Department of Mathematics and Computer Science
Process Analytics

The Generation of interpretable Counterfactual Examples by finding minimal Edit Sequences using Event Data in Complex Processes

Master Thesis

Olusanmi A. Hundogan

Supervisors:
Xixi Lu
Yupei Du
August 7, 2022

Abstract

Contents

Chapter 1

Introduction

1.1 Motivation

Many processes, often medical, economical, or administrative in nature, are governed by sequential events and their contextual environment. Many of these events and their order of appearance play a crucial part in the determination of every possible outcome[48]. With the rise of AI and the increased abundance of data in recent years, several techqniques emerged that help to predict the outcomes of complex processes in the real world. A field that focuses on modelling processes is Process Mining (PM).

Research in the Process Mining discipline has shown that is possible to predict the outcome of a particular process fairly well[27, 45]. For instance, in the medical domain, models have been shown to predict the outcome or trajectory of a patient's condition[32]. In the private sector, process models can be used to detect faults or outliers. The research discipline Deep Learning has shown promising results within domains that have been considered difficult for decades. The Moravex Paradox[1], which postulates that machines are capabable of doing complex computations easily while failing in tasks that seem easy to humans such as object detection or language comprehension, does not hold anymore. Meaning that with enough data to learn, machines are capable of learning highly sophisticated tasks, better than any human. The same holds for predictive tasks. However, while many prediction models can predict certain outcomes, it remains a difficult challenge to understand their reasoning.

This difficulty arises from models, like neural networks, that are so-called *blackbox models*. Meaning, that their inference is imcomprehensible, due to the vast amount of parameters involved. This lack of comprehension is undesirable for many fields like IT or finance. Not knowing why a loan was

given, makes it impossible to rule out possible biases. Knowing what will lead to a system failure, will help us knowing how to avoid it. In critical domains like medicine, the reasoning behind decisions become crucial. For instance, if we know that a treatment process of a patient reduces the chances for survival, we want to know which treatment step is the critical factor we ought to avoid. To summarise, knowing the outcome of a process often leads us to questions on how to change it. Formally, we want to change the outcome of a process instance, by making it maximally likely, with as little interventions as possible[37]. Figure 1.1 is a visual representation of the desired goal.

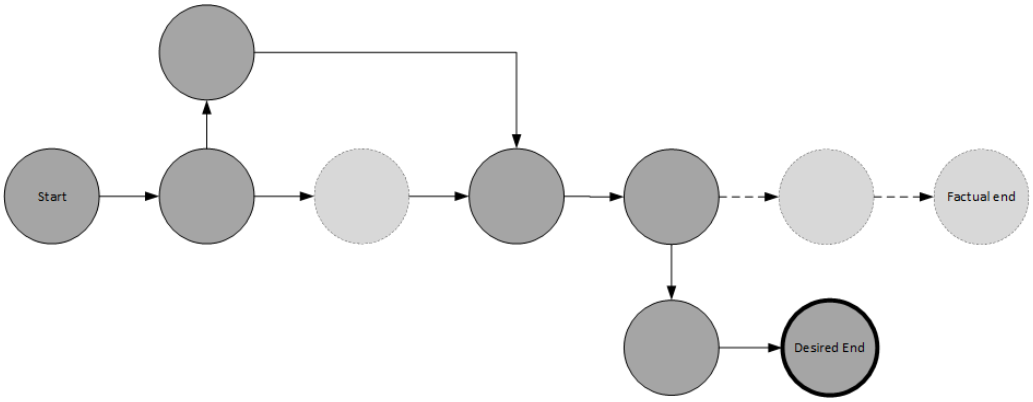


Figure 1.1: This figure illustrates a model, that predicts a certain trajectory of the process. However, we want to change the process steps in such a way, that it changes the outcome.

One-way to better understand the Machine Learning (ML) models lies within the eXplainable AI (XAI) discipline. XAI focuses the developments of theories, methods, and techniques that help explaining blackbox models models to humans. Most of the discipline's techniques produce explanations that guide our understanding. Explanations can come in various forms, such as IF-THEN rules[37, p.90] or feature importances[37, p.45]. but some are more comprehensible for humans than others.

A prominent and human-friendly approach are *counterfactuals*[37, p. 221]. Counterfactuals within the AI framework help us to answer hypothetical "what-if" questions. Basically, if we know *what* would happen *if* we changed the execution of a process instance, we could change it for the better. In this thesis, we will raise the question, how we can use counterfactuals to change the trajectory of a process models' prediction towards a desired outcome. Knowing the answers not only increases the understanding of blackbox models, but also help us avoid or enforce certain outcomes.

1.2 Problem Space

In this paper, we will approach the problem of generating counterfactuals for processes. The literature has provided a multitude of techniques to generate counterfactuals for AI models, that are derived from static data[1]. However, little research has focussed on counterfactuals for dynamic data[2].

For process data, the literature often uses terms like structured and semi-structured, as they are related to the staticity and dynamicity. Both, structuredness and semi-structuredness, often relate to the data model, in which we structure the information at hand. As static data neither changes over time nor changes its structure, we can use structured data-formats such as tables to capture the information and each data point is an independent entity. We can take the MNIST dataset[16] or Iris dataset[3, 17] as examples for structured and static data. In both datasets, all data points are independent and have the same amount of attributes. In contrast, semi-structured data does not have to follow these stric characteristics. Here, data points often belong to a group of data points which constitutes the full entity. Furthermore, the attributes of each data point may vary. The grouping mechanism could take the form of associative links, class associations or temporal cause-effect relationships. Examples of these are Part-of-Speech datasets like Penn Treebank set[33]. Here, we often associate each data point with a sentence. However, the temporal relationship between words is debatable and hence whether the data is *dynamic* as well. Hence, not all semi-structured data sets are dynamic and vice versa. However, structured data will almost always be static, with the exception of time-series. Lastly, there is also unstructured data, which does not incorporate any specific data model. Corpora like the Brown dataset[19], for instance, are collections of text heavy unstructured information. In Figure 1.2, we show various examples of data.

A major reason, why there has not been much research on counterfactuals for dynamic semi-structured data, emerges from a multitude of challenges, when dealing with counterfactuals and sequences. Three of these challenges are particularly important.

First, counterfactuals within AI attempt to explain outcomes which never occured. A *what-if* questions often refer to hypothetical scenarios. Therefore, there is no evidential data, from which we can infer predictions. Subsequently, this lack of evidence further complicates the evaluation of generated counterfactuals. In other words, you cannot validate the correctness of a theoretical outcome that has never occured.

