATEGY Fourth and finally, there is also the possibility to split the trace into samples by sliding a window along it. This results in $l - w + 1$ samples for a trace of length l and a window width w. Taking Figure 21 as an example, the window would be $w = 2$ timesteps wide. This approach solves the problem of unequal sample lengths and facilitates batch construction. Unfortunately, it only allows the model to use a maximum of w timesteps per sample for training. This might cause the model to miss potential long-term dependencies. As both Evermann et al. [23] and Schönig et al. [64] use this format, and it directly opposes the findings of Klinkmüller et al. [48], we investigate it. Furthermore, we are convinced that use of a windowed training data format misses out on LSTM potential.

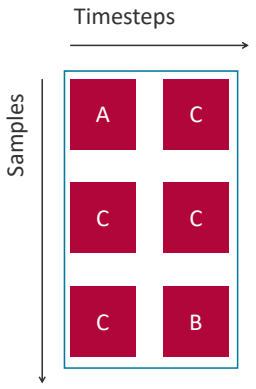


Figure 21: Batch construction by sliding a window along a sequence $\langle A, C, C, B \rangle$. A batch is boxed in a blue rectangle, and its dimensions are written beside it.

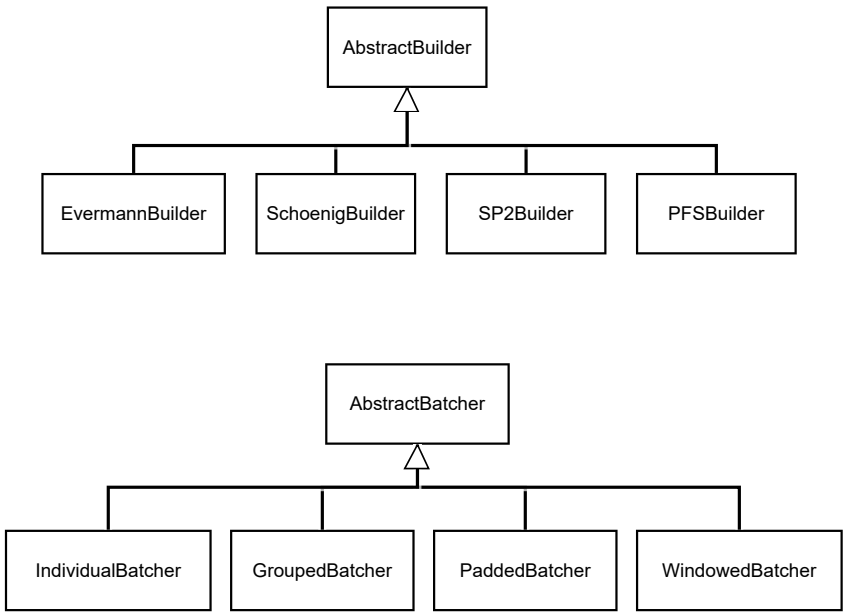


Figure 22: Unified Modeling Language (UML) diagram of the abstract base class inheritance structure of Builders and Batchers

5.2 A TRAINING FRAMEWORK FOR SEQUENTIAL DATA

We realize that making different process prediction approaches comparable is not only a technical but also a data-related challenge. No established benchmarking datasets for predictive process monitoring exist yet, which complicates comparison. As a result, more and more datasets are used, and demand more and more work to establish comparisons. To mitigate this situation, we designed a software framework that helped us compare the sixteen total model-batching-strategy combinations on a variety of datasets in an extensible way. We want to make this framework accessible to future researchers to facilitate model development and sharing of implementations.

The proposed framework provides a simple training frontend and integrates two concepts: Builders and Batchers. As Figure 22 shows, the two concepts are introduced as abstract base classes from which the actual model and batching strategy implementations are derived.

Builders construct Keras models and also define the structure of the training and test data. This is important as models may not only

have one input layer but two or more. These pre-formatted data sets are not structured into batches yet. One such Builder is implemented for every model type by inheriting from the abstract base class `AbstractBuilder`.

Batchers take the pre-formatted data sets and structure them into batches. For each batching strategy, a class inherits from the abstract base class `AbstractBuilder`.

The `model_runner` frontend ties these two concepts together with training logic and provides a basic command-line interface as Figure 23 shows. It allows for flexible training of a model with a specific Builder and a specific Batcher. The frontend permits placing the training process on a specific GPU and allows defining the output directory for the model and performance measurement files. The training algorithm also implements early stopping, thresholds of which can be configured via a configuration file.

With the help of this framework, future researchers only need to subclass `AbstractBuilder`, and can directly evaluate the model performance on any given dataset without having to set up the whole training environment. By merely sharing their Builder and Batcher implementations, it will be much easier for other researchers to reproduce findings. The framework and detailed documentation is hosted under github.com/flxw/nitro4ppm.

In this chapter, we explained how Keras requires the dimensions of samples in a batch to be constant, and outlined four strategies to deal with this requirement. The strategies present ways to construct batches from variable-length sequential data that honor Keras' requirement. We also presented how the training framework integrates these strategies, and facilitates training models with multiple strategies on multiple datasets. In the next chapter, we explain which datasets we use to train the models, and how each model was configured.

```
usage: model_runner.py [-h] [--gpu GPU]
                       [--output OUTPUT]
                       {evermann,schoenig,sp2,pfs}
                       {padding,grouping,individual,windowing}
                       datapath

The network training runner script of nitro4ppm!

positional arguments:
  {evermann,schoenig,sp2,pfs}
                        Which type of model to train.
  {padding,grouping,individual,windowing}
                        Which strategy to use for feeding
                        the data into the model.
  datapath              Path of dataset to use for training.

optional arguments:
  -h, --help            show this help message and exit
  --gpu GPU             CUDA ID of which GPU the model
                        should be placed on to
  --output OUTPUT       Target directory to put model
                        and training statistics
```

Figure 23: The `model_runner` command-line frontend for the training framework

6 METHODOLOGY

In this chapter, we detail how we processed the datasets, and which other methods we used to get the models ready for training. To prepare the data, we follow the KDD process, a routine to prepare data for Data Mining use cases. First, we present and justify our data selection in Section 6.1. We then lead through the pre-processing phase in Section 6.2, and follow up with the transformation phase in Section 6.3. This section also covers the used logic to engineer the features for the SP2 and PFS models. Finally, the system configuration and the used training strategies are presented in Section 6.4.

The reimplementations of Evermann's and Schönig's approaches are referred to as EVM and SCH from here on.

6.1 DATA SELECTION

The first step of the KDD process is about picking the right data for the right problem. In Section 1.2, we discuss the challenge posed by the great variety of datasets in predictive process monitoring. While we can not establish a standard, we ensure comparability of our results to a variety of works by evaluating the four models on eight datasets:

- BPIC11, an event log from cases in a Gynaecology department of a Dutch Academic Hospital [9]
- BPIC12, an event log for loan and overdraft applications from a Dutch Financial Institute [10]
- BPIC15, five event logs that contain building permit application cases of approximately four years from Dutch municipalities. One log contains data from one municipality. The process is the same, but local deviations are present. The logs are referred to as BPIC15-1 to BPIC15-5 in the following [12]
- HelpDesk, a log from a ticketing management process of the help desk of an Italian software company [37]

The four logs cover a spectrum of process complexity. We take the increasing number of distinct activities and variance in trace length to be indicators of complexity. Table 8 sums up these properties.

The least complex traces are found in HelpDesk and BPIC12, with a small variance in trace length and a tiny number of different activities. While the HelpDesk log contains events from a rigid, predefined

Dataset	min TL	max TL	$\overline{TL}$	σ_{TL}	Traces	Events	Activities
HelpDesk	1	14	3.60	1.19	3804	13 710	9
BPIC12	3	96	12.56	11.33	13 087	164 506	23
BPIC15-4	1	116	44.91	14.63	1053	47 293	356
BPIC15-3	3	124	42.35	16.05	1409	59 681	383
BPIC15-5	6	154	51.10	15.82	1156	59 083	389
BPIC15-1	2	101	43.55	16.74	1199	52 217	398
BPIC15-2	1	132	53.31	20.32	832	44 354	410
BPIC11	1	1 814	131.49	194.01	1 143	150 291	524

Table 8: Properties of the traces contained in the used datasets, sorted by activity count. TL abbreviates trace length

process, process models were also easily mined from BPIC12 in competition submissions [4]. We include the HelpDesk log for comparability, but we believe that it is not well suited for the prediction use case, since it only consists out of dictionary-encoded activity names and timestamps. Furthermore, its cases seem to follow a very well defined ITIL-style process of very little variability.

The process in BPIC11 seems to be the most complex, with long traces and a high number of distinct activities. Process models were barely obtained from it during the competition [16].

BPIC15 resides relatively in the middle of the two others in terms of complexity. The mined process models from it were more complex than for BPIC12 [33].

Additionally, the datasets allow us to compare our findings to the following works on next-element predictions. When comparing, it is important to differentiate between the works that focused on case-specific predictions like us and those that focused on entire event streams:

- Predicting the next element in a stream, not specific to a case
  - BPIC12: Evermann et al. [23]
- Predicting the next element for a specific case
  - BPIC12 & HelpDesk: Böhmer et al. [15]
  - BPIC12 & HelpDesk: Tax et al. [72]

After the logs are chosen, the contained data can be preprocessed.

6.2 DATA PRE-PROCESSING

In the second step of the KDD process [24], the data is pre-processed to prepare it for a data mining use case. In this step, we eliminate

generally known properties that hinder machine learning model performance. This encompassed three steps for each dataset:

1. Filter for completed events
2. Drop all columns which contain only a single value
3. Eliminate features which correlate strongly

In step 1, the `lifecycle:transition` feature was filtered so that only completed events were left. This reduces dimensionality and improves comparability because BPIC11 only contains completed events.

In step 2, the `lifecycle:transition` feature was removed in every dataset since it had been filtered before. No other features exhibited zero entropy.

In step 3, the correlation between features was calculated in a pairwise fashion. Correlating features often harm prediction accuracy [50]. We calculated the correlation with the bias-corrected version of Cramér's V to incorporate categorical features [14]. The heatmaps that highlight strong correlations between different features are presented in Appendix A. We removed a feature if its correlation with another feature was large in relation to the other correlations. As evidenced in Figure A.49, the BPIC12 dataset with its small number of features did not require any removals. A heatmap for the HelpDesk log was not created, as it only contained two features. Table 9 summarizes which features were removed and which ones were kept after the three steps.

Dataset	Omitted features	Remaining features
HelpDesk		`concept:name`, `time:timestamp`
BPIC11	`lifecycle:transition`, `Producer code`, `Activity code`, `Specialism code`	`time:timestamp`, `concept:name`, `org:group`, `Number of executions`, `Activity code`, `Producer code`, `Specialism code`, `Section`
BPIC12	`lifecycle:transition`	`org:resource`, `concept:name`, `time:timestamp`
BPIC15	`lifecycle:transition`, `activityNameNL`, `activityNameEN`, `action_code`, `dueDate`	`time:timestamp`, `dateFinished`, `planned`, `concept:name`, `monitoringResource`, `org:resource`, `question`

Table 9: Omitted features during pre-processing

The activities described above were conducted in JupyterLab notebooks [60], where Anaconda [8] was used to create a stable development environment. The OpyenXes [2] library proved to be especially useful for transferring raw XES logs into more usable data types. This concludes step two of the KDD process.

6.3 DATA TRANSFORMATION

After selecting suitable features, the third step of the KDD process [24] suggests the transformation of these features into representations that are better received by predictive models. Feature engineering is an additional part of this step. Thus, the constructions of the SP-2 and PFS features are explained in the following subsections.

Transforming Basic Features

The features remaining after selection can be divided into three categories: timestamps, numerical and categorical. The transformation applied to each category of feature is described briefly in the following:

Timestamps are converted to numerical features for each trace, i.e., each timestamp t_i is made relative to the beginning of a trace by replacing it with the result of $t_i - t_0$. Relative timestamps have been shown to work with sequential data [51].

Each numerical feature x is normalized using the min-max method with min- and max-values specific to each trace. This also includes the relative timestamps. For the traces that are so short that $\mathrm{min}(x)$ equals $\mathrm{max}(x)$, an edge case is introduced:

$$\mathrm{normalize}(x) = \begin{cases} \frac{x - \mathrm{min}(x)}{\mathrm{max}(x) - \mathrm{min}(x)} & \text{if } \mathrm{min}(x) \neq \mathrm{max}(x) \\ 1 & \text{otherwise} \end{cases}$$

Categorical features were encoded using one-hot encoding, since no ordinal relationships were present in the data, and the dimensions were not too large.

Engineering the SP-2 Features

SP-2 features were proposed in the SPiCe submission by Shibata et al. [66]. In a binary bag-of-words feature vector, they mark whether an element has occurred in the past. As such, these features are engineered iteratively. Listing 1 shows the construction of the feature vector in Python. Table 6 in Section 4.2 illustrates how such a vector could look like.

For every trace `t`, a new feature vector `sp2_df` is created and the occurrence of the first activity, contained in the column `target_col`, is marked inside it (line 7). A loop begins over the remaining steps, where each previous row inside `sp2_df` is copied into the currently indexed row (line 17). After copying, the presence of the current activity gets marked (line 18). This repeats itself until the trace is processed completely, leaving behind a complete SP-2 feature vector for a trace.

```
1  # Dataframe initialization with zeroes
2  sp2_df = pd.DataFrame(columns=activity_labels,
3                        index=range(0,len(t)),
4                        dtype=np.bool)
5  for col in sp2_df.columns: sp2_df[col].values[:] = 0
6
7  # mark first occuring activity
8  cname = "{0}{1}".format(sp2_prefix, t[target_col][0])
9  sp2_df[cname].values[0]  = 1
10
11 # copy over values from last row and
12 # set activity labels accordingly
13 for i in range(1,len(t)):
14     first_activity_name = t[target_col].iloc[i]
15     col = "{0}{1}".format(sp2_prefix,first_activity_name)
16
17     sp2_df.values[i] = sp2_df.values[i-1]
18     sp2_df[col].values[i] = 1
```

Listing 1: SP-2 feature generation code for a single trace `t` and a specific target column `target_col`.

Engineering the PFS Features

As presented in Section 4.3, PFS features encode subsequence occurrences. Klinkmüller et al. claim that these features can help models cover a broader variety of relationships [48].

The features for the PFS model were created with the help of the PrefixSpan algorithm implementation from the *prefixspan-py* library [75]. Similarly to the SP-2 features, they are based on the target variable.

```
1  prefixspan_traces = PrefixSpan(encoded_traces)
2  closed_sequences = prefixspan_traces.topk(25, closed=True)
```

Listing 2: Obtaining closed sequences with *prefixspan-py*

As Listing 2 shows, prefixspan-py greatly facilitates obtaining subsequences. The depicted function call returns the top 25 closed sub-

sequences ranked by their support metric. These subsequences are picked as features to be encoded. We choose the number of subsequence features $l = 25$. With this choice, the selected subsequence features are well above the minimum support of 0.05 proposed by Klinkmüller et al. [48].

After mining the sequences, a loop is executed for every trace `t` - shown in Listing 3. This loop iteratively constructs the PFS feature vector.

First, the binary feature vector `subseq_df` is initialized (line 3). For each index `i`, it is checked whether any of the mined subsequences starts at that position (lines 14-25). This is done by peeking ahead of `i` for the length of the subsequence, indicated by `j`. As in the case of the SP-2 features, the occurrence of a subsequence is marked with a boolean flag in the feature vector `subseq_df` from the row that it occurred in onwards.

Constructing Prediction Labels

For each event e_t at a given timestep t, the prediction label $\#_{activityname}(e_{t+1})$ was constructed, essentially picking the activity name from the following event. For the target of the final event of a trace, a marker for denoting the end of the sequence was introduced with `EOS`.

To finally work with the data, we needed to identify the column containing the activity name. The XES standard states that this column is called `concept:name` [3], which is subsequently used.

Splitting the Datasets

To verify model performance on unseen data, it is common to set aside a portion of the dataset for testing purposes.

We split all datasets into training and validation sets of complete traces. While 25% of the traces were used for validation, the remaining 75% were used for training purposes [50]. The traces were shuffled and stratified to contain an approximately similar distribution of trace lengths. This avoids possible biases towards trace lengths. The event order in a trace was not changed.

With the logs selected, preprocessed, transformed, and organized in separate sets, the models can be trained. How we approached this will be shown in the next section.

```
# Initialize the feature vector
# subsequences have not happened yet
subseq_df = pd.DataFrame(columns=subseq_labels,
                         index=range(0,len(t)),
                         dtype=np.bool)
subseq_df[:].values[:] = False
tlen = len(t)

# Dictionary encode activity names, entire trace
activity_codes = t[target_col].map(name_to_int)

for i in range(0, tlen):
  # loop through all subsequences
  for subseq_idx, subseq in enumerate(ps):
    if tlen <= i+len(subseq): continue

    # check if subsequence starts at offset j+i
    # abort at first mismatch
    subsequence_found = True
    j = 0

    while subsequence_found and j < len(subseq):
      if subseq[j] != activity_codes[j+i]:
        subsequence_found = False
        j += 1

    # if subseq took place, subsequence_found is still true
    if subsequence_found:
      subseq_df.values[j+i-1:,subseq_idx] = True
```

Listing 3: Subsequence feature generation code for a trace `t`

6.4 TRAINING CONFIGURATION

This section highlights the model configuration, and the employed hyper-parameters for each model. The batch sizes of each batching strategy are discussed, too.

We made use of the framework described in Chapter 5, which implements early stopping, and saves model snapshots when its validation loss hits a new record low. We configured it to stop training if the validation loss did not improve for ten epochs.

The implementation of the padding strategy required a small cue: The padding values incurred a significant memory overhead so that the prepared BPIC11 data would not fit into GPU memory. For this reason, we filtered out 20% of the longest traces.

Network	EVM	SCH	SP2	PFS
Optimizer	SGD	RMSprop		
Loss	Categorical crossentropy			
Weight initializer	Random normal	None	Glorot normal	
Epochs	50	100	150	150
Features	Activity name	All event attributes		

Table 10: Used hyper-parameters for each model

The models themselves were trained with different optimizers, and various initialization strategies for weights, depending on the original author's choice [23, 64]. Table 10 shows these hyper-parameters per network side-by-side and recapitulates the used event attributes. The batch size is also an important hyper-parameter. It depends on the particular batching strategy and is shown in Table 11.

To conduct the experiments, Docker containers were built with an Anaconda environment inside them [21]. Using a version of Docker for GPU applications running on NVIDIA hardware [17], each network-batch-formatting combination was trained and evaluated on a single NVIDIA K80 GPU. The HPI FutureSOC Lab [32] provided the computational infrastructure and multiple GPUs so that multiple models could be trained and evaluated simultaneously. The complete code is available at github.com/flxw/master-thesis-code.

This concludes the presentation of the methodology. We highlighted the processes which the used logs were obtained from and the characteristics of the traces contained in each. We chose the logs so that they cover a spectrum of process complexity in terms of activity count and trace length. Then, highlighted the pre-processing through filtering of correlating values. Afterward, we covered the transformation of the data into machine-readable input with min-max-normalization and one-hot-encoding. We ended the chapter with an explanation of the conditions under which the experiments were run. The results of the experiments are presented and discussed in the next chapter.

Dataset	Individual	Grouping	Padding	Windowing
HelpDesk	1	3.60 (1.20)	99	19
BPIC11	1	2.80 (4.99)	7	1054
BPIC 12	1	129.14 (333.58)	89	939
BPIC 15-1	1	10.33 (11.45)	9	365
BPIC 15-2	1	6.17 (7.57)	5	316
BPIC 15-3	1	12.27 (19.28)	10	414
BPIC 15-4	1	9.17 (14.59)	8	334
BPIC 15-5	1	9.52 (14.36)	7	420

Table 11: Batch sizes for the different datasets and batching strategies. Since the grouped batches vary in size, their size is stated as *average (standard deviation)*

7 EVALUATION

In this chapter, we present our evaluation criteria and the findings which we got from them. We evaluate the four models with a three-pronged strategy, targeting the following characteristics:

1. **Accuracy** - the share of correct next-activity predictions
2. **Training time** - the amount of time required for training
3. **Stability** - the change in prediction accuracy with progress

We measure accuracy because it is a good indicator of how often the model predicts the right class, and permits comparisons to other works. Training time is of interest because a real-world application should use as little computing resources as possible. We forgo inference timing measurements because few predictive process monitoring solutions are deployed in application scenarios yet. The stability criterion permits making a judgment about the behavior of accuracy over time. The works of Francescomarino et al. [25] and Klinkmüller et al. [48] inspired this measure. It provides an understanding of how the prediction accuracy of a model changes as the trace grows longer. This can facilitate building trust in the model, as indicated in the introduction of the thesis [15, 48]. During the evaluation, results for BPIC logs and the HelpDesk dataset should not be compared too closely, as the BPIC logs are a lot more resourceful in terms of data attributes.

We discuss the acurracy of the models in Section 7.1, followed by the discussion of the required training time in Section 7.2. We go on to discuss stability in Section 7.3, and conclude the chapter in Section 7.4.

7.1 ACCURACY

The accuracy on unseen data is a good indicator for how well a model predicts. It is the share of correct predictions that match their target labels among the number of total predictions:

$$\frac{n_{correct}}{n_{total}}$$

In the following paragraphs, we present the maximum accuracies obtained on the validation set of each log. For each log, we show a plot of the different model accuracies, grouped by batching strategy. The presentation of the logs is ordered by process complexity, similar to Table 8. We end the section with a verdict of the observations.

