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. 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

Abstract

Contents

Chapter 1

Introduction

1.1 Motivation

The rapid growth of data generated by large-scale information systems leads to new opportunities for society and businesses. By the end of 2020, the total amount 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, analyses or predictions.

A remarkable subset of this data is described as *event data*, which is generated by *process-aware information systems*, which manage, execute and monitor 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 exploding.

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 and analysis of business processes, in contrast to traditional offline and project-based approaches, that are not integrated with the remaining IT infrastructure of a company. The integrated and continuous application of process mining is realized by a *business process monitoring system*, which are a key success factor for many organizations, since they 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 connecting between the information system and the monitoring system, that provides the analytical capabilities. Figure 1.1 visualizes such an infrastructure and the connection between the systems and the internal and external process stakeholders.

Traditional process mining tends to be backward-looking [5], i.e. the focus relies on analyzing and understanding past executions of a process rather than providing value for running process instances in form of predictions or recommendations. Businesses can develop a

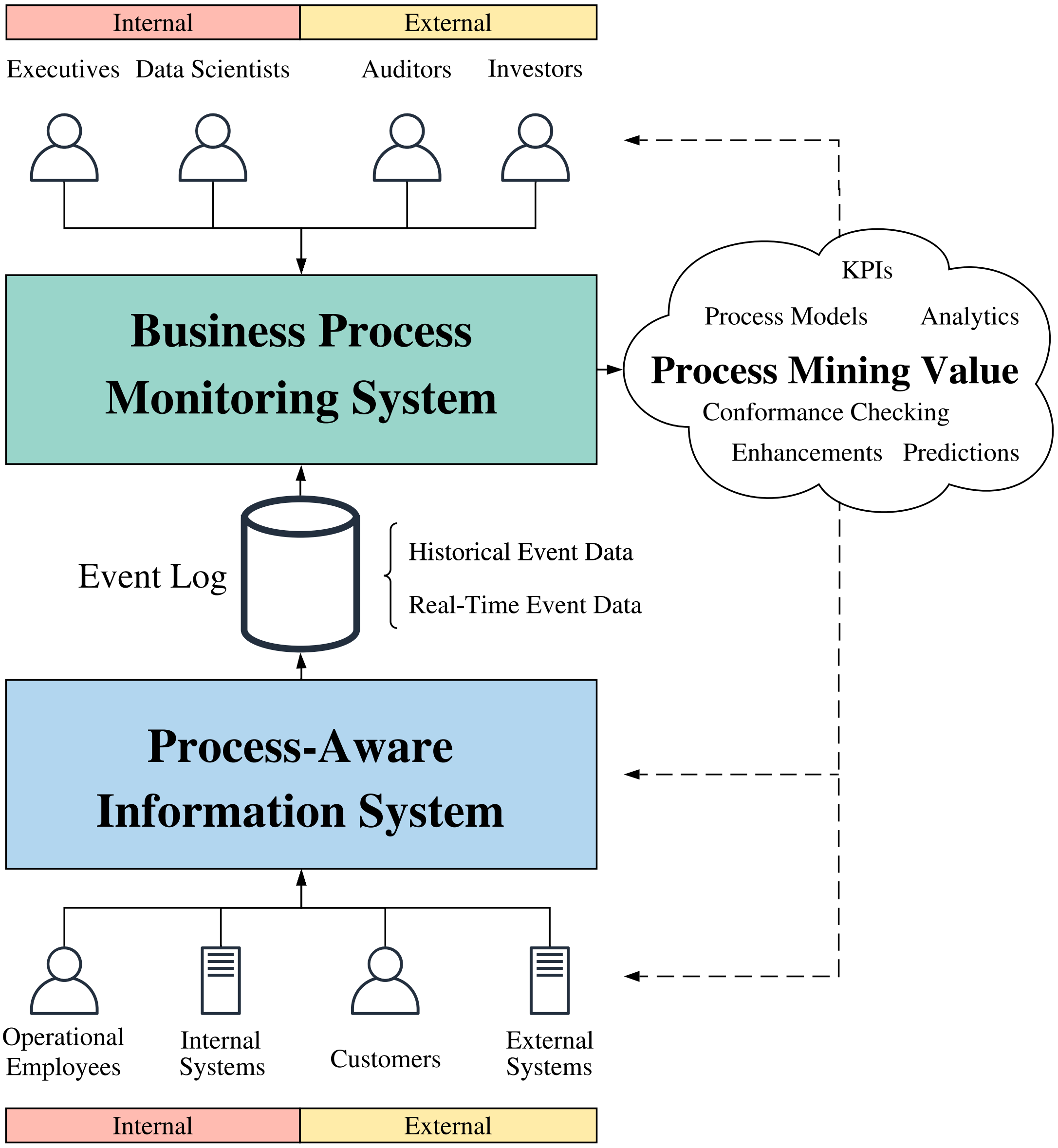

Figure 1.1: Business process monitoring allows to continuously apply process mining techniques in an automated fashion in order to generate value for internal and external process stakeholders.

competitive advantage, if their process mining solution offers predictive capabilities, that allow to predict the future of a running process instance. For example, if it is known beforehand, that a running process instance will probably exceed its deadline, measures can be initiated before damage occurs. Therefore, including the forward-perspective is crucial for a 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 the targets they predict. Many approaches derive the prediction purely from the control flow of process instance ignoring additional data attributes in the event log. Especially, almost no approach is able to consider textual data for process prediction. In addition, most of the prediction methods focus on a single prediction dimension only, for example they just predict the remaining time until a process instance is finished. But depending on the context also 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/declined that can be predicted.