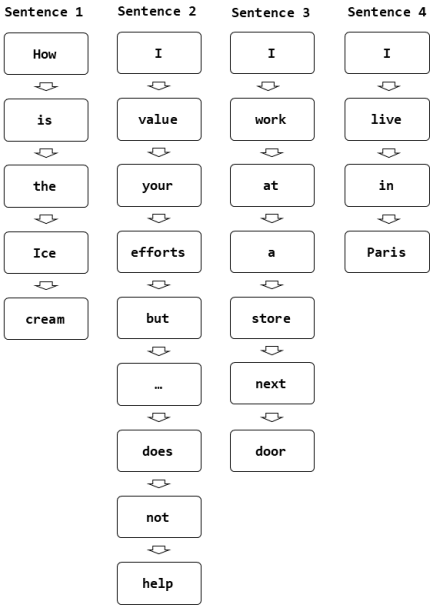
[1]With static data, we refer to data that does not change over a time dimension.

[2]With dynamic data, we refer to data that has a temporal relationship as a major component, which is also inherently sequential

s.length	s.width	p.length	p.width	variety
6.5	2.8	4.6	1.5	Versicolor
5.8	2.7	4.1	1.0	Versicolor
6.7	3.3	5.7	2.5	Virginica
4.6	3.4	1.4	0.3	Setosa
6.4	3.2	5.3	2.3	Virginica
5.9	3.0	4.2	1.5	Versicolor
7.4	2.8	6.1	1.9	Virginica
5.5	2.4	3.8	1.1	Versicolor
5.6	2.5	3.9	1.1	Versicolor
5.0	3.4	1.5	0.2	Setosa
6.9	3.1	5.4	2.1	Virginica
5.5	2.5	4.0	1.3	Versicolor
5.7	2.6	3.5	1.0	Versicolor
5.8	2.7	3.9	1.2	Versicolor
7.6	3.0	6.6	2.1	Virginica
6.7	3.3	5.7	2.1	Virginica
5.0	3.5	1.6	0.6	Setosa
7.7	2.8	6.7	2.0	Virginica
6.4	2.7	5.3	1.9	Virginica
7.7	3.8	6.7	2.2	Virginica
5.2	3.5	1.5	0.2	Setosa
5.7	3.8	1.7	0.3	Setosa

(a) Shows an excerpt of the MNIST dataset. This is a structured dataset.

(b) Shows a number of heterogenous documents. A dataset like this would be called unstructured.



(c) Shows multiple seqeuences of words. Each word forms a sentence. Therefore, this data is semi-structured.

Figure 1.2: Figure shows schematic examples of static structured, dynamic semistructured data and unstructured data.

Second, sequential data is highly variable in length, but process steps have complicated factors, too. The sequential nature of the data impedes the tractability of many problems due to the combinatorial explosion of possible sequences. Furthermore, the data generated is seldomly one-dimensional or discrete. Henceforth, each dimension's contribution can vary in dependance of its context, the time and magnitude.

Third, process data often requires knowledge of the causal structures that produced the data in the first place. However, these structures are often hidden and it is a NP-hard problem to elicit them[50].

These challenges make the field, in which we can contribute a vast endeavor.

1.3 Related Literature

Many researchers have worked on counterfactuals and PM. Here, we will combine the important concepts and discuss the various contributions to this thesis.

1.3.1 Generating Counterfactuals

The topic of counterfactual generation as explanation method was introduced by Wachter et al. in 2017[49]. The authors defined a loss function which incorporates the criteria to generate a counterfactual which maximizes the likelihood for a predefined outcome and minimizes the distance to the original instance. However, the solution of Wachter et al. did not account for the minimalisation of feature changes and does not penalize unrealistic features. Furthermore, their solution cannot incorporate categorical variables.

A newer approach by Dandl et al. incoporates four main criteria for counterfactuals (see section 2.3) by applying a genetic algorithm with a multi-objective fitness function[13]. This approach strongly differs from gradient-based methods, as it does not require a differentiable objective function. However, their solution was only tested on static data.

1.3.2 Generating Counterfactual Sequences

When it comes to sequential data most researchers work on ways to generate counterfactuals for natural language. This often entails generating univariate discrete counterfactuals with the use of Deep Learning techniques. Martens and Provost and later Krause et al. are early examples of counterfactual NLP research[28, 34]. Their approach strongly focuses on the manipulation

of sentences to achieve the desired outcome. However, as Robeer et al. puts it, their counterfactuals do not comply with *realisticness*[42].

Instead, Robeer et al. showed that it is possible to generate realistic counterfactuals with a Generative Adversarial Model (GAN)[42]. They use the model to implicitly capture a latent state space and sample counterfactuals from it. Apart from implicitly modelling the latent space with GANs, it is possible to sample data from an explicit latent space. Examples of these approaches often use an encoder-decoder pattern in which the encoder encodes a data instance into a latent vector, which will be peturbed and then decoded into a a similar instance[35, 51]. By modelling the latent space, we can simply sample from a distribution conditioned on the original instance. Bond-Taylor et al. provides an overview of the strengths and weaknesses of common generative models.

Eventhough, a single latent vector model can theoretically produce multivariate sequences, it may still be too restrictive to capture the combinatorial space of multivariate sequences. Hence, most of the models within Natural Language Processing (NLP) were not used to produce a sequence of vectors, but a sequence of discrete symbols. For process instances, we can assume a causal relation between state vectors in a sequential latent space. We call models that capture a sequential latent state-space which has causal relations *dynamic*[30]. Early models of this type of dynamic latent state-space models are the well-known *Kalman-Filter* for continous states and Hidden Markov Model (HMM) for discrete states. In recent literature, many techniques use Deep Learning to model complex state-spaces. The first models of this type were developed by Krishnan et al.[28, 29]. Their Deep Kalman Filter (DKF) and subsequent Deep Markov Model (DMM) approximate the dynamic latent state-space by modeling the latent space given the data sequence and all previous latent vectors in the sequence. There are many variations[10, 18, 30] of Krishnan et al.'s model, but most use Evidence Lower-Bound (ELBO) of the posterior for the current Z_t given all previous $\{Z_{t-1}, \ldots, Z_1\}$ and X_t[21].

1.3.3 Generating Counterfactual Time-Series

Within the *multivariate time-series* literature two recent approaches yield ideas worth discussing.