ACCURACY ON HELPDESK Figure 24 displays the different accuracies on the validation set from the HelpDesk log. With the individual, grouping strategies, all four models score above 0.80. As expected, the windowing strategy impacts accuracy significantly, and all accuracies see large degradations. The SP2 model is impacted the least by the windowing strategy, presumably because its SP2 features capture the history that is cut away. While the SCH, SP2 and PFS models do not show strong reactions to changes in the first three batching strategies, the EVM model fails with the padding strategy.

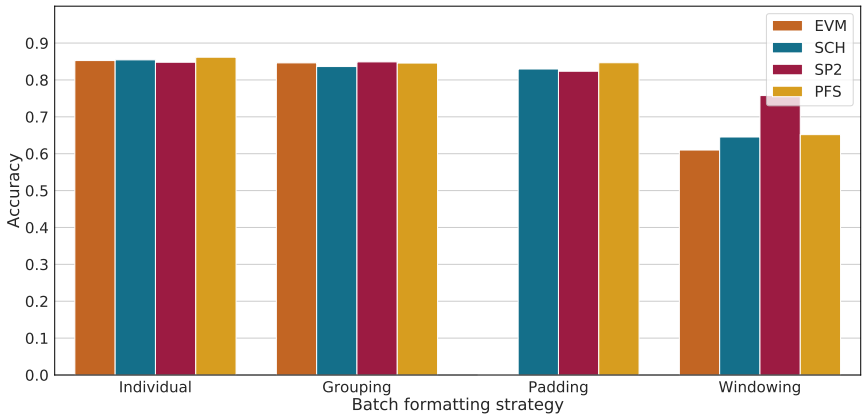


Figure 24: Best accuracies on the validation set of HelpDesk

ACCURACY ON BPIC12 The process contained in BPIC12 is more complicated than the one captured in the HelpDesk log. The accuracy measurements on it are visualized in Figure 25. Again, all four models exhibit very similar accuracies for the individual and grouping batching strategies, where all score above 0.75. On the padding strategy, the EVM model barely reaches an accuracy above 0, while the others score above 0.8. On all of the three strategies that supply the complete trace to the model, the accuracies are very similar. The windowing strategy leads to more significant differences between the accuracies. Here, the SP2 models outperform all other models. The worst accuracy is shown by the EVM model on all four strategies, although the difference to the next-best accuracy is small.

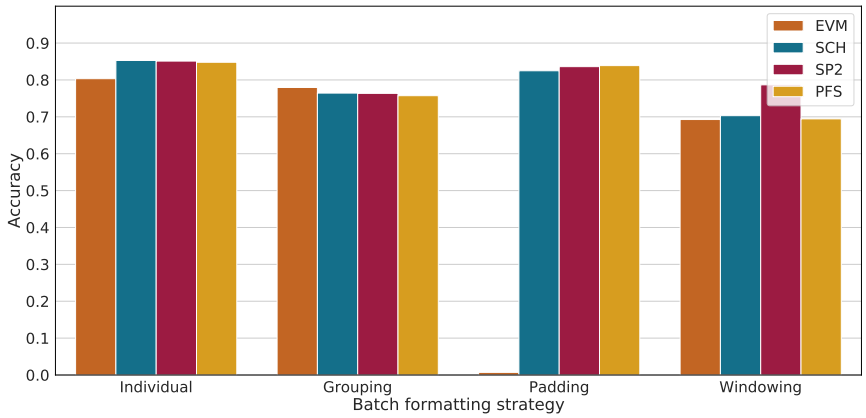


Figure 25: Best accuracies on the validation set of BPIC12

ACCURACY ON BPIC15 The accuracies obtained on the validation sets of the BPIC15 logs are visualized in Figure 26 to Figure 30. The results are very similar, so we describe the measurements in a grouping fashion per strategy and highlight common themes:

First, the accuracies of all four models are very similar across the individual and grouping strategies. The biggest difference in accuracy is 0.1.

Second, the SCH model always outperforms the other models slightly on the individual strategy.

Third, on the padding strategy, the SP2 model always is the most accurate. On the same strategy, the EVM model fails.

Fourth, the SP2 model gives the highest accuracies on the grouping, padding and windowing strategies. On the windowing strategy, its accuracies are the highest of all models by up to 0.2. The highest accuracies across all strategies are produced by the SP2 model on any BPIC15 dataset.

Fifth, the EVM model accuracy suffers the most from the windowing strategy. It is a general rule that, on any BPIC15 dataset, the EVM model performs second-best on the individual strategy, while it ranks last on any other strategy. Nonetheless, the differences to the next-best accuracy are always less than 0.2, except for when the padding strategy is in use.

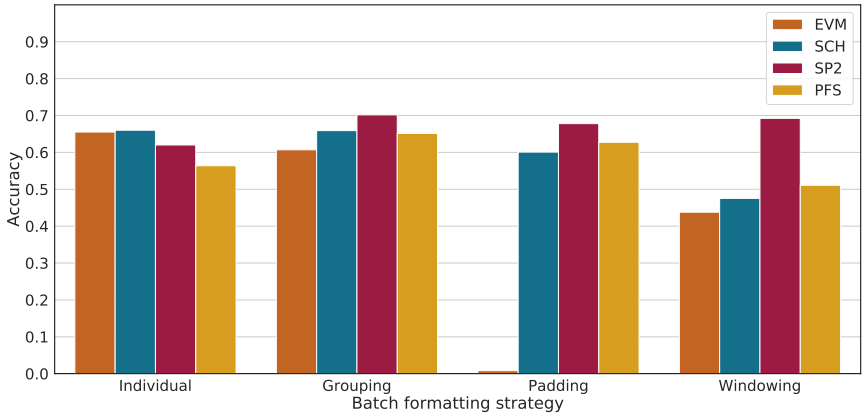


Figure 26: Best accuracies on the validation set of BPIC15-1

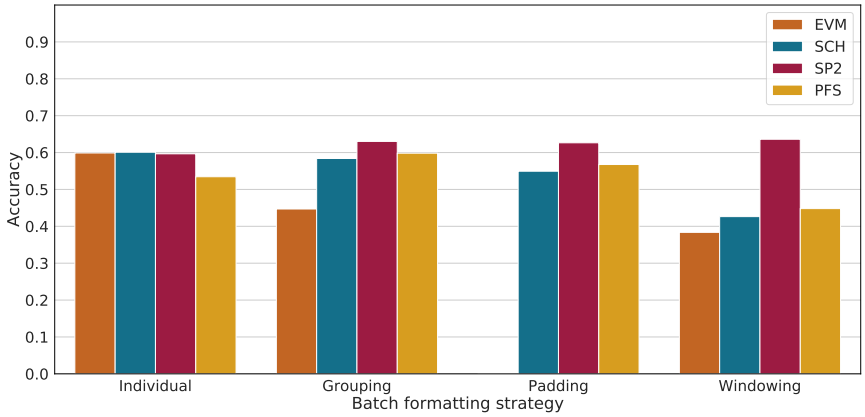


Figure 27: Best accuracies on the validation set of BPIC15-2

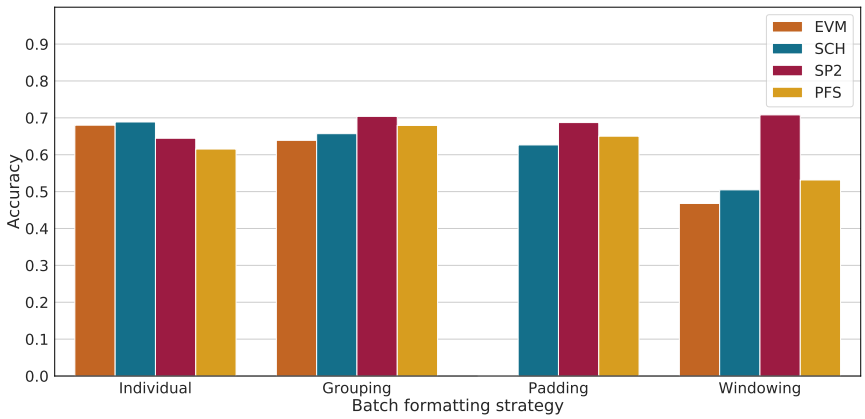


Figure 28: Best accuracies on the validation set of BPIC15-3

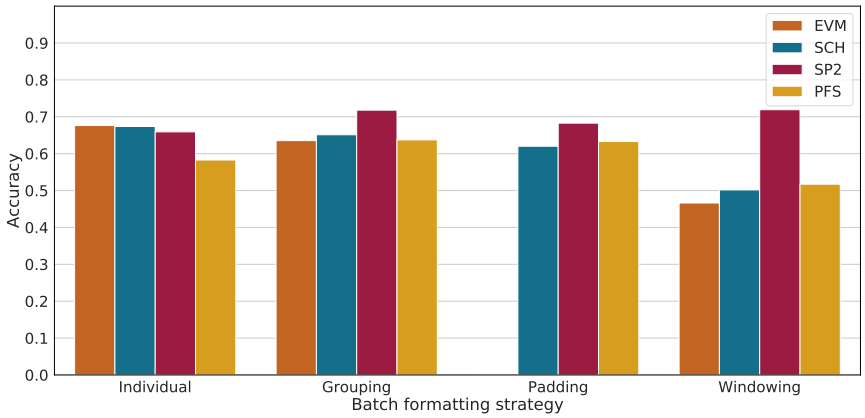


Figure 29: Best accuracies on the validation set of BPIC15-4

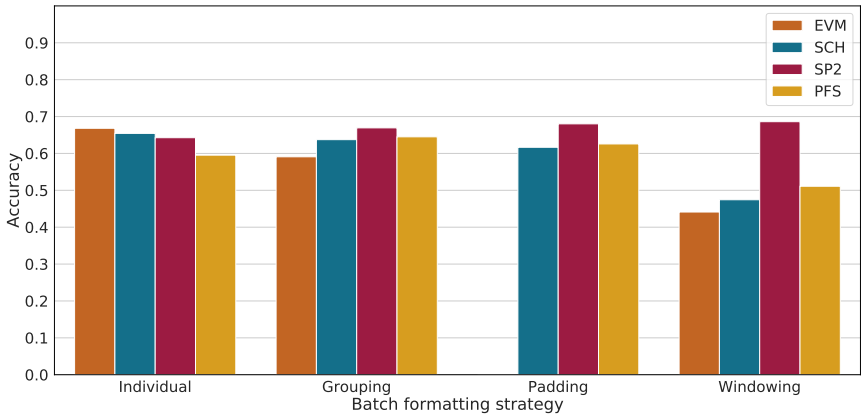


Figure 30: Best accuracies on the validation set of BPIC15-5

ACCURACY ON BPIC11 BPIC11 captures the most complex process. The validation accuracies that we obtained for this log are depicted in Figure 31.

With the individual strategy, the EVM, SCH and PFS models score above 0.6. The SP2 model reaches 0.4.

In the case of the grouping strategy, the accuracies of all four models are very similar between 0.65 and 0.7.

The padding strategy sees another failure of the EVM model, while the other three score just above 0.6.

The windowing strategy pushes the accuracies below 0.5, with the SP2 model scoring exactly 0.5.

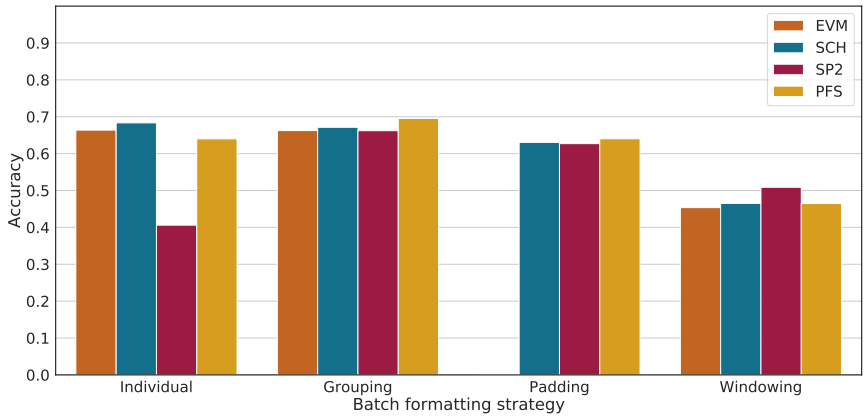


Figure 31: Best accuracies on the validation set of BPIC11

VERDICT ON ACCURACY Looking at the accuracy across all datasets, we detected five commonalities:

First, the batching strategies that supply the complete trace to the model lead to very similar results.

Second, the grouping strategy benefits high accuracies. The mean of the four model accuracies on different batching strategies shows this in Table 12. The mean value is highest with the grouping strategy in six out of eight cases. On BPIC12 however, the grouping strategy delivers relatively low accuracies.

Third, Table 12 on mean accuracies also reveals that the prediction accuracy goes down with increasing complexity of the process. The models are most accurate on the HelpDesk log, and become more inaccurate toward BPIC11.

Fourth, the windowing strategy always results in underperforming models. Nonetheless, the SP2 model copes with this strategy in the most efficient way, almost reaching maximum accuracies on BPIC15.

Fifth, the EVM model frequently was the most inaccurate, although with small differences to the next-best measurement.

Strategy / Dataset	Individual	Grouping	Padding	Windowing
HelpDesk	0.854	0.844	0.625	0.666
BPIC12	0.839	0.766	0.627	0.719
BPIC15-4	0.648	0.660	0.484	0.551
BPIC15-3	0.657	0.670	0.491	0.553
BPIC15-5	0.516	0.518	0.482	0.423
BPIC15-1	0.625	0.655	0.479	0.529
BPIC15-2	0.583	0.565	0.436	0.473
BPIC11	0.598	0.674	0.474	0.473

Table 12: Accuracy means for all models across a batching strategy. The grouping strategy is generally the highest. The rows are sorted by process complexity.

This concludes the presentation of the accuracy results. We continue with the presentation of the timing measurements in the following section.

7.2 TRAINING TIME

The training time that a model requires for an epoch is important to gauge the efficiency of its training process. As in the previous section, we discuss the measurements per log and finish with a verdict. For each log, a plot is shown that presents the mean epoch training time per model, grouped by batching strategy. The plots do not share the same scale on the y-axis. During the following paragraphs, it is important to keep in mind that the batch size has a direct effect on the training time since it corresponds to the number of weight adjustments that need to be calculated.

TRAINING TIMES ON HELPDESK How long training an epoch took with a model and a particular strategy on the HelpDesk log is shown in Figure 32. Training with the grouping strategy takes the least amount of time and the most by far with the individual strategy. The timings are very similar for all models on the same strategy with less than 10s difference.

TRAINING TIMES ON BPIC12 Figure 33 depicts the epoch training times for BPIC12. As for the HelpDesk log, the timings for models on the same strategy are very similar. By far, they are the highest for each model on the individual strategy at approximately 10 minutes. The other strategies take significantly less time.

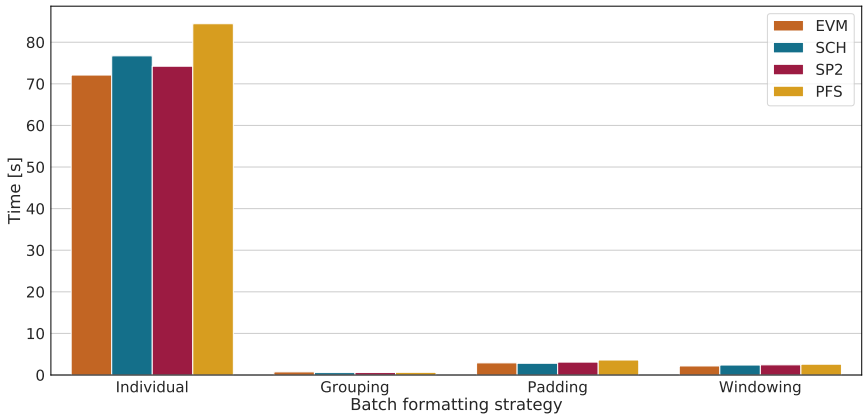


Figure 32: Training times measured on HelpDesk

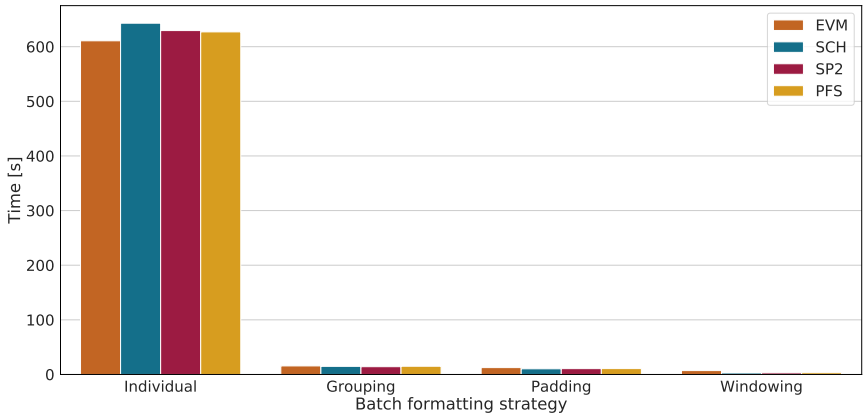


Figure 33: Training times measured on BPIC12

TRAINING TIMES ON BPIC15 The training timings taken during the model training on the BPIC15 log are shown in Figure 34 to Figure 38. The measurements are very similar across the five datasets in the log, which is why we explain them in a collective fashion.

The SCH, SP2 and PFS models always need a very similar amount of training time per epoch. Training the EVM models often needs less time than the others, in some cases even less than half. Across all five datasets, it is also possible to see that the individual strategy takes by far the most time. Significantly less time is needed by the padding strategy, although it still ranks third. The grouping strategy needs a little less time, and the windowing strategy trains fastest by far.

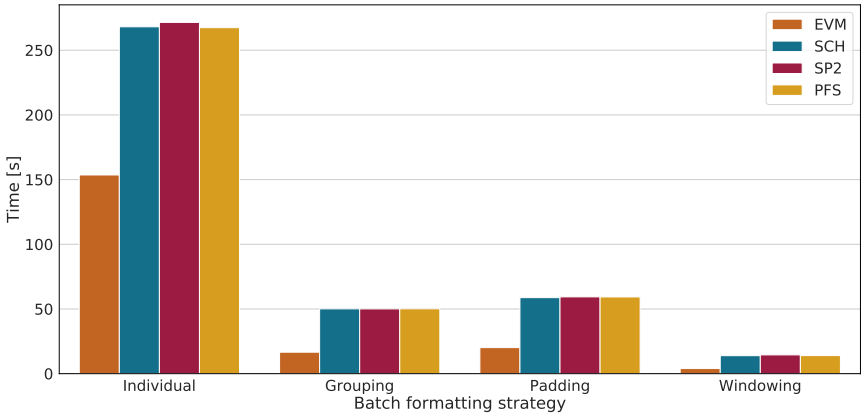


Figure 34: Training times measured on BPIC15-1

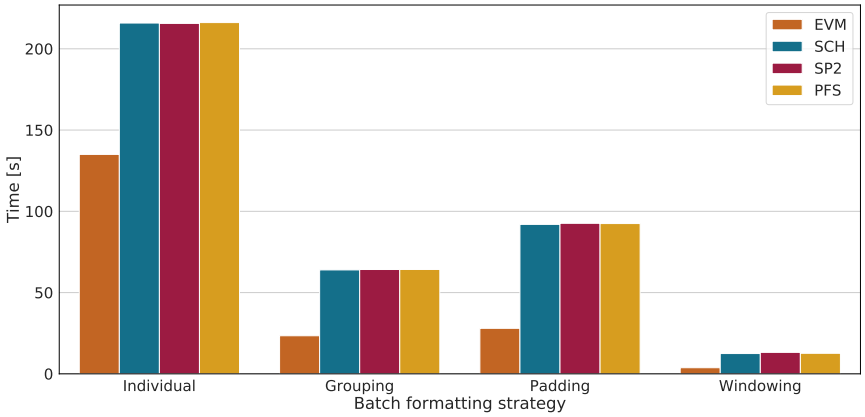


Figure 35: Training times measured on BPIC15-2

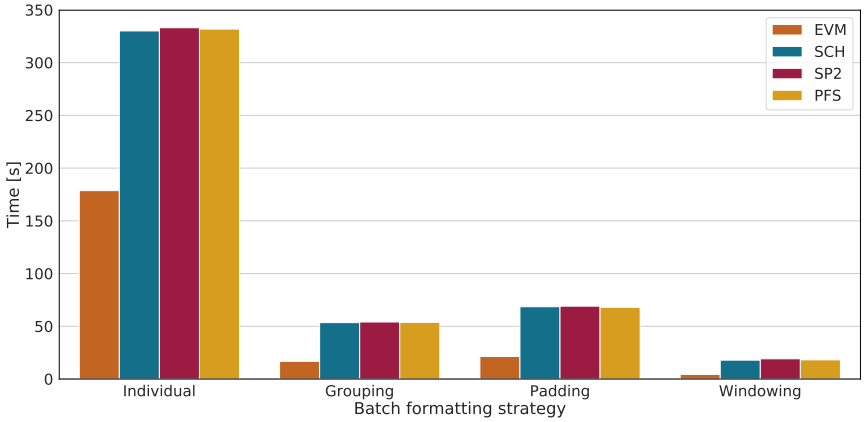


Figure 36: Training times measured on BPIC15-3

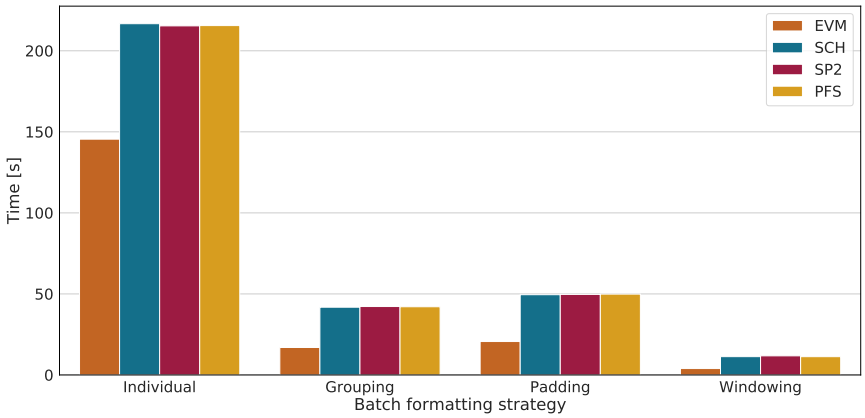


Figure 37: Training times measured on BPIC15-4

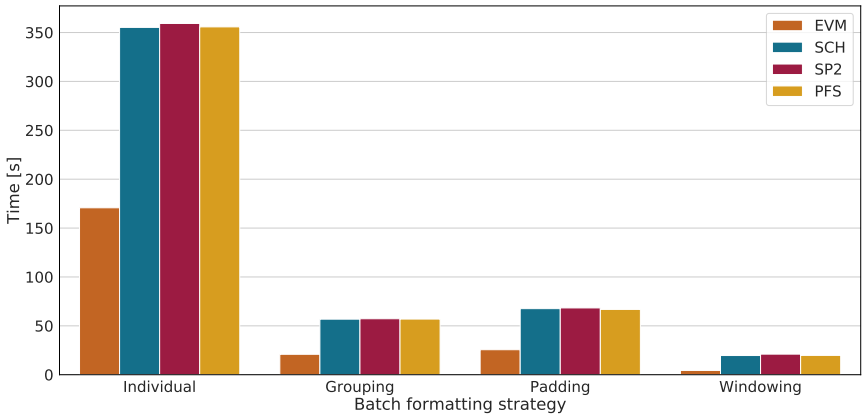


Figure 38: Training times measured on BPIC15-5

TRAINING TIMES ON BPIC11 Figure 39 shows the training times measured on BPIC11. On this plot, it is visible that the EVM model takes approximately half the training time of the other models. As before, the other models require about the same time for an epoch for the same batching strategy. The individual strategy causes the longest epochs. The grouping strategy speeds up training by approximately a minute. The padding strategy incurs a big drop in training times of around 6 minutes, and the windowing strategy is again the fastest.