Precisely, given an event log with past executions of a process and a running (i.e. not completed) process instance, we would like to answer the following questions:

- What will happen next?
- When will it happen?
- What is the most likely future path of the instance?
- When will the instance finish?
- What is the outcome of the instance?

...

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 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. Since most current approach are not able to handle textual data, we would like to know to what extend textual data can improve the quality of process prediction. Furthermore, we want to evaluate different design choices and text models for text-aware process prediction and discuss potential trade-offs.

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

> **RQ1** To what extend can the utilization of textual data improve the performance of process prediction?

RQ2 How does the choice of the text model and other parameters influence the prediction results?

RQ3 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 notations, definitions and concepts used in this contribution are introduced. This includes an introduction to process mining, text mining, supervised learning and LSTM neural networks. Chapter 3 summarizes relevant scientific contributions which focus on the problem of prediction in process mining and gives an overview of already available methods and their capabilities. In Chapter 4 the novel text-aware process prediction model as 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, in Chapter 7 the conclusion is given by wrapping up the key results as well as 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, long short-term memory networks are presented. Furthermore, necessary formal definitions and notations that are introduced.

2.1 Processes and Process Mining

A *business process* is a collection of activities that are performed in a specific order to archive a goal [6]. 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. A case can be for example a patient 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 trinity of case, activity and timestamp is called *event*. An event can have more attributes, for example resource, costs or transactional information.

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) [7], which can be extracted from databases. A table-based representation of an artificial event log about patient treatment in a hospital can be seen in Table 2.1.

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 [8].

Traditionally, process mining is divided into a set of subdisciplines mainly process discovery, conformance checking, process enhancement and process analytics [9]. Process discovery aims to generate process models out of event data in order to understand 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.

ID	**Activity**	**Timestamp**	**Resource**	**Cost**	**Comment**
0	Register patient	01.02.2020:14.12	SYSTEM	0	-
	Consultation	01.02.2020:14.34	John Brown, MD	24.32	The patient reports persistent nausea.
	Blood test	01.02.2020:15.12	Kim Smith	14.23	Tests: Complete blood count
	Evaluate test result	01.02.2020:16.35	John Brown, MD	38.67	No abnormalities in the complete blood count.
	Release patient	01.02.2020:17.24	SYSTEM	0	-
1	Register patient	02.02.2020:08.20	SYSTEM	0	-
	Consultation	02.02.2020:14.12	Jana Simpson, MD	24.32	Noticeable tachycardia. No chro are known.
	MRI	02.02.2020:14.12	Sara Taylor, MD	352.87	-
	Release patient	02.02.2020:14.12	SYSTEM	0	-
2	Register patient	02.02.2020:09.08	SYSTEM	0	-
	Consultation	02.02.2020:09.14	Jana Simpson, MD	24.32	The patient has severe leg injurie due to a motorcycle accident.
	Patient hospitalized	02.02.2020:09.20	Mike Johnson	130.37	-
...	...	...	...	...	...

Table 2.1: Artifical event log of patient treatment in a hospital

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 2.1 (Sequence)**.** A *sequence* of length $n \in \mathbb{N}_0$ over a set A is an ordered collection of elements defined by 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 $1 \leq i \leq n$. 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. The set of all possible sequences over A is given with $A^* = \bigcup\limits_{i \in \mathbb{N}_0} A^i$.

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 $1 \leq i \leq n$. 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 $0 \leq k \leq n$.

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

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

2.3 Events, Traces, Event Logs

Definition 2.2 (Event)**.** An *event* is defined by tuple $e = (a, c, t, d_1, \dots, d_m) \in \mathcal{C} \times \mathcal{A} \times \mathcal{T} \times \mathcal{D}_1 \times \cdots \times \mathcal{D}_m = \mathcal{E}$ 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 $\mathcal{C} \times \mathcal{A} \times \mathcal{T}$.

Each attribute $d \in \mathcal{D}$ of an event (including activity, timestamp and case ID) can be accessed by a 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 $\mathcal{D}_i = V^*$ for $1 \leq i \leq m$ and some fixed vocabulary V.

Definition 2.3 (Trace)**.** A *trace* is a finite and non-empty 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 $1 \leq i < j \leq |\sigma|$.

By lifting the projection functions to sequences a trace can be transformed into a sequence of attributes by applying the projection function to the trace. For example, $\pi_{\mathcal{A}}(\sigma)$ gives the sequence of the activities of the events in σ.