First, Delaney et al. introduces a case-based reasoning to generate counterfactuals[15]. Their method uses existing counterfactual instances, or *prototypes*, in the dataset. Therefore, it ensures, that the proposed counterfactuals are *realistic*. However, case-based approaches strongly depend on the *representativeness* of the prototypes[37, p. 192]. In other words, if the model displays behaviour, which is not capture within the set of prototypical

instances, most case-based techniques will fail to provide viable counterfactuals. The likelihood of such a break-down increases due to the combinatorial explosion of possible behaviours if the *true* process model has cycles or continuous event attributes. Cycles may cause infinite possible sequences and continous attributes can take values on a domain within infinite negative and positive bounds. These issues have not been explored in the paper of Delaney et al., as it mainly deals with time series classification[15]. However, despite these shortcomings, case-based approaches may act as a valuable baseline against other sophisticated approaches.

The second paper within the multivariate time series field by Ates et al. also uses a case-based approach[5]. However, it contrasts from other approaches, as it does not specify a particular model but proposes a general framework instead. Hence, within this framework, individual components could be substituted by better performing components. Describing a framework, rather than specifying a particular model, allows to adapt the framework, due to the heterogeneous process dataset landscape. In this paper, we will also introduce a framework that allows for flexibility depending on the dataset.

1.3.4 Generating Counterfactuals for Business Processes

So far, none of the models have been applied to process data.

Within PM, Causal Inference has long been used to analyse and model business processes. Mainly, due to the causal relationships underlying each process. However, early work has often attempted to incorporate domain-knowledge about the causality of processes in order to improve the process model itself[6, 24, 43, 52]. Among these, Narendra et al. approach is one of the first to include counterfactual reasoning for process optimization[38]. Oberst and Sontag use counterfactuals to generate alternative solutions to treatments, which lead to a desired outcome[39]. Again, the authors do not attempt to provide an explanation of the models outcome and therefore, disregard multiple viability criterions for counterfactuals in XAI. Qafari and W. M. P. van der Aalst published the most recent paper on the counterfactual generation of explanations[41]. The authors, use a known Structural Causal Model (SCM), to guide the generation of their counterfactuals. However, this approach requires a process model which is as close as possible to the *true* process model. For our approach, we assume that no knowledge about the dependencies are known.

Within the XAI context, Tsirtsis et al. develop the first explanation method for process data[47]. However, their work closely resembles the work of Oberst and Sontag and treat the task as Markov Decision Process

(MDP)[39]. This extension of a regular Markov Process (MP) assumes that an actor influences the outcome of a process given the state. This formalisation allows the use of Reinforcement Learning (RL) methods like Q-learning or SARSA. However, this often requires additional assumptions such as a given reward function and an action-space. For counterfactual sequence generation, there is no obvious choice for the reward function or the action-space.

Nonetheless, both Tsirtsis et al. and Oberst and Sontag contribute an important idea. The idea of incrementally generating the counterfactual instead of the full sequence. Hsieh et al. has recently published an approach that builds on the same notion of incremental generation. Their approach has a very similar structure to our approach and appears to be the only one that we can compare our counterfactuals against.

For this reason, we dedicate an **[section xxx]** to their work to highlight key differences and similarities. However, to understand the differences and similarities, we first have to establish some core concept. Therefore, we dedicate **[a number of]** sections to highlight those diffenrences and similarities in **[chapter reference]**. In this section, we only discuss their approach, briefly.

The authors recognised that some processes have critical events, which govern the overall outcome. Hence, by simply avoiding the undesired outcome from critical event to critical event, it is possible to limit the search space and compute viable counterfactuals. They use an extension of DiCE to generate counterfactuals. However, this approach requires concrete knowledge about these critical points. We propose a Framework that avoids this constraint.

To our knowledge, the authors are also the first authors that try to optimize their counterfactual process generation based on criterions that ensure their viability. However, in our approach, we use different operationalisations quantify the criterions.

1.4 Research Question

As we seek to make data-driven process models interpretable, we have to understand the exact purpose of this thesis. Hence, we will establish the challenges that are open and how this thesis attempts to solve them.

Having discussed the previous work on counterfactual sequence generation, a couple of challenges emerge. First, we need to generate on a set of criteria and therefore, require complex loss and evaluation metrics, that may or may not be differiantiable. Second, they cannot to be logically impossible, given the data set. Hence, we have to restrict the space to counterfactuals of viable solutions, while being flexible enough to not just copy existing data in-

stances. Third, using domain knowledge of the process significantly reduces the practicality of any solution. Therefore, we have to develop an approach, which requires only the given log as input while not relying on process specific domain knowledge. This begs the question, whether there is a method to generate sequential counterfactuals that are viable, without relying on process specific domain knowlege. In terms of specific research questions we try to answer:

[Update this image.]

RQ: Which existing counterfactual approaches can be applied to generate sequences?

RQ1: To what extent can we generate counterfactuals that fulfill the criterions to be viable?

RQ2: How does an algorithm, which optimizes multiple quality metrics perform against other approaches?

We approach these questions, by proposing a schematic framework which allows the exploration of several independent components. Figure **??** shows the conceptual framework of the base approach visually.

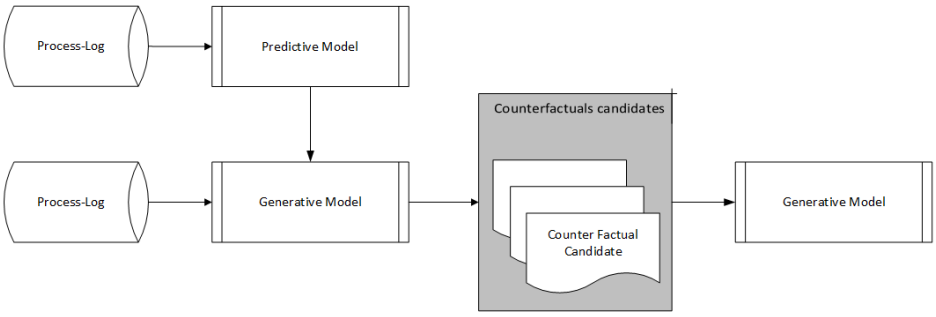


Figure 1.3: This figure shows a simplified schematic representation of the framework which is explored in this thesis.

The framework contains three parts. First, we need a pretrained predictive component which we aspire to explain. The component should be capable of *accurately* predicting the outcome of a process at any step. The accuracy-condition is favorable but not necessary. If the component is accurately modelling the real world, we can draw real-world conclusions from the explanations generated. If the component is inaccurate, the counterfactuals only explain the prediction decisions and not the real world. The second part requires a generative component. The generative component needs to generate viable sequential counterfactuals which are logically *plausible*. A

plausible counterfactual is one whose outcome can be predicted by the predictive component. If the predictive component cannot predict the counterfactual sequence, we can assume that the generative model is *unfaithful* to the predictive component, we want to explain. The third component is the evaluation metric upon which we decide the viability of the counterfactual candidates.