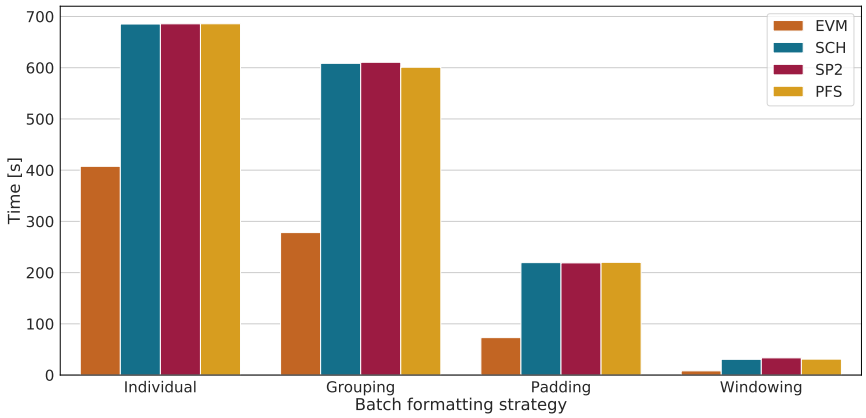


Figure 39: Training times measured on BPIC11

VERDICT ON TRAINING TIMES Across all datasets, we gathered the following observations. The individual strategy leads to the longest training times overall. This is not surprising, as Table 8 on page 48 reveals that it creates hundreds of batches per epoch - more than with any other strategy. Each batch incurs a weight adjustment, which leads to additional required computing time.

The grouping strategy leads to significant reductions in training times, depending on how diverse the trace lengths are. Table 8 also explains why the reduction is not as prominent with BPIC11: It exhibits up to 1814 different trace lengths, while the other datasets only exhibit 5% to 10% as many.

With the padding strategy, it is possible to overcome the limitations imposed by trace lengths and construct batches out of any number of traces. This makes it easier to optimize the batch size, which can lead to faster training times.

Fastest of all is the windowing strategy. It produces a constant number of timesteps per sample and allows for arbitrary batch sizes. The short samples additionally reduce the amount of calculation that needs to be done.

On each strategy, the SCH, SP2 and PFS models take approximately the same time to train. The EVM model trains up to 50% faster, depending on the dataset.

7.3 STABILITY

As we stressed in the introduction, the stability of the prediction accuracy along the progress of a case can strongly impact the level of trust that users put into a model [52].

In this section, we show the stability plots produced on each log, one for each batching strategy. We discuss the logs in the order of increasing process complexity and finish with a verdict. For space reasons, we only present the most informative graph per log in this section, and enclose the others in Appendix B.

Each figure for a batching strategy and a dataset contains four curves, one for each model. The curves show the model accuracy along the progression of all traces inside the validation set in steps of 5%. The stability on the HelpDesk log was calculated in steps of 10%, as most cases in it are only 13 steps long. The following paragraphs explain the curves in further detail.

STABILITY ON HELPDESK On the HelpDesk log, it can be seen again that the batching strategies have a substantial impact on the accuracy in different stages.

Figure 40 shows the stability of each model with the individual strategy. The four curves start around 0.8 and linearly rise to 0.85 until 60% progress. At the 60% mark, all curves show a small correction, before going up toward an accuracy of 1 at the end.

Figure B.55, Figure B.56, and Figure B.57 show the stability on the other strategies. With the grouping strategy, the curves are very similar, except for the EVM accuracy. Its curve starts around 0.8 and accelerates towards 0.5 at 90% progress. Then, it shoots up to 1 at the end.

The padding strategy makes all four models very unstable. After the SCH, SP2 and PFS curves start around 0.8, the curves see degradations and fluctuation around the 50% mark, with the PFS model completely failing at 0 accuracy at the end. The EVM accuracy curve does not move above 0.

The windowing strategy makes the curves very jittery. All curves start at 0 accuracy and stay below 0.5 accuracy until 60% progress. At this point, the curves shoot up above 0.8 and finish at an accuracy of 1. Only the PFS curve finishes at 0.5.

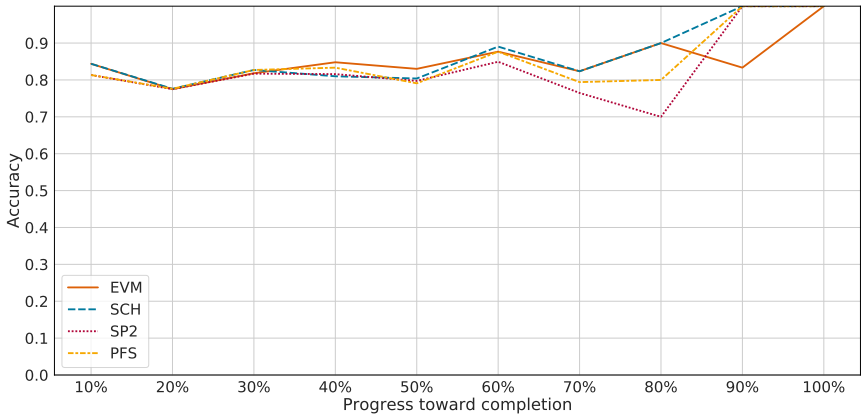


Figure 40: Model stability, individual strategy, HelpDesk

STABILITY ON BPIC12 The stability of the predictions on the BPIC12 log is discussed in this paragraph. Figure 41 shows the stability of each model with the individual strategy on BPIC12. The four curves start around 0.8 and immediately drop by 0.2. The SCH, SP2 and PFS curves swiftly recover by 15% and rise to 0.9 at 35% progress The EVM curve recovers slower from the initial drop, rejoining the others at 20%. From 35% onwards, all curves slowly sink toward 0.7 at 85% and rise sharply toward 1 at the end.

Figure B.58 shows the stability of the grouping strategy. The trajectories of all four curves are mostly as described for the individual strategy. A small, but notable difference is that the curves of the SCH, SP2 and PFS models are much closer together, and the EVM curve consistently set lower.

In Figure B.59, the stability for the padding strategy is shown. The curve trajectories are again very similar to the two aforementioned stability plots, but the overall accuracies are lower. The EVM model reacts strongly to the padding strategy. Its accuracy curve now starts at 0 accuracy, shortly reaches 0.15 around 80% and then drops back to 0 accuracy at the end. The other three curves show a stronger depression around the 75% mark, dropping down to an accuracy of 0.5, before recovering to 1 at the end. The SCH curve even drops to 0.4.

Finally, the stability plot of the windowing strategy in Figure B.60 reveals entirely new curve trajectories. The SP2 curve starts at 0.8, drops by 0.1, and recovers back to 0.9 until 50% progress. At this point, there is a sharp decline to 0.7. The curve recovers just as sharply, and slowly degrades to 0.6, and shoots up to 1 in the end. The EVM, SCH and PFS curves show a similar trajectory as the SP2 curve, albeit less accurate by around 0.150.

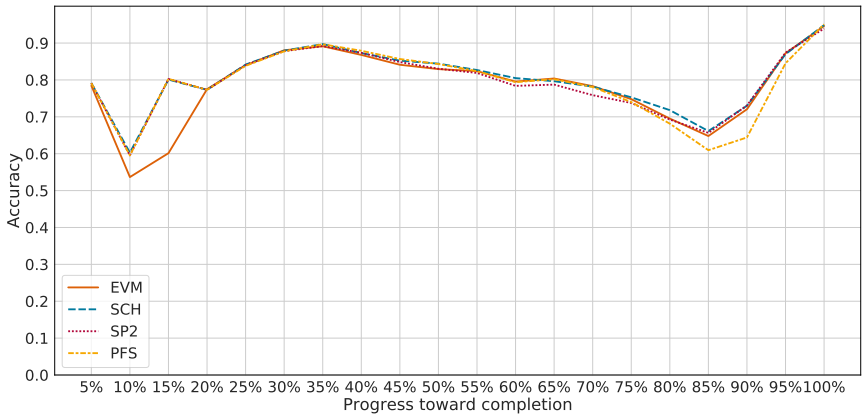


Figure 41: Model stability, individual strategy, BPIC12

STABILITY ON BPIC15 The BPIC15 log with its five datasets is discussed in this paragraph. The figures Figure 42, Figure 43 and Figure 44 show the stabilities for the individual, grouping and windowing strategies on BPIC15-5. In the appendix, Figure B.61 to Figure B.77 illustrate the remaining curves for the remainder of the log. The curves across BPIC15-1 to BPIC15-5 are very similar within a specific strategy, which is why we give a general description of the observations for each strategy in the following.

In the stability plot of the individual strategy in Figure 42, all four curves follow the same trend. They start around 0.7 and sink to 0.6 at 15%. At this point, the PFS curve is 0.15 below the rest and stays at this distance until the end. After 15%, the curves rise back up to 0.7 at 35%. From there on, they descend to 0.5 at 95%. Then, all three curves see an increase of 0.2, with the SP2 curve ending at 0.75. The SCH curve ends at 0.7, and the PFS curve at 0.6.

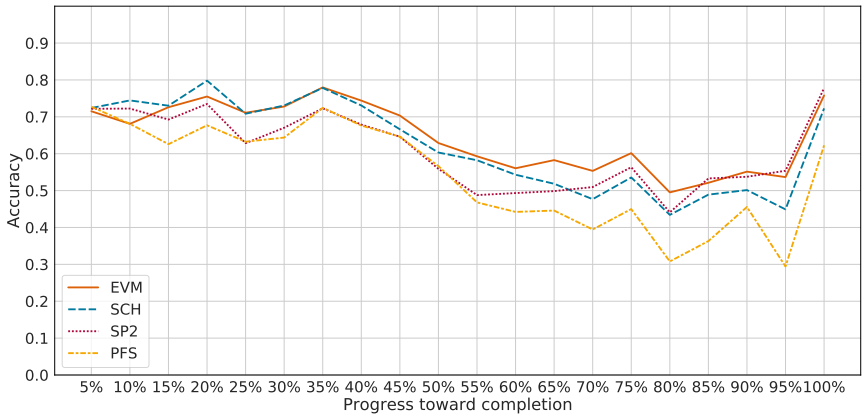


Figure 42: Model stability, individual strategy, BPIC15-5

The stability plot for the grouping strategy in Figure 43 shows very similar curve trajectories. In contrast to the individual strategy, the

four curves are a lot closer together, and the EVM curve has moved to the lower end. This is visible on the other BPIC15 datasets, too.

The plots for the padding strategy show steadily declining curves for SCH, SP2 and PFS models from 0.7 at the beginning towards an accuracy of 0.4 around 90% progress. Then, the curves jump up by 0.2 to 0.3 to finish around 0.7. The EVM curve does not surface above 0. For BPIC15-1 and BPIC15-4, the SCH curve also dips down after the start. On BPIC15-1, the PFS curve also dips down, leaving the SP2 curve as the sole remainder on the steadily declining trajectory described.

The plots for the windowing strategy in Figure 44 paint a simple picture. The EVM, SCH and PFS curves start around 0.6, and descend toward 0.3, spiking up toward 0.5 at the end. The SP2 curve hovers above, starting around 0.8, and linearly descends to 0.55 until 90% progress. Then, it also jumps up by 0.3 at the end of the plot.

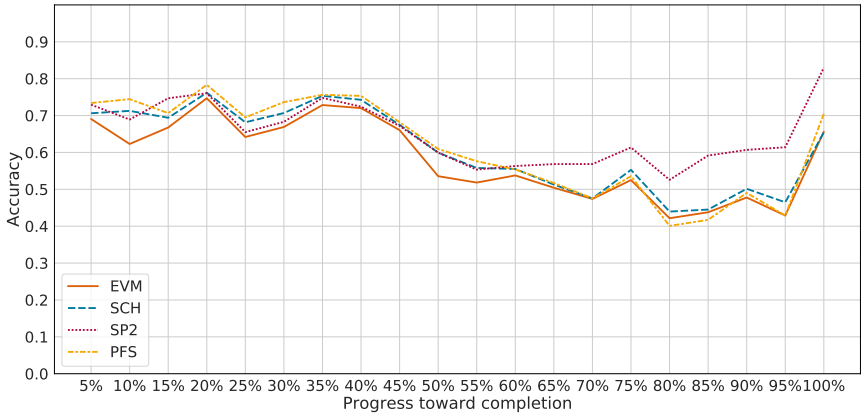


Figure 43: Model stability, grouping strategy, BPIC15-5

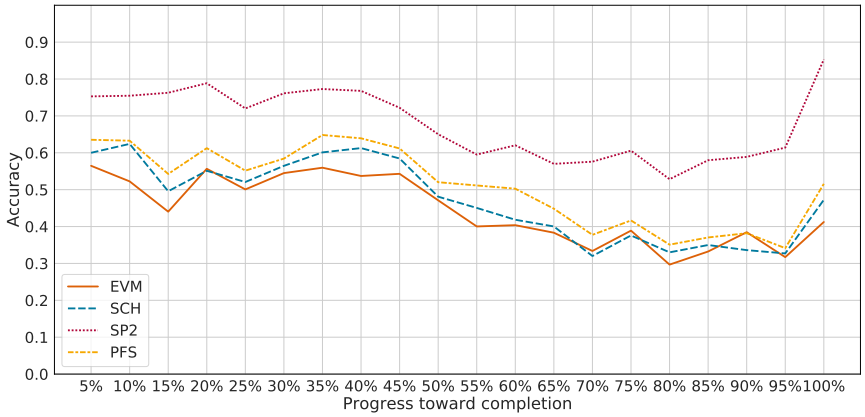


Figure 44: Model stability, windowing strategy, BPIC15-5

STABILITY ON BPIC11 The most informative plot from the BPIC11 stability measurements is shown in Figure 45 for the grouping strategy. It illustrates how the accuracy is relatively constant for the EVM, SP2 and PFS models around 0.7. Also, the plot shows well how close the curves of the three models are. All curves drop slightly in the beginning and then stay constant until the end, where there is another small drop. The SP2 model accuracy drops heavier in the beginning and stays constant around 0.65.

Little do the curves change for the individual strategy. Only two minor differences surface: The SCH and PFS curves are further apart, and the SP2 curve drops asymptotically from 0.65 in the beginning to 0.1 at the end.

With the padding strategy, the SCH, SP2 and PFS curves start around 0.75, drop to approximately 0.55 at 55% and stay constant until the end of the process. There is no accuracy drop in the end. The EVM curve stays level at 0 accuracy.

The windowing strategy makes the initial drop from all four models more pronounced from 0.75 in the beginning to just above 0.4 from 35% onwards. The SP2 model accuracy is approximately 0.1 higher than the others until sinking to their level 80% From there on, the curves stay constant.

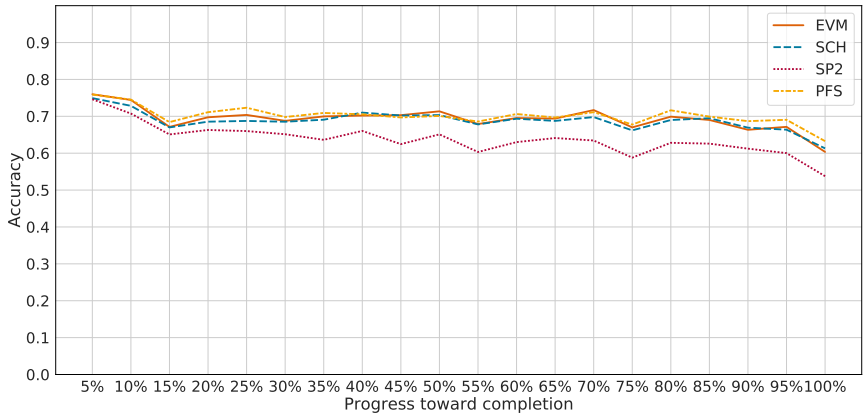


Figure 45: Stability curves for grouping strategy on BPIC11

VERDICT ON STABILITY In our verdict on the stability measurements across all datasets, we focus on five key observations.

First, we observed that many accuracy curves started with a drop in accuracy. Similarly, accuracy spiked up towards the end of the process, like on the BPIC12 log. The BPIC12 log contains traces from a process that covers a loan application. On this log, the accuracies taper off until 85% progress, when they spike up again.

Figure 46 reveals a process model that was mined from BPIC12. Seeing the model in context with the stability curve reveals that the accuracy drops and spikes could be related to the varying degrees of

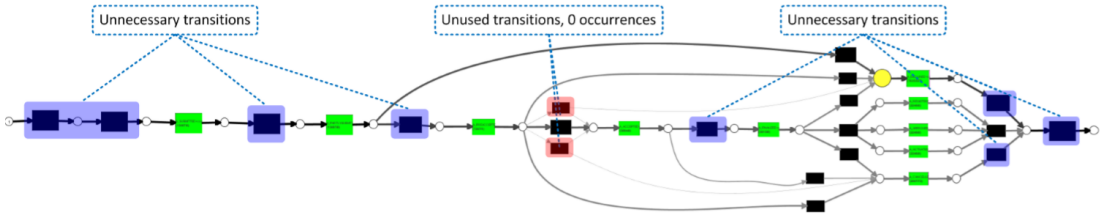


Figure 46: A mined process model from BPIC12. Illustration taken from [4]

complexity in different stages of the process. Every application begins in the same way, i.e., variability in the first stages of the process is low, which might cause accuracy to be high. As the process progresses, the variability increases, which correlates with the accuracy decline until 80% progress. The processes in BPIC12 always end either on approval or dismissal of the application, boosting accuracy from the moment that no more splits have to be anticipated. This could hint at a connection between prediction accuracy and complexity of the stage of a process.

Second, we realized that the grouping strategy did not only contribute to high overall results but brought curves closer together for the four models. This might be a symptom of a harmonizing effect of the strategy. We visualized this effect in Figure 47. The heatmap shows the standard deviation among the best accuracies of the SCH, SP2 and PFS models for each dataset and strategy. It is clear that the grouping strategy leads to the lowest standard deviations.

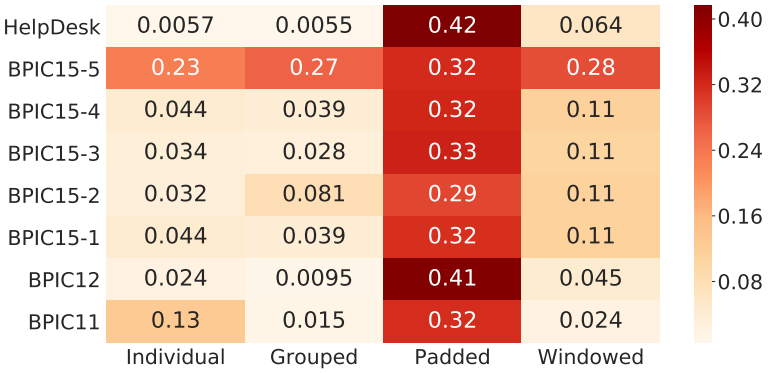


Figure 47: The grouping strategy often reduces the standard deviation between the highest accuracies of the models

Third, within a batching strategy, all models produce accuracy curves that follow the same trend. On the padding strategy, the EVM model consistently breaks down.