Definition 2.4 (Event log)**.** An *event log* or *trace log* $\mathbb{L} = \{\sigma_1, \sigma_2, \dots, \sigma_k\}$ 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

Text mining describes all techniques to generate value out of unstructured or semi-structured textual data. It combines concepts of natural language processing, machine learning and data mining [10]. 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 e-mail, where text information is assigned to sender, subject, message etc. In our setting, a document $d \in V^*$ (i.e. textual data) is always a sequence of words from a fixed vocabulary V. 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 text data that can be interpreted by a computer, a text model has to be build using the text corpus. Popular text models are Bag-of-words, Bag-of-n-gram, Paragraph vector (a.k.a. Doc2Vec) [11] and Latent Dirichlet Allocation [12]. 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 [13].

2.5 Supervised Learning

In *supervised learning* an unknown function is learned (i.e. approximated) from a set of example input-output pairs [14]. In contrast, *unsupervised learning* does not require examples pattern and is about finding pattern in the data. An input instance is usually described by a set of *feature variables* X and the output is defined by a *target variable* y. If the target variable y is continuous, we refer to this as a regression problem, if however it is discrete variable with a finite range of values, the learning problem is called classification problem. Given a *training set* of input-output pairs $\{(X_1, y_1), (X_2, y_2), \dots, (X_m, y_m)\}$, that were generated from an unknown function $y = f(X)$, the goal is to approximate a hypothesis function $h(X)$, which is close to 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 generalizes 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 not reliable on unseen data, the hypothesis usually *overfits* the training data. In this case, the model complexity, i.e. the number of parameters is higher than justified by the true function. 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 prediction. Therefore, the prediction accuracy is always limited by the randomness of the true distribution.

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 [15]. This approach addresses the well-known vanishing and exploding gradient problem [16] of traditional recurrent neural networks by introducing more complex LSTM cells as hidden units. The proposed architecture has been improved several times [17] [18] and 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 [19].

Gated recurrent units (GRU) [20] 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 [21].

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 [22]. These networks can learn and compute complex functions in supervised learning settings, where input and output pattern 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 [23].

Recurrent neural networks extend traditional feed forward 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 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 $\boldsymbol{h}_t$.

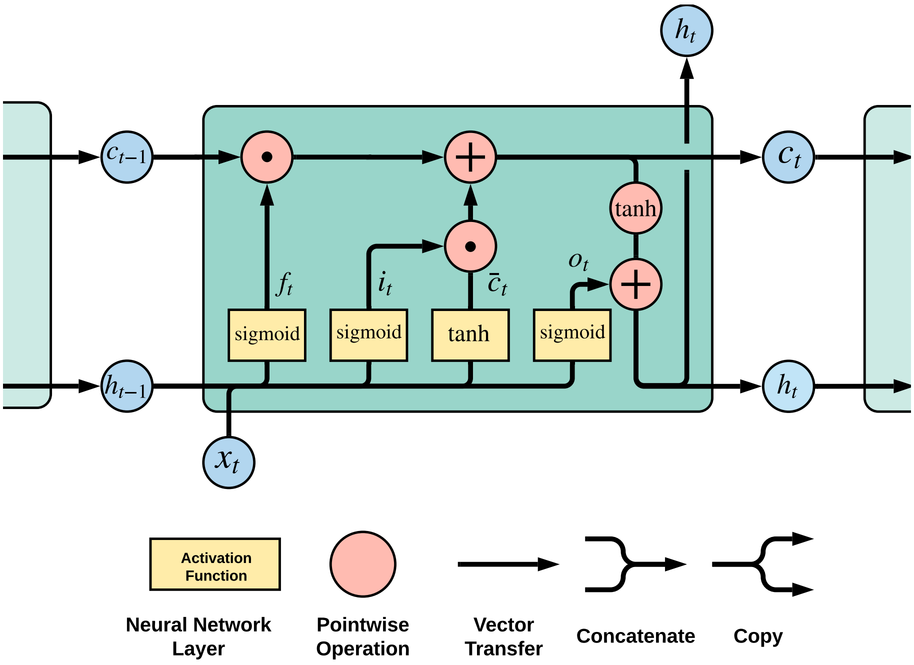

Figure 2.1: An LSTM module with four sublayers that manipulate the cell state and compute the module's output. Graphic adapted from [24].

The input vector $\boldsymbol{x}_t$ is concatenated with the previous hidden output $\boldsymbol{h}_{t-1}$ and fed 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 $\bar{\boldsymbol{c}}_t$) and what the output of the layer will be (output gate $\boldsymbol{o}_t$ leading to $\boldsymbol{h}_t$). The sublayer apply sigmoid$(x) = \frac{1}{1+\exp(-x)}$

or $\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ activation functions elementwise, leading to the following equations:

$$\boldsymbol{f}_t = \mathrm{sigmoid}(\boldsymbol{W}_f \cdot (\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) + \boldsymbol{b}_f)$$

$$\boldsymbol{i}_t = \mathrm{sigmoid}(\boldsymbol{W}_i \cdot (\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) + \boldsymbol{b}_i)$$

$$\bar{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_c \cdot (\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) + \boldsymbol{b}_c)$$

$$\boldsymbol{o}_t = \mathrm{sigmoid}(\boldsymbol{W}_o \cdot (\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) + \boldsymbol{b}_o)$$

$\boldsymbol{W}_f$, $\boldsymbol{W}_i$, $\boldsymbol{W}_c$ 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 $\bar{\boldsymbol{c}}_t$, where the layer computations $\boldsymbol{f}_t$ and $\boldsymbol{i}_t$ determine the proportions by a pointwise multiplication ($\odot$) with the cell states.

$$\boldsymbol{c}_t = f_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \bar{\boldsymbol{c}}_t$$

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.