For the evaluation, we have to test the following hypotheses:

RQ1-H1: If we use a viability function which incorporates multiple criterions to determine counterfactuals, we consistently retrieve more viable counterfactuals, than randomly generating one.

RQ1-H2: The generated counterfactuals consistently outperform the most viable counterfactuals among examples in the dataset.

RQ2-H1: The results of the counterfactual are comparable to other existing literature.

Chapter 2

Background

This chapter explores the most important concepts for this workk. Hence, we will focus on the problem domain, starting with an overview about PM. Afterwards, we discuss the nature of the data, we handle in this thesis by discussing *Multivariate Time-Series*. Next, we introduce counterfactuals and establish how we characterise *viable* counterfactuals.

2.1 Process Mining

This thesis will focus on processes and the modelling of process generated data. Hence, it is important to establish a common understanding for this field.

2.1.1 A definition for Business Processes

Before elaborating on Process Mining, we have to establish the meaning of the term *process*. The term is widely-used and therefore has a rich semantic volume. A process generally refers to something that advances and changes over time[14]. Despite, legal or biological processes being valid understandings, too, we focus on *business processes*.

An example is a loan application process in which an applicant may request a loan. The case would then be assessed and reviewed by multiple examiners and end in a final decision. The loan might end up in an approval or denial. The *business* part is misleading as these processes are not confined to commercial settings alone. For instance, a medical business process may cover a patients admission to a hospital, followed by a series of diagnostics and treatments and ending with the recovery or death of a patient. Another example from a Human Computer Interaction (HCI) perspective would be

an order process for an online retail service like Amazon. The buyer might start the process by adding articles to the shopping cart and proceeding with specifying their bank account details. This order process would end with the submission or receival of the order.

All of these examples have a number of common characteristics. They have a clear starting point which is followed by numerous intermediary steps and end in one of the possible sets of outcomes. For this work we will mainly follow the understanding outlined in W. van der Aalst et al.[48]. Each step, including start and end points, is an process event which was caused by an *activity*. Often, both terms, *event* and *activity*, are used interchangeably. However, there are subtle differences, which will become important later in this thesis. For now, we understand an event as something that happens at a specific point in time. The driving question is *when* the event happens. In contrast, an activity is related to the content of an event. Here, we ask *what* happens at a point in time. For instance, if we apply for a loan that requires an approval by one person and aferwards a second approval, we can call both activities **APPROVAL**. Although both activities are fundamentally the *same*, they happen at different points in time. Henceforth, both events remain *different*. Mainly, because one can argue that both events have varying time dependent contexts. For instance, an approval at daytime might be caused by different reasons, than an event caused at nighttime.

Each process event may contain additional information in the form of event attributes. If a collection of events *sequentially* relate to a single run through a process, we call them *process instance* or *trace*. These instances do not have to be completed. Meaning, the trace might end prematurely. In line with the aforementioned examples, these process instances could be understood as a single loan application, a medical case or a buy order. We can also attach process instance related information to each instance. Examples would be the applicants location, a patients age or the buyers budget. In its entirety, a business process can be summarised as a *graph*, a *flowchart* or another kind of visual representation. Figure 2.1's graphical representation is an example of such a *process map*[48].

In conclusion, in this thesis a *business process* refers to

> *A finite series of discrete events with one or more starting points, intermediary steps and end points. Each intermediate step has at least one precedent and at least one antecedent step.*

However, we have to address a number of issues with this definition.

First, it excludes infinite processes like solar system movements or continuous processes such as weather changes. There may be valid arguments

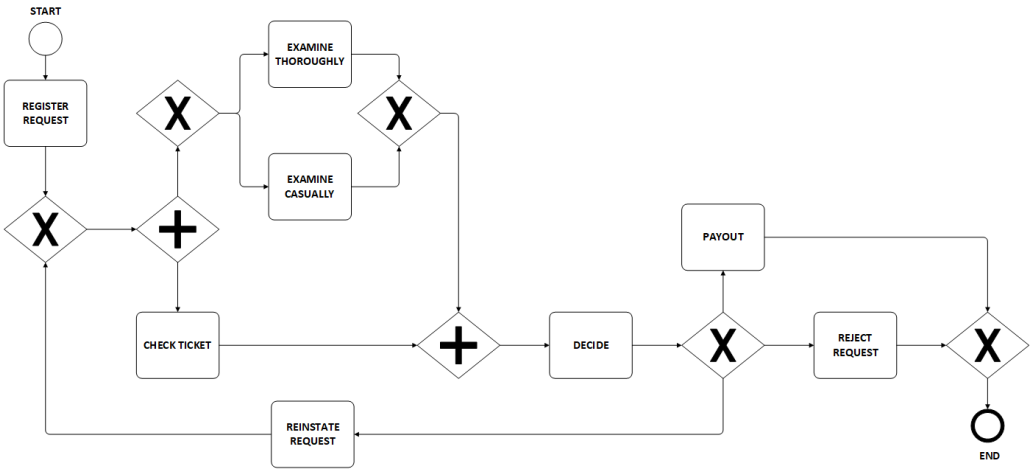


Figure 2.1: This graph shows an example of a Business Process Modell Notation (BPMN) process map.

to include processes with these characteristics, but they are not relevant for this thesis.

Second, in each example, we deliberately used words that accentuate modality such as *may*, *can* or *would*. It is important to understand that each process anchors its definition within an application context. Hence, what defines a business process is indisputably subjective. For instance, while an online marketplace like Amazon might be interested in the process from the customers first click to the successful shipment, an Amazon vendor might only be interested in the delivery process of a product.

Third, the example provided in Figure 2.1 may not relate to the *real* underlying data generating process. As process *models* are inherently simplified, they may or may not be accurate. The *true* process is often unknown. Therefore, we will distinguish between the *true process* and a *modelled process*. The *true process* is a hypothetical concept whose *true* structure remains unknown. In, contrast, a process *model* simplifies and approximates the characteristics of the *true process*.

2.1.2 What is Process Mining?

Having established a definition for a process, we next discuss *Process Mining*. This young discipline has many connections to other fields that focus on the modeling and analysis of processes such as Continuous Process Improvement (CPI) or Business Process Management (BPM)[48]. However, its data-centric approaches originate in Data Mining. The authors W. van der Aalst et al. describe this field as a discipline "to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge

from event logs readily available in today's (information) systems"[48]. The discipline revolves around the analysis of event logs. A event log is a collection of process instances, which are retrieved from various sources like an Information System (IS) or database. Logs are often stored in data formats such as Comma Seperated Values (CSV) or eXtensible Event Stream (XES)[48].