Fourth, the windowing strategy incurs very unstable curves. While this statement seems to be especially true for the longer processes in

BPIC11 and BPIC15, the windowing strategy seems to have less of an impact with shorter processes as evidenced with BPIC12.

Fifth, the SP2 model best handles the missing history on the windowing strategy. Its curves are the highest among the windowing strategy measurements on any dataset. Also, the SP2 accuracy curves on the windowing strategy are among those with the smallest fluctuations.

We connect these observations to the other findings from the other verdicts in the following section.

7.4 DISCUSSION

In this section, we connect the observations made in the previous sections on accuracy, timing, and stability of the models. Furthermore, we discuss resulting learnings and potential reasons.

The EVM model consistently underperforms, although it trains faster. Furthermore, it does not seem to work at all with the padding strategy. The SCH model works better, although both architectures are very similar. A fundamental difference between the EVM and SCH models is an embedding layer, which we suspect to be the cause for the lower accuracy. We believe that the embedding layer requires more data points per target class than currently included to perform well. In the HelpDesk log, there is a tiny number of classes and a large number of traces, which could be a reason for the EVM model scoring above 0.85. Also, we noted that the EVM model produced relatively better accuracies with the individual strategy than on the others. This might be due to more frequent adjustments of the word embedding.

We find that the grouping strategy frequently delivers the best results in terms of speed and accuracy. Nonetheless, there is potential for optimization, as the relatively low accuracy on BPIC12 suggests. The reason for this accuracy can be found in Table 8 and Table 11 in Chapter 6. BPIC12 has the highest number of traces, and a relatively even distribution of lengths, resulting in a small standard deviation. This makes the strategy place too many traces into a single batch, causing lost optimization opportunities. With this in mind, the grouping strategy should be enhanced to split batches if they exceed a certain size threshold.

We were also able to confirm Klinkmüller et al. and their statement that the windowing strategy leads to unstable results [48]. While it may be a performant strategy to use for time-series prediction, it does

not work very well for predicting the future of a single case.

Furthermore, we expected the models to become more accurate toward the end of the process. This did not turn out to be the case, but instead, we saw a connection of process complexity and accuracy. This connection was visible both in absolute accuracy and in prediction stability measurements. Table 12 on page 63 shows that the mean accuracy of the three top-performing models on each dataset goes down as the process complexity goes up. In the stability measurements, it was possible to see that the models gave very accurate predictions in phases of low variability. We showed in the preceding section that process phases with few decision points coincide with phases of higher accuracy.

It is worth reflecting on this point. If a machine learning model is incapable of understanding the choices made in certain phases of a process, it could either be a data problem or a process problem. In the latter case, a potential reason could be that decisions in phases of low accuracy are not standardized or overly complicated. Using the stability information in this way could create a feedback loop for process optimization.

After discussing potential reasons for our findings and consequences thereof, we compare our accuracy measurements to other works on the same datasets.

7.5 COMPARISON TO OTHER WORKS

We end the evaluation with a comparison of our best results with publications that also based their results on the BPIC12 and HelpDesk logs. The comparison is presented in Table 13.

We want to stress that while Evermann et al. [23] were able to obtain an accuracy of 0.768 for predicting the next event on BPIC12, they did not focus on specific cases. They focused on the whole event stream, and thus did not plan to predict the future of a single case.

Böhmer et al. and Tax et al. worked on next-activity prediction for a particular case. Böhmer et al. used a method based on probability distributions and obtained 0.77 on both logs. Tax et al. made use of neural networks, and obtained 0.71 on the BPIC12 log and 0.76 on the HelpDesk log.

The comparison table shows that all four models deliver higher accuracies on both logs. The PFS model surpasses the accuracies in the other works by 0.09 or more on the HelpDesk log. On the BPIC12 log, the SCH model outperforms the other works by 0.08 or more. We obtained these accuracies with the individual strategy.

Model	HelpDesk	BPIC12
Evermann et al. [23]	-	0.768
Böhmer et al. [15]	0.77	0.77
Tax et al. [72]	0.71	0.76
PFS	0.862	0.848
SCH	0.854	0.853
SP2	0.848	0.850
EVM	0.853	0.802

Table 13: Accuracy comparison to published numbers

In this evaluation, we discussed the accuracy, training time and accuracy stability of the models on the individual logs. We determined that the grouping strategy delivers the most promising results, but can still be optimized. Furthermore, we noted that the complexity of the traces in the log has a significant impact on stability. Depending on the amount of variability in different stages of the process, model accuracy varies dramatically. We could confirm that our models outperform recently published approaches. The next chapter describes ways to improve and continue our research and concludes the thesis with a summary of the results.

8 CONCLUSION

In order to present our conclusion, we divide this chapter into two sections. In Section 8.1, we recapitulate the results of our work. We elaborate on our limitations and connect them to steps with which to carry research further in Section 8.2.

8.1 SUMMARY OF THE RESULTS

In this thesis, we compared four different neural network architectures on three criteria in predicting the next activity in a running case. The comparison was implemented in Python and Keras and yielded a neural network training framework for sequence prediction. It allows for quickly training different network architectures with different batch construction strategies for variable-length traces. The dataset for training can be exchanged.

The framework not only allows for training numerous model-strategy-dataset combinations but also tackles an issue within current research: comparability. Most publications in predictive process monitoring use different datasets and do not publicize their implementation, thus making comparisons hard. It is considered standard practice in machine learning research to make code and datasets accessible for reproduction [62]. By using our framework, researchers only need to make the implementations of the base classes accessible to the public along with the data. Because the framework covers a large part of typical training logic, development time is also reduced.

To establish a baseline for the comparison, we implemented two of the four neural network models from publications by Evermann et al. and Schönig et al. [23, 64]. We then ported an approach from NLP to predictive process monitoring, resulting in a third model [66]. By adapting the input features of the ported model in accordance with the findings of another publication, we obtained a fourth model [48]. In the order of introduction, we refer to the four models as EVM, SCH, SP2, and PFS.

In our evaluation, we focused on prediction accuracy, training time, and the stability of the prediction accuracy over time. The SP2 model consistently ranked at the top or among the best performing models, which we attribute to the use of SP-2 features. In comparison with recently published accuracy numbers on the BPIC12 log, the SCH

and SP2 models were more accurate by 0.07 and attained 0.853 and 0.850 [15, 23]. On the HelpDesk log, the PFS and SCH models outperformed the published numbers, too, and reached accuracies of 0.862 and 0.854.

Furthermore, we learned that grouping traces by their length to produce batches contributes to the highest accuracies and takes the least training time. Generally, the batching strategies which supply complete history to the model perform best. The strategy of using a single trace per batch sees the smallest accuracy gains and requires the most time to train.

Another finding of the measurements was that process complexity seemed to influence prediction accuracy. The more activities a trace had, and the longer it was, the less accurate the predictions became. This connection was also visible in the stability measurements when process complexity went up or down in a stage of the process: in a mined process model, process phases with few decision points coincided with phases of higher accuracy in the stability curve.

Finally, we could confirm findings from Klinkmüller et al. that omitting history from process traces negatively impacts prediction accuracy [48].

8.2 FUTURE WORK AND LIMITATIONS

In this section, we present items which could carry this research further in publications or subsequent master thesis. We cluster the future work by topic and connect our limitations where applicable.

BENCHMARKING DATASET Currently, predictive process monitoring suffers from not having standardized benchmarking datasets for researchers to evaluate their algorithms on. This forces researchers to resort to a multitude of datasets, or to even use non-public data. In our case, we used the BPIC12 and HelpDesk logs for evaluation and accuracy comparison. We feel that the HelpDesk log with its two columns can neither be considered a realistic nor an optimal source of data for this use case. This is why we advocate for standard benchmarking datasets, which are common in many other machine learning disciplines. We believe that it is time to draw up such datasets with real data or close to real-world data, representing industry-typical processes. By having different datasets with different characteristics, prediction approaches could be evaluated along different dimensions. As there is already a tendency to use BPIC logs, the next year's challenge could be on predictive process monitoring. There, a log could be advertised as a future standard to evaluate on. Otherwise, the number of datasets that researchers have to use to ensure comparability will only ever increase.

IMBALANCED CLASSES Predictive models learn better with a balanced target class distribution. The prediction targets, activities in our case, are not distributed uniformly across traces. This distribution problem is also commonly encountered in NLP. Weighting classes based on their occurrence is a possibility to deal with it. Keras offers a mechanism to set the weights on a per-batch basis, which could see accuracy gains [41]. We did not pursue this possibility, as our goal was to evaluate different types of models and batching strategies, and not to maximize the performance of a single model.

FEATURE ENGINEERING AND HYPER-PARAMETER TUNING The PFS model uses a binary feature vector. Every bit inside it encodes the occurrence of a single subsequence. We chose that the vector should encode the presence of the 25 closed subsequences with the highest support of the respective dataset. We chose not to optimize the number of encoded subsequences, as our goal was different from maximizing accuracy gains from feature engineering. However, the chosen threshold should be varied and explored further, as it most likely differs for each dataset. It could also be made relative to a minimum support value so that the PFS feature vector covers a different number of features for each dataset. Furthermore, none of the model's hyper-parameters, e.g., hidden layer dimensions or activation functions, were optimized. This would incur further gains in accuracy.

DATA AUGMENTATION Image data for machine learning applications is often augmented to increase the number of available training samples. In this case, images are rotated, transformed, stretched or cropped and saved as separate samples [76]. More data generally leads to higher accuracies and more robust models. In the interest of improving model performance for predictive process monitoring, augmentation methods should be explored with which to produce more traces from existing logs.

LISTING OF BEST EVENT ATTRIBUTES The predictive power that a feature possesses can be measured with different methods. If, e.g., a feature by itself exactly indicates what the next activity is going to be, its predictive power is very high. During the analysis of a log for prediction, it is essential to isolate features with high predictive power and discard those with low predictive power. Most event logs are saved in the XES format, which guarantees the type of information that is stored in an attribute. In the interest of speeding up feature selection in the future, we believe that it is worthwhile to investigate the predictive power across XES standard attributes for a large number of logs. From the findings, a list could be made of those attributes which are most likely to possess high predictive power and incur accuracy gains.

WORD EMBEDDINGS Word embeddings for NLP applications are trained on huge text corpora like Wikipedia and help map dictionary-encoded words to a large feature space. In Section 2.4, we hinted at the fact that the weights for word embeddings can be imported from pre-trained models to avoid training on a similarly large dataset and save time. This principle could be transferred to predictive process monitoring by providing pre-trained embeddings for process categories. As evidenced in Section 7.4, this is tied to larger datasets. It is worth exploring how well activity names are similar across processes, to gauge the reusability of a word embedding produced on one process for another.

PROCESS OPTIMIZATION We have seen that the accuracy of a model can be influenced by the variability of the process in a specific stage. For stages in a process where there are not many activities or any decisions, prediction accuracy tends to be high. Assuming that all data on which decisions are usually based are provided to the model, we believe that accuracy drops can indicate optimization potential in the process. If the model can not understand the connections in the data, it could mean that the control flow decisions are overly complicated or do not follow a standard. Whether this is the case, and what process fallacies can be detected based on accuracy fluctuation should be investigated.

APPENDIX

A

CORRELATION HEATMAPS

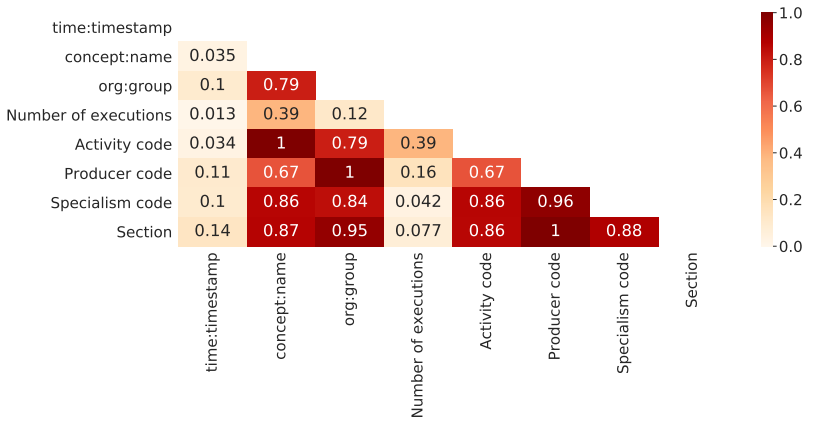


Figure A.48: Heatmap of correlating values within the BPIC11 dataset, obtained using pair-wise application of Cramér's V

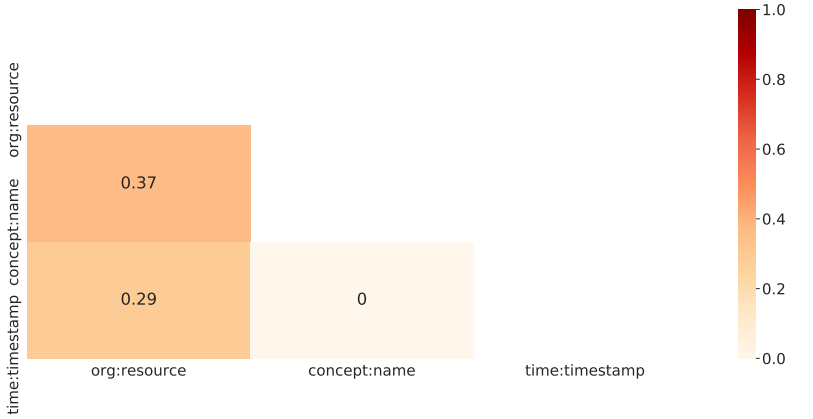


Figure A.49: Heatmap of correlating values within the BPIC12 dataset, obtained using pair-wise application of Cramér's V

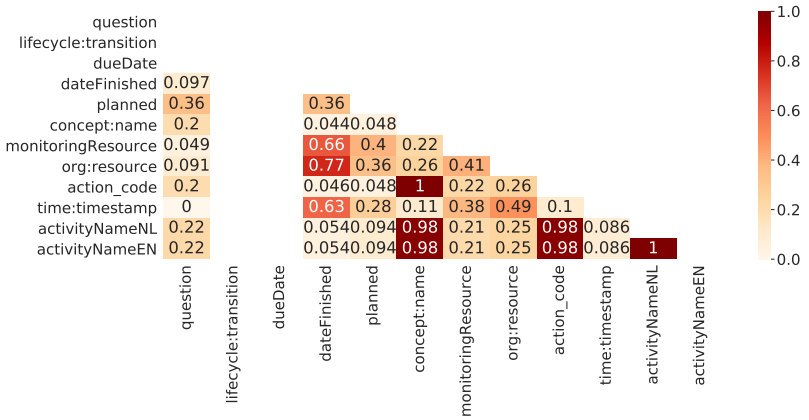


Figure A.50: Heatmap of correlating values within the BPIC15-1 dataset, obtained using pair-wise application of Cramér's V

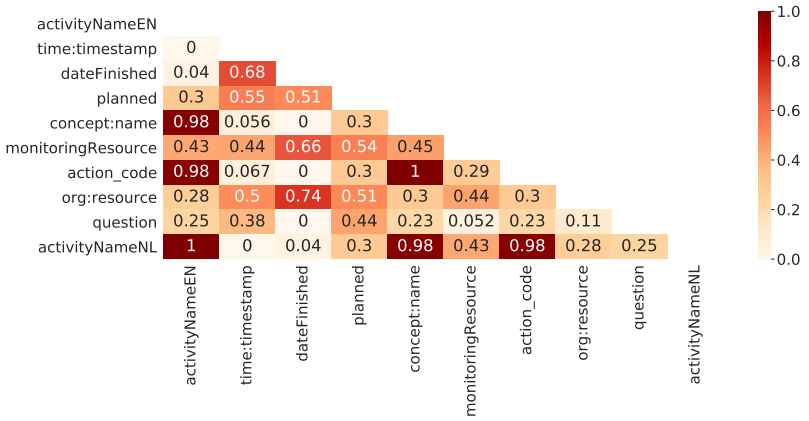


Figure A.51: Heatmap of correlating values within the BPIC15-2 dataset, obtained using pair-wise application of Cramér's V

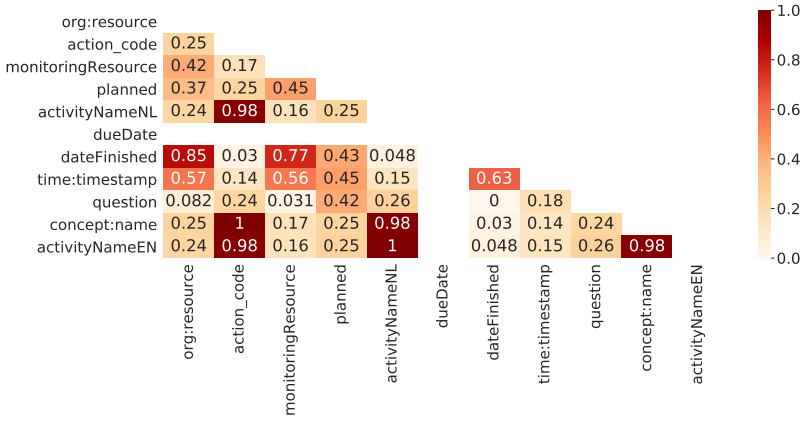


Figure A.52: Heatmap of correlating values within the BPIC15-3 dataset, obtained using pair-wise application of Cramér's V

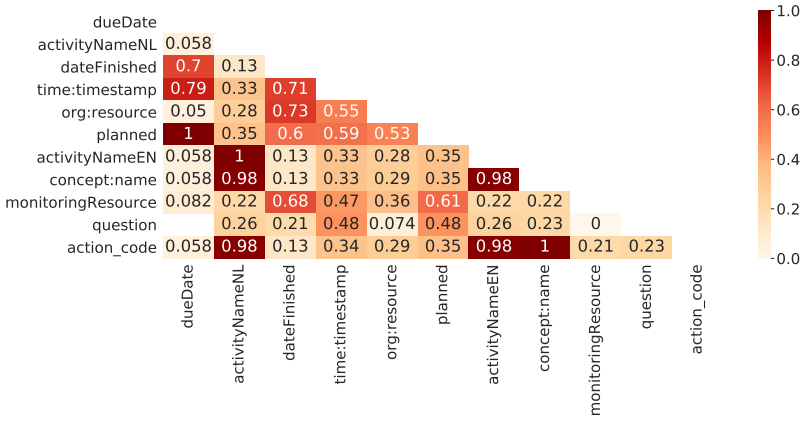


Figure A.53: Heatmap of correlating values within the BPIC15-4 dataset, obtained using pair-wise application of Cramér's V

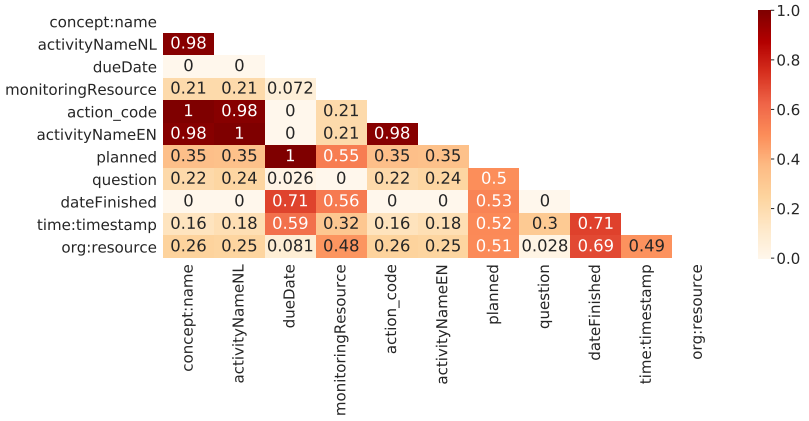


Figure A.54: Heatmap of correlating values within the BPIC15-5 dataset, obtained using pair-wise application of Cramér's V