Chapter 3

Related Work

The prediction of the future of an process instance has been an important subfield in the process mining research, that aims 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 models or process models to construct a predictor that generalizes from a historical event log.

van Dongen et al. presented five different non-parametric regression predictors for forecasting the total cycle time of an unfinished case[25]. The estimates are based on activity occurrences, activity duration and attributes.

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 remaining time of case [26]. 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 remaining time of a case using the activity and timestamp data of an event log [27].

Rogge-Solti and Weske showed how a stochastic Petri net can be used to predict the remaining 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 a running traces to similar historical traces [29]. A decision tree is then used to predict the next activity and the remaining 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[30]. 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[31].

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[32].

The work by Navarin et al. adopts the idea of using an LSTM network [32] and extends the encoding to additional data attributes associated with each event[33] to predict the remaining time of an case.

Polato et al. presented a set of approaches that use support vector regression for remaining time prediction[34]. 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 [26] 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 [35].

Park and Song showed how LSTM-based predictions can be used to solve a resource allocation problem, leading to direct recommendations for process improvement [36].

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. [25]	2008	Regression	✓	✗	Remaining time
van der Aalst et al. [26]	2011	Transition system	✗	✗	Remaining time
Pandey et al. [27]	2011	Hidden Markov	✗	✗	Remaining time
Rogge-Solti and Weske [28]	2013	Stochastic Petri Net	✗	✗	Remaining time
Ceci et al. [29]	2014	Sequence Tree Decision Tree	✓	✗	Next activity Remaining time
Teinemaa et al. [30]	2016	Random Forest Logistic regression	✓	✓	Case outcome
Evermann et al. [31]	2016	LSTM	✗	✗	Next activity
Tax et al. [32]	2017	LSTM	✗	✗	Next activity Future path Next event time Remaining time
Navarin et al. [33]	2017	LSTM	✓	✗	Remaining time
Polato et al. [34]	2018	Transition system SVR Naive Bayes	✓	✗	Next activity Future path Remaining time
Park and Song [36]	2019	LSTM	✓	✗	Next activity Next event time
This contribution	2020	LSTM	✓	✓	Next activity Future path Next event time Remaining time Case outcome

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. A first approach has been presented in [30], where traces with text 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 control flow and additional numerical, categorical and textual data. 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 current processes give an competitive advantage.

4.1 Overview

The goal of the framework is to learn prediction functions $f_{\mathrm{a}}, f_{\mathrm{t}}, f_{\mathrm{o}}$ and f_{c}, that predict the next activity, timestamp, case outcome and case cycle time (time between first and last event) given any prefix $hd^k(\sigma)$ trace of a case. Precisely, the prediction functions should approximate the true future of the case, such that

$$\begin{aligned}
f_{\mathrm{a}}(hd^k(\sigma)) &= \begin{cases} \text{PROCESS END} & \text{if } |\sigma| = k \\ \pi_{\mathcal{A}}(\sigma(k+1)) & \text{else} \end{cases} \\
f_{\mathrm{t}}(hd^k(\sigma)) &= \begin{cases} \pi_{\mathcal{T}}(\sigma(k)) & \text{if } |\sigma| = k \\ \pi_{\mathcal{T}}(\sigma(k+1)) & \text{else} \end{cases} \\
f_{\mathrm{o}}(hd^k(\sigma)) &= \pi_{\mathcal{A}}(\sigma(|\sigma|)) \\
f_{\mathrm{c}}(hd^k(\sigma)) &= \pi_{\mathcal{T}}(\sigma(|\sigma|)) - \pi_{\mathcal{T}}(\sigma(1)).
\end{aligned}$$

The proposed framework 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 an event log $\mathbb{L} = \{\sigma_1, \dots, \sigma_l\}$ with historical 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 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 encoded based on a textual model. The text model is an exchangeable component and is fit to the text corpus, that is extracted from the text data in the event log.

Each encoded prefix sequence with its desired prediction target values corresponds to one training example for an LSTM network, that realizes the predictions. The target values of a prefix sequence are the activity and timestamp (relative to the case start) of the next event as well as the case outcome and case cycle time. The case outcome can be for example a binary label, ie. a label telling if the case is successful or has failed or if it has been approved or declined. In some applications the case outcome is defined by the final event of a case. For completed cases the next activity is an artificial "PROCESS END" activity with the same timestamp as the final event. 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 can be used to predict the next event, outcome and cycle time of new unseen and unfinished cases, that are currently monitored.