2.1.3 The Challenges of Process Mining

As mentioned in chapter 1, process data modelling and analysis is a challenging task. W. van der Aalst et al. mentions a number of issues that arise from processes[48].

The first issue arises from the quality of the data set. Process logs are seldomly collected with the primary goal of mining information and hence, often appear to be of subpar quality for information mining purposes. The information is often incomplete, due to a lack of context information, the ommision of logged process steps, or wrong levels of granularity[48].

This issue is exacerbated by the second major issue with process data. Mainly, its complexity. Not only does a process logs' complexity arise from the variety of data sources and differing levels of complexity, but also from the datas' characteristics. The data can often be viewed as multivariate sequence with discrete and continuous features and variable length. This characteristic alone creates problems explored in section 2.2. However, the data is also just a *sample* of the process. Hence, it may not reflect the real process in its entirety. In fact, mining techniques need to incorporate the *open world assumption* as the original process may generate unseen process instances[48].

A third issue which contributes to the datasets' incompleteness and complexity is a phenomenon called *concept drift*[48]. This phenomenon relates to the possibility of changes in the *true* process. The change may occur suddenly or gradually and can appear in isolation or periodically. An expression of such a drift may be a sudden inclusion of a new process step or domain changes of certain features. These changes are not uncommon and their likelihood increases with the temporal coverage and level of granularity of the dataset[48]. In other words, the more *time* the dataset covers and the higher its detail, the more likely a change might have occured over the time.

All three issues relate to the *representativeness* of the data with regards to the unknown *true* process that generated the data. However, they also represent open challenges that require research on their own. For our purpose, we have to assume that the data is representative and its underlying process is static. These assumptions are widely applied in the body of process

mining literature[27, 45].

2.2 Multivariate Time-Series Modelling

The temporal and multivariate nature of process instance often turns PM into a Multivariate Time-Series Modelling problem. Therefore, it is necessary to establish an understanding for this type of data structure.

The data which is mined in Process Mining is typically a multivariate time-series. It is important to establish the characteristics of time-series.

2.2.1 What are Time Series Models?

A time series can be understood as a series of observable values and depend on previous values. The causal dependence turns time-series into a special case of sequence models. Sequences do not *have to* depend on previous values. They might depend on previous and future values or not be interdependent at all. An example of a sequence model would be a language model. Results in NLP, that the words in a sentences for many languages do not seem to only depend on prior words but also on future words[20]. Hence, we can assume that a human has formulated his sentence in the brain before expressing it in a sequence of words. In contrast to sequences, time series cannot depend on future values. The general understanding of *time* is causal and forward directed. The notion of time relates to our understanding of *cause and effect*. Hence, we can decompose any time series in a precedent (causal) and an antecedent (effect) part[30]. A time series model attempts to capture the relationship between precedent and antecedent.

2.2.2 The Challenges of Time Series Modelling

The analysis of unrestricted sequential opens up a myriad of challenges. First, sequential data introduces a combinatorial set of possible realisations (often called *productions*). For instance, a set of two objects $\{A, B\}$ *produces* 7 theoretical combinations ($\{\emptyset\}$, $\{A\}$, $\{B\}$, $\{A, B\}$, $\{B, A\}$, $\{A, A\}$, $\{B, B\}$). Just by adding C and then D to the object set increases the number of combinations to 40 and 341 respectively. Second, sequential data may contain cyclical patterns which increase the number of possible productions to infinity[50]. Both, the combinatorial increase and cycles, yield a set of a countable infinite number of possible productions. However, as processes may also contain additional information a third obstacle arises. Including additional information extends the set to an uncountable number of possible

productions. With these obstacles in mind, it often becomes intractable to compute an exact model.

Hence, we have to include restrictive assumptions to reduce the solution space to a tractable number. A common way to counter this combinatorial explosion is the inclusion of the *Granger Causality* assumption[2]. This idea postulates the predictive capability of a sequence given its preceding sequence. In other words, if we know that C must be followed by D, then 341 possible combinations reduce to 156. All of these possible 156 combinations are now temporally-related and hence, we speak of a *time-series*.

However, the prediction of sequences recontextualises the issue to two new questions: First, if we know the precedence of a time-series, what is the antecedent? And second, if we can predict the antecedent accurately, what caused it? We often use data-driven AI-methods like Hidden-Markov-Models or Deep Learning to solve the first question. However, the second question is more subtle. At first glance, it is easy to believe that both questions are quite similar, because we could assume that the precedent causes the antecedent. Meaning, that we can use the data available to elicit sequential correlative patterns. In reality, the latter question is much more difficult as data often does not include any information about the inter-relationships. To illustrate this difficulty, we could say that the presence of C causes D. But if D also appears to be valid in a sequence 'AABD', it cannot be caused by the presence of C alone.

Answering this question requires additional tools within the XAI framework. One such method is the focus of this thesis and is further explored in section 2.3.

2.3 Counterfactuals

Counterfactuals are an important explanatory tool to understand a models' cause for decisions. Generating counter factuals is main focus of this thesis. Hence, we will establish the most important chateristics of counterfactuals in this section.

2.3.1 What are Counterfactuals?

Counterfactuals have various definitions. However, their semantic meaning refers to "*a conditional whose antecedent is false*"[11]. A simpler definition from Starr states that counterfactual modality concerns itself with "*what is not, but could or would have been*". Both definitions are related to linguistics and philosophy. Within AI and the mathematical framework various formal

definitions can be found in the causal inference[22] literature. A prominent figure within the causal inference discipline is Pearl et al., who postulates that a "*kind of statement – an 'if' statement in which the 'if' portion is untrue or unrealized – is known as a counterfactual*"[40]. What binds all of these definitions is the notion of causality within *what-if* scenarios.

For this paper, we use the understanding established within the XAI context. Within XAI, counterfactuals act as a prediction which "*describes the smallest change to the feature values that changes the prediction to a predefined output*" according to Molnar[37, p. 212]. Note that XAI mainly concerns itself with the explanation of *models*, which are always subject to inductive biases and therefore, inherently subjective. The idea behind counterfactuals as explanatory tool[1]is simple. We understand the outcome of a model, if we know *what* outcome would occur *if* we changed its input. For instance, lets declare a sequence 1 as $ABCDE\boldsymbol{FG}$. Then a counterfactual $ABCDE\boldsymbol{XZ}$ would tell us that **F** (probably) caused **G** in sequence 1. As counterfactuals only address explanations of one model result and not the model as a whole, they are called *local* explanations[37, p. 212]. According to Molnar *Valid* counterfactuals satisfy **four** criteria[37, p. 212]:

Similarity: A counterfactual should be similar to the original instance. If the counterfactual to sequence 1 was $A\boldsymbol{A}CDE\boldsymbol{XZ}$ we would already have difficulties to discern whether B or F or both caused G at the end of sequence 1. Hence, we want to be able to easily compare the counterfactual with the original. We can archive this by either minimizing their mutual distance.