B

PREDICTION STABILITY

HELPDESK

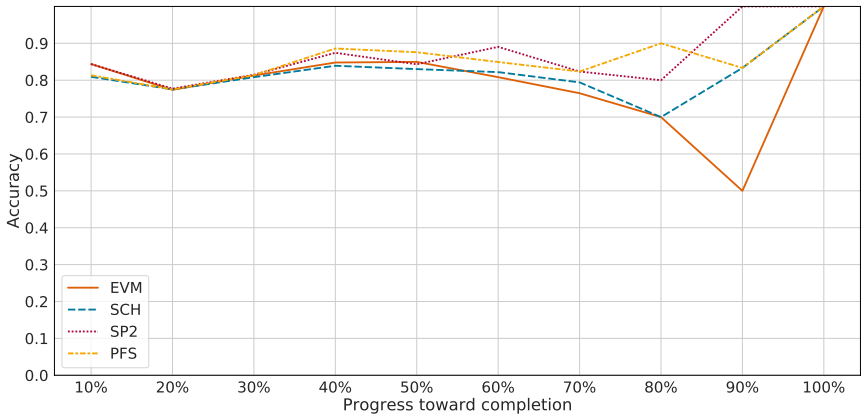


Figure B.55: Model stability, grouping strategy, HelpDesk

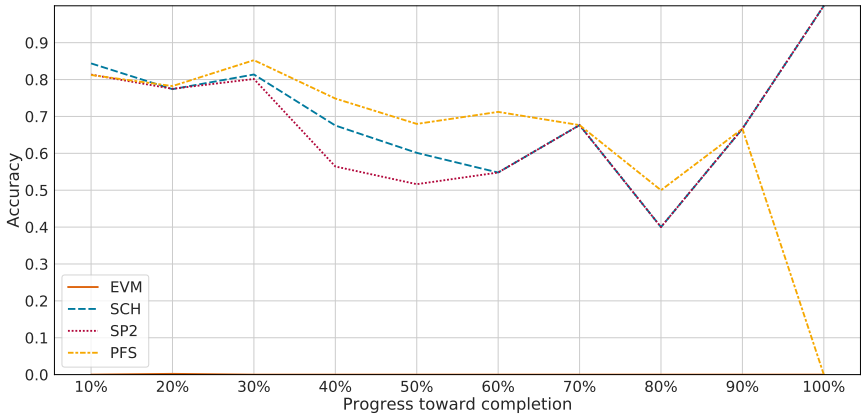


Figure B.56: Model stability, padding strategy, HelpDesk

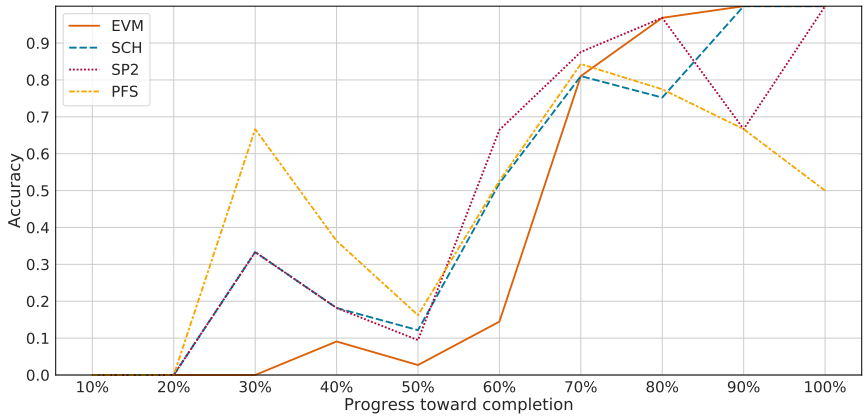


Figure B.57: Model stability, windowing strategy, HelpDesk

BPIC12

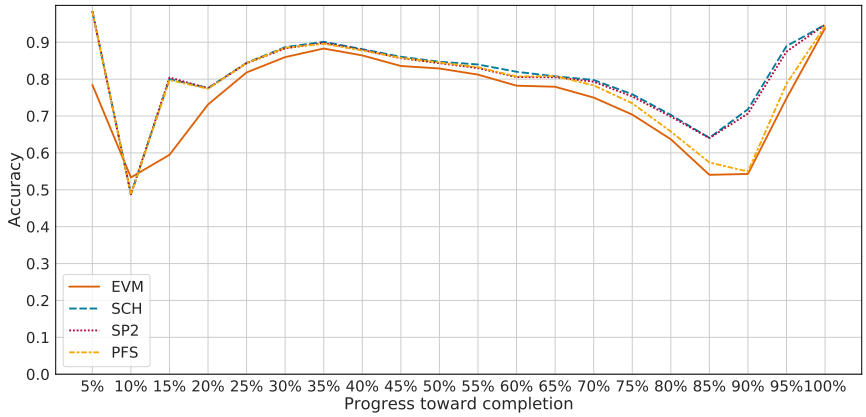


Figure B.58: Model stability, grouping strategy, BPIC12

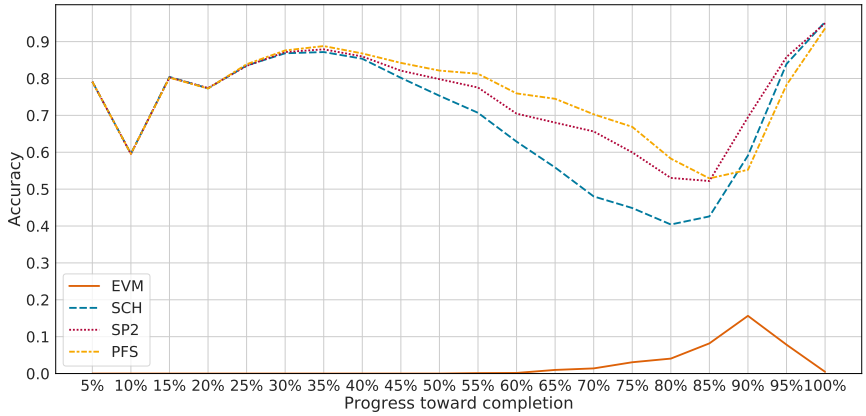


Figure B.59: Model stability, padding strategy, BPIC12

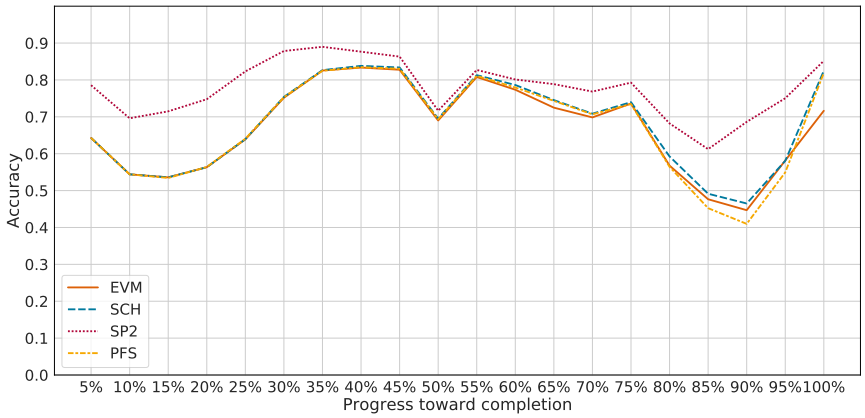


Figure B.60: Model stability, windowing strategy, BPIC12

BPIC15-1

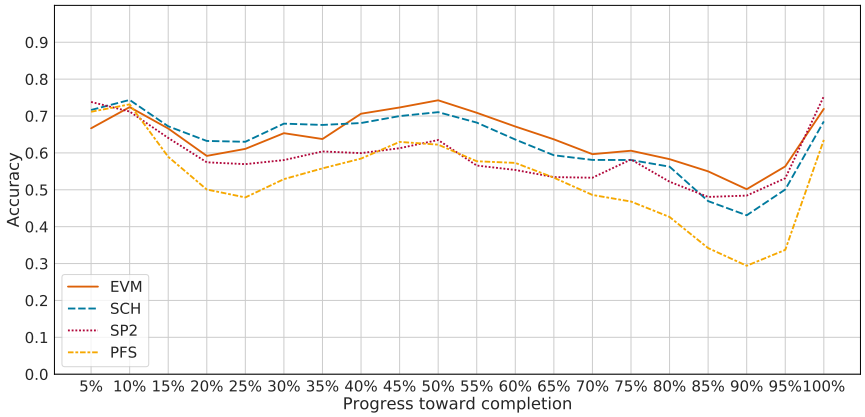


Figure B.61: Model stability, individual strategy, BPIC15-1

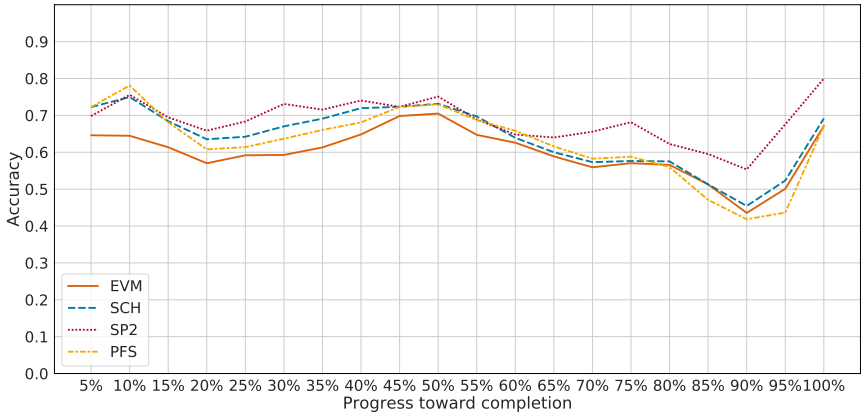


Figure B.62: Model stability, grouping strategy, BPIC15-1

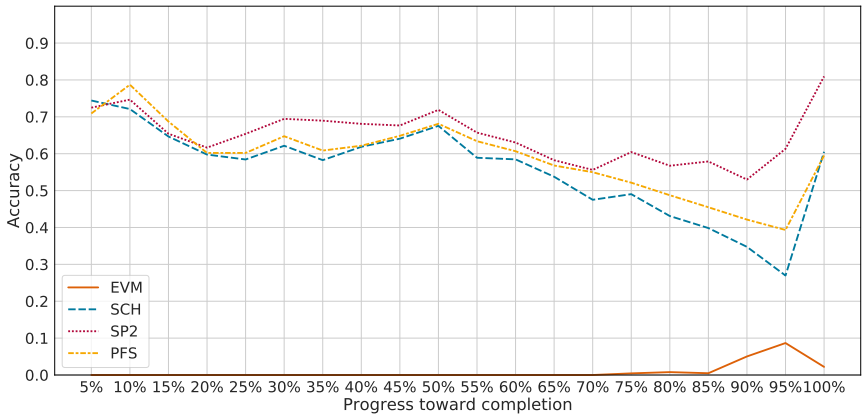


Figure B.63: Model stability, padding strategy, BPIC15-1

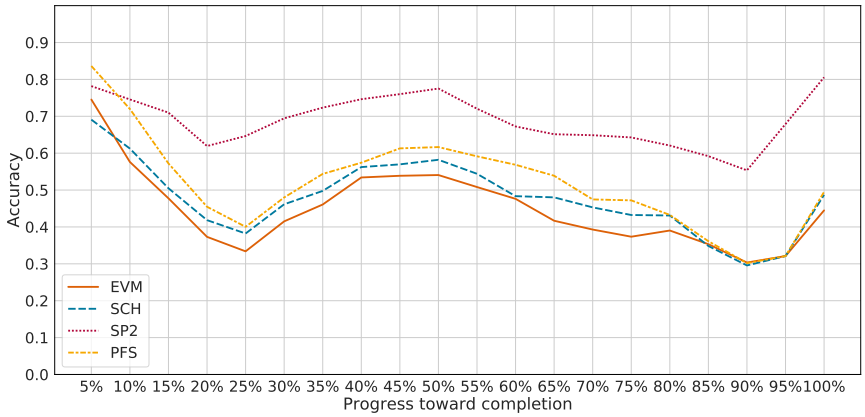


Figure B.64: Model stability, windowing strategy, BPIC15-1

BPIC15-2

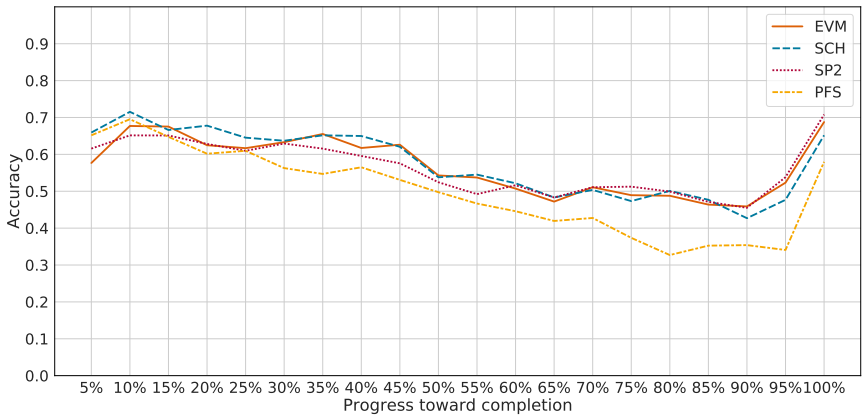


Figure B.65: Model stability, individual strategy, BPIC15-2

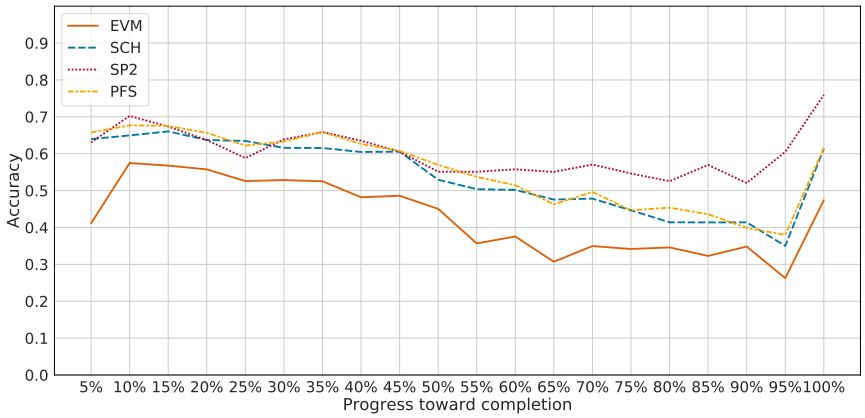


Figure B.66: Model stability, grouping strategy, BPIC15-2

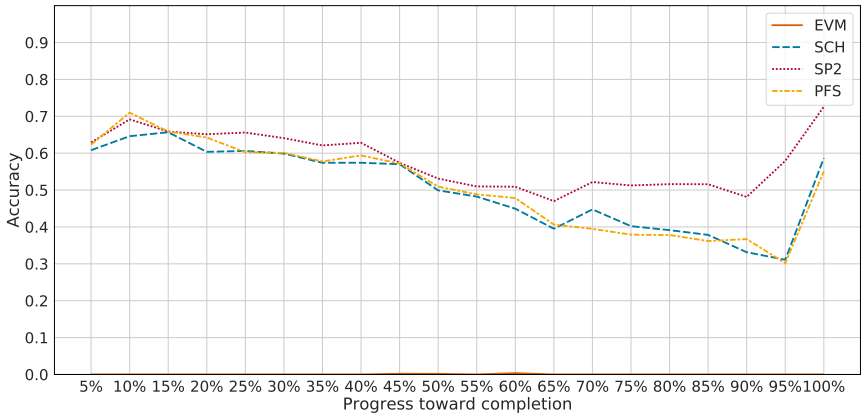


Figure B.67: Model stability, padding strategy, BPIC15-2

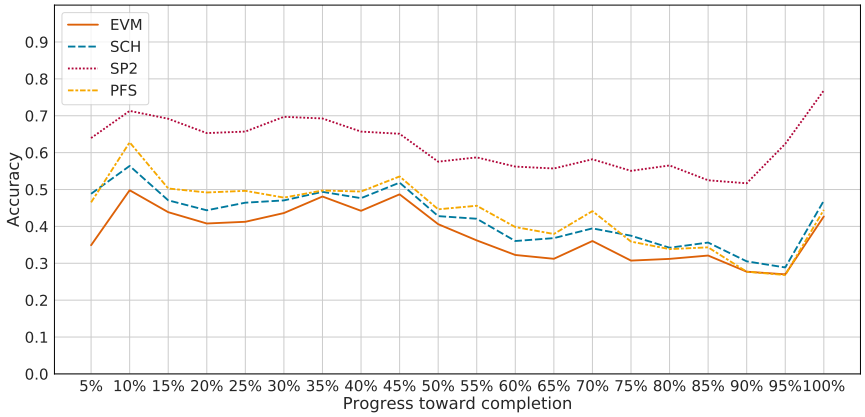


Figure B.68: Model stability, windowing strategy, BPIC15-2

BPIC15-3

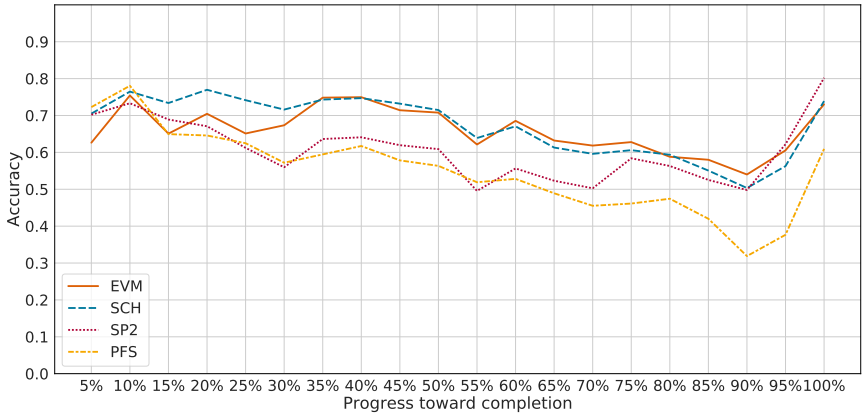


Figure B.69: Model stability, individual strategy, BPIC15-3

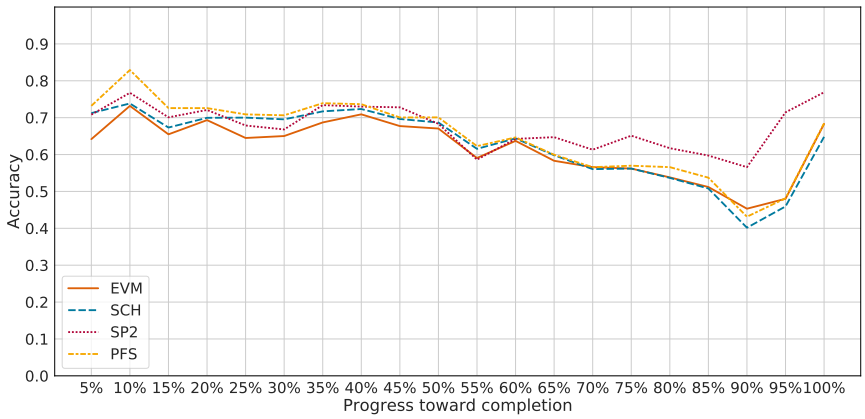


Figure B.70: Model stability, grouping strategy, BPIC15-3

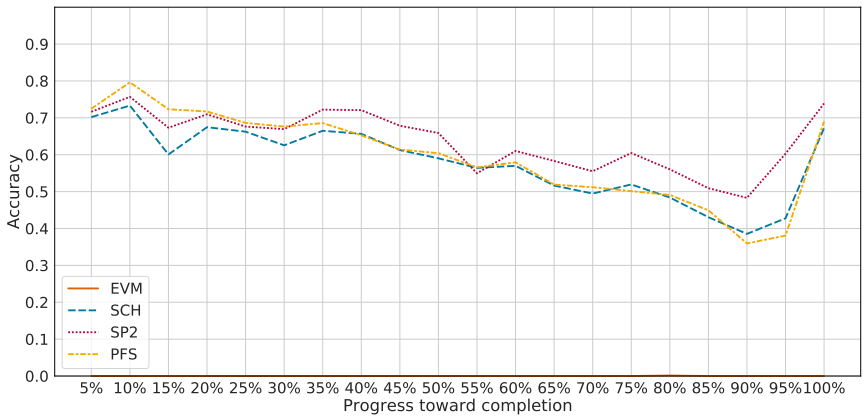


Figure B.71: Model stability, padding strategy, BPIC15-3

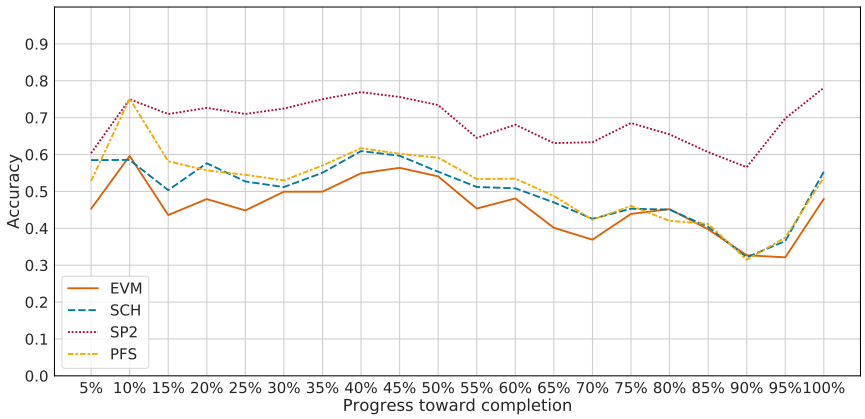


Figure B.72: Model stability, windowing strategy, BPIC15-3

BPIC15-4

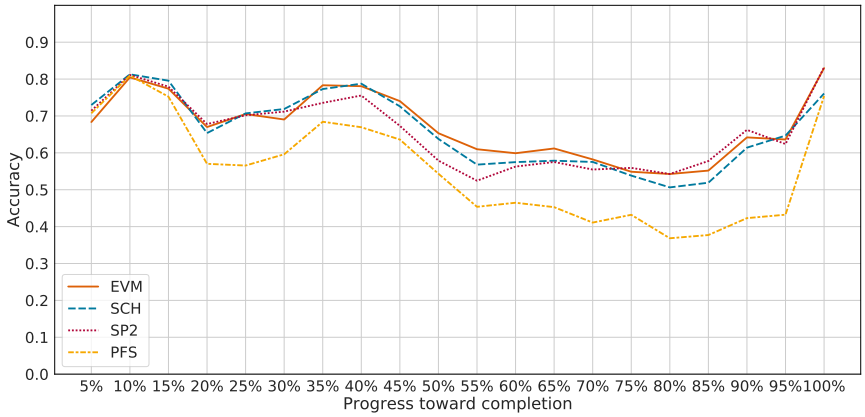


Figure B.73: Model stability, individual strategy, BPIC15-4

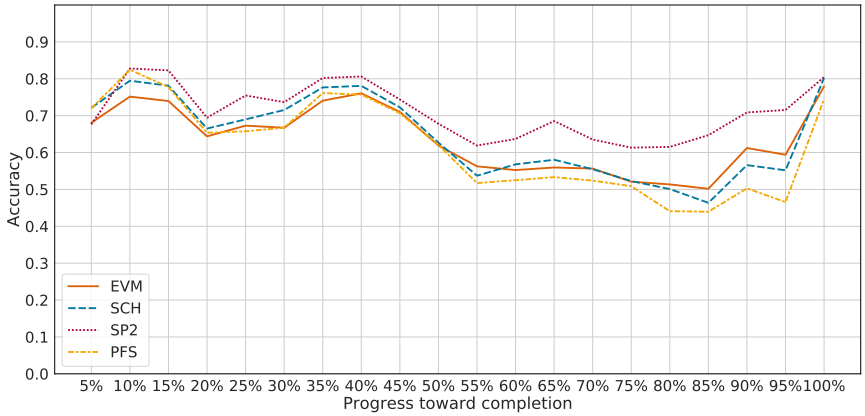


Figure B.74: Model stability, grouping strategy, BPIC15-4

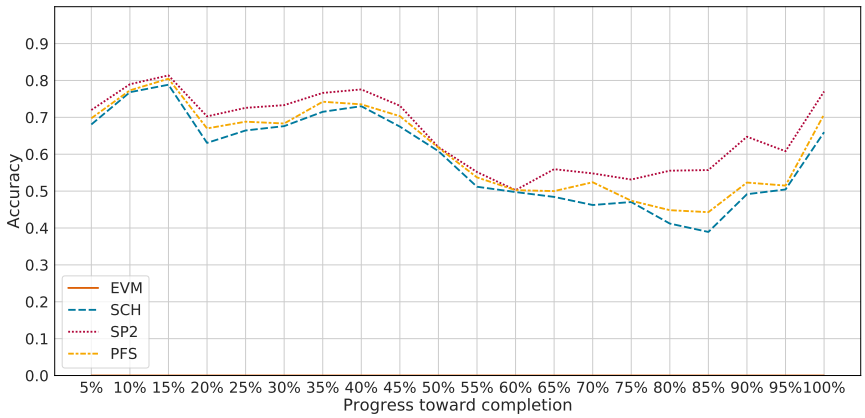


Figure B.75: Model stability, padding strategy, BPIC15-4

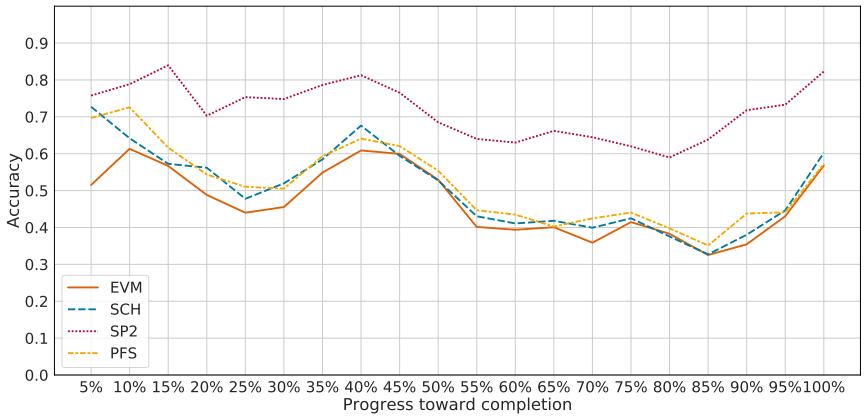


Figure B.76: Model stability, windowing strategy, BPIC15-4

BPIC15-5

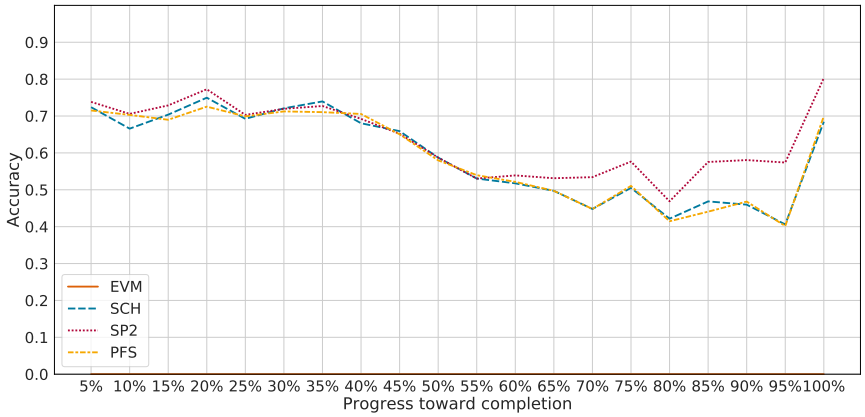


Figure B.77: Model stability, padding strategy, BPIC15-5

BPIC11

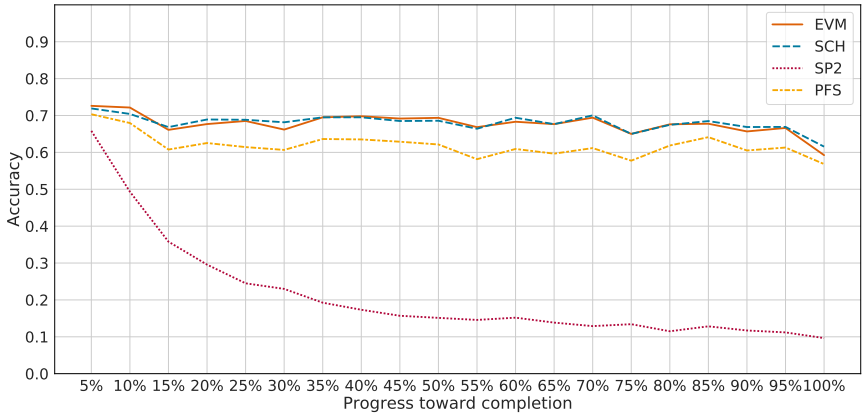


Figure B.78: Model stability, individual strategy, BPIC11

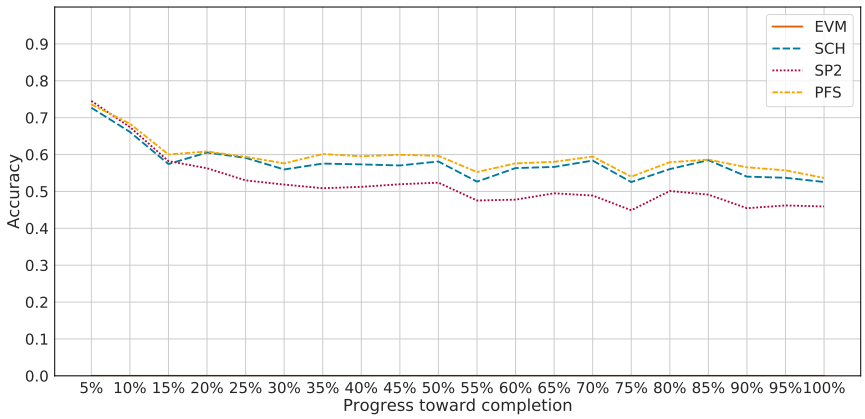


Figure B.79: Model stability, padding strategy, BPIC11

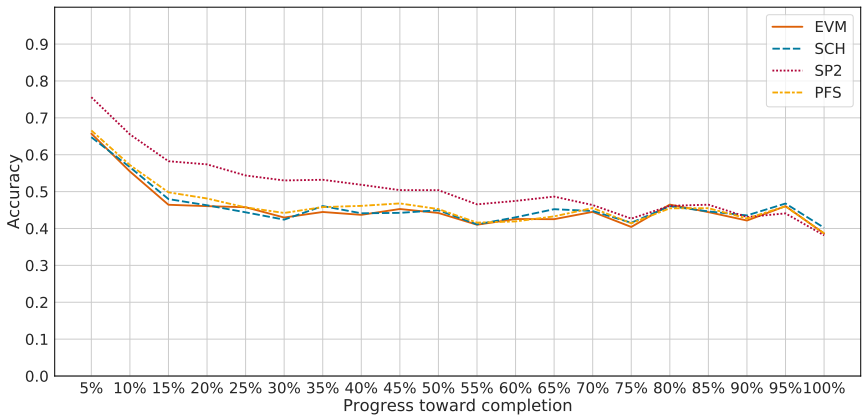


Figure B.80: Model stability, windowing strategy, BPIC11

C

ACCURACY AND LOSS CURVES

HELPDESK

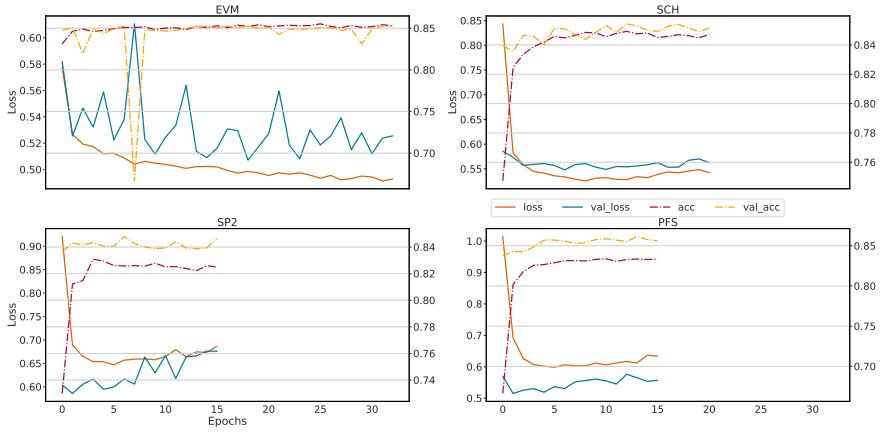


Figure C.81: Accuracy vs. loss, individual strategy, HelpDesk

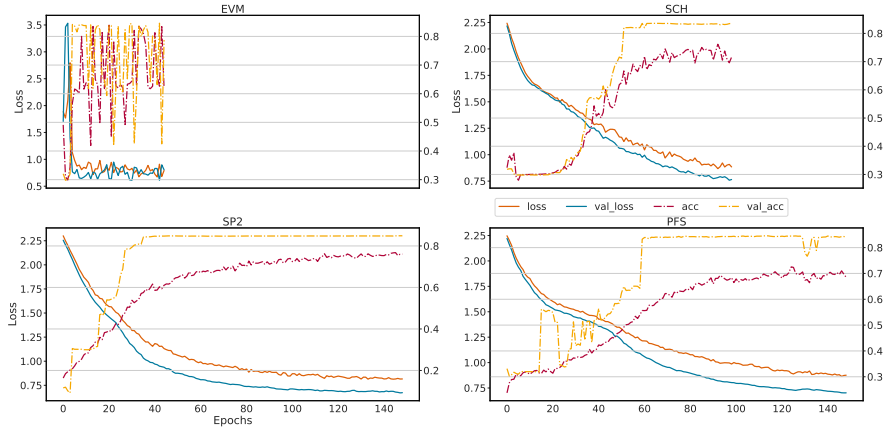


Figure C.82: Accuracy vs. loss, grouping strategy, HelpDesk

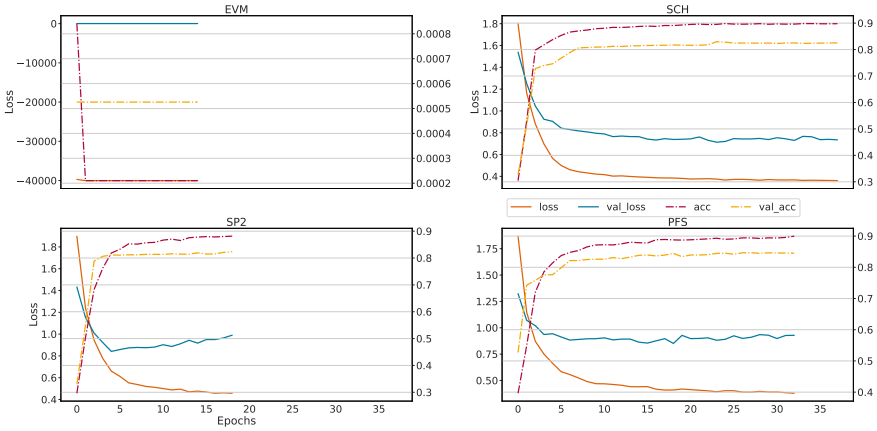


Figure C.83: Accuracy vs. loss, padding strategy, HelpDesk

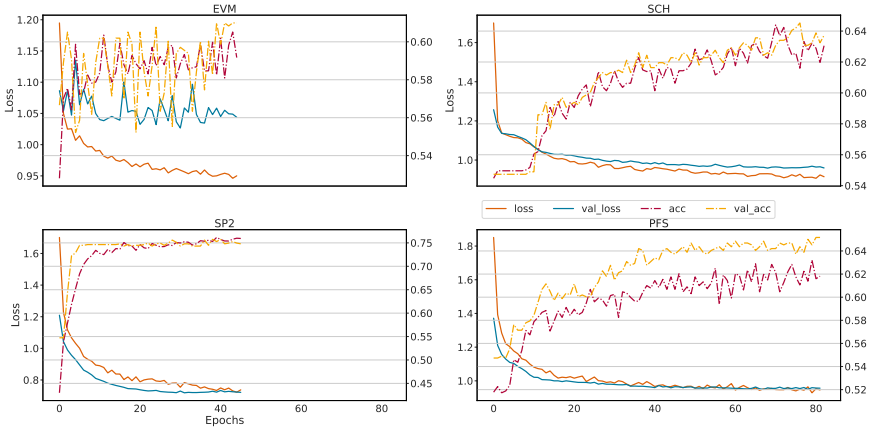


Figure C.84: Accuracy vs. loss, windowing strategy, HelpDesk

BPIC11

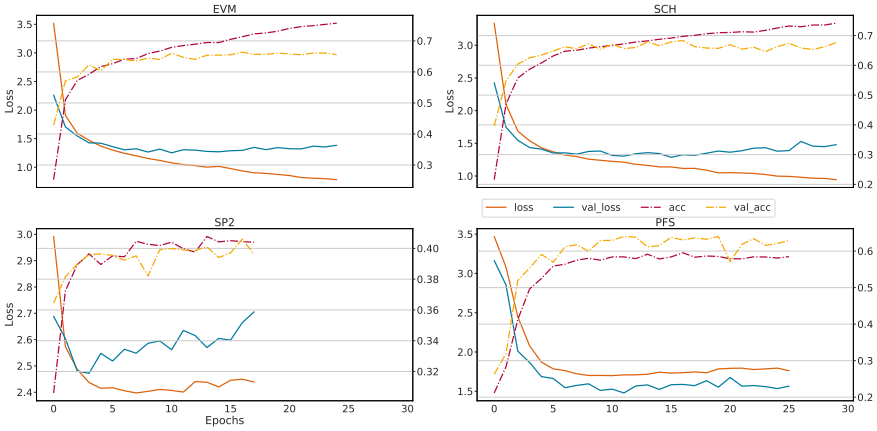


Figure C.85: Accuracy vs. loss, individual strategy, BPIC11

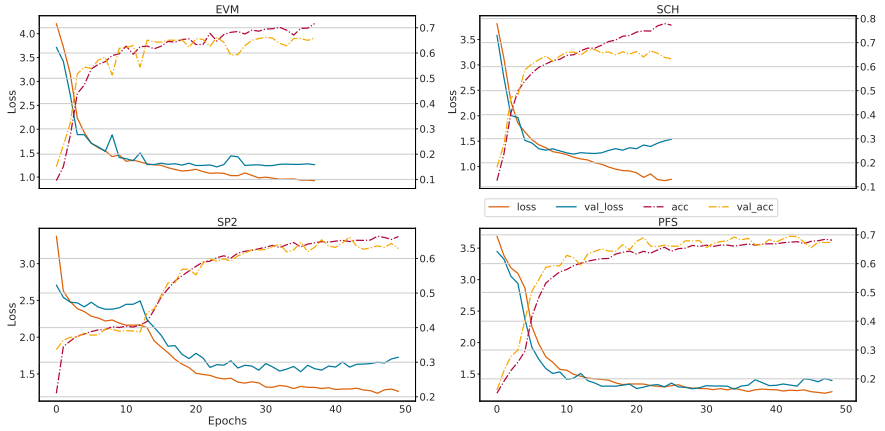


Figure C.86: Accuracy vs. loss, grouping strategy, BPIC11

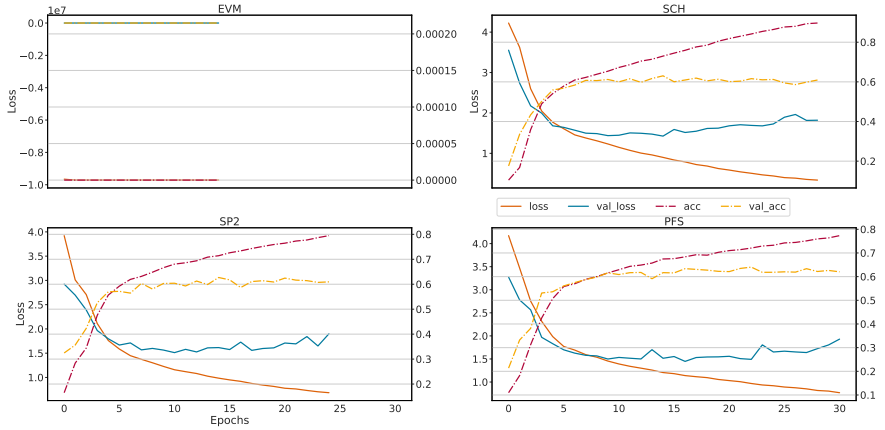


Figure C.87: Accuracy vs. loss, padding strategy, BPIC11

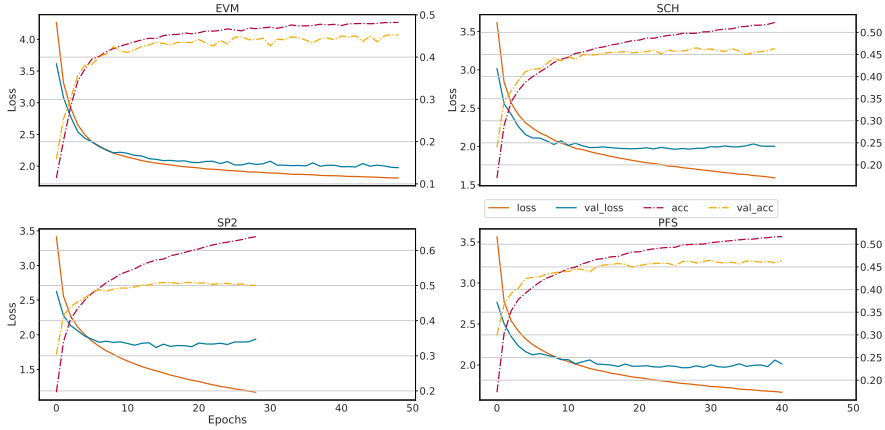


Figure C.88: Accuracy vs. loss, windowing strategy, BPIC11

BPIC12

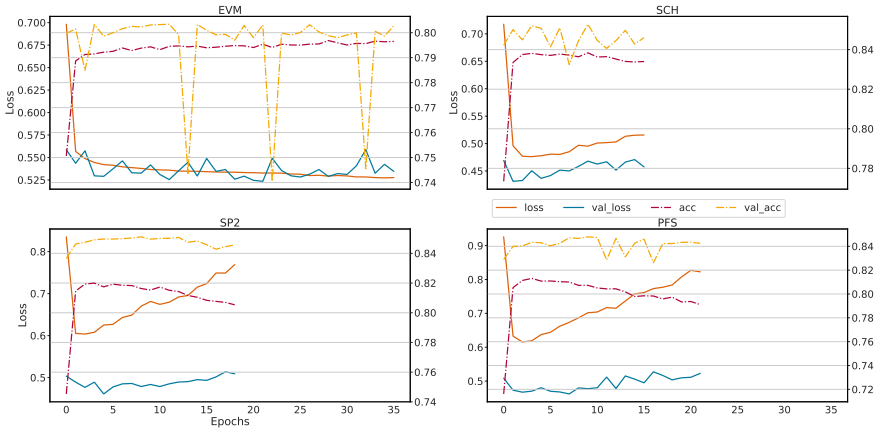


Figure C.89: Accuracy vs. loss, individual strategy, BPIC12

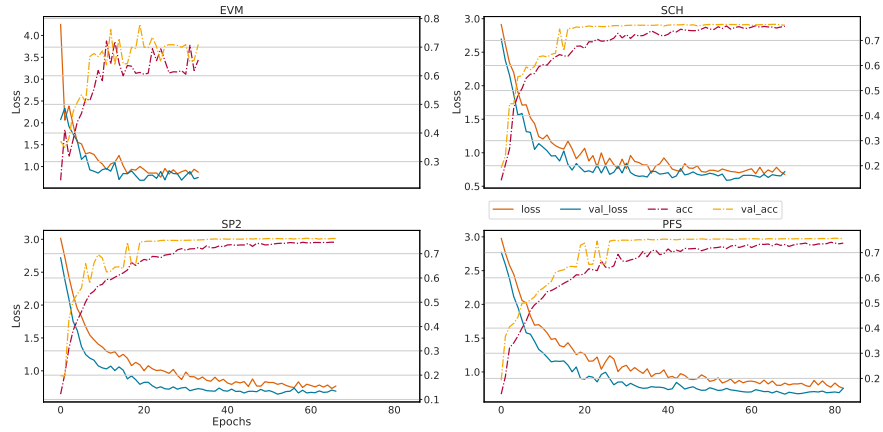


Figure C.90: Accuracy vs. loss, grouping strategy, BPIC12

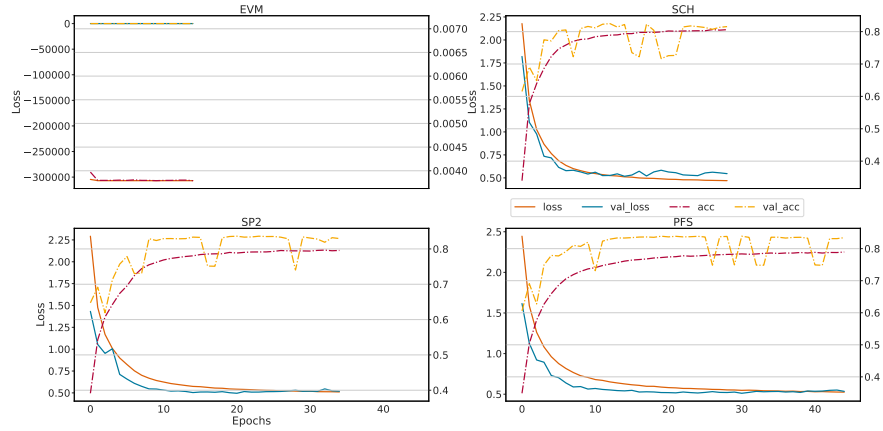


Figure C.91: Accuracy vs. loss, padding strategy, BPIC12

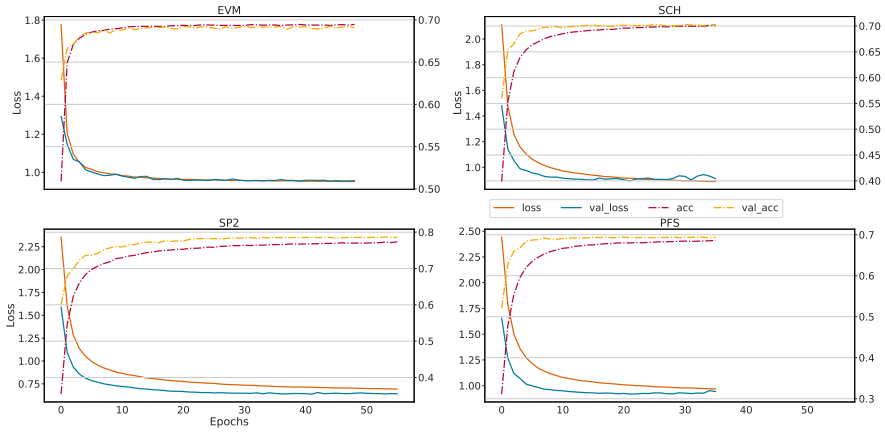


Figure C.92: Accuracy vs. loss, windowing strategy, BPIC12

BPIC15-1

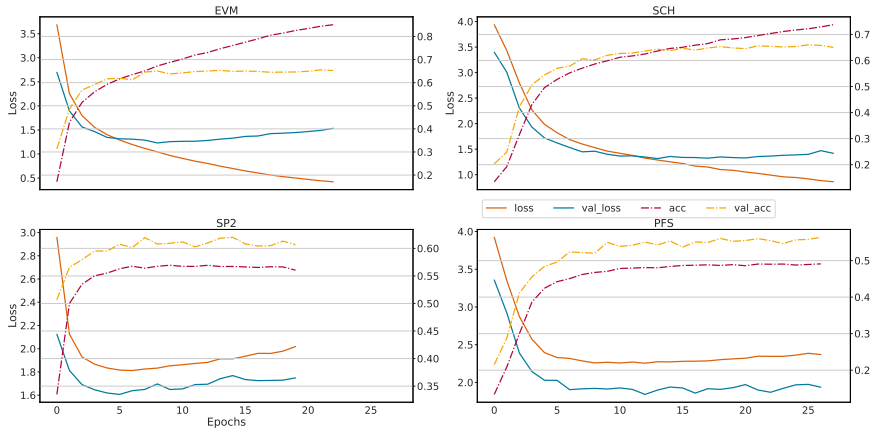


Figure C.93: Accuracy vs. loss, individual strategy, BPIC15-1

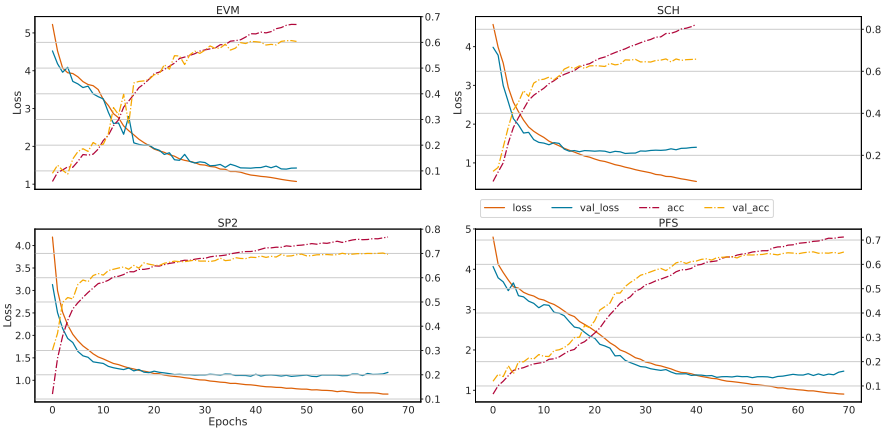


Figure C.94: Accuracy vs. loss, grouping strategy, BPIC15-1

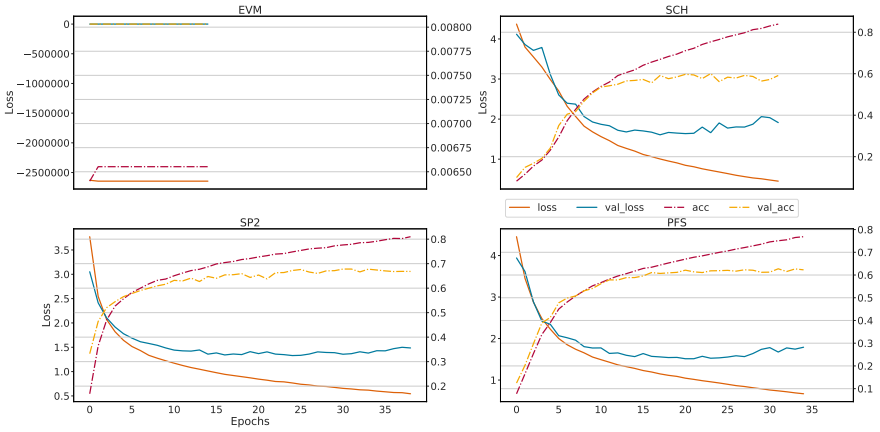


Figure C.95: Accuracy vs. loss, padding strategy, BPIC15-1

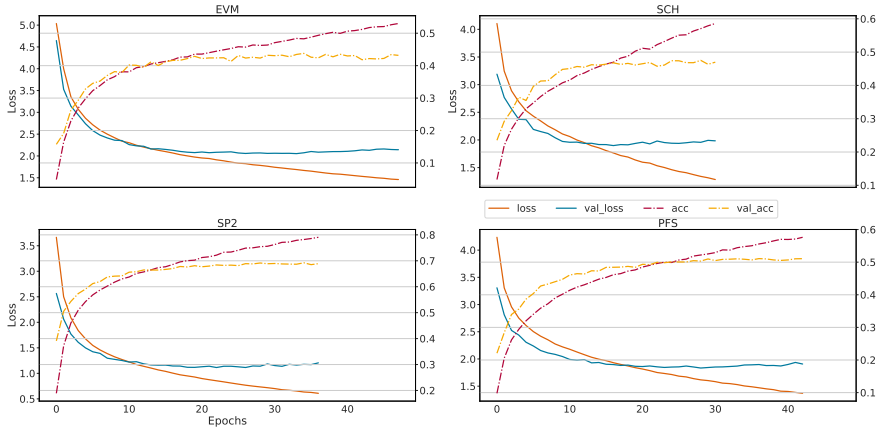


Figure C.96: Accuracy vs. loss, windowing strategy,BPIC15-1

BPIC15-2

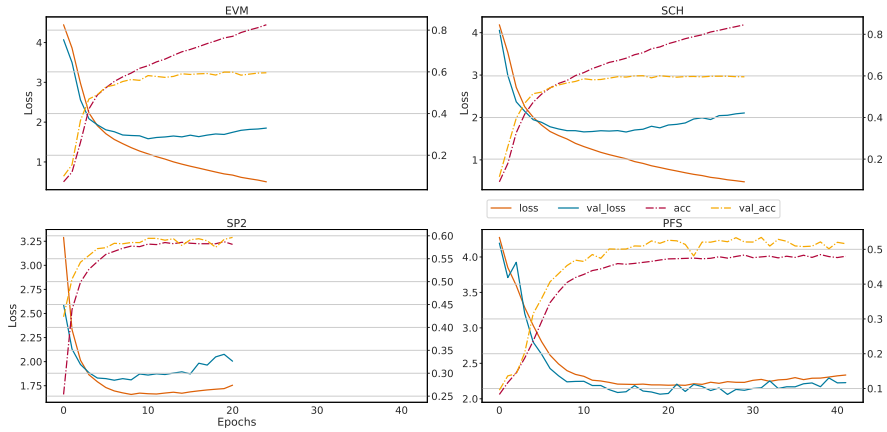


Figure C.97: Accuracy vs. loss, individual strategy, BPIC15-2

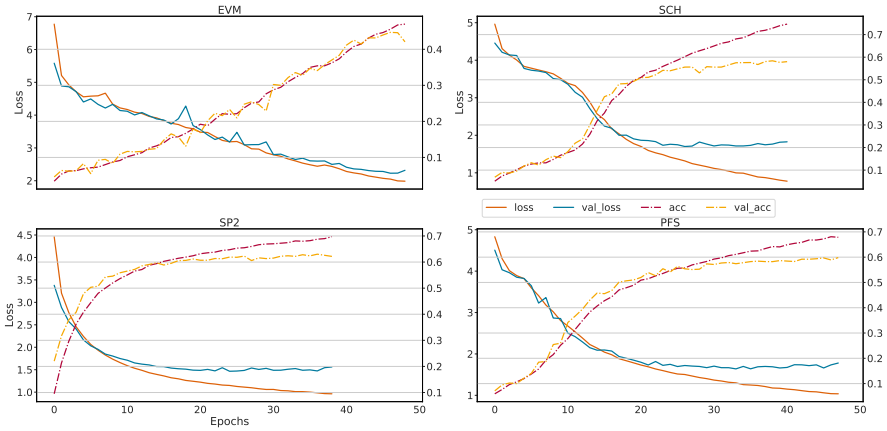


Figure C.98: Accuracy vs. loss, grouping strategy, BPIC15-2

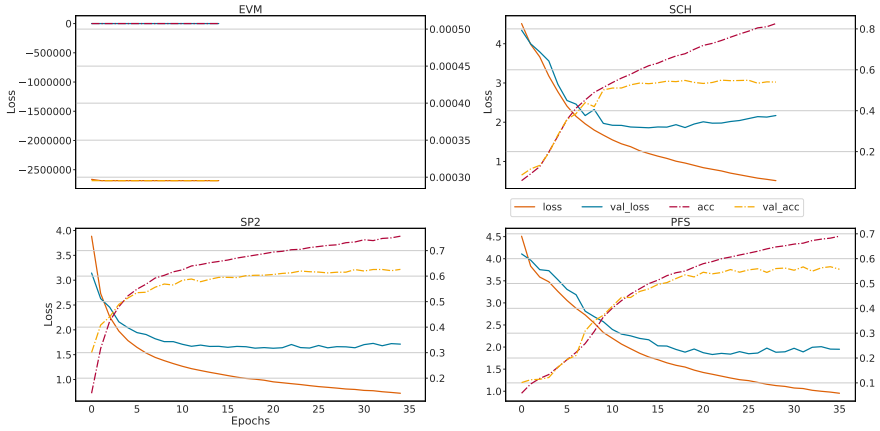


Figure C.99: Accuracy vs. loss, padding strategy, BPIC15-2

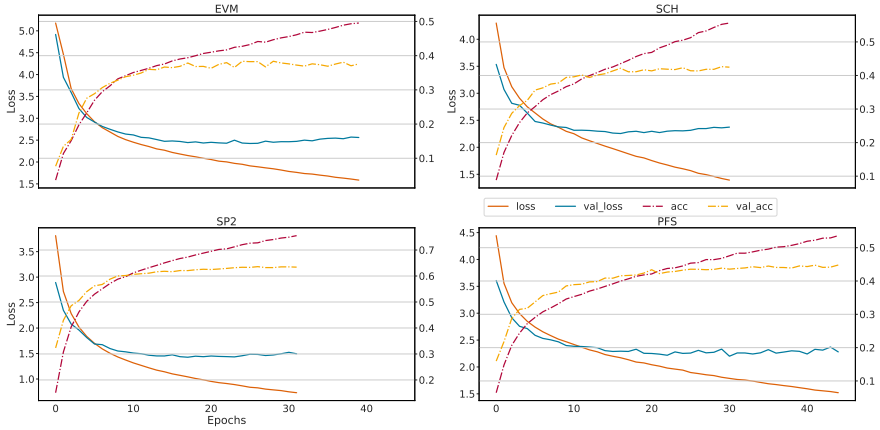