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, a set of encoding functions $enc_k \colon \mathcal{E}^k \to (\mathbb{R}^n)^k$ is realized by the encoding component, that encodes (prefix-)traces of length k to vector sequences of the same size, i.e. $enc_k(\sigma) = \langle \boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_k} \rangle$ with $\sigma = \langle e_1, e_2, \dots, e_k \rangle$. 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. $e_i \in \mathcal{C} \times \mathcal{A} \times \mathcal{T} \times \mathcal{D}_1^{\text{num}} \times \dots \times \mathcal{D}_r^{\text{num}} \times \mathcal{D}_1^{\text{cat}} \times \dots \times \mathcal{D}_s^{\text{cat}} \times \mathcal{D}_1^{\text{text}} \times \dots \times \mathcal{D}_u^{\text{text}}$. Each encoded event vector $\boldsymbol{x_i}$ is the concatenation of set of feature vectors, which are constructed from the event data.

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

The activity of the event is represented by vector $\boldsymbol{a_i}$ using *one-hot encoding*. Given the set of possible activities $\mathcal{A}$, an arbitrary but fixed ordering over is introduced with a bijective index function $index_{\mathcal{A}} \colon \mathcal{A} \to \{1, \dots, |\mathcal{A}|\}$. 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 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. 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

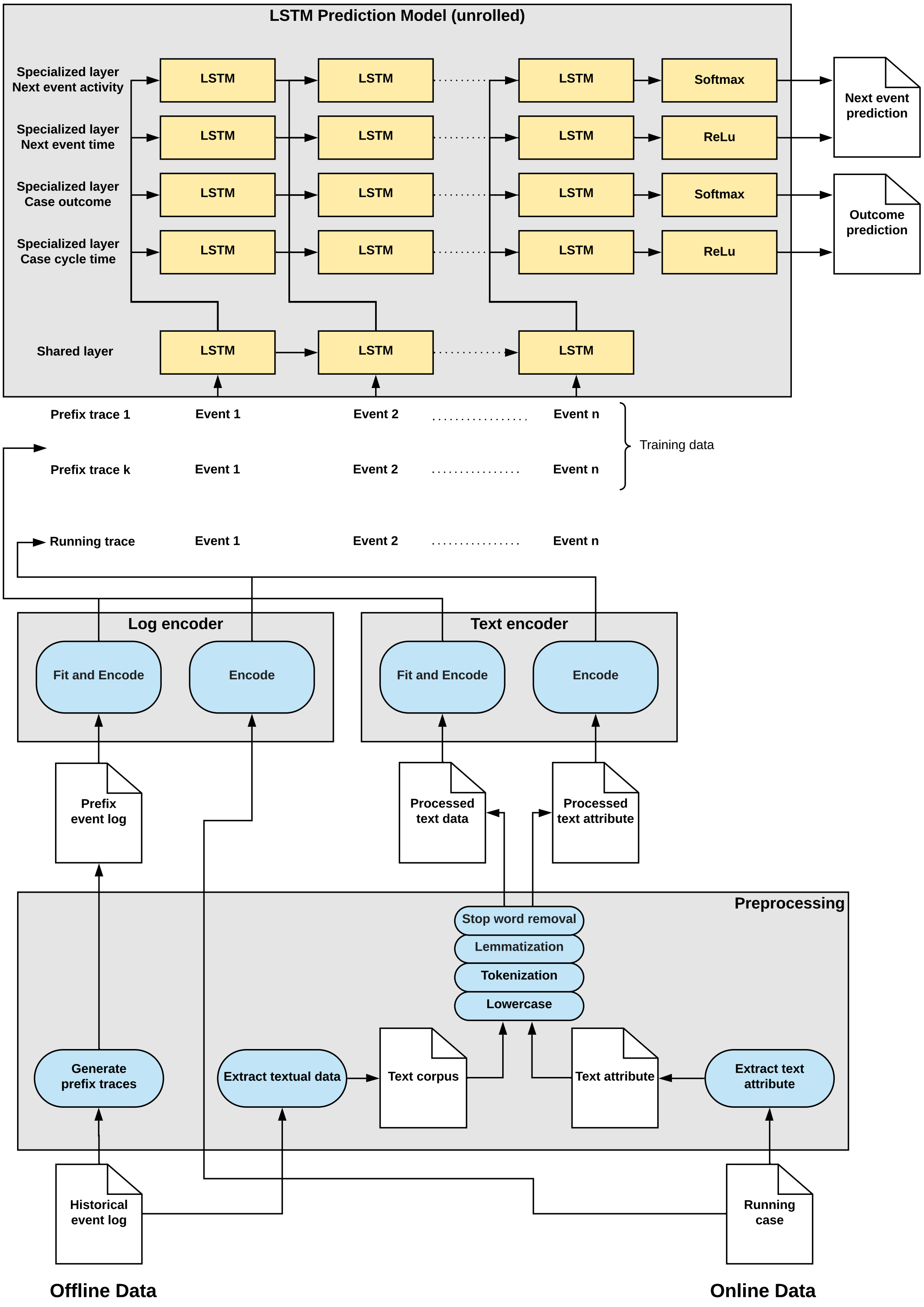

Figure 4.1: Overview of the prediction framework.

