Chair of Process and Data Science

RWTH AACHEN UNIVERSITY

Text-Aware Predictive Monitoring of Business Processes with LSTM Neural Networks

Master's Thesis

Author : **David Benedikt Georgi**

Supervisors : Dr.-Ing. M. Seran Uysal
M. Sc. Marco Pegoraro

Examiners : Prof. h. c. Dr. h. c. Dr. ir. Wil van der Aalst
Prof. Dr.-Ing. Ulrik Schroeder

Submission date : YYYY-MM-DD

This work is submitted to the institute

i9 Process and Data Science (PADS) Chair, RWTH Aachen University

Acknowledgments

I would like to my express my deep gratitude to Professor van der Aalst, who inspired and educated me during my entire master's studies and gave me the chance to write this thesis. I could profit from his outstanding scientific excellence and his didactic skills during all his delightful courses, which I will miss for sure.

Furthermore, I would like to thank Professor Schroeder, who kindly agreed to support me as the co-examiner of my thesis.

I would also like to thank my daily supervisors Seran and Marco, who accompanied and heavily supported me on the whole writing process of my thesis. I could always rely on their masterly guidance through all the obstacles I have encountered.

A sincere thank you is dedicated to Katja and Philip for their diligent proofreading.

Last but no least, I would like thank my parents for their unconditional love and support during my whole life. I would have never been able to succeed through my academic journey without their help.

Abstract

The prediction of real-time business processes using event data of historical executions is a critical capability of business process monitoring systems. Existing process prediction methods are limited in terms of the type of data they are able to utilize and the prediction tasks they can perform. This work describes the design, implementation and evaluation of a novel text-aware process prediction model based on long short-term memory (LSTM) neural networks and natural language models. The model is able to take categorical, numerical and textual attributes in event data into account to predict the activity and timestamp of the next event, the outcome and the cycle time of a process instance. The experiments show that the text-aware model is able to outperform state-of-the-art process prediction on simulated and real-world event logs containing textual data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The rapid growth of data generated and collected in large-scale information systems leads to new opportunities for society and businesses. By the end of 2020, the total amount of generated data is estimated to be 44 trillion gigabytes of which 90% has been created in the last two years [1]. In order to benefit from the massive amount of data, efficient solutions are required, that are able to extract potential value in form of models, insights and predictions.

A remarkable subset of this data can be described as *event data*, which is generated by *process-aware information systems*, which define, manage and execute business processes [2]. With the non stopping rise of digitization of business processes, increasingly more event data becomes utilizable, thus the potential value of this data is rising sharply.

The scientific engagement aiming to discover, analyze and improve real processes based on event data led to *process mining*. Process mining bridges the gab between the data-driven characteristic of data science and the process-centric view of process science [3]. The ongoing success of progress mining in research has been transferred to businesses, that successfully offer or utilize this technology. Celonis, which is often considered as one of the biggest commercial providers of process mining, has been valued 2.5 billion dollar only 9 years after the company was founded [4].

Modern process mining software tends to focus on continuous monitoring of business processes, in contrast to traditional offline and project-based approaches, that are not integrated within the remaining IT infrastructure of a company. The integrated and continuous application of process mining mostly realized by *business process monitoring systems*, which are a key success factor for many organizations. These systems allow to understand and supervise all processes of a company in real-time during the execution of the processes. The core idea of this approach is to automate process mining and keep a persistent data connection between the information system and the monitoring system, that provides the analytical capabilities. Figure 1.1 visualizes such an infrastructure and the interaction between the systems and internal and external process stakeholders. The operational employees, customers and connected software systems interact with the process-aware information system and are part of the processes themselves. In contrast, executives, data scientist, auditors and investors are interested in supervising the processes using the business process monitoring system, that is able to produce valuable insights.

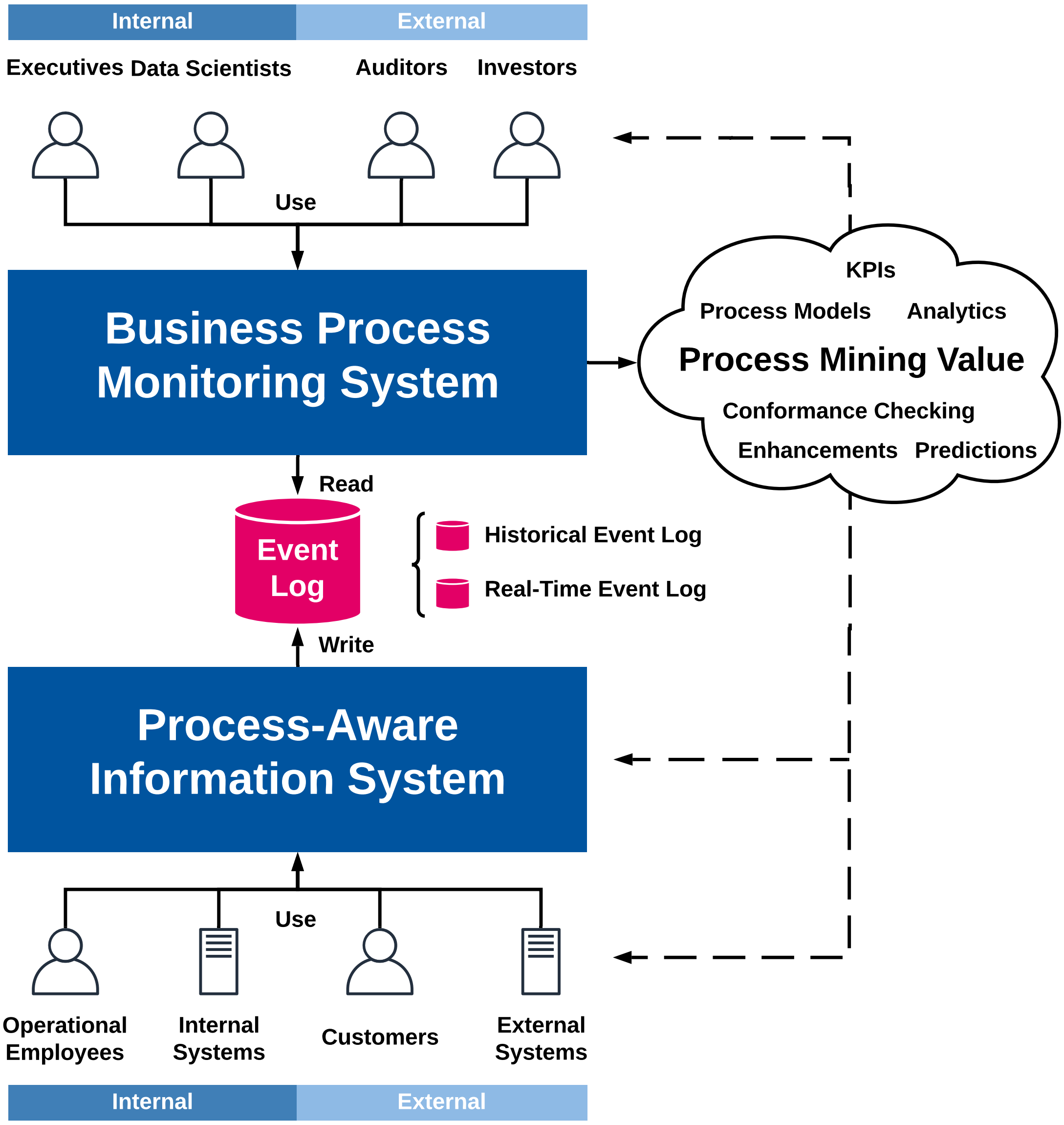


Figure 1.1: Business process monitoring allows to continuously apply process mining in an automated fashion in order to generate value for internal and external process stakeholders using the event data generated by a process-aware information system.

Traditional process mining tends to be backward-looking [5], i.e. the main focus relies on analyzing and understanding past executions of a process rather than providing insights for running process instances in form of predictions or recommendations. Businesses can develop a competitive advantage, if their process mining solution also offers predictive capabilities to anticipate the future of a running process instance. For example, if it can be expected, that a running process instance will likely exceed its deadline, measures can be initiated before damage occurs. Therefore, including the forward perspective is crucial for any competitive process mining software, especially in the context of business process monitoring.

1.2 Problem Statement

Although, many approaches for process prediction have been suggested in the literature (see Chapter 3), current solutions are limited regarding the data they are able to consider and prediction task they can perform. Many approaches derive their prediction purely from the control flow of a process instance ignoring additional data attributes in the event log. Most notably, almost no approach is able to consider textual data for process prediction. However, textual data is highly available in many systems and might hold important information, that can be used to improve the prediction performance. For example, millions of emails are sent everyday and their content influences processes inside of organizations. In addition, most of the existing prediction methods focus on a single prediction task only, for example they just predict the remaining time or cycle time of a process instance. Depending on the context, information about the next event or the future path of process instance can be of interest. In some scenarios, processes instances have an outcome like success/failure or accepted/rejected that can be predicted.

In data science, predictions are usually derived using *predictive inference* [6], i.e. correlations in past observations are used to estimate target variables for new observations. In the context of processes, past observations come in the form of historical event log data, that has been logged during the execution of a process.

This leads to the following problem statement: Given an event log with past executions of a process holding numerical, categorical and textual data and a running (i.e. not completed) process instance, a prediction model is required that is able to reliably perform the following prediction tasks in order to answer the corresponding questions for a process owner:

- Next activity prediction: What will happen next in the running process instance?
- Next event time prediction: When will the event happen?
- Case outcome prediction: What is the outcome of the process instance?
- Case cycle time prediction: When will the process instance finish?
- Future path prediction: What is the most likely sequence of future activities of the process instance?

A general description of such a process prediction model is illustrated in Figure 1.2. Existing models follow the same framework, but they are less flexible in terms of input data types and prediction targets. Including textual data for process prediction brings up new research questions, which are concretized in the next section.

1.3 Research Goals and Questions

This thesis aims to improve current state-of-art approaches for process prediction in order to extend the capabilities of process monitoring software. The main research goal is

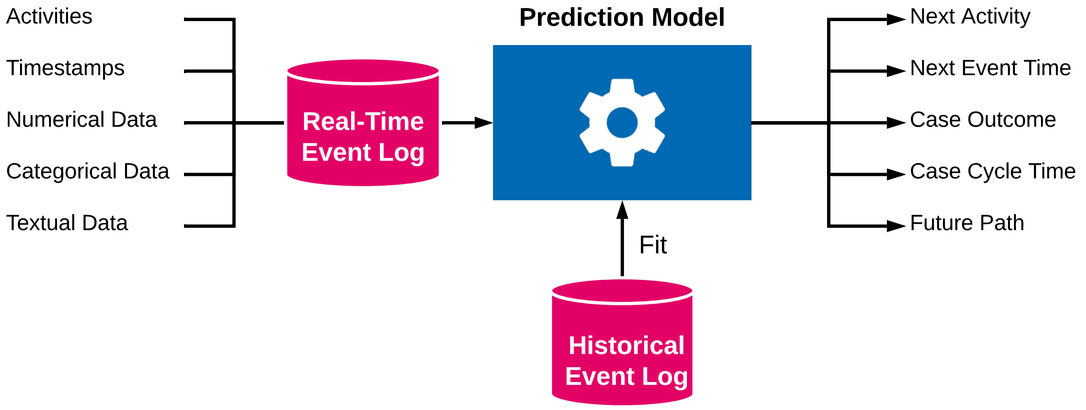


Figure 1.2: Predictive business process monitoring includes a prediction model, that is able to predict the future of running cases using historical event data. Current approaches differ in terms of the considered input data, the underlying prediction method and the prediction targets.

to design, implement and evaluate a predictive model for event data that is able to take advantage of additional textual data associated with each event in process. Since most current approach are not able to handle textual data, the goal is to investigate if and to what extend textual data can improve the quality of process prediction. Furthermore, different design choices and text models for text-aware process prediction need to be evaluated and potential trade-offs have to be discussed. Finally, the text-aware process prediction model has to be compared to current state-of-the-art process prediction methods.