Figure C.100: Accuracy vs. loss, windowing strategy,BPIC15-2

BPIC15-3

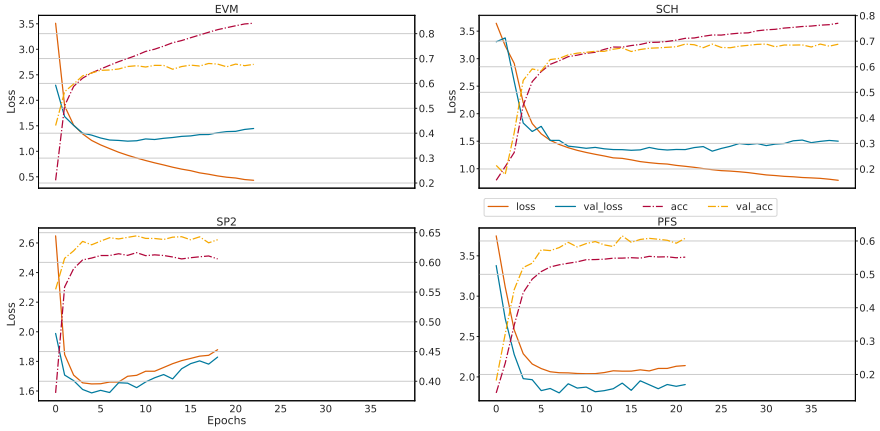


Figure C.101: Accuracy vs. loss, individual strategy, BPIC15-3

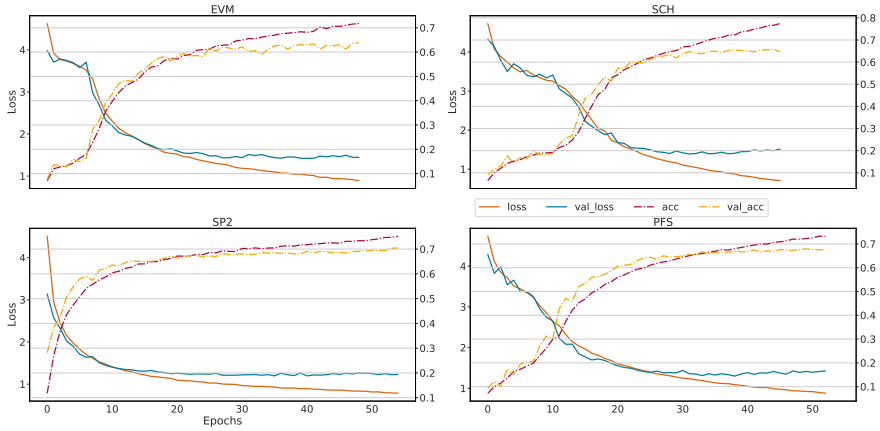


Figure C.102: Accuracy vs. loss, grouping strategy, BPIC15-3

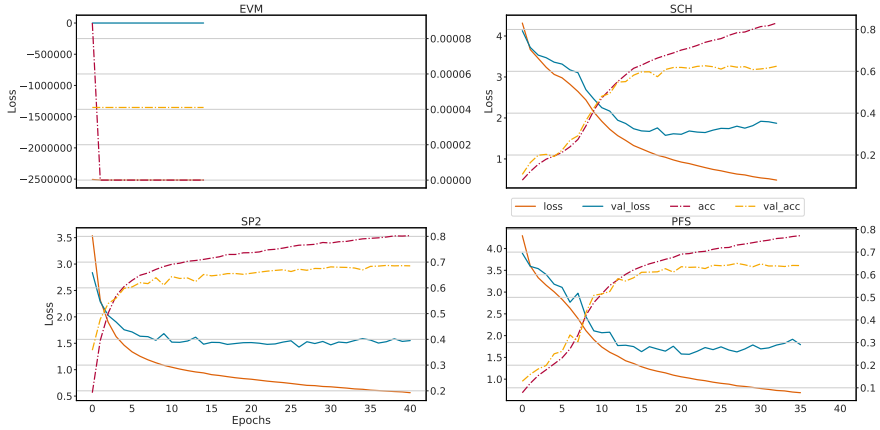


Figure C.103: Accuracy vs. loss, padding strategy, BPIC15-3

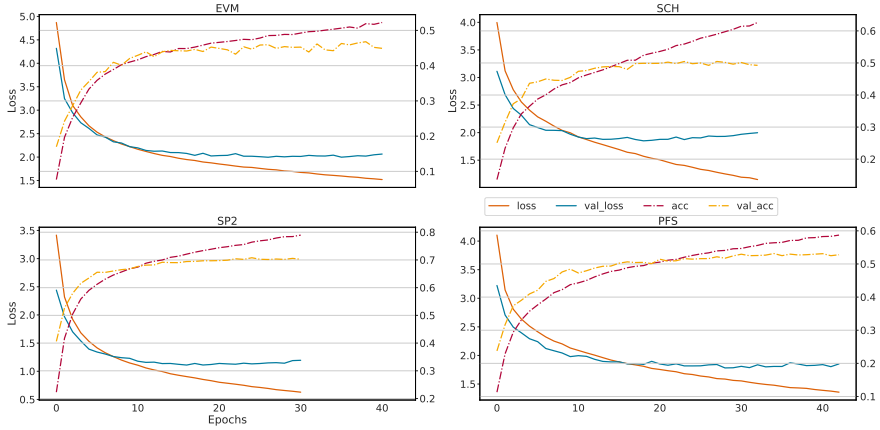


Figure C.104: Accuracy vs. loss, windowing strategy,BPIC15-3

BPIC15-4

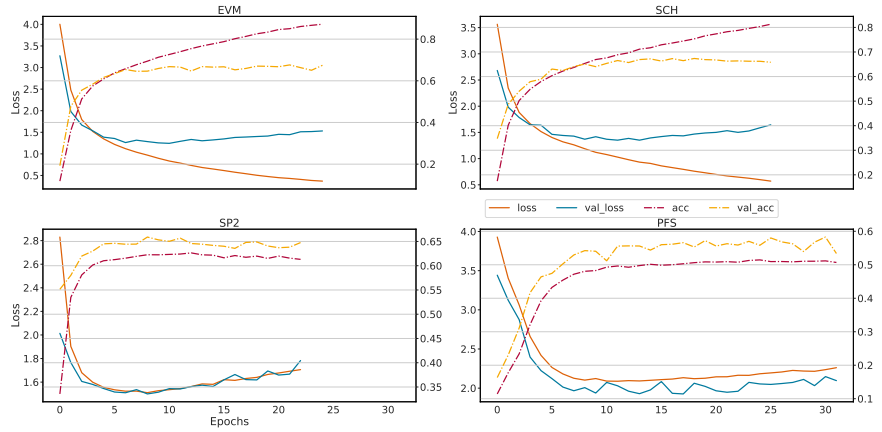


Figure C.105: Accuracy vs. loss, individual strategy, BPIC15-4

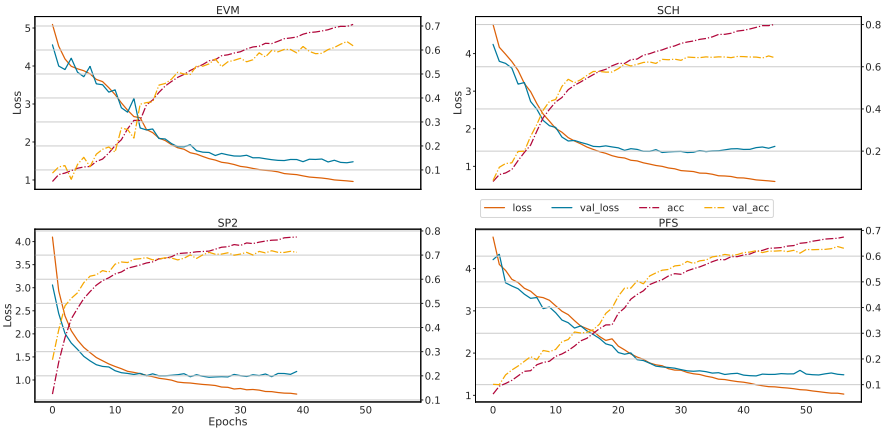


Figure C.106: Accuracy vs. loss, grouping strategy, BPIC15-4

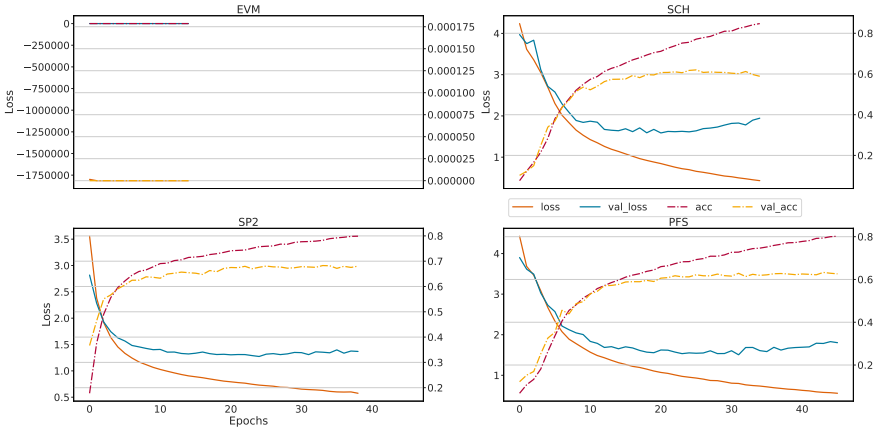


Figure C.107: Accuracy vs. loss, padding strategy, BPIC15-4

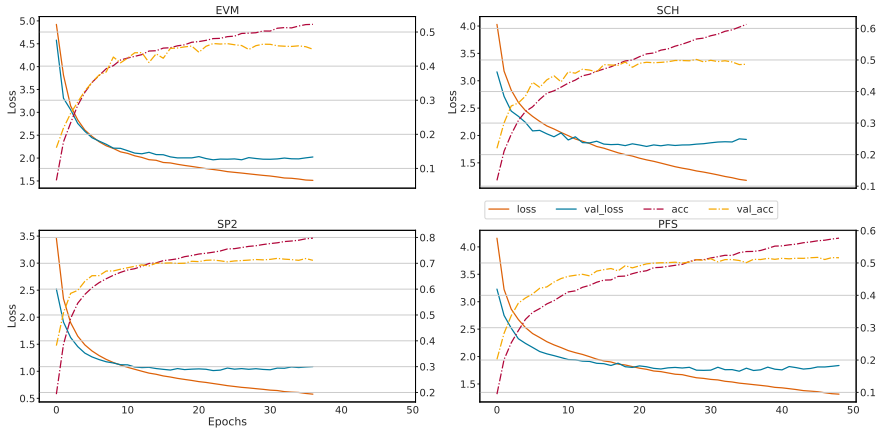


Figure C.108: Accuracy vs. loss, windowing strategy,BPIC15-4

BPIC15-5

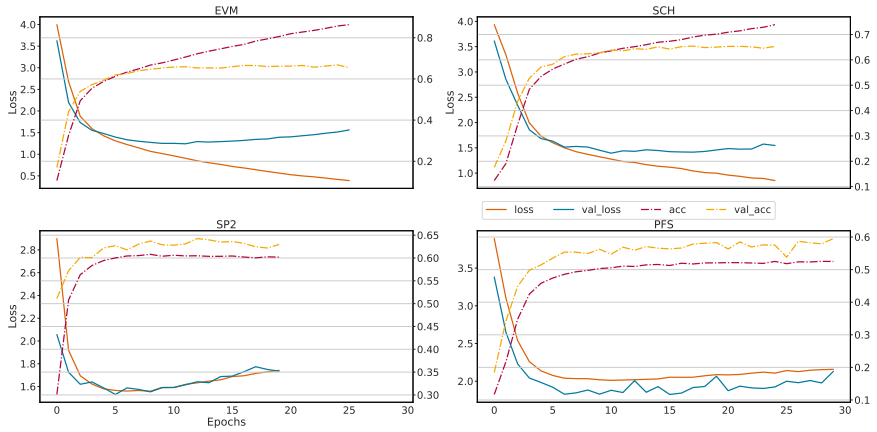


Figure C.109: Accuracy vs. loss, individual strategy, BPIC15-5

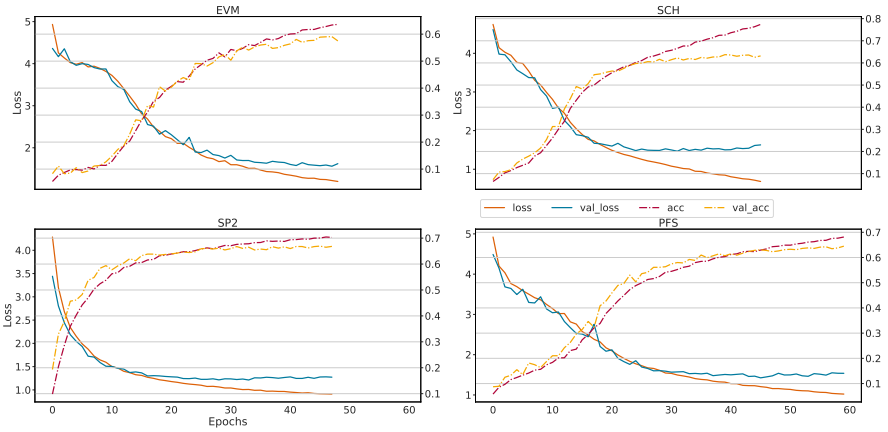


Figure C.110: Accuracy vs. loss, grouping strategy, BPIC15-5

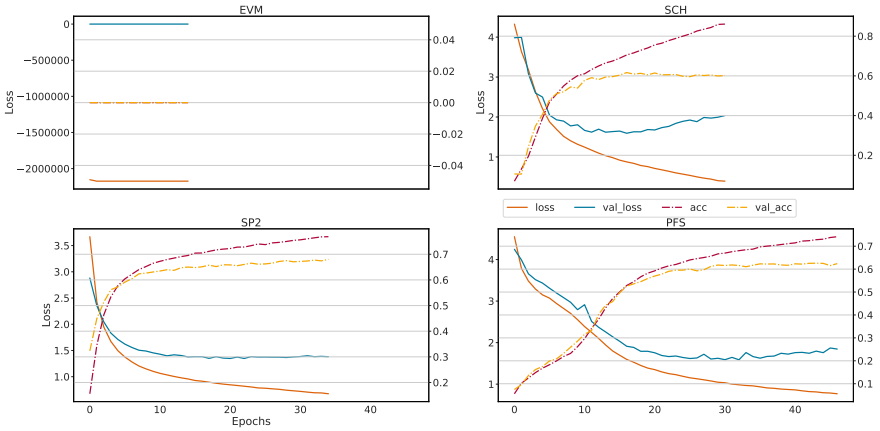


Figure C.111: Accuracy vs. loss, padding strategy, BPIC15-5

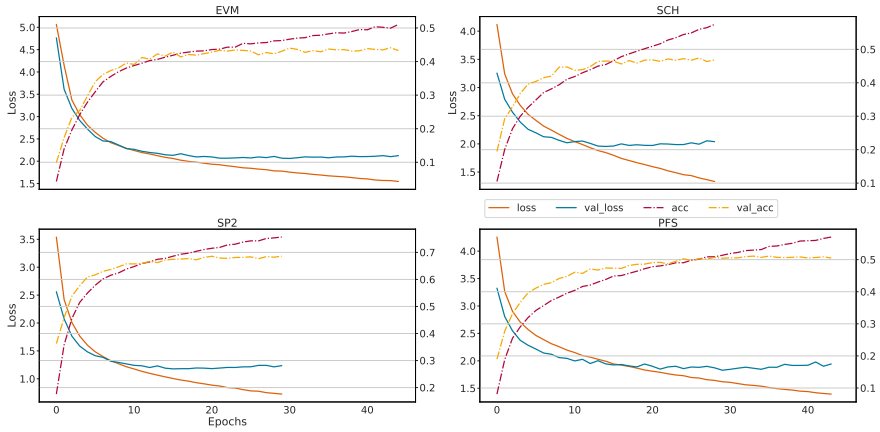


Figure C.112: Accuracy vs. loss, windowing strategy,BPIC15-5

BIBLIOGRAPHY

[3] Wil M. P. van der Aalst. *Process Mining: Data Science in Action*. 2nd ed. Heidelberg: Springer, 2016. ISBN: 978-3-662-49850-7. DOI: 10.1007/978-3-662-49851-4.

[4] A Adriansyah and JCA M Buijs. "Mining process performance from event logs: The bpi challenge 2012." In: *Case Study. BPM Center Report BPM-12-15, BPMcenter. org*. Citeseer. 2012.

[6] Thomas Allweyer. *BPMN 2.0*. BoD, 2010.

[9] *BPI Challenge 2011 Dataset*. https://data.4tu.nl/repository/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54. Accessed: 2018-08-20.

[10] *BPI Challenge 2012 Dataset*. https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f. Accessed: 2018-08-15.

[11] *BPI Challenge 2013 Dataset*. https://data.4tu.nl/repository/uuid:7e326e7e-8b93-4701-8860-71213edf0fbe. Accessed: 2018-12-22.

[12] *BPI Challenge 2015 Dataset*. https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1. Accessed: 2018-12-20.

[13] *BPI Challenge 2017 Dataset*. https://data.4tu.nl/repository/uuid:7e326e7e-8b93-4701-8860-71213edf0fbe. Accessed: 2018-08-16.

[14] Wicher Bergsma. "A bias-correction for Cramér's V and Tschuprow's T." In: *Journal of the Korean Statistical Society* 42.3 (2013), pp. 323–328.

[15] Kristof Böhmer and Stefanie Rinderle-Ma. "Probability based Heuristic for Predictive Business Process Monitoring." In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer. 2018, pp. 78–96.

[16] RP Jagadeesh Chandra Bose and WMP van der Aalst. "Analysis of Patient Treatment Procedures: The BPI Challenge Case Study." In: (2011).

[19] Chiara Di Francescomarino, Chiara Ghidini, Fabrizio Maria Maggi, Giulio Petrucci, and Anton Yeshchenko. "An eye into the future: leveraging a-priori knowledge in predictive business process monitoring." In: *International Conference on Business Process Management*. Springer. 2017, pp. 252–268.

[20] Peter F. Drucker. "Knowledge-Worker Productivity: The Biggest Challenge." In: *California Management Review* 41.2 (1999), pp. 79–94. DOI: 10.2307/41165987. eprint: https://doi.org/10.2307/41165987. URL: https://doi.org/10.2307/41165987.

[22] *Environmental permit application process ("wabo"), coselog project - municipality 4*. https://doi.org/10.4121/uuid:e8c3a53d-5301-4afb-9bcd-38e74171ca32. Accessed: 2018-12-22.

[23] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. "A Deep Learning Approach for Predicting Process Behaviour at Runtime." In: *Business Process Management Workshops*. 2016.

[24] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "From data mining to knowledge discovery in databases." In: *AI magazine* 17.3 (1996), p. 37.

[25] Chiara Di Francescomarino, Marlon Dumas, Fabrizio Maria Maggi, and Irene Teinemaa. "Clustering-Based Predictive Process Monitoring." In: *CoRR* abs/1506.01428 (2015).

[26] Chiara Di Francescomarino, Chiara Ghidini, Fabrizio Maria Maggi, and Fredrik Milani. "Predictive Process Monitoring Methods: Which One Suits Me Best?" In: *CoRR* (2018).

[27] Paul A Gagniuc. *Markov Chains: From Theory to Implementation and Experimentation*. John Wiley & Sons, 2017.

[28] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks." In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323.

[29] Yoav Goldberg and Omer Levy. "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method." In: *arXiv preprint arXiv:1402.3722* (2014).

[30] Klaus Greff, Rupesh K Srivastava, Jan Koutnik, Bas R Steunebrink, and Jürgen Schmidhuber. "LSTM: A search space odyssey." In: *IEEE transactions on neural networks and learning systems* 28.10 (2017), pp. 2222–2232.