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

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

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

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.

Categorical attributes are encoded using one-hot-encoding in the same way as the activity, i.e. $d_{ij} = \mathbb{1}_{\mathcal{D}_j^{\text{cat}}}$. Textual attributes are vectorized via a dedicated text model, that is explained in Section 4.4.

All in all, the encoding results in vector of size

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

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 $\sigma = \langle e_1, \dots, e_k \rangle$ 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	Time since previous event
t_i^2	Time since case start
t_i^3	Time since first recorded event
t_i^4	Time since midnight
t_i^5	Time since last Monday
t_i^6	Time since last January 1 00:00

Table 4.2: Time-based features as part of event encoding for an event e_i.

Using the time features a set of time-dependent trends 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 in the process [37]. Most real-life 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 capture the time difference between the current event and midnight, the current event and the most recent monday and the current event and the beginning of the 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 usually has much more customers in summer, which can affect the process execution in many ways. Each feature $t_i^1, \ldots, t_i^6$ 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 representationu using a text model. Useful in that context means, that texts with similar semantic meanings should also have similar representations. The vector representation of text 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 by text encoding component. First, all text 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*. 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 that appear in the corpus and is indexed by an bijective index function $index_V \colon V \to \{1, 2, \ldots, |V|\}$. As text vectorization techniques rely on the Bag of Words, Bag of N-Grams, Paragraph Vector or Latent Dirichlet Allocation as a text model.

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. Letters are converted to lowercase
2. 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 in 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".

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

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.

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 [38]. 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 in the document indexed with i it.

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 score is the tfidf score of a word in a document. consul

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 tfidf scores can excluded from the vocabulary.

4.4.3 Bag of N-Grams

The *Bag of N-Grams* 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 tfidf 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 Latent Dirichlet Allocation

The Latent Dirichlet Allocation presented by Blei et al. in 2003 [12] is generative statistical text model, that represents documents as a mixture of a fixed number of topics, depending on the words of the document. Therefore, a document is encoded as a vector whose dimension is equal to the number of chosen topics and each component indicates the degree of affiliation between the document and the corresponding topic. The topics are are not generated manually, but by the model itself.

4.4.5 Paragraph Vector

The *Paragraph Vector* 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 [11]. The idea is inspired by the word embedding model presented by Bengio et al. [39], 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*)[40].

The core idea of the Paragraph Vector model is to extend the model in [39] in a way, that an additional document or paragraph vector, that is unique per document is trained together with the word vectors. 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

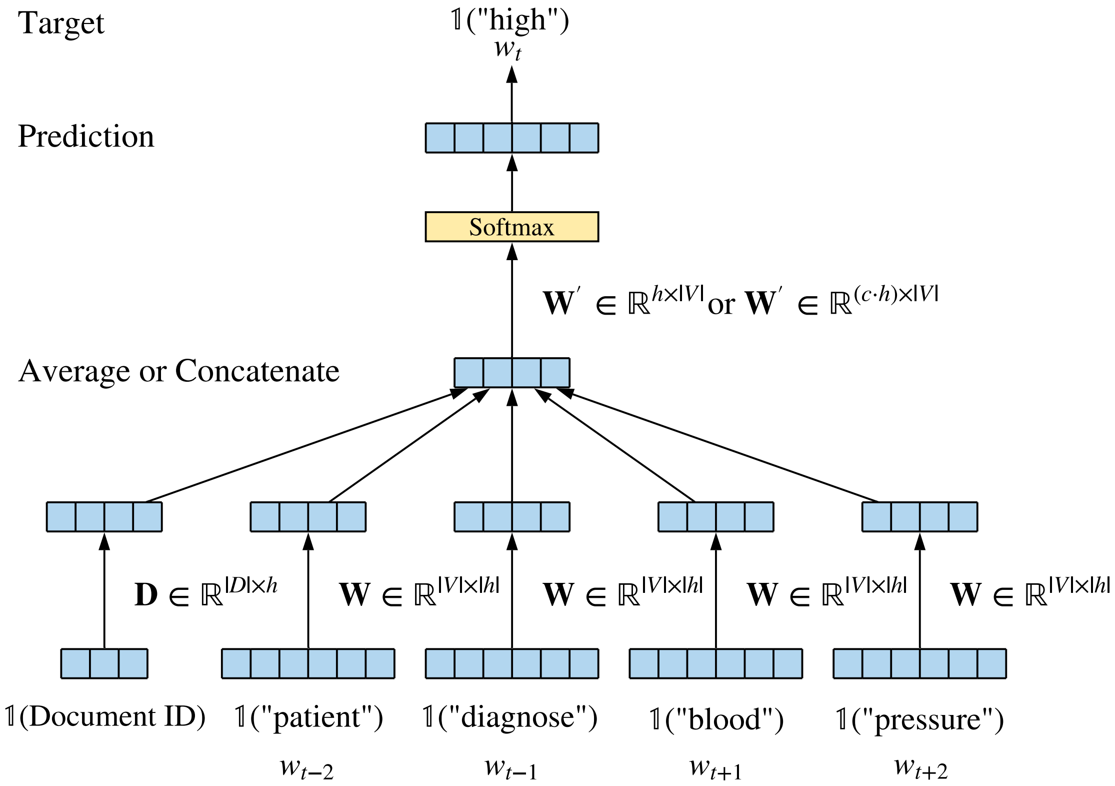
Target
𝟙("high")
w_t
Prediction
Softmax
$\mathbf{W}' \in \mathbb{R}^{h \times |V|}$ or $\mathbf{W}' \in \mathbb{R}^{(c \cdot h) \times |V|}$
Average or Concatenate
$\mathbf{D} \in \mathbb{R}^{|D| \times h}$
$\mathbf{W} \in \mathbb{R}^{|V| \times |h|}$
$\mathbf{W} \in \mathbb{R}^{|V| \times |h|}$
$\mathbf{W} \in \mathbb{R}^{|V| \times |h|}$
$\mathbf{W} \in \mathbb{R}^{|V| \times |h|}$
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 $\boldsymbol{W}$.