These goals lead to the formulation of the following three research questions:

1. To what extend can the utilization of textual data improve the performance of process prediction?
2. How does the choice of the text model and other parameters influence the prediction results?
3. What are the advantages and disadvantages of the approach compared to existing methods?

1.4 Contribution

1.5 Thesis Structure

This thesis is structured in seven chapters. In Chapter 2 the basic notations, definitions and concepts used in this contribution are introduced. This includes an brief introduction to process mining, text mining, supervised learning and LSTM neural networks. Chapter 3 summarizes relevant scientific contributions, which focus on process prediction in process mining, and gives an overview of already available methods and their capabilities. In Chapter 4 a novel text-aware process prediction model as the main conceptional contribution is presented. Moreover, Chapter 5 covers the implementation details of the model on a technical level. In Chapter 6 the performance of the new approach is evaluated and compared to current state-of-the-art prediction methods. Finally, the conclusion is given in Chapter 7 by wrapping up the key achievements and discussing the limitations of the approach. Furthermore, an outlook towards future potential research questions on process prediction in the context of business process monitoring is given.

Chapter 2

Preliminaries

In this chapter the basic concepts of process mining, text mining, supervised learning and long short-term memory networks are presented. Furthermore, necessary formal definitions and notations are introduced.

2.1 Processes and Process Mining

Definition 2.1 (Business Process)**.** A *business process* is a collection of activities that are performed in a specific order to achieve a goal [7]. A single execution of a process is a *case* or *process instance*, which is identified by a case ID.

Each performed activity belongs to specific case and is completed at a certain time. For example, a case can be a patient treated in a hospital, a customer journey or an online order. The time on which an activity for a certain case is performed is specified by a timestamp. The subject identified by the case ID, the performed activity and the timestamp form an *event*. An event can have additional attributes, describing for example the resource, costs or transaction information associated with the event.

If the execution of a business process is logged by an information system, the resulting event data is called *event log*. Depending on the format of the event log, it can also contain additional data on case level. Typical formats for event logs, are comma-separated values (CSV) and eXtensible Event Stream (XES) [8], which can be extracted from databases of process-aware information systems. A table-based representation of an artificial event log describing patient treatments in a hospital can be seen in Table 2.1. Besides the case ID, activity and timestamp, the event log also contains information about the identity executing the activity, the costs of the activity and a text comment as an example for a categorical, numerical and textual attribute.

Definition 2.2 (Process Mining)**.** *Process mining* is the discipline that covers all approaches aiming to generate value out of event data. As an umbrella term, process mining includes or utilizes concepts of business process management, data mining, business process intelligence, big data, workflow management, business process monitoring [3] as well as machine learning [9].

Traditionally, process mining is divided into a set of subdisciplines mainly process discovery, conformance checking, process enhancement and process analytics [10]. Process

ID	Activity	Timestamp	Resource	Costs	Comment
0	Register patient	2020-02-01 14:12	SYSTEM	0	-
	Consultation	2020-02-01 14:34	J. Brown, MD	24.32	The patient reports persistent nausea.
	Blood test	2020-02-01 15:12	K. Smith	14:23	Tests: Complete blood count
	Evaluate test	2020-02-01 16:35	J. Brown, MD	38.67	No abnormalities in the complete blood count.
	Release patient	2020-02-01 17:24	SYSTEM	0	-
1	Register patient	2020-02-02 08:20	SYSTEM	0	-
	Consultation	2020-02-02 14:12	J. Simpson, MD	24.32	Noticeable tachycardia. No chronic pre-existing conditions are known.
	MRI	2020-02-02 16:10	Sara Taylor, MD	352.87	-
	Release patient	2020-02-02 18:33	SYSTEM	0	-
2	Register patient	2020-02-02 09:08	SYSTEM	0	-
	Consultation	2020-02-02 09:14	J. Simpson, MD	24.32	The patient has severe leg injuries due to a motorcycle accident.
	Hospitalization	2020-02-02 09:20	M. Johnson	130.37	-
...	...	...	...	...	...

Table 2.1: An artificial event log of patient treatments in a hospital. Each line corresponds to one event. The events are grouped by their case IDs, that each represent a single patient.

discovery aims to generate process models out of event data in order to understand the control flow of a process and enable further analysis. Conformance checking is about comparing the intended and observed behavior of a process and identifying deviations. On top of these diagnostic approaches, process enhancement deals with the improvement of processes regarding compliance, performance or complexity.

Finally, process analytics focuses on metric and performance evaluation of processes. Similar to conformance checking, this term is closely related to business process monitoring, a rising subfield enabling the analysis of running business processes in real-time. Driven by the fast and ongoing development of quantitative prediction methods in data science and machine learning, also prediction-based methods have been applied to event data. These methods add the forward perspective to business process monitoring and deal with forecasting the future of a running process instance, which is also the main focus of this work.

2.2 Basic Notations and Sequences

The set $\mathbb{N}$ denotes the set of all natural numbers $\{1, 2, 3, \dots\}$ and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ denotes the set of natural numbers including 0. The set of natural numbers up to n is noted as $[n] = \{1, 2, \dots, n\} \subset \mathbb{N}$ with $[0] = \emptyset$.

Definition 2.3 (Sequence)**.** A *sequence* of length $n \in \mathbb{N}_0$ over a set A is an ordered collection of elements defined by a function $\sigma\colon [n] \to A$, which assigns each index an element of A. A sequence of length n is represented explicitly as $\sigma = \langle a_1, a_2, \dots, a_n \rangle$ with $a_i \in A$ for $1 \le i \le n$. In addition, $\langle\rangle$ is the empty sequence of length 0.

Given a set A, A^n describes the set of all sequences $\langle a_1, a_2, \dots, a_n \rangle$ over A of length n. The set A^0 is defined as $\{\langle\rangle\}$, the set that only contains the empty sequence. $A^* = \bigcup_{i \in \mathbb{N}_0} A^i$ is the set of all sequences over A and $A^+ = \bigcup_{i \in \mathbb{N}} A^i = A^* \setminus \{\langle\rangle\}$ is the set of all sequences over A with a length of at least 1.

Given sequences σ_1 and σ_2, the concatenation of both sequences is denoted by $\sigma_1 \cdot \sigma_2$. Moreover, the i-th element of a sequence $\sigma = \langle a_1, a_2, \dots, a_n \rangle$ is accessed using $\sigma(i) = a_i$ for $1 \le i \le n$. The length of a sequence is denoted by $|\sigma|$. For a sequence $\sigma = \langle a_1, a_2, \dots, a_n \rangle$, the function $hd^k(\sigma) = \langle a_1, a_2, \dots, a_k \rangle$ gives the prefix of length k of σ and $tl^k(\sigma) = \langle a_{k+1}, a_{k+2}, \dots, a_n \rangle$ the suffix of length $n-k$ for $0 \le k \le n$. Note that $\sigma = hd^k(\sigma) \cdot tl^k(\sigma)$ for $0 \le k \le n$ and any sequence σ.

A function $f \colon A \to B$ can be lifted element-wise to sequences over A, precisely:

$$f(\sigma) = \begin{cases} \langle\rangle & \text{if } \sigma = \langle\rangle \\ \langle f(a_1), f(a_2), \dots, f(a_n) \rangle & \text{else} \end{cases}$$

If an element $a \in A$ appears in a sequence $\sigma \in A^*$, the set membership notation $a \in \sigma$ is used for simplification.

2.3 Events, Traces, Event Logs

Based on the definition of sequences, the concepts of events, traces and event logs can be formalized.

Definition 2.4 (Event)**.** An *event* is a tuple $e = (c, a, t, d_1, \dots, d_m) \in \mathcal{C} \times \mathcal{A} \times \mathcal{T} \times \mathcal{D}_1 \times \dots \times \mathcal{D}_m = \mathcal{E}$ where $c \in \mathcal{C}$ is the case ID, $a \in \mathcal{A}$ is the executed activity and $t \in \mathcal{T}$ is the timestamp of the event. Furthermore, each event contains a fixed number $m \in \mathbb{N}_0$ of additional attributes $d_1, \dots, d_m$ in their corresponding domains $\mathcal{D}_1, \dots, \mathcal{D}_m$. In case that no additional attribute data is given ($m = 0$) the event space $\mathcal{E}$ (set of all possible events) is reduced to its minimal form $\mathcal{C} \times \mathcal{A} \times \mathcal{T}$.

Each attribute $d \in \mathcal{D}$ of an event (including activity, timestamp and case ID) can be accessed by a corresponding projection function $\pi_D \colon \mathcal{E} \to \mathcal{D}$. For example, the activity a of an event e is retrieved by $\pi_{\mathcal{A}}(e) = a$.

Throughout this thesis, $\mathcal{C} = \mathbb{N}_0$, $|\mathcal{A}| < \infty$ and $\mathcal{T} = \mathbb{R}$ is assumed, where $t \in \mathcal{T}$ is given in Unix time, precisely the number of seconds since 00:00:00 UTC on 1 January 1970 minus the applied leap seconds. Each additional attribute is assumed to be numerical, categorical or textual, i.e. $\mathcal{D}_i = \mathbb{R}$, $|\mathcal{D}_i| < \infty$ or $\mathcal{D}_i = \Sigma^*$ for $1 \le i \le m$ and some fixed language-dependent alphabet Σ.

Definition 2.5 (Trace)**.** A *trace* is a finite sequence of events $\sigma = \langle e_1, e_2, \dots \rangle \in \mathcal{E}^*$ with increasing timestamps, i.e. $\pi_{\mathcal{T}}(e_i) < \pi_{\mathcal{T}}(e_j)$ for $1 \le i < j \le |\sigma|$.

A trace can be transformed into a sequence of attributes by applying a projection function to the trace. For example, $\pi_{\mathcal{A}}(\sigma)$ gives the sequence of the activities of the events in σ. The sequence of activities is also called *path* or *trace variant*.

Definition 2.6 (Event Log)**.** An *event log* or *trace log* $\mathbb{L} = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ is a set of traces, where each event of a trace is unique in the log and all events of a trace share a case IDs, which is unique per trace.

2.4 Text Mining

With the consideration of textual data in event logs, the concept of *text mining* becomes relevant. Text mining describes all techniques that aim to generate value out of unstructured or semi-structured textual data. It combines concepts of natural language processing, machine learning and data mining [11].

The base object in text mining is a *document* containing textual data. The textual data can be completed unstructured, i.e. it does not conform to a pre-defined data model, or semi-structured, like in an email, where text information is assigned to sender, subject, message etc. In this setting, a document $d \in \Sigma^*$ (i.e. textual data) is always a sequence of symbols from a fixed alphabet Σ. A collection of documents is called *text corpus*, which forms the basis for many text mining techniques.

In order to derive a mathematical representation of the textual data that can be interpreted by learning algorithms, a text model has to be build using the whole text corpus. Popular text models are Bag-of-words [12], Bag-of-n-gram, Paragraph vector (a.k.a. Doc2Vec) [13] and Latent Dirichlet Allocation [14]. Most models do not work with the raw text data, but require a text normalization step, where the text is cleaned from linguistic variation as well as meaningless words and symbols [15].

2.5 Supervised Learning

In *supervised learning* an unknown function is learned (i.e. approximated) from a set of example input-output pairs [16]. In contrast, in *unsupervised learning* no target outputs are available and the machine tries to identify pattern in the data **??**.

An input instance in the supervised scenario is usually described by a set of *feature variables* X and the output is defined by a *target variable* y. The target variable y is either continuous (regression problem) or discrete (classification problem). Given a *training set* of input-output pairs $\{(X_1, y_1), (X_2, y_2), \ldots, (X_m, y_m)\}$, that were generated from an unknown function $y = f(X)$, the goal is to compute a hypothesis function $h(X)$, which is as close as possible to the true function f(X), i.e. $h(X) \approx f(X)$.

The challenge in supervised learning is to generalize from the training set of input-output pairs in such a way, that the learned hypothesis function $h(x)$ can also successfully predict the target variable for unseen problem instances. In order to evaluate a hypothesis, the function is tested on a separate *test set* of input-output pairs, which has not been used for the construction of $h(X)$.