[31] Christian W Günther and E Verbeek. "XES-standard definition (2014)." In: *BPMcenter. org* (2013).

[33] Ube van der Ham. "Benchmarking of five dutch municipalities with process mining techniques reveals opportunities for improvement." In: *Business Process Intelligence Challenge* 2015 (2015).

[34] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. "FreeSpan: frequent pattern-projected sequential pattern mining." In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2000, pp. 355–359.

[35] Matheus Hauder, Simon Pigat, and Florian Matthes. "Research Challenges in Adaptive Case Management: A Literature Review." In: *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations* (2014), pp. 98–107.

[36] Jeffrey Heinz and James Rogers. "Estimating Strictly Piecewise Distributions." In: *ACL*. 2010.

[37] *Helpdesk Dataset*. http://dx.doi.org/10.17632/39bp3vv62t.1. Accessed: 2018-12-22.

[38] Marcin Hewelt and Mathias Weske. "A Hybrid Approach for Flexible Case Modeling and Execution." In: *Business Process Management Forum - BPM Forum 2016, Rio de Janeiro, Brazil, September 18-22, 2016, Proceedings*. 2016, pp. 38–54. DOI: 10.1007/978-3-319-45468-9_3. URL: https://doi.org/10.1007/978-3-319-45468-9_3.

[39] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen." In: *Diploma, Technische Universität München* 91.1 (1991).

[40] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory." In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[42] Sebastian Huber, Marian Fietta, and Sebastian Hof. "Next Step Recommendation and Prediction Based on Process Mining in Adaptive Case Management." In: *Proceedings of the 7th International Conference on Subject-Oriented Business Process Management*. S-BPM ONE '15. Kiel, Germany: ACM, 2015, 3:1–3:9. ISBN: 978-1-4503-3312-2. DOI: 10.1145/2723839.2723842. URL: http://doi.acm.org/10.1145/2723839.2723842.

[44] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures." In: *International Conference on Machine Learning*. 2015, pp. 2342–2350.

[47] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On large-batch training for deep learning: Generalization gap and sharp minima." In: *ICLR*. 2017.

[48] Christopher Klinkmüller, Nick RTP van Beest, and Ingo Weber. "Towards Reliable Predictive Process Monitoring." In: *International Conference on Advanced Information Systems Engineering*. Springer. 2018, pp. 163–181.

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[50] Max Kuhn and Kjell Johnson. *Applied predictive modeling*. Vol. 26. Springer, 2013.

[51] Stefan Lessmann. *Business Analytics & Data Science lecture*. Winter term 2015/2016.

[52] A. Metzger, P. Leitner, D. Ivanović, E. Schmieders, R. Franklin, M. Carro, S. Dustdar, and K. Pohl. "Comparing and Combining Predictive Business Process Monitoring Techniques." In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45.2 (2015), pp. 276–290. ISSN: 2168-2216. DOI: 10.1109/TSMC.2014.2347265.

[53] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[54] Theodore Panagacos. *The Ultimate Guide to Business Process Managment: Everything You Need to Know and how to Apply it to Your Organization*. Amazon, 2012.

[56] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth." In: *icccn*. IEEE. 2001, p. 0215.

[57] Hasso Plattner and Alexander Zeier. *In-memory data management: technology and applications*. Springer Science & Business Media, 2012.

[58] Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano de Leoni. "Data-aware remaining time prediction of business process instances." In: *2014 International Joint Conference on Neural Networks (IJCNN)* (2014), pp. 816–823.

[59] Alexandros Potamianos and Filippos Kokkinos. "Structural Attention Neural Networks for improved sentiment analysis." In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*. 2017, pp. 586–591.

[61] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should i trust you?: Explaining the predictions of any classifier." In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM. 2016, pp. 1135–1144.

[62] Stuart Russell, Peter Norvig, and Artificial Intelligence. "A modern approach." In: *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs* 25.27 (1995), pp. 79–80.

[63] Jürgen Schmidhuber. "Deep learning in neural networks: An overview." In: *Neural networks* 61 (2015), pp. 85–117.

[64] Stefan Schönig, Richard Jasinski, Lars Ackermann, and Stefan Jablonski. "Deep Learning Process Prediction with Discrete and Continuous Data Features." In: *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering*. s.l., 2018. URL: https://eref.uni-bayreuth.de/41777/.

[66] Chihiro Shibata and Jeffrey Heinz. "Predicting Sequential Data with LSTMs Augmented with Strictly 2-Piecewise Input Vectors." In: *ICGI*. 2016, pp. 137–142.