one-hot encoded words and a one-hot encoded document IDs 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.

4.5 Network Architecture and Training

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. The network consists of an input layer, an shared LSTM layer, an a specialized LSTM layer for each target, and an fully connected output layer for each target. Furthermore, layer normalization [41] is applied after each LSTM layer, which standardizes the hidden output in order to speed up the training convergence.

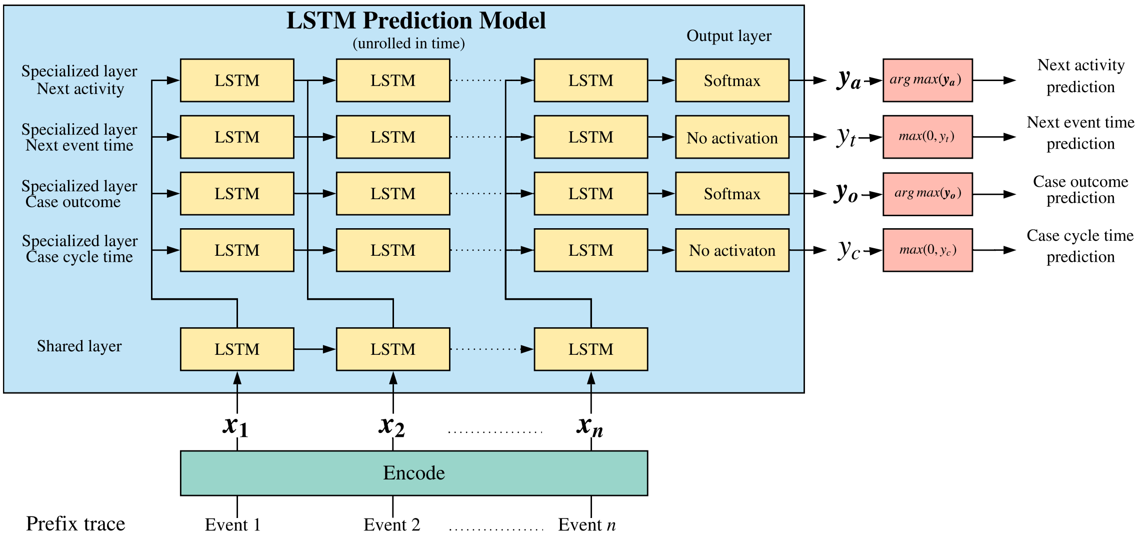

Figure 4.3: LSTM model to simultaneously predict the next activity ($\boldsymbol{y_a}$), next event time(y_t), case outcome ($\boldsymbol{y_o}$) and case cylce time (y_c) for an encoded prefix trace $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}$.

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 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

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

The training set of encoded prefix traces is represented by an 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 [42] with zeros vectors. Hence, a prefix trace of encoded events $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}$ is represented in the training set by a 2-dimensional matrix $(\boldsymbol{0}, \ldots, \boldsymbol{0}, \boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n})$, 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 by a backpropagation through time (BPTT) algorithm that updates the weights of the network using the update rules of the Adam optimizer with Nesterov momentum [43]. 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 $\text{CE} = -\sum_{i=1}^{k} y_i \cdot \log \hat{y}_i$.

4.6 Predictive Business Processing Monitoring

During online business process monitoring, predictions are realized by a forward-pass of the encoded running cases 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.

Chapter 5

Implementation

Chapter 6

Evaluation

6.1 Datasets and

6.2 Next Event Prediction

6.3 Remaining Time Prediction

6.4 Outcome Prediction

Chapter 7

Conclusion

Bibliography

[1] Jacquelyn Bulao. How much data is created every day in 2020?, 2020. URL https://web.archive.org/web/20200728232536/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. doi: 10.1007/978-3-642-00899-3_1. URL https://doi.org/10.1007/978-3-642-00899-3_1.