A hypothesis is assumed to generalize well, if its prediction performance is high on the training set as well as on test set. However, if the prediction performance is high on the training set, but performs poorly on unseen data, the hypothesis is *overfitting* with respect to the training data. In this case, the model fails to generalize. In contrast, if the model is too simple to fit any data from training set, the hypothesis is *underfitting*.

In many real-world applications, the true function $f(X)$ is stochastic, i.e. we need to estimate a conditional probability function $P(Y|X)$ (classification problem) or a conditional expectation $E(Y|X)$ (regression problem) for the prediction. Therefore, the prediction accuracy is always limited by the randomness of the true distribution. Typical supervised learning methods are linear regression, support vector machines, decision trees or neural networks including long short-term memory networks.

2.6 Long Short-Term Memory Networks

Long short-term memory (LSMT) is an advanced recurrent neural network architecture for sequential data originally presented by Hochreiter and Schmidhuber in 1997 [17]. This approach addresses the well-known vanishing or exploding gradient problem [18] of traditional recurrent neural networks by introducing more complex LSTM cells as hidden units. The proposed architecture has been improved several times [19] [20] and is considered as one of the most successful recurrent neural network models. Although, LSTM networks have been available for a long time, the breakthrough of this technology is dated around 2016 after many success stories of LSTM in combination with large data sets and GPU hardware have been reported for sequence to sequence tasks like text translation [21].

Gated recurrent units (GRU) [22] are the competing gating mechanism by Cho et al. that have fewer parameters and perform similar to LSTM. However, more recent studies show, that LSTM outperforms GRU consistently in neural machine translation tasks [23].

A simple feedforward neural networks consists of an input layer, arbitrarily many hidden layers and an output layer, where each layer consists of neurons that compute and output the weighted sum of the cells of the previous layer that has been passed to an non-linear activation function [24]. These networks can learn and compute complex functions in supervised learning settings, where input and output pattern in form of vectors are provided. The network computes a loss function for each training pattern and adjusts its weights with gradient descents using a back-propagation algorithm in order to minimize the loss function [25].

Recurrent neural networks extend traditional feedforward networks with backfeeding connections between hidden layers. This enables the network to keep a state across inputs and allows the neural network to process arbitrarily long sequences of input data while learning temporal dependencies.

In LTSM networks the layers are replaced by more complex LSTM modules, where each module contains four different sublayers. The architecture of a single LSTM module is shown in Figure 2.1. The module uses as input the state $\boldsymbol{c}_{t-1}$ and the hidden output $\boldsymbol{h}_{t-1}$ of the module in the previous time step as well as the output of the previous layer $\boldsymbol{x}_t$ to compute a new cell state $\boldsymbol{c}_t$ and a hidden output $\boldsymbol{h}_t$.

The input vector $\boldsymbol{x}_t$ is concatenated with the previous hidden output $\boldsymbol{h}_{t-1}$ and transferred to four neural network layers, which are designed to decide what part of the cell state will remain (forget gate $\boldsymbol{f}_t$), how it is updated (update gate $\boldsymbol{i}_t$ and $\tilde{\boldsymbol{c}}_t$) and what the output of the layer will be (output gate $\boldsymbol{o}_t$ leading to $\boldsymbol{h}_t$ considering the updated cell state $\boldsymbol{c}$). The sublayer apply $\text{sigmoid}(x) = \frac{1}{1+\exp(-x)}$ or $\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ activation functions elementwise to vectors, leading to the following equations:

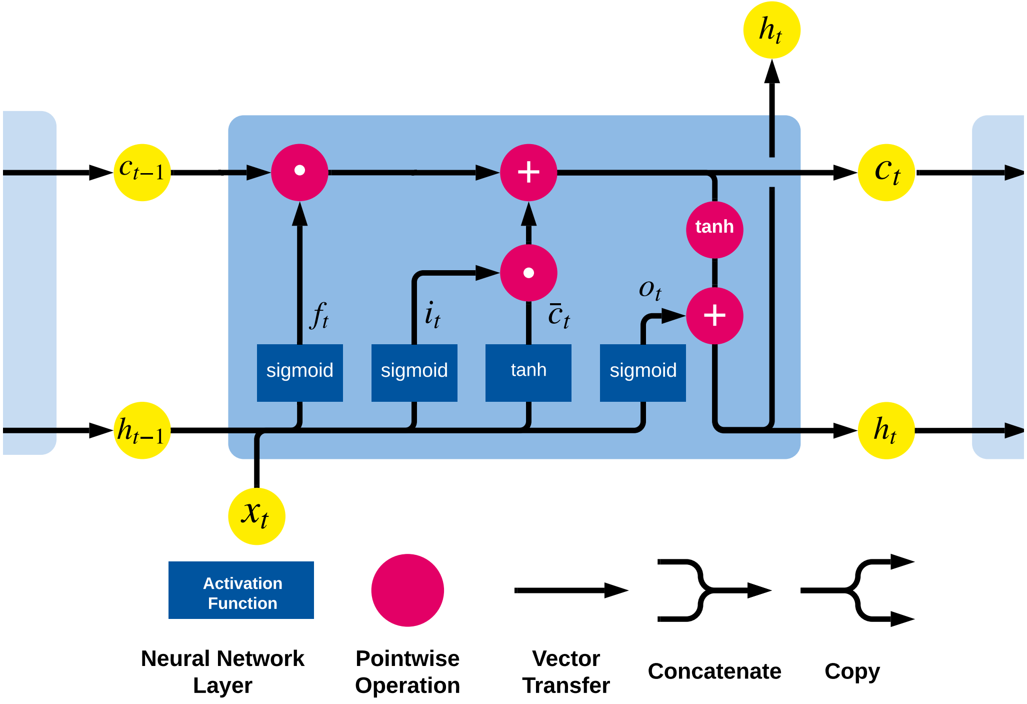


Figure 2.1: LSTM module with four sublayers that manipulate the cell state and compute the module's output. Graphic adapted from [26].

$$\begin{aligned}
\boldsymbol{f}_t &= \text{sigmoid}(\boldsymbol{W}_f \cdot (\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) + \boldsymbol{b}_f)\\
\boldsymbol{i}_t &= \text{sigmoid}(\boldsymbol{W}_i \cdot (\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) + \boldsymbol{b}_i)\\
\tilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_c \cdot (\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) + \boldsymbol{b}_c)\\
\boldsymbol{o}_t &= \text{sigmoid}(\boldsymbol{W}_o \cdot (\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) + \boldsymbol{b}_o)
\end{aligned}$$

$\boldsymbol{W}_f$, $\boldsymbol{W}_i$, $\boldsymbol{W}_c$ and $\boldsymbol{W}_o$ are the sublayer's learned weights and $\boldsymbol{b}_f$, $\boldsymbol{b}_i$, $\boldsymbol{b}_c$ and $\boldsymbol{b}_o$ are the corresponding biases.

The new cell state $\boldsymbol{c}_t$ is then a combination of the old cell state $\boldsymbol{c}_{t-1}$ and the result of the update gate $\tilde{\boldsymbol{c}}_t$, where the layer computations $\boldsymbol{f}_t$ and $\boldsymbol{i}_t$ determine the proportions by a pointwise multiplication ($\odot$) with the cell states.

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t$$

The result of the output gate $\boldsymbol{o}_t$ is pointwise multiplied with the tanh-activated new cell state to calculate the hidden output $\boldsymbol{h}_t$ of the module.

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$

LSTM networks are able to backpropagate a more stable error with this gating mechanism, such that these networks are much more capable of learning complex functions for sequences compared to standard recurrent neural networks. As a machine learning model that naturally supports sequential data, LSTM networks are a suitable prediction model for processes.

Chapter 3

Related Work

The prediction of the future course of a process instance has been an important subfield in process mining research, aiming to enhance process monitoring capabilities. Depending on the use case, for example predicting time-related attributes, the future path or the the outcome of a case can be of interest. Most approaches presented in the literature either use machine learning or process models to construct a predictor, that generalizes from a historical event log.

Five different non-parametric regression predictors for forecasting the total cycle time of an unfinished case have been presented by van Dongen et al. [27]. The estimates are based on activity occurrences, activity duration and attributes.

Furthermore, van der Aalst et al. proposed to build a transition system using a set, bag or sequence abstraction, which is annotated with time-related data in order to predict the cycle time of case [28]. The core idea of this approach is to replay unfinished cases on the learned transition system and compute the prediction using the annotated data.

Pandey et al. use a hidden markov model to predict the cycle time of a case using the activity and timestamp data of an event log [29].

Rogge-Solti and Weske showed how a stochastic Petri net can be used to predict the cycle time of a process instance. The model naturally supports parallelism in business processes and considers future events, which are expected to occur.

Ceci et al. presented an approach, where a sequence tree is learned in order relate running traces to similar historical traces [31]. A decision tree is then used to predict the next activity and the cycle time of a case.

Teinemaa et al. applied text vectorization techniques like bag-of-n-grams (BoNG), Latent Dirichlet Allocation (LDA) and Paragraph Vectors (PV) to textual data of processes in order to predict a binary label describing the process outcome [32]. In this approach random forest and logistic regression classifiers for each prefix length of a trace are trained.

Most recently, several authors have applied recurrent neural networks in form of LSTM networks for process prediction. Evermann et al. encode events using an embedding matrix as it is known for word embeddings. The embedded events are then used as input for an LSTM network that predicts the next activity [33].

Tax et al. use an one-hot-encoding of the activity and the timestamp of an event to predict the activity and timestamp of the next event. This is done by using a two-layered LSTM network [34].

The work by Navarin et al. adopts the idea of using an LSTM network [34] and extends the encoding to additional data attributes associated with each event [35] to predict the cycle time of an case.

Polato et al. presented a set of approaches that use support vector regression for cycle time prediction [36]. In this work the authors implement different encoding for events including simple one-hot-encoding and a more advanced state based encoding using transition systems. Furthermore, they enhance the approach in [28] by taking additional data attributes into account.

Teinemaa et al. reported an in-depth review and benchmark of outcome-oriented predictive process monitoring approaches. The study showed that aggregated encoding like counting frequencies of activities as most reliable encoding for outcome-prediction [37].

Park and Song showed how LSTM-based predictions can be used to solve a resource allocation problem, leading to direct recommendations for process improvement [38].

A comparison of the process prediction methods is presented in Table 3.1.

Contribution	**Year**	**Model(s)**	**Data-Aware**	**Text-Aware**	**Predictions**
Van Dongen et al. [27]	2008	Regression	✓	✗	Cycle time
Van der Aalst et al. [28]	2011	Transition system	✗	✗	Cycle time
Pandey et al. [29]	2011	Hidden Markov	✗	✗	Cycle time
Rogge-Solti and Weske [30]	2013	Stochastic Petri net	✗	✗	Cycle time
Ceci et al. [31]	2014	Sequence tree Decision tree	✓	✗	Next activity Cycle time
Teinemaa et al. [32]	2016	Random forest Logistic regression	✓	✓	Case outcome
Evermann et al. [33]	2016	LSTM	✗	✗	Next activity
Tax et al. [34]	2017	LSTM	✗	✗	Next activity Next event time Cycle time Future path
Navarin et al. [35]	2017	LSTM	✓	✗	Cycle time
Polato et al. [36]	2018	Transition system SVR Naive Bayes	✓	✗	Next activity Cycle time Future path
Park and Song [38]	2019	LSTM	✓	✗	Next activity Next event time
This contribution	2020	LSTM	✓	✓	Next activity Next event time Cycle time Case outcome Future path

Table 3.1: Comparison of process prediction methods.

Chapter 4

Text-Aware Process Prediction

Text-aware process prediction aims to utilize unstructured text information in event data to improve predictions for unfinished cases. While many prediction methods have been applied to event data, almost none of them are able to handle textual data. Nevertheless, a lot of textual information in the context of processes is available, for example in form of business emails or notes by employees or customers. These texts in natural language might hold process-critical information and should consequently be considered for process prediction. However, taking advantage of textual data remains a major challenge, since natural language is predominantly context-sensitive and ambiguous [39]. A first approach has been presented by Teinemaa et al. [32], where traces with textual data are encoded as vectors and a random forest classifier is learned for each prefix length.