Sparcity: In line with the notion of similarity, we want to change the original instance only minimally. Multiple changes impede the understanding of causal relationships in a sequence.

Feasibility: Each counterfactual should be feasible. In other words, impossible values are not allowed. As an example, a sequence $ABCDE\boldsymbol{1G}$ would not be feasible if numericals are not allowed. Typically we can use data to ensure this property. However, the *open-world assumption* impedes this solution. With *open-world*, we mean that processes may change and introduce behaviour that has not been measured before. Especially for long and cyclical sequences, we have to expect previously unseen sequences.

[1]There are other explanatory techniques in XAI like *feature importances* but counterfactuals are considered the most human-understandable

Likelihood: A counterfactual should produce the desired outcome if possible. This characteristic is ingrained in Molnar's definition. However, as the model might not be persuaded to change its prediction, we relax this condition. We say that we want to increase the likelihood of the outcome as much as possbile. If the counterfactual $ABCDE\boldsymbol{XZ}$ ends with Z but this sequence is highly unrealistic, we cannot be certain of our conclusion for sequence 1. Therefore, we want the outcome's lieklihood to be at least higher under the counterfactual than under the factual instance.

All four criteria allow us to assess the viability of each generated counterfactual and thus, help us to define an evaluation metric for each individual counterfactual. However, we also seek to optimise certain qualities on the population level of the counterfactual candidates.

Diversity: We typically desire multiple diverse counterfactuals. One counterfactual might not be enough to understand the causal relationships in a sequence. In the example above, we might have a clue that F causes G, but what if G is not only caused by F? If we are able to find counterfactuals $\boldsymbol{V}BCDEF\boldsymbol{H}$ and $ABCDE\boldsymbol{XZ}$ but all other configurations lead to G, then we know positions 1 and 6 cause G.

Realism: For a real world application, we still have to evaluate their *reasonability* within the applied domain. This is a characteristic that can only be evaluated by a domain expert.

We refer to both sets of viability criterions as *individual viability* and *population viability*. However, to remains concise, we will use *viability* to refer to the individual criterions only. We will explicitly mention *population viability* if we refer to criterions that concern the poulation.

2.3.2 The Challenges of Counterfactual Sequence Generation

The current literature surounding counterfactuals exposes a number of challenges when dealing with counterfactuals.

The most important disadvantage of counterfactuals is the Rashommon Effect[37, ch.9.3]. If all of the counterfactuals are viable, but contradict eachother, we have to decide which of the *truths* are worth considering.

This decision reveals the next challenge of evaluation. Although, the criteria can support us with the decision, it remains an open research question

how to evaluate counterfactuals according to Carvalho et al. So far, no one was able to establish a standardized evaluation protocol[25]. Every automated measure comes with implicit assumptions and they cannot guarantee a realistic explanations. Furthermore, we attempt to explain something with – in simple terms – *experiences* that never actually occured. We still need domain experts to assess their *plausibility*.

The generation of counterfactual sequences contribute to both former challenges, due to the combinatorial expansion of the solution space. This problem is common for counterfactual sentence generation and has been adressed within the NLP. However, as process mining data not only consist of discrete objects like *words*, but also event and case features, the problem remains a daunting task. So far, little work has gone into the generation of multivariate counterfactual sequences like process instances.

2.4 Formal Definitions

Before diving into the rest of this thesis, we have to establish preliminary definitions, we use in this work. With this definitions, we share a common formal understanding of mathematical descriptions of every concept used within this thesis.

2.4.1 Process Logs, Cases and Instance Sequences

We start by formalising the event log and its elements. We use a medical process as an example to provide a better semantic understanding. An event log is denoted as L. Here, L could be as database which logs the medical histories of all patients in a hospital.

We assume the database to log all interactions, be it therapeutic or diagnostic and store them as an event with a unique identifier. Let $\mathcal{E}$ be the universe of these event identifiers and $E \subseteq \mathcal{E}$ a set of events. The set E could consist, for instance, of a patients first session with a medical professional, then a diagnostic scan, followed by therapie sessions, surgery and more.

All of these interactions with one patient make up a case, which has a unique identifier, too. Let C be a set of case identifiers and $\pi_\sigma : E \mapsto C$ a surjective function that links every element in E to a case $c \in C$ in which c signifies a specific case. The function allows us to associate every event within the database to a single patient. The function's surjective property ensures for each case there exists at least one event.

For a set of events $E \subseteq \mathcal{E}$, we use a shorthand s^c being a particular sequence $s^c = \langle e_1, e_2, \ldots, e_t \rangle$ with c as case identifier and a length of t. Each

s is a trace of the process log $s \in L$. To understand the difference between c and s, we can say, that c is the ID for the case of patient X. Henceforth, s^c reflects all interactions that the database has logged for patient X.

These events are ordered in the sequence, in which they occured for patient X. Therefore, let $\mathcal{T}$ be the time domain and $\pi_t : E \mapsto \mathcal{T}$ a non-surjective linking function which strictly orders a set of events. In other words, every event in the database maps to one point in time. If the database logs every event on a daily basis, then all possible dates in history constitute $\mathcal{T}$. However, not every day has to be linked to a case as π_t is non-surjective.

Let $\mathcal{A}$ be a universe of attribute identifiers, in which each identifier maps to a set of attribute values $\overline{a}_i \in \mathcal{A}$. An attribute identifier describes everything the database might store for a patient, such as heart-rate or blood sugar level. If the database logs the heart-rate, then heart-rates of -42 beats-per-minute are not possible. Hence, $\overline{a}_i$ can per definition only map to positive integers.

Let $\overline{a}_i$ correspond to a set of possible attribute values by using a surjective mapping function $\pi_A : \mathcal{A} \mapsto A$. Then, each event e_t consists of a set $e_t = \{a_1 \in A_1, a_2 \in A_2, \ldots, a_I \in A_I\}$ with the size $I = |\mathcal{A}|$, in which each a_i refers to a value within its respective set of possible attribute values. In other words, every event consists of a set of values. If the event was recorded after a physio therapeutic session, then a_1 might be the specific degree to which you can move your ligaments and a_2 a description for the type of activity. If the event was recorded after a breast-cancer scan, the a_1, a_2 and a_3 might relate to the specific diameter, the threat-level and again an indicator for the activity type. Conversely, we define a mapping from an attribute value to its respective attribute identifier $\pi_{\overline{a}} : A \mapsto \mathcal{A}$. Hence, we can map every event attribute value back to its attribute identifier.