[3] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition.* Springer, 2016. ISBN 978-3-662-49850-7. doi: 10.1007/978-3-662-49851-4. URL https://doi.org/10.1007/978-3-662-49851-4.

[4] Ryan Browne. How three friends turned a college project into a $2.5 billion software unicorn, 2019. URL https://web.archive.org/web/20200125191917/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. URL http://dl.acm.org/citation.cfm?id=3275386.

[6] 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, Li-

jie 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. doi: 10.1007/978-3-642-28108-2_19. URL `https://doi.org/10.1007/978-3-642-28108-2_19`.

[7] 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. URL `http://ceur-ws.org/Vol-592/PaperDemo07.pdf`.

[8] 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. URL `http://ceur-ws.org/Vol-1920/BPM_2017_paper_192.pdf`.

[9] 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. doi: 10.1007/978-3-319-19069-3_19. URL `https://doi.org/10.1007/978-3-319-19069-3_19`.

[10] 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. doi: 10.1162/coli.2008.34.1.125. URL `https://doi.org/10.1162/coli.2008.34.1.125`.

[11] 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. URL `http://proceedings.mlr.press/v32/le14.html`.

[12] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003. URL `http://jmlr.org/papers/v3/blei03a.html`.

[13] 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. ISBN 9780135041963. URL `http://www.worldcat.org/oclc/315913020`.

[14] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach, Third International Edition.* Pearson Education, 2010. ISBN 978-0-13-207148-2. URL `http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0136042597,00.html`.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL `https://doi.org/10.1162/neco.1997.9.8.1735`.

[16] 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. URL `http://proceedings.mlr.press/v28/pascanu13.html`.

[17] Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000. doi: 10.1162/089976600300015015. URL `https://doi.org/10.1162/089976600300015015`.

[18] 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. doi: 10.1109/TNNLS.2016.2582924. URL `https://doi.org/10.1109/TNNLS.2016.2582924`.

[19] 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. URL `http://arxiv.org/abs/1609.08144`.

[20] 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. doi: 10.3115/v1/d14-1179. URL `https://doi.org/10.3115/v1/d14-1179`.

[21] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural machine translation architectures. *CoRR*, abs/1703.03906, 2017. URL `http://arxiv.org/abs/1703.03906`.

[22] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. doi: 10.1016/j.neunet.2014.09.003. URL `https://doi.org/10.1016/j.neunet.2014.09.003`.

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

[24] Chris Olah. Understanding LSTM networks, 2015. URL `http://web.archive.org/web/20200528192048/http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Accessed = 2020-06-02.

[25] 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. doi: 10.1007/978-3-540-88871-0_22. URL `https://doi.org/10.1007/978-3-540-88871-0_22`.

[26] 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. doi: 10.1016/j.is.2010.09.001. URL `https://doi.org/10.1016/j.is.2010.09.001`.

[27] 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. doi: 10.4108/icst.collaboratecom.2011.247129. URL `https://doi.org/10.4108/icst.collaboratecom.2011.247129`.

[28] 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. doi: 10.1007/978-3-642-45005-1_27. URL `https://doi.org/10.1007/978-3-642-45005-1_27`.

[29] 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. doi: 10.1007/978-3-319-11812-3_5. URL `https://doi.org/10.1007/978-3-319-11812-3_5`.

[30] 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. doi: 10.1007/978-3-319-45348-4_23. URL `https://doi.org/10.1007/978-3-319-45348-4_23`.

[31] 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 Work-*

shops, Rio de Janeiro, Brazil, September 19, 2016, Revised Papers, volume 281 of *Lecture Notes in Business Information Processing*, pages 327–338, 2016. doi: 10.1007/978-3-319-58457-7_24. URL `https://doi.org/10.1007/978-3-319-58457-7_24`.

[32] 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. doi: 10.1007/978-3-319-59536-8_30. URL `https://doi.org/10.1007/978-3-319-59536-8_30`.

[33] 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. doi: 10.1109/SSCI.2017.8285184. URL `https://doi.org/10.1109/SSCI.2017.8285184`.

[34] 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. doi: 10.1007/s00607-018-0593-x. URL `https://doi.org/10.1007/s00607-018-0593-x`.

[35] 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. doi: 10.1145/3301300. URL `https://doi.org/10.1145/3301300`.

[36] 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. doi: 10.1109/ICPM.2019.00027. URL `https://doi.org/10.1109/ICPM.2019.00027`.

[37] 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. doi: 10.1109/TNNLS.2013.2278313. URL `https://doi.org/10.1109/TNNLS.2013.2278313`.

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

[39] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, 2003. URL `http://jmlr.org/papers/v3/bengio03a.html`.

[40] 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. URL `http://arxiv.org/abs/1301.3781`.

[41] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL `http://arxiv.org/abs/1607.06450`.

[42] Mahidhar Dwarampudi and N. V. Subba Reddy. Effects of padding on lstms and cnns. *CoRR*, abs/1903.07288, 2019. URL `http://arxiv.org/abs/1903.07288`.

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

Acknowledgments