In this chapter a novel approach for text-aware process prediction is presented that considers the control flow and additional numerical, categorical and textual data of the process. The model is able to captures temporal dependencies between events, seasonal variability and concept drifts using an event-wise encoding and a sequential LSTM prediction model. The main application scenario for the model is inside of real-time business process monitoring software, where prediction capabilities for running processes give a competitive advantage.

4.1 Overview

The framework is designed to approximate prediction functions $f_\mathrm{a}, f_\mathrm{t}, f_\mathrm{o}$ and f_c, that predict the next **a**ctivity, next **t**imestamp, case **o**utcome and case **c**ycle time given any prefix $hd^k(\sigma)$ of length $1 \leq k \leq |\sigma|$ of the full (unkown) trace $\sigma = \langle e_1, \dots, e_{|\sigma|} \rangle \in \mathcal{E}^+$. The next activity prediction function $f_\mathrm{a} \colon \mathcal{E}^+ \to \mathcal{A} \cup \{\text{END OF PROCESS}\}$ returns the activity of the next event or an artificial END OF PROCESS activity, if the given trace is already completed, precisely:

$$f_\mathrm{a}(hd^k(\sigma)) = \begin{cases} \text{END OF PROCESS} & \text{if } k = |\sigma| \\ \pi_\mathcal{A}(\sigma(k+1)) & \text{else} \end{cases}$$

Furthermore, $f_{\mathrm{t}}\colon \mathcal{E}^+ \rightarrow \mathbb{R}$ gives the timestamp of the next event in relation to the last event in the prefix:

$$f_{\mathrm{t}}(hd^k(\sigma)) = \begin{cases} \pi_{\mathcal{T}}(\sigma(|\sigma|)) - \pi_{\mathcal{T}}(\sigma(|\sigma|-1)) & \text{if } k = |\sigma| \\ \pi_{\mathcal{T}}(\sigma(k+1)) - \pi_{\mathcal{T}}(\sigma(k)) & \text{else} \end{cases}$$

The case outcome of trace can be a binary label, ie. a label describing if the case has been successful or has failed. In some applications the case outcome is defined by the activity of the final event of a case. In that scenario,the outcome function $f_{\mathrm{o}}\colon \mathcal{E}^+ \rightarrow \mathcal{A}$ returns the last activity of the trace:

$$f_{\mathrm{o}}(hd^k(\sigma)) = \pi_{\mathcal{A}}(\sigma(|\sigma|))$$

Finally, $f_{\mathrm{c}}\colon \mathcal{E}^+ \rightarrow \mathbb{R}$ returns the total duration of the process, i.e. the time difference between the first and the last event of the trace:

$$f_{\mathrm{c}}(hd^k(\sigma)) = \pi_{\mathcal{T}}(\sigma(|\sigma|)) - \pi_{\mathcal{T}}(\sigma(1)).$$

All functions are approximated via an LSTM model using historical event log data, i.e. a set of completed traces with full information about the course of the process instances. The goal is to generalize from the completed traces in such a way, that the prediction error is minimized for new unseen and uncompleted traces. Due to the probabilistic behavior of real-world processes, the prediction accuracy is always limited by the randomness of the process behavior.

The proposed framework (see Figure 4.1) consists of a preprocessing, encoding and prediction model component, which operate in an offline and online phase. In the offline phase, a historical event log with completed traces of a business process is used to fit the encoding and prediction component. Given a historical event log $\mathbb{L} = \{\sigma_1, \ldots, \sigma_l\}$ with completed traces, the set of all prefix traces $\mathbb{L}_{\text{prefix}} = \{hd^k(\sigma) \mid \sigma \in \mathbb{L}, 1 \leq k \leq |\sigma|\}$ is computed and each trace is encoded as a sequence of event vectors. The encoding component distinguish between categorical or numerical data that can be encoded directly and textual data, that is prepossessed and encoded based on a text model. The text model is an exchangeable component and is fit to the text corpus, that is extracted from the textual data in the event log $\mathbb{L}$.

Each encoded prefix sequence with its desired prediction target values according to $f_{\mathrm{a}}, f_{\mathrm{t}}, f_{\mathrm{o}}$ and f_{c} corresponds to one training example for an LSTM network, that finally realizes the predictions. The total number of training examples that can be generated out of the log is $\sum_{\sigma \in \mathbb{L}} |\sigma|$, which is exactly the number of events in the log.

In the online phase, the model applies the learned prediction function to predict the activity and time of the next event as well as the outcome and cycle time of new unseen and unfinished traces (real-time event log), that are monitored by the business process monitoring system.

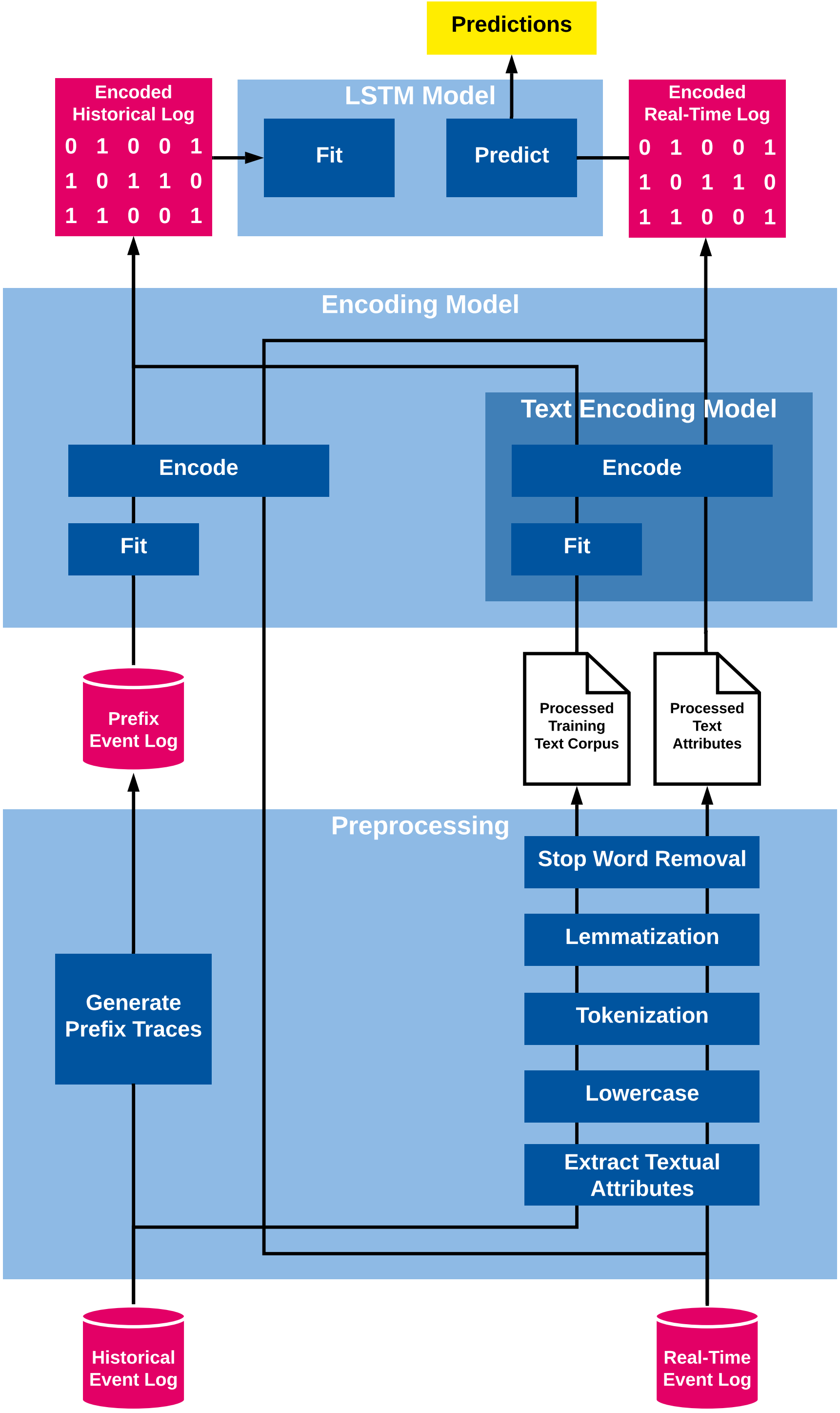


Figure 4.1: Overview of the text-aware process prediction framework. Predictions for real-time processes are realized by an LSTM model that is fitted using an encoded representation of all prefixes of the historical event log. The encoding of textual attributes is realized by a text preprocessing pipeline and an exchangeable text encoding model.

4.2 Event Encoding

In the offline training phase as well as during online prediction, traces are encoded as sequences of event vectors. The prefix log $\mathbb{L}_{\text{prefix}}$ is encoded as training set in the offline phase, while in the online phase, running cases are encoded for prediction. Strictly speaking, an encoding function is realized by the encoding component, that transforms (prefix) traces of length k to vector sequences of the same size, i.e. $enc(\sigma) = \langle \boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_k} \rangle$ with $\sigma = \langle e_1, e_2, \ldots, e_k \rangle$ for $k \in \mathbb{N}$. Each event e_i is encoded as a fixed-length vector using the activity, timestamp and additional categorical, numerical and textual data, that is associated with each event. We assume to have $r \in \mathbb{N}_0$ numerical, $s \in \mathbb{N}_0$ categorical and $u \in \mathbb{N}_0$ textual attributes, i.e. $e_i \in \mathcal{C} \times \mathcal{A} \times \mathcal{T} \times \mathcal{D}_1^{\text{num}} \times \cdots \times \mathcal{D}_r^{\text{num}} \times \mathcal{D}_1^{\text{cat}} \times \cdots \times \mathcal{D}_s^{\text{cat}} \times \mathcal{D}_1^{\text{text}} \times \cdots \times \mathcal{D}_u^{\text{text}}$. Each encoded event vector $\boldsymbol{x_i}$ is the concatenation of set of feature vectors, which are constructed from the event data.

$$\boldsymbol{x_i} = (\boldsymbol{a_i}, \boldsymbol{t_i}, \boldsymbol{d}_{i1}^{\text{num}}, \ldots, \boldsymbol{d}_{ir}^{\text{num}}, \boldsymbol{d}_{i1}^{\text{cat}}, \ldots, \boldsymbol{d}_{is}^{\text{cat}}, \boldsymbol{d}_{i1}^{\text{text}}, \ldots, \boldsymbol{d}_{iu}^{\text{text}})$$

The activity of the event is represented by vector $\boldsymbol{a_i}$ using *one-hot encoding.* More precisely, given the set of possible activities $\mathcal{A}$, an arbitrary but fixed ordering over $\mathcal{A}$ is introduced with a bijective index function $index_{\mathcal{A}} \colon \mathcal{A} \to \{1, \ldots, |\mathcal{A}|\}$. Using this function, the activity is encoded as a vector of size $|\mathcal{A}|$, where the component $index_{\mathcal{A}}(\pi_{\mathcal{A}}(e))$ has value 1 and all the other components have value 0. We write $\mathbb{1}_{\mathcal{A}} \colon \mathcal{A} \to \{0,1\}^{\mathcal{A}}$ to describe the function that realizes such an one-hot encoding transformation for the set of all activities $\mathcal{A}$. The timestamp of the event is used to compute a six-dimensional vector $\boldsymbol{t_i}$ of time-related features, which is explained in detail in Section 4.3.

Additional attributes of the events are encoded based on their type, i.e. if they are numerical, categorical or textual. Categorical attributes are encoded using one-hot encoding in the same way as the activity, i.e. $\boldsymbol{d}_{ij}^{\text{cat}} = \mathbb{1}_{\mathcal{D}_j^{\text{cat}}}(\pi_{\mathcal{D}_j^{\text{cat}}}(e_i))$. Textual attributes are vectorized via a dedicated text model, that is explained in Section 4.4. The dimensions of the text vectorizations can be tuned individually per attribute using the parameter z_j, which is the encoding dimension of the j-th textual attribute.