The following part is not necessarily connected with *what* is stored within the database symbolically, but rather *how* it is represented in the database or during processing.

We require a set of functions F to map every attribute value to a representation which can be processed by a machine. Let $\pi_d : A_i \mapsto \mathbb{N}$ be a surjective function, which determines the dimensionality of a_i and also F be a set of size I containing a representation function for every named attribute set. We denote each function $f_i \in F$ as a mapper to a vector space $f_i : a_i \mapsto \mathbb{R}_i^d$, in which d represents the dimensionality of an attribute value $d = \pi_d(A_i)$. For instance categorical variables will can map to a one-hot-encoded vector. Numerical values like heart-beat might be recorded in scalar form.

With these definitions, we denote any event $e_t \in s^c$ of a specific case c as a vector, which concatenates every attribute representation f_i as $\mathbf{e}_t^c = [f_1; f_2; \ldots; f_I]$. Therefore, $\mathbf{e}_t^c$ is embedded in a vector space of size D which

is the sum of each individual attribute dimension $D = \sum_i \pi_d(A_i)$. In other words, we concatenate all representations, whether they are scaler or vectors to one final vector representing the event. Furthermore, if we refer to a specific named attribute set A_i as a name, we will use the shorthand $\overline{a}_i$.

Figure 2.2 shows a schematic representation of a log L, a case c and an event e.

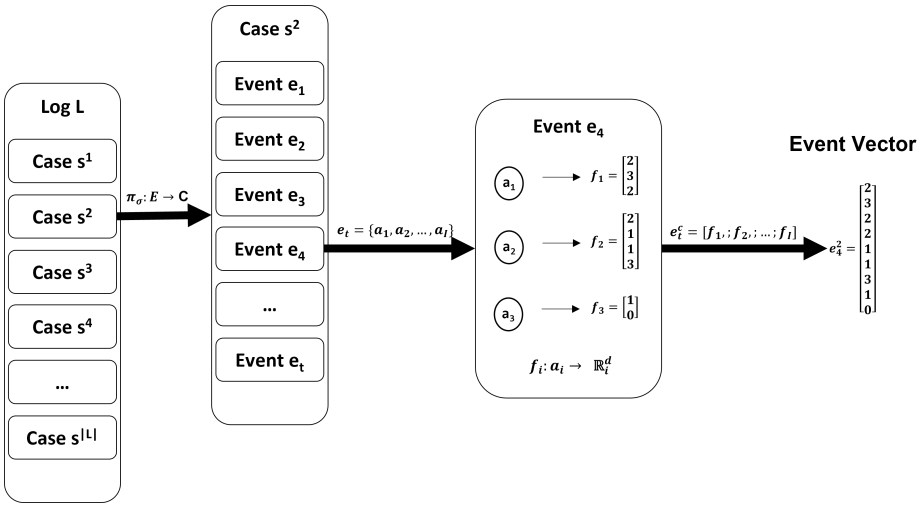


Figure 2.2: This figure shows the representation of a log L which contains anumber of cases s. Case s^2 contains a number of events e_t. Each events has attribute values a_i, which are mapped to vector spaces of varying dimensions. At last, all of the vectors are concatenated.

2.4.2 State-Space Models

Generally speaking, every time-series can be represented as a state-space model[26]. Within this framework the system consists of *input states* for *subsequent states* and *subsequent outputs*. A mathematical form of such a system is shown in Equation 2.1.

$$\begin{aligned} \mathbf{z}_{t+1} &= h(t, \mathbf{z}_t, \mathbf{u}_t) \\ \mathbf{e}_t &= g(t, \mathbf{z}_t, \mathbf{u}_t) \\ \mathbf{z}_{t+1} &:= \frac{d}{dt}\mathbf{z}_t \end{aligned} \tag{2.1}$$

Here, $\mathbf{u}_t$ represents the input, $\mathbf{z}_t$ the state at time t. The function h maps t, $\mathbf{z}_t$ and $\mathbf{u}_t$ to the next state $\mathbf{z}_{t+1}$. The event $\mathbf{e}_t$ acts as an output computed by function g which takes the same input as h. The variables $\mathbf{z}_t$, $\mathbf{u}_t$ and $\mathbf{e}_t$ are vectors with discrete or continuous features. The distinction of $\mathbf{z}_{t+1}$ and $\mathbf{e}_t$

decouples *hidden*[2] states, from *observable* system outputs. Figure 2.3 shows a graphical representation of these equations.

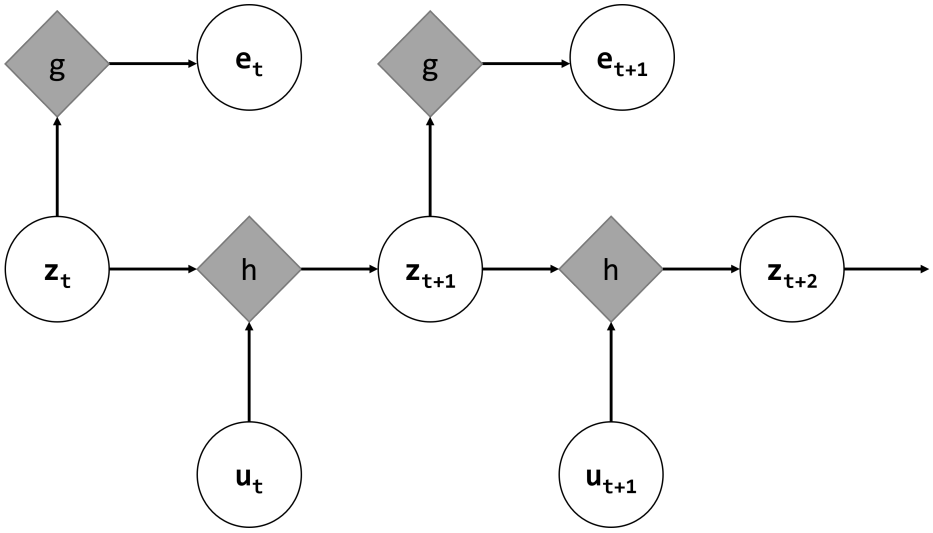


Figure 2.3: This figure shows a simplified graphical representation of a state-space model. Each arrow represents the flow of information.