[67] Siva Sivaganesan. "Predictive Inference: An Introduction (Seymour Geisser)." In: *SIAM Review* 36.3 (1994), pp. 519–520.

[68] Ramakrishnan Srikant and Rakesh Agrawal. "Mining sequential patterns: Generalizations and performance improvements." In: *International Conference on Extending Database Technology*. Springer. 1996, pp. 1–17.

[69] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[70] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the inception architecture for computer vision." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[71] Niek Tax, Irene Teinemaa, and Sebastiaan J van Zelst. "An Interdisciplinary Comparison of Sequence Modeling Methods for Next-Element Prediction." In: *arXiv preprint arXiv:1811.00062* (2018).

[72] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. "Predictive Business Process Monitoring with LSTM Neural Networks." In: *CAiSE*. 2017.

[76] Martin Thoma. "Analysis and Optimization of Convolutional Neural Network Architectures." Masters's Thesis. Karlsruhe, Germany: Karlsruhe Institute of Technology, June 2017. URL: https://martin-thoma.com/msthesis/.

[79] Mathias Weske. "Business process management architectures." In: *Business Process Management*. Springer, 2012, pp. 333–371.

ONLINE SOURCES

[1] *A Note to Techniques in Convolutional Neural Networks and Their Influences II*. URL: http://yeephycho.github.io/2016/08/02/A-reminder-of-algorithms-in-Convolutional-Neural-Networks-and-their-influences-II/ (visited on 01/09/2019).

[2] *A python implementation of the XES standard that is based on the Java implementation OpenXes*. URL: https://github.com/opyenxes/OpyenXes (visited on 11/12/2018).

[5] *Ahogrammer | List of pretrained word embeddings*. URL: http://ahogrammer.com/2017/01/20/the-list-of-pretrained-word-embeddings/ (visited on 10/11/2018).

[7] *An applied introduction for LSTMs for text generation*. URL: https://medium.freecodecamp.org/applied-introduction-to-lstms-for-text-generation-380158b29fb3 (visited on 11/08/2018).

[8] *Anaconda - The Most Popular Python Data Science Platform*. URL: http://anaconda.com (visited on 11/12/2018).

[17] *Build and run Docker containers leveraging NVIDIA GPUs*. URL: https://github.com/NVIDIA/nvidia-docker (visited on 11/12/2018).

[18] *Case Management For The Knowledge Worker Era*. URL: https://jvzoggel.com/2016/08/16/case-management-for-the-knowledge-worker-era/ (visited on 12/28/2018).

[21] *Enterprise Container Platform | Docker*. URL: https://www.docker.com/ (visited on 11/12/2018).

[32] *HPI Future SOC Lab*. URL: https://hpi.de/en/research/future-soc-lab.html (visited on 11/12/2018).

[41] *How to set weights for imbalanced classes in Keras*. URL: https://datascience.stackexchange.com/questions/13490/how-to-set-class-weights-for-imbalanced-classes-in-keras (visited on 12/27/2018).

[43] *Introduction to word embedding and Word2Vec*. URL: https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa (visited on 11/08/2018).

[45] *Just in Time - Idea*. URL: https://www.economist.com/news/2009/07/06/just-in-time (visited on 12/06/2018).

[46] *Keras Documentation*. URL: https://keras.io/ (visited on 11/12/2018).

[55] *Pandas get dummies function documentation*. URL: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html (visited on 12/27/2018).

[60] *Project Jupyter*. URL: http://jupyter.org (visited on 11/12/2018).

[65] *Sequence Prediction Challenge (SPICE)*. URL: http://spice.lif.univ-mrs.fr/ (visited on 09/22/2018).

[73] *Text Generation With LSTM Recurrent Neural Networks*. URL: https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/ (visited on 11/08/2018).

[74] *The Unreasonable Effectiveness Of Recurrent Neural Networks*. URL: https://karpathy.github.io/2015/05/21/rnn-effectiveness/ (visited on 01/07/2019).

[75] *The shortest yet efficient Python implementation of the sequential pattern mining algorithm PrefixSpan*. URL: http://git.io/prefixspan-py (visited on 11/12/2018).

[77] *Understanding LSTM Networks – colah's blog*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 10/12/2018).

[78] *Waikato Environment for Knowledge Analysis (Weka)*. URL: https://www.cs.waikato.ac.nz/~ml/weka/ (visited on 10/13/2018).

[80] *When does Keras reset an LSTM state?* URL: https://stackoverflow.com/questions/43882796/when-does-keras-reset-an-lstm-state (visited on 12/07/2018).

DECLARATION

I certify that the material contained in this thesis is my own work and does not contain significant portions of unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation.

Hiermit versichere ich, dass diese Arbeit selbständig verfasst wurde und dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt wurden. Diese Aussage trifft auch für alle Implementierungen und Dokumentationen im Rahmen dieses Projektes zu.

Potsdam, January 25, 2019

Felix Wolff