Feature Vector	Construction	Dimension	Description
$\boldsymbol{a_i}$	$\mathbb{1}_{\mathcal{A}}(\pi_{\mathcal{A}}(e_i))$	$\|\mathcal{A}\|$	One-hot encoding of the activity.
$\boldsymbol{t_i}$	See Section 4.3	6	Time-based feature vector.
$\boldsymbol{d}_{ij}^{\text{num}}$	$\widehat{\pi_{\mathcal{D}_j^{\text{num}}}(e_i)}$	1	Normalized value of the j-th numerical attribute
$\boldsymbol{d}_{ij}^{\text{cat}}$	$\mathbb{1}_{\mathcal{D}_j^{\text{cat}}}(\pi_{\mathcal{D}_j^{\text{cat}}}(e_i))$	$\|\mathcal{D}_j^{\text{cat}}\|$	One-hot encoding of the j-th categorical attribute.
$\boldsymbol{d}_{ij}^{\text{text}}$	See Section 4.4	z_j (parameter)	Fixed-length vectorization of the j-th text attribute.

Table 4.1: Feature vectors as part of the event encoding $\boldsymbol{x_i}$ for an event e_i.

All additional numerical attributes $\pi_{\mathcal{D}_i^{\text{num}}}(e_i)$ are scaled to the interval $[0,1]$ to improve learning efficiency using min-max normalizing. The scaling for a numerical feature x is realized with the transformation

$$\hat{x} = \frac{x - \min(x)}{\max(x) - \min(x)},$$

where $\min(x)$ is the lowest and $\max(x)$ is the highest value x can take. If the limits are not bounded conceptually, the lowest or highest value of x in the event log is used for scaling.

All in all, the encoding of an event e_i results in a vector of size

$$|\boldsymbol{x_i}| = |\mathcal{A}| + r + \sum_{i=1}^{s} |\mathcal{D}_i| + \sum_{j=1}^{u} z_j + 6.$$

An overview of each feature vector that is part of the event encoding is provided in Table 4.1.

4.3 Capturing Temporal Dependencies

A set of time-based features is computed from the timestamp data in the event log in order to profit from time-related pattern in the process. As part of the complete encoding $\boldsymbol{x}_i$ for an event e_i in a (prefix) trace $\sigma = \langle e_1, \ldots, e_k \rangle$, a time vector $\boldsymbol{t_i} = (t_i^1, t_i^2, t_i^3, t_i^4, t_i^5, t_i^6)$ of dimension 6 is computed.

Feature	Description
t_i^1	Normalized time in seconds since previous event
t_i^2	Normalized time in seconds since case start
t_i^3	Normalized time in seconds since first recorded event
t_i^4	Normalized time in seconds since midnight
t_i^5	Normalized time in seconds since last Monday
t_i^6	Normalized time in seconds since last January 1 00:00

Table 4.2: Time-based features as part of event encoding $\boldsymbol{x_i}$.

Using these features time-related correlations can be captured and utilized for prediction. The feature t_i^1 describes the time difference between the current event e_i and previous event e_{i-1}, while t_i^2 gives the time difference between the current event and the first event of the case e_1, i.e. the time since the start of the case. Furthermore, t_i^3 is the time difference between the current event and the first event that is recorded in the log. This feature indicates the absolute time position of an event in the data. This information is important to detect concept drift [40] in the process. Most real-world processes are not static, i.e. the behavior of the process changes over time. For example, in earlier process executions customers might have been informed by a letter, whereas in more recent cases customers are messaged using email or app notifications. Therefore, the knowledge about the absolute time of the events can be used to relate cases in similar periods of time.

The features t_i^4, t_i^5 and t_i^6 describe the time of the event in relation to the day, week and year. They are used to capture daily, weekly and seasonal trends. For example, some activities might only be executed during office hours or before the weekend. Also, many businesses expect seasonally fluctuating demand, for example a booking platform for vacation usually has much more customers in summer, which can affect the process execution in many ways. Each feature $t_i^1, \dots, t_i^6$ is min-max normalized such that $t_i^j \in [0, 1]$ for $j \in [6]$. A summary of all time-related features can be seen in Table 4.2.

4.4 Text Vectorization

In order to prepare the textual data of the event log for a prediction model, the texts have to be encoded in a compact, finite and useful numerical vector representation using a text model. Useful in that context means, that texts with similar semantic meanings should also have similar representations. The vector representation of textual data is an important aspect in *Natural Language Processing* (NLP). Extracting the meaning of textual information remains a challenge even for humans, because textual data is unstructured, language dependent and domain specific. Many words are ambiguous, for example the word 'apple' might denote a fruit or a global technology company. In addition, grammatical variations and the importance of context in language makes extracting the semantic meaning even more difficult for computers.

In our setting, the text vectorization for textual attributes is realized in a two step procedure. First, all textual data associated with the events in the corresponding textual attribute is collected in a so called *text corpus*. Each document in the text corpus is then preprocessed in order to filter out linguistic noise or useless information. This step is called *text normalization* [15]. Finally, the normalized text corpus is used to build up a *vocabulary* and a text vectorization technique is applied to encode the text of the attribute into a fixed-length vector. The vocabulary of a text corpus is a set V of all relevant words (after filtering) that appear in the corpus, which is indexed by a bijective index function $index_V \colon V \rightarrow \{1, 2, \dots, |V|\}$. As text vectorization techniques the *Bag of Words*, *Bag of N-Grams*, *Paragraph Vector* and *Latent Dirichlet Allocation* text models are considered, which are introduced in more detail in the Sections 4.4.2 to 4.4.5.

4.4.1 Text Normalization

In the text normalization step each document of the text corpus is transformed by a preprocessing pipeline which consists of the following four steps:

1. All letters are converted to lowercase
2. The document is tokenized (i.e. splitted) by word
3. Each word is lemmatized
4. Stop words are filtered out

The first step eliminates all capital letters in the text. In the tokenenization step a document is split up into a sequence of words. Each word is then lemmatized, i.e. it is converted to its canonical form. The idea is to unify words that have a very similar meaning and filter out grammatical variations. For example, the words 'go', 'going', 'goes', 'gone' and 'went' are all transformed to the basic form 'go'. Lemmatization significantly reduces the complexity of textual data accepting a hopefully small loss of information.

Ultimately, all stop words are filtered out of each document. Stop words are words with low

Step	Transformation	Example Document
0	Original	'The patient has been diagnosed with high blood pressure.'
1	Lowercase	'the patient has been diagnosed with high blood pressure.'
2	Tokenization	⟨'the', 'patient', 'has', 'been', 'diagnosed', 'with', 'high', 'blood', 'pressure', '.'⟩
3	Lemmatization	⟨'the', 'patient', 'have', 'be', 'diagnose', 'with', 'high', 'blood', 'pressure', '.'⟩
4	Stop word filtering	⟨'patient', 'diagnose', 'high', 'blood', 'pressure'⟩

Table 4.3: Preprocessing transformation of an example document containing a single sentence.

information value like 'the', 'a', 'of' or 'here'. Stop word lists are language dependent and can be more or less aggressive at filtering. Usually they contains articles, auxiliary verbs, prepositions and generic verbs like 'be' and 'have'. In addition, punctuation marks and numerical symbols are excluded. An example application of the text normalization pipeline is shown in Table 4.3. Using the corpus of normalized documents (i.e. texts), a text model is fitted in the offline phase to realized the encoding of the textual attributes.

4.4.2 Bag of Words

The *Bag of Words* (BoW) model is a simple text model, which represents documents based on the *term frequencies* of its words, while ignoring their order [12]. Given the learned vocabulary V, a document is represented by a vector of size $|V|$, where the i-th component gives the number of occurrences of the word indexed with i in the document.

Since this approach does not reflect the prior distributions of words in the corpus, i.e. how likely certain words appear in a document in general, the term frequencies are usually normalized by the so-called *inverse document frequency* (idf) of a word. The inverse document frequency indicates the specificity of a word in the corpus and is computed by dividing the total number of documents by the number of documents that contain the specific word and scaling that value logarithmically. The resulting value is the tf-idf score of a word in a document.

The Bag of Words model is easy to build and effective for certain applications, but limited in several ways. First, the model completely ignores the order of words, which is often crucial for understanding the semantic meaning. For example, the sentences 'The patient's health state went from stable to critical.' and 'The patient's health state went from critical to stable.' would result in the same vector representation, while the meaning is clearly inverted. Second, the vector representations are sparse and of high dimensionality since they depend of the size of the vocabulary. However, the dimension can be reduced by limiting the size of the vocabulary. For example, words with low term frequencies can be excluded from the vocabulary. Unseen words, that appear during online prediction and are not part of the learned vocabulary are not considered.

4.4.3 Bag of N-Grams

The *Bag of N-Grams* (BoNG) model is a generalization of the Bag of Words model, which addresses the missing word order awareness of the latter. Instead of single words, the vocabulary consists of n-tuples of words, that appear consecutive in the corpus. The unigram model ($n = 1$) is equivalent to the BoW model. For the bigram model ($n = 2$), the vocabulary consists of pairs of words that appear next to each other in the documents. For example, for the preprocessed document ⟨'patient', 'diagnose', 'high', 'blood', 'pressure'⟩, the pairs ('patient', 'diagnose'), ('diagnose', 'high'), ('high', 'blood') and ('blood', 'pressure') are taken into the vocabulary. For $n > 2$ n-tuples are generated accordingly. The feature vector is constructed by computing the tf-idf score for each vocabulary entry like in the BoW model.

Compared to the BoW model, n-grams also take the order of words into account, which is beneficent in many scenarios. However, the vocabulary size is usually even higher than in the BoW model. In order to generate more compact vectors, distributed text representations are needed for larger text corpora and vocabularies.

4.4.4 Paragraph Vector

The *Paragraph Vector* model (also known as *Doc2Vec*), originally presented by Le and Mikolov in 2014, is an unsupervised algorithm that learns distributed fixed length vector representations for documents of variable length using a simple feedforward neural network [13]. The idea is inspired by the word embedding model presented by Bengio et al. [41], which can learn distributed fixed-length vector representations for words. In this model words are mapped to vectors, that are trained to predict words from its context, i.e. words that appear before or after the target word in the training documents. Several variants of this approach exits, notably the *Continuous Bag of Words* model, which ignores the order of the words in the context and the *Continuous Skip-gram* model, which predicts the skip-gram context for a word vector (also known as *Word2Vec*) [42].

The core idea of the Paragraph Vector model is to extend the model in [41] in a way, that an additional document (i.e. paragraph) vector is trained together with the word vectors, that is unique per document. Fig. 4.2 show the architecture of the distributed memory variant of the Paragraph Vector model (PV-DM). Its is realized by a neural network, that takes one-hot encoded words and an one-hot encoded document ID as input. These are mapped to vector representation via weight matrices $\boldsymbol{D}$ and $\boldsymbol{W}$, which are learned during training with gradient descent. The distributed representations are then averaged or concatenated to a vector in order to predict the one-hot encoded target word using another mapping via $\boldsymbol{W'}$ and a softmax activation function. The training set is constructed using a sliding window over every document, such that the input is the context of the target word and the document ID. After training, each column in $\boldsymbol{D}$ represents the distributed encoding of the corresponding document.

The network is also able to learn a representation for new unseen documents by an inference step. In this phase, the word matrix $\boldsymbol{W}$ and the prediction matrix $\boldsymbol{W'}$ are fixed and only the document vector is trained. The paragraph vector model tends to perform better than non-distributed models, however since new documents are vectorized via inference, a bigger training corpus of documents is usually required.