The body of literature for state-space models is too vast to discuss them in detail[3]. However, for process mining, we can use this representation to discuss the necessary assumptions for process mining. In line with the process-definition in section 2.1, we can understand the event log as a collection of the observable outputs of a state-space model. The state of the process is hidden as the *true* process which generated the data cannot be observed as well. The time t is a step within the process. Hence, we will treat t as a discrete scalar value to denote discrete sequential time steps. Hence, if we have $\sigma = \{a, b, b, c\}$, then t, describes the index of each element in σ. The input $\mathbf{u}_t$ represents all context information of the process. Here, $\mathbf{u}_t$ subsumes observable information such as the starting point and process instance-related features. The functions h and g determine the transition of a process' state to another state and its output over time. Note, that this formulation disregards any effects of future timesteps on the current timestep. Meaning, that the state transitions are causal and therefore, ignorant of the future. As we establish in section 2.1, we can assume that a process is a discrete sequence, whose transitions are time-variant. In this framework, we try to identify the

[2]A state does not have to be hidden. Especially, if we know the process and the transition rules. However, those are often inaccessible if we only use log data. Instead, many techniques try to approximate the hidden state given the data instead.

[3]For an introduction to state-space models see: XXX

parameters of the functions h and g. Knowing the functions, it becomes simple to infer viable counterfactuals. However, the function parameters are often unknown and therefore, we require probablistic approaches.

We can formulate Equation 2.1 probablistically as shown in Equation 2.2.

$$\mathbb{E}[p\left(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h\right)] = \int z_{t+1} \cdot p\left(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h\right) \tag{2.2}$$

$$\mathbb{E}[p\left(x_t \mid t, z_{1:T}, u_{1:T}, \theta_g\right)] = \int x_t \cdot p\left(x_t \mid t, z_{1:T}, u_{1:T}, \theta_g\right)$$

Note, that h and g are substitued with probability density functions parametrized with θ_h and θ_g. T signifies the full sequence including future timesteps. Both expectations are intractable as they require integrating over n-dimensional vectors. To solve the intractability, we characterize the system as a *Hidden Markov Process* and Probalistic Graphical Model (PGM). This framework allows us to leverage simplifying assumptions such as the independece from future values and *d-seperation*.

These characteristics change the probabilities in Equation 2.2 to Equation 2.3:

$$p\left(z_{t+1} \mid z_{1:t}, u_{1:t}, \theta_h\right) = \prod_{1}^{t} p\left(z_t \mid z_{1:t}, u_t, \theta_h\right) \tag{2.3}$$

$$p\left(x_t \mid z_{1:t}, \theta_g\right) = \prod_{1}^{t} p\left(x_{t-1} \mid z_{1:t}, \theta_f\right) \tag{2.4}$$

For $p\left(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h\right)$, we ignore future timesteps, as T changes into t. *d-seperation* allows us to ignore all $\mathbf{e}_t$ of previous timesteps. The graphical form also decomposes the probability into a product of probabilities that each depend on all previous states and its current inputs. Previous $\mathbf{e}_t$ are ignored due to *d-seperation*. $p\left(x_t \mid t, z_{1:T}, u_{1:T}, \theta_g\right)$ only depends on its current state, which is in line with HMMs. Note, that we deliberately not assume a *strong Markov Property*, as the Deep Learning-Framework allows us to take all previous states into account. The *strong Markov Property* would assume that only the previous state suffices. At last, we assume that we do not model automatic or any other process whose state changes without a change in the input or previous states. Hence, we remove the dependency on the independet t variable. Only the previous states $z_{1:T}$ and the input information $\mathbf{u}_t$ remain time-dependent.

In this probablistic setting, the generation of counterfactuals, amounts to drawing samples from the likelihood of Equation 2.3. We then use the samples to reconstruct the most-likely a counterfactual $e^*_{1:t}$. Hence, our goal is to maximize both likelihoods.

2.5 Representation

To process the data in subsequent processing steps, we have to discuss the way we will encode the data. There are a multitude of ways to represent a log. We discuss 4 of them in this thesis.

First, we can choose to concentrate on *event-only-representation* and ignore feature attributes entirely. However, feature attributes hold significant amount of information. Especially in the context of using counterfactuals for explaining models as the path of a process instance might strongly depend on the event attributes. Similar holds for a *feature-only-representation*

The first is a *single-vector-representation* with this representation we can simply concatenate each individual representation of every original column. This results in a matrix with dimensions (case-index, max-sequence-length, feature-attributes). The advantage of having one vector is the simplicity with which it can be constructed and used for many common frameworks. Here, the entire log can be represented as one large matrix. However, eventhough, it is simple to construct, it is quite complicated to reconstruct the former values. It is possible to do so by keeping a dictionary which holds the mapping between original state and transformed state. However, that requires every subsequent procedure to be aware of this mapping.

Therefore, we decide to keep the original sequence structure of events as a seperate matrix and complementary to the remaining event attributes. If required, we turn the label encoded activities ad-hoc to one-hot encoded vectors. Thus, this *hybrid-vector-representation* grants us greater flexibility. However, we now need to process two matrices. The first matrix has the dimensions (case-index, max-sequence-length) and the latter (case-index, max-sequence-length, feature-attributes).

2.6 Long-Short-Term Memory Models

In order to explain the decisions of a prediction we have to introduce a predictive model, which needs to be explained. Any sequence model suffices. Additionally, the model's prediction do not have to be accurate. However, the more accurate the model can capture the dynamics of the process, the

better the counterfactual functions as an explanation of these dynamics. This becomes particularly important if the counterfactuals are assessed by a domain expert.

In this thesis, the predictive model is an Long Short-Term Memory (LSTM) model. LSTMs are well-known models within Deep Learning, that use their structure to process sequences of variable lengths[23]. LSTMs are an extension of Recurrent Neural Networks (RNNs). We choose this model as it is simple to implement and can handle long-term dependencies well.

Generally, RNNs are Neural Networks (NNs) that maintain a state h_{t+1}. The state ist computed and then propagated to act as an additional input alongside the next sequential input of the instance x_{t+1}. The hidden state h is also used to compute the prediction o_t for the current step. The formulas attached to this model are shown in

$$h_{t+1} = \sigma(Vh_t + Ux_t + b) \tag{2.5}$$

$$o_t = \sigma(Wh_t + b) \tag{2.6}$$

Here, W, U and V are weight matrices that are multiplied with their respective input vectors h_t, x_t. b is a bias vector and σ is a nonlinearity function. LSTM fundamentally work similarly, but have a more complex structure that allows to handle long-term dependencies better. They manage this behaviour by introducing addional state vectors, that are also propagated to the following step. We omit discussing these specifics in detail, as their explanation is not further relevant for this thesis. For our understanding it is enough to know that h_t holds all the necessary state information. Figure **??** shows a schematic representation of an RNN.