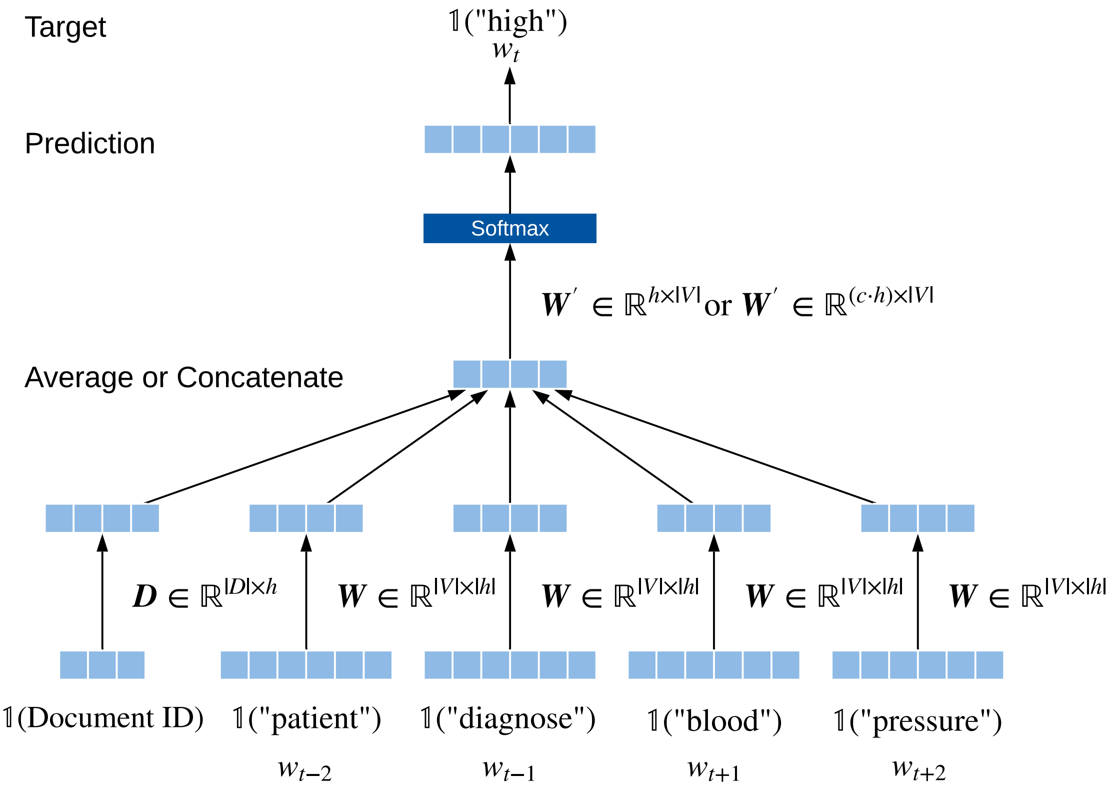

Target
𝟙("high")
w_t
Prediction
Softmax
$\boldsymbol{W}' \in \mathbb{R}^{h\times|V|}$ or $\boldsymbol{W}' \in \mathbb{R}^{(c\cdot h)\times|V|}$
Average or Concatenate
$\boldsymbol{D} \in \mathbb{R}^{|D|\times h}$
$\boldsymbol{W} \in \mathbb{R}^{|V|\times|h|}$
$\boldsymbol{W} \in \mathbb{R}^{|V|\times|h|}$
$\boldsymbol{W} \in \mathbb{R}^{|V|\times|h|}$
$\boldsymbol{W} \in \mathbb{R}^{|V|\times|h|}$
𝟙(Document ID)
𝟙("patient")
𝟙("diagnose")
𝟙("blood")
𝟙("pressure")
w_{t-2}
w_{t-1}
w_{t+1}
w_{t+2}

Figure 4.2: The distributed memory Paragraph Vector model (PV-DM) is designed to predict a word w_t from its context and derives fixed-length representation of documents and words via the learned matrices $\boldsymbol{D}$ and $\boldsymbol{W}$. The dimension of the representations are defined before by parameter h.

4.4.5 Latent Dirichlet Allocation

The *Latent Dirichlet Allocation* (LDA) presented by Blei et al. in 2003 [14] is generative statistical text model, which represents documents as a mixture of a small and fixed number of topics. Topics are defined by probability distribution over all words in the vocabulary and are also learned by the model in an unsupervised manner. The underlying assumption of the LDA model is that the documents were created by a statistical process, that sampled the words of the document from sampled topics. A document is encoded as a vector, whose dimension is equal to the number of topics and each component indicates the probability that the corresponding topic distribution was chosen to sample a word in the document. The distributions and therefore document encoding can be learned with statistical inference algorithms like Monte Carlo Simulation, Variational Bayes or maximum likelihood estimation using the words of each document. Compared to the other text models, LDA can generate very compact documents encodings, since the documents are only described by their degree of affiliation to each topic. Similar to the Bag of Words model, LDA does not consider the order of words in the document, which can be disadvantageous for some text corpora.

4.5 Network Architecture and Training

The encoded historical prefix event log is used to fit an LSTM model. The LSTM network is designed to be trained with all prediction targets (next activity, next event time, case outcome and case cycle time) at once, in order to benefit from correlations between these. In the basic variant, the network consists of an input layer, an shared LSTM layer, an specialized LSTM layer for each prediction target and an fully connected output layer for each target. Furthermore, layer normalization [43] is applied after each LSTM layer, which standardizes the hidden output in order to speed up the training convergence.

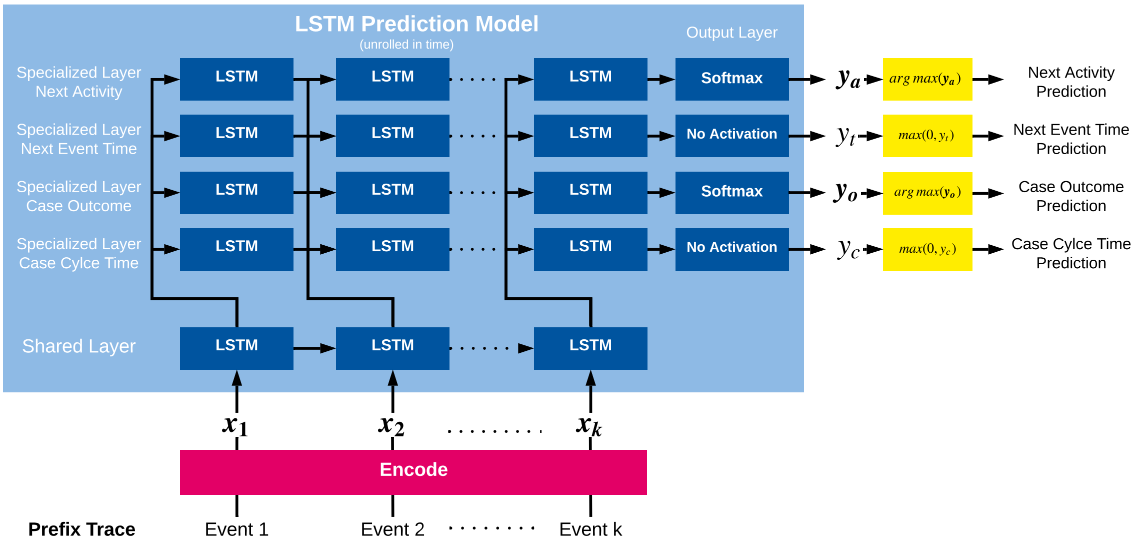


Figure 4.3: LSTM model to simultaneously predict the next activity ($\boldsymbol{y_a}$), next event time(y_t), case outcome ($\boldsymbol{y_o}$) and case cylce time (y_c) for an encoded prefix trace $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}$.

The fully connected output layer uses a softmax activation function for the next activity and case outcome prediction to estimate the probability for each target value. The softmax function normalizes a vector of real numbers into another vector of same dimension, such that all component are in the interval [0,1] and the sum of all component is equal to 1. Hence, the transformed vector can be interpreted as a probability distribution, while keeping the proportions of the original vector. The softmax function is described with

$$\text{Softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \text{ for } i = 1, \dots, n \text{ and } \boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{R}^n.$$

The whole training set of encoded prefix traces is represented for efficiency by a 3-dimensional matrix of real values, where the three dimensions specify the prefix traces, the events per prefix trace and the features per event. Since the prefix traces have different length, shorter traces are pre-padded [44] with zeros vectors. Hence, a prefix trace of encoded events $\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_n}$ is represented in the training set by a 2-dimensional matrix $(\boldsymbol{0}, \dots, \boldsymbol{0}, \boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_n})$, such that the zero vectors fill up shorter traces to the length of the longest trace in the training set. All prefix traces together form the 3-dimensional training matrix.

The training is realized using a learning algorithm based on stochastic gradient descent and backpropagation through time (BPTT), which update the weights of the network using the update rules of the Adam optimizer with Nesterov momentum [45]. The loss for numerical prediction values $\hat{y}$ and the true value y is the absolute error $\text{AE}(\hat{y}, y) = |\hat{y} - y|$, while the loss for categorical prediction values is computed using the categorical cross entropy error $\text{CE}(\boldsymbol{\hat{y}}, \boldsymbol{y}) = -\sum_{i=1}^{k} y_i \cdot \log \hat{y}_i$.

4.6 Predictive Business Processing Monitoring

During online business process monitoring, predictions are realized by a forward-pass of the encoded, uncompleted traces through the LSTM model. The component with the highest value of the softmax outputs for the next activity ($\boldsymbol{y_a}$) and the case outcome ($\boldsymbol{y_o}$) indicates the categorical prediction. The output values for the next event time (y_t) and case duration (y_c) are clipped to 0 for negative outputs and the normalization is reverted in order to compute the final prediction value.

With every new event, that is registered by the business process monitoring system, the prediction can be updated with another forward-pass through the model. This guarantees a continuous forward-projection on running process instances, that is updated whenever new information in form of new events becomes available. By predicting the timestamp and activity of the next event in an iterative way, the complete future path of running process instance can be predicted. The path ends, when the model predicts the artificial END OF PROCESS activity.

Chapter 5

Implementation

This chapter covers the implementation of the text-aware process prediction model presented in the previous chapter. In Section 5.1 the underlying technology is depicted, whereas in Section 5.2 the architecture of the implementation is defined.

5.1 Technology

The implementation of the text-aware process prediction model is purely based on Python 3.6 [46]. The set of Python packages that are utilized for the implementation are summarized in Table 5.1. All packages follow the open-source development model and are mostly community-driven.

Package	**Developer(s)**	**Purpose**
PM4Py [47]	Fraunhofer Institute for Applied Information Technology	Event log parsing and handling
TensorFlow [48]	Google Brain Team et al.	Construction and training of LSTM model
NTLK [49]	Bird et al.	Text preprocessing
Scikit-learn [50]	Cournapeau et al.	Bag of Words and Bag of N-Gram tf-idf encoding
Gensim [51]	Řehůřek et al.	Latent Dirichlet Allocation and Paragraph Vector encoding

Table 5.1: List of Python packages used for implementation.

PM4Py [47] is a Python package developed by the Fraunhofer Institute for Applied Information Technology, which offers a wide range of process mining algorithms and event log operations for the Python environment. It is used for event log parsing and its internal event log representation.

TensorFlow [48] is a dataflow oriented framework originally developed by Google, which includes a diverse set of neural network models and serves with its LSTM implementation using the Keras API [52].

Furthermore, the packages NTLK [49], Scikit-learn [50] and Gensim [51] are applied for the preprocessing and encoding of textual data. NTLK is used to realize the tokenization and word lemmatization, whereas the implementation of the text models is supported by the Scikit-learn (Bag of Words, Bag of N-Gram) and Gensim (LDA, Paragraph Vector) packages.

5.2 Model Implementation

The interface of the text-aware process prediction model (tapp) is realized through a class `TappModel`, which implements the functions `fit()`, `predict()` and `evaluate()`, that can be used to fit the model to an event log with historical data, compute prediction for an event log with uncompleted traces and evaluate the performance of the prediction model. The `TappModel` manages the underlying LSTM model and can be configured with the number of shared and specialized layers as well as the dimension of the hidden neurons per LSTM layer.

The encoding of traces as described in Section 4.2, 4.3 and 4.4 is computed in the `LogEncoder` module, which is controlled by the `TappModel`. The `LogEncoder` is also fitted to the historical log and can transform (prefix) traces to the encoded matrix format via functions `fit()` and `transform()`. It is configured with one out of the four text encoding models described in Section 4.4 (`BowTextEncoder`, `BongTextEncoder`, `LDATextEncoder` and `PVTextEncoder`), that realizes the vectorization of textual data. Just like the `LogEncoder`, the text encoding models offers the functions `fit()` and `transform()` adapt to the historical data and compute the text encoding.

During the fitting phase of the `LogEncoder`, all possible values for categorical attribute are indexed and the parameters for the normalization of numerical attributes are computed. With regard to the text encoding model, the indexed vocabulary of all words (after preprocessing) in the historical event log is built during fitting. In case of the Paragraph Vector model, the encoding network is trained with the training corpus.

The text-aware process prediction model can be used in any business process monitoring software to provide the prediction capabilities using the methods of the `TappModel`. In order to evaluate the performance of the model, an experiment using the `Evaluation` module is performed, which is described in-depth in Chapter 6. The module reads an event log using PM4Py, divides it into a training and test log and measures the prediction performance of differently configured instances of the text-aware process prediction model. An overview of the different components of the implementation is shown in Figure 5.1.

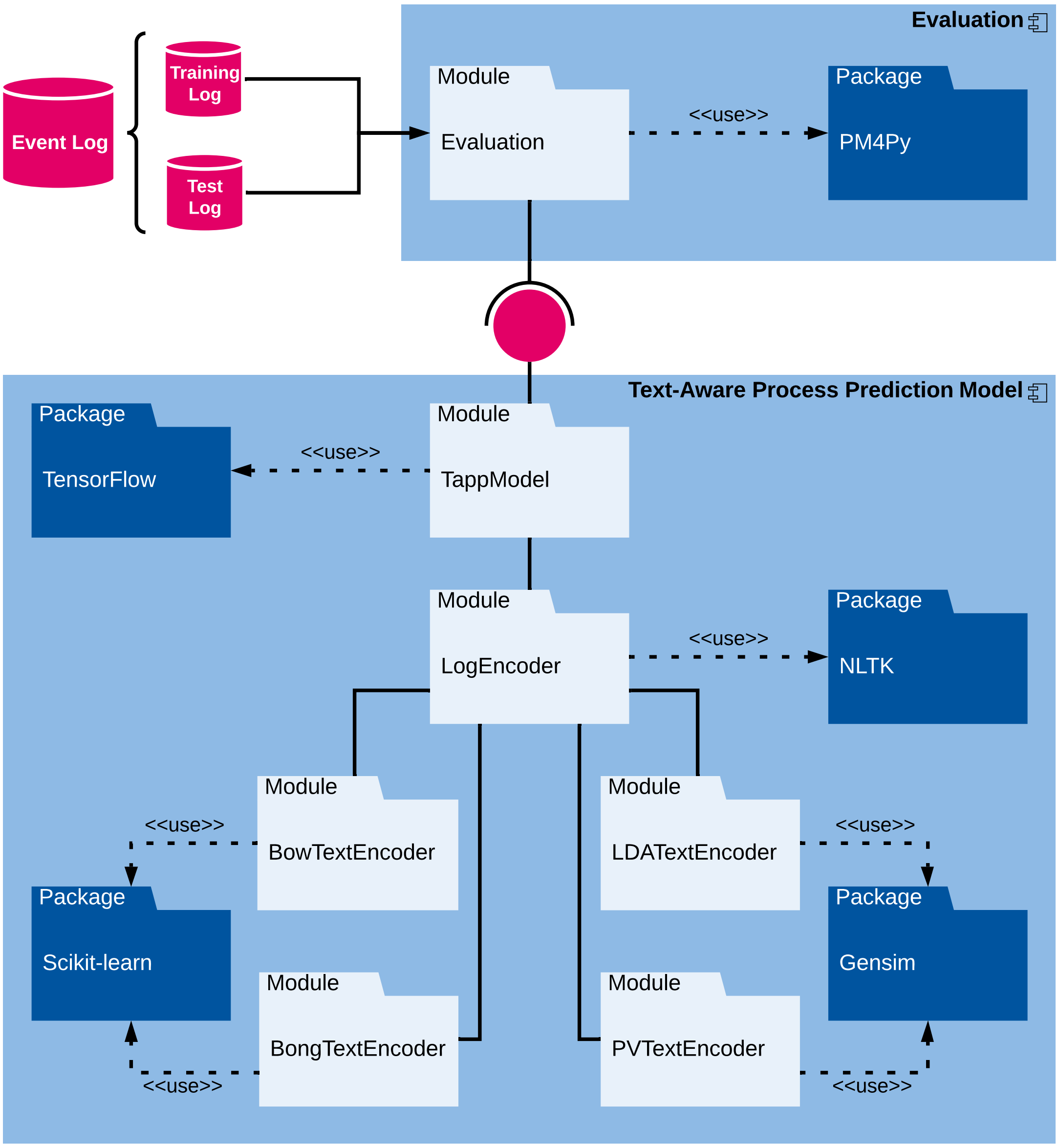


Figure 5.1: Component diagram of the implementation.

Chapter 6

Evaluation

In this chapter, the performance of the text-aware process prediction model is evaluated based on real-life event data. First, the data sets and evaluation method are described. Then, the performance of differently parameterized models on the data sets is evaluated and analyzed in-depth.

6.1 Data Sets

The prediction model is evaluated on real-world and simulated event logs, which are described in the following. An overview of the key properties of the logs is summarized in Table 6.1.

Job Application (Simulated) This event log describes a simple job application process. First, the applicant submits an application in the system. Then, a curriculum vitae (CV) and a cover letter are uploaded in an random order, where the CV is uploaded first with 75% probability. When both documents have been received, the applicant is either directly rejected or invited to an interview. After the interview a decision is made and the applicant gets an job offer or is rejected. The CV contains text information about the skills of the applicant. The probability for an interview invitation is exactly the relative share of technology related skills. After the interview, an email is sent to the applicant with the final decision. After that, the applicant gets an job offer or is rejected. Furthermore, noise is added by introducing a 3% probability that the applicant will stop the process after every event. The time of each event is determined by sampling from a normal distribution, which is unique per activity. The text information in the CV and the email is generated by sampling multiple times from sets of sentences. For example, if an applicant gets a job offer, the generated email contains text fragments from typical job offer emails. With this text generation mechanism, all texts in the event log are unique, but the words and sentences in the texts correlate with the path of the corresponding case.

werk.nl (Real-world) This event log describes customer's journeys of the Employee Insurance Agency commissioned by the Dutch Ministry of Social Affairs and Employment. The log is aggregated from two anonymized data sets, that were provided in the BPI Challenge 2016 [53], containing click data of logged in customers from the official website werk.nl and phone call data from the call center. Both data sets are joined based on the customer ID to derived a detailed view on customer contacts. For each phone call the costumer's question is available as a text attribute in English. In addition, the customer's age (grouped) and gender is considered as additional attributes. The event log is filtered to remove outlier activities (threshold <0.5%) and infrequent trace variants (2 or less traces with the same variant). After prepossessing, 18 distinct activities and 1001 trace variants remain, such that the process can be described as quite unstructured.

Help Desk (Real-world)

Data Set	**Job Application**	**werk.nl**	**Help Desk**
Number of cases	20 000	15 001	
Number of trace variants	19	1 001	
Number of events	132 268	55 220	
Events per case (mean)	6.6134	3.681	
Median case duration (days)	2.4661	0.224	
Mean case duration (days)	2.2108	0.713	
Number of activities	9	18	
Number of words (pre filtering)	1 774 587	247 010	
Number of words (post filtering)	760 504	98 915	
Vocabulary size (pre filtering)	131	1 203	
Vocabulary size (post filtering)	91	815	

Table 6.1: Overview of evaluated data sets.

6.2 Evaluation Method

Each event log is evaluated in a consistent procedure. In the first step, the event log is separated into a training and test log. The training log consists of the first 2/3 chronologically ordered traces and is used to fit the prediction model to the historical data. The remaining 1/3 of traces are used to measure the prediction performance. For each trace σ in the test log, all prefixes $hd^k(\sigma)$ of length $2 \leq k \leq |\sigma| - 1$ are considered as instances for prediction. For prefix traces with only one event, predictions seem to be less stable as also observed in [34] and are therefore not included for the metric computation. The LSTM network is trained with at most 100 epochs and the learning rate is initialized with 0.001. During the training of the LSTM model, 20% of the training log is used for validation: If the error on the validation log is not decreasing anymore for 5 epochs, the training rate is reduced, and if the error is not decreased for 10 epochs, the training is stopped in order to avoid overfitting. Furthermore, the LSTM layers use dropout [54] of 20% during training as an additional measure against overfitting.

For classification (i.e. categorical prediction) task, like next event and outcome prediction, the accuracy is utilized as metric. The accuracy is computed as the number of correct predictions t divided by the total number of predictions n, i.e.

$$\text{accuracy} = \frac{t}{n} = \frac{\#\text{ correct predictions}}{\#\text{ total predictions}} \in [0, 1].$$

For regression tasks, like the next event time and the case cycle time predictions, the mean absolute error (MAE) is computed to measure the prediction performance. The mean absolute error indicates the average absolute difference between the predicted value $\hat{y}$ and the true value y, precisely

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i| \in [0, \infty).$$

This error metric is favored, since it gives a more intuitive interpretation and is less sensitive to outliers compared to similar metrics like the mean squared error (MSE). An accuracy of 1 and a MAE of 0 would be the most desirable.

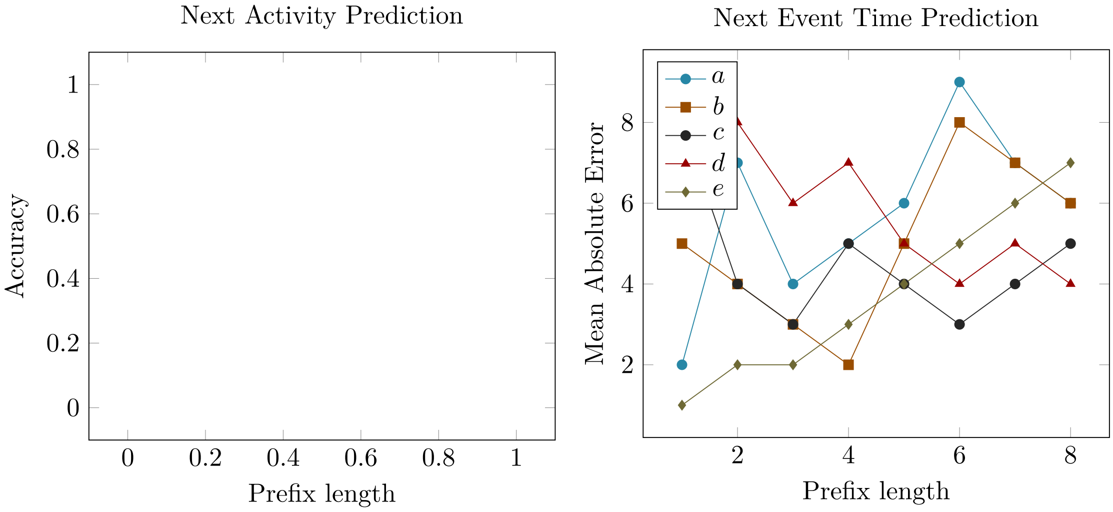
The prediction model is evaluated with each presented text model and different encoding lengths for the text. For the Bag of Words and Bag of N-Gram model the encoding length is adjusted by only considering the most frequent terms for the vocabulary. The encoding length of the non-textual data depends on the considered attributes and their number of the distinct values in the event logs. The Bag of N-Gram model is used with bigrams ($n = 2$). The Paragraph Vector model is trained for 15 epochs.

6.3 Next Event Prediction

		Job Application		**werk.nl**		**Help Desk**	
Text Model	Encoding Length	Accuracy	MAE	Accuracy	MAE	Accuracy	MAE
Text-Aware Process Prediction (LSTM)							
BoW	50			0.5274	0.0778		
BoW	100			0.5293	0.0776		
BoW	500			0.5229	0.0785		
BoNG	50			0.5288	0.0788		
BoNG	100			0.5242	0.0798		
BoNG	500			0.5214	0.0782		
PV	10			0.5153	0.0785		
PV	20			0.5206	0.0789		
PV	100			0.5199	0.0785		
LDA	10			0.5244	0.0786		
LDA	20			0.5214	0.0782		
LDA	100			0.5271	0.0788		
LSTM baseline							
-	0			0.5128	0.0785		

Table 6.2: Experimental results for the next event prediction.

6.4 Outcome and Case Cycle Time Prediction



(a) A subfigure

(b) A subfigure

Figure 6.1: Next activity prediction.

Table 6.3: Experimental results for the case cycle time prediction.

Chapter 7

Conclusion

7.1 Results and Achievements

7.2 Limitations and Future Research

The text-aware process prediction model is able to take advantage of textual data in predictive business process monitoring and can improve the prediction quality by exploiting correlations in (textual) data. The generalizability of the evaluation in Chapter 6 is limited due the small amount of evaluated data sets. To further validate the approach, more event logs with textual data need to evaluated. However, the data in common use cases like help desk or hospital processes is maintained under high privacy regulations. Since textual data can not easily be anonymized for evaluation and is highly sensitive, the data acquisition remains challenging.

In certain contexts, a high and reliable prediction performance is not sufficient as the interpretability of prediction model is necessary. LSTM-based methods are usually unable to deliver insights about the construction of the prediction and the influence of individual feature variables. The frequently observed trade-off between prediction performance and interpretability in machine learning, is also recognizable in process prediction [55]. While in this contribution the prediction performance has been prioritized, interpretability text-aware prediction models could be viable. Nevertheless, the utilization of textual data is an additional barrier and current interpretable methods based on process models can not be naturally extended for this purpose.

Another for the most part unsolved problem is to identify causality in processes. In order to derive predictions for processes, correlations in the event data are sufficient. But when predictive methods are applied to support decision making, it is important to identify the main forces that really *influence* the future path of the process. The ascertainment of causality is a significant harder problem, since it requires a much deeper understanding of the individual process. Therefore, tailor made methods might be necessary, that are specific to the field of use. Gained insights could then be utilized to improve process prediction by not only considering event data, but also background knowledge of the process.

Bibliography

[1] Jacquelyn Bulao. How much data is created every day in 2020?, 2020. URL `https://techjury.net/blog/how-much-data-is-created-every-day/`. Accessed = 2020-07-31.

[2] Wil M. P. van der Aalst. Process-aware information systems: Lessons to be learned from process mining. *Trans. Petri Nets Other Model. Concurr.*, 2:1–26, 2009.

[3] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[4] Ryan Browne. How three friends turned a college project into a $2.5 billion software unicorn, 2019. URL `https://www.cnbc.com/2019/11/21/celonis-raises-290m-series-c-funding-round-at-2point5b-valuation.html`. Accessed = 2020-04-20.

[5] Wil M. P. van der Aalst. Process mining and simulation: a match made in heaven! In *Proceedings of the 50th Computer Simulation Conference, SummerSim 2018, Bordeaux, France, July 09-12, 2018*, pages 4:1–4:12. ACM, 2018.

[6] Seymour Geisser. *Predictive inference: An introduction*. Chapman and Hall, 1993.

[7] Wil M. P. van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, R. P. Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos C. A. M. Buijs, Andrea Burattin, Josep Carmona, Malú Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano de Leoni, Pavlos Delias, Boudewijn F. van Dongen, Marlon Dumas, Schahram Dustdar, Dirk Fahland, Diogo R. Ferreira, Walid Gaaloul, Frank van Geffen, Sukriti Goel, Christian W. Günther, Antonella Guzzo, Paul Harmon, Arthur H. M. ter Hofstede, John Hoogland, Jon Espen Ingvaldsen, Koki Kato, Rudolf Kuhn, Akhil Kumar, Marcello La Rosa, Fabrizio Maria Maggi, Donato Malerba, R. S. Mans, Alberto Manuel, Martin McCreesh, Paola Mello, Jan Mendling, Marco Montali, Hamid R. Motahari Nezhad, Michael zur Muehlen, Jorge Munoz-Gama, Luigi Pontieri, Joel Ribeiro, Anne Rozinat, Hugo Seguel Pérez, Ricardo Seguel Pérez, Marcos Sepúlveda, Jim Sinur, Pnina Soffer, Minseok Song, Alessandro Sperduti, Giovanni Stilo, Casper Stoel, Keith D. Swenson, Maurizio Talamo, Wei Tan, Chris Turner, Jan Vanthienen, George Varvaressos, Eric Verbeek, Marc Verdonk, Roberto Vigo, Jianmin Wang, Barbara Weber, Matthias Weidlich, Ton Weijters, Lijie Wen, Michael Westergaard, and Moe Thandar Wynn. Process mining manifesto. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand,*

France, August 29, 2011, Revised Selected Papers, Part I, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer, 2011.

[8] Eric Verbeek, Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. XES tools. In Pnina Soffer and Erik Proper, editors, *Proceedings of the CAiSE Forum 2010, Hammamet, Tunisia, June 9-11, 2010*, volume 592 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[9] Fabian Veit, Jerome Geyer-Klingeberg, Julian Madrzak, Manuel Haug, and Jan Thomson. The proactive insights engine: Process mining meets machine learning and artificial intelligence. In Robert Clarisó, Henrik Leopold, Jan Mendling, Wil M. P. van der Aalst, Akhil Kumar, Brian T. Pentland, and Mathias Weske, editors, *Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Modeling (BPM 2017), Barcelona, Spain, September 13, 2017*, volume 1920 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

[10] Maikel L. van Eck, Xixi Lu, Sander J. J. Leemans, and Wil M. P. van der Aalst. PM ˆ2 : A process mining project methodology. In Jelena Zdravkovic, Marite Kirikova, and Paul Johannesson, editors, *Advanced Information Systems Engineering - 27th International Conference, CAiSE 2015, Stockholm, Sweden, June 8-12, 2015, Proceedings*, volume 9097 of *Lecture Notes in Computer Science*, pages 297–313. Springer, 2015.

[11] Rada Mihalcea. *The Text Mining Handbook: Advanced Approaches to Analyzing Unstructured Data* ronen feldman and james sanger (bar-ilan university and ABS ventures) cambridge, england: Cambridge university press, 2007, xii+410 pp; hardbound, ISBN 0-521-83657-3. *Comput. Linguistics*, 34(1):125–127, 2008.

[12] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[13] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1188–1196. JMLR.org, 2014.

[14] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[15] Dan Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2009.

[16] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach, Third International Edition*. Pearson Education, 2010.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[18] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1310–1318. JMLR.org, 2013.

[19] Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

[20] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE Trans. Neural Networks Learn. Syst.*, 28(10):2222–2232, 2017.

[21] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[22] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014.

[23] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural machine translation architectures. *CoRR*, abs/1703.03906, 2017.

[24] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[25] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[26] Chris Olah. Understanding LSTM networks, 2015. URL `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Accessed = 2020-06-02.

[27] Boudewijn F. van Dongen, R. A. Crooy, and Wil M. P. van der Aalst. Cycle time prediction: When will this case finally be finished? In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, November 9-14, 2008, Proceedings, Part I*, volume 5331 of *Lecture Notes in Computer Science*, pages 319–336. Springer, 2008.

[28] Wil M. P. van der Aalst, M. H. Schonenberg, and Minseok Song. Time prediction based on process mining. *Inf. Syst.*, 36(2):450–475, 2011.

[29] Suraj Pandey, Surya Nepal, and Shiping Chen. A test-bed for the evaluation of business process prediction techniques. In Dimitrios Georgakopoulos and James B. D. Joshi, editors, *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 2011, Orlando, FL, USA, 15-18 October, 2011*, pages 382–391. ICST / IEEE, 2011.

[30] Andreas Rogge-Solti and Mathias Weske. Prediction of remaining service execution

time using stochastic petri nets with arbitrary firing delays. In Samik Basu, Cesare Pautasso, Liang Zhang, and Xiang Fu, editors, *Service-Oriented Computing - 11th International Conference, ICSOC 2013, Berlin, Germany, December 2-5, 2013, Proceedings*, volume 8274 of *Lecture Notes in Computer Science*, pages 389–403. Springer, 2013.

[31] Michelangelo Ceci, Pasqua Fabiana Lanotte, Fabio Fumarola, Dario Pietro Cavallo, and Donato Malerba. Completion time and next activity prediction of processes using sequential pattern mining. In Saso Dzeroski, Pance Panov, Dragi Kocev, and Ljupco Todorovski, editors, *Discovery Science - 17th International Conference, DS 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*, volume 8777 of *Lecture Notes in Computer Science*, pages 49–61. Springer, 2014.

[32] Irene Teinemaa, Marlon Dumas, Fabrizio Maria Maggi, and Chiara Di Francescomarino. Predictive business process monitoring with structured and unstructured data. In Marcello La Rosa, Peter Loos, and Oscar Pastor, editors, *Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings*, volume 9850 of *Lecture Notes in Computer Science*, pages 401–417. Springer, 2016.

[33] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. A deep learning approach for predicting process behaviour at runtime. In Marlon Dumas and Marcelo Fantinato, editors, *Business Process Management Workshops - BPM 2016 International Workshops, Rio de Janeiro, Brazil, September 19, 2016, Revised Papers*, volume 281 of *Lecture Notes in Business Information Processing*, pages 327–338, 2016.

[34] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. Predictive business process monitoring with LSTM neural networks. In Eric Dubois and Klaus Pohl, editors, *Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings*, volume 10253 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2017.

[35] Nicolò Navarin, Beatrice Vincenzi, Mirko Polato, and Alessandro Sperduti. LSTM networks for data-aware remaining time prediction of business process instances. In *2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017, Honolulu, HI, USA, November 27 - Dec. 1, 2017*, pages 1–7. IEEE, 2017.

[36] Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano de Leoni. Time and activity sequence prediction of business process instances. *Computing*, 100 (9):1005–1031, 2018.

[37] Irene Teinemaa, Marlon Dumas, Marcello La Rosa, and Fabrizio Maria Maggi. Outcome-oriented predictive process monitoring: Review and benchmark. *ACM Trans. Knowl. Discov. Data*, 13(2):17:1–17:57, 2019.

[38] Gyunam Park and Minseok Song. Prediction-based resource allocation using LSTM and minimum cost and maximum flow algorithm. In *International Conference on Process Mining, ICPM 2019, Aachen, Germany, June 24-26, 2019*, pages 121–128. IEEE, 2019.

[39] Ramzan Talib, Muhammad Kashif, Shaeela Ayesha, and Fakeeha Fatima. Text min-

ing: Techniques, applications and issues. *International Journal of Advanced Computer Science and Applications*, 7, 2016.

[40] R. P. Jagadeesh Chandra Bose, Wil M. P. van der Aalst, Indre Zliobaite, and Mykola Pechenizkiy. Dealing with concept drifts in process mining. *IEEE Trans. Neural Networks Learn. Syst.*, 25(1):154–171, 2014.

[41] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, 2003.

[42] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[43] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[44] Mahidhar Dwarampudi and N. V. Subba Reddy. Effects of padding on lstms and cnns. *CoRR*, abs/1903.07288, 2019.

[45] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

[46] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual.* CreateSpace, Scotts Valley, CA, 2009.

[47] Alessandro Berti, Sebastiaan J. van Zelst, and Wil M. P. van der Aalst. Process mining for python (pm4py): Bridging the gap between process- and data science. *CoRR*, abs/1905.06169, 2019.

[48] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[49] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python.* O'Reilly, 2009.

[50] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011.

[51] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50. ELRA, 2010.

[52] François Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[53] Marcus Dees and Boudewijn F. van Dongen. BPI Challenge 2016, 2016. URL `https://data.4tu.nl/collections/BPI_Challenge_2016/5065538`. Accessed = 2020-09-12.

[54] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[55] Niek Tax, Irene Teinemaa, and Sebastiaan J. van Zelst. An interdisciplinary comparison of sequence modeling methods for next-element prediction. *CoRR*, abs/1811.00062, 2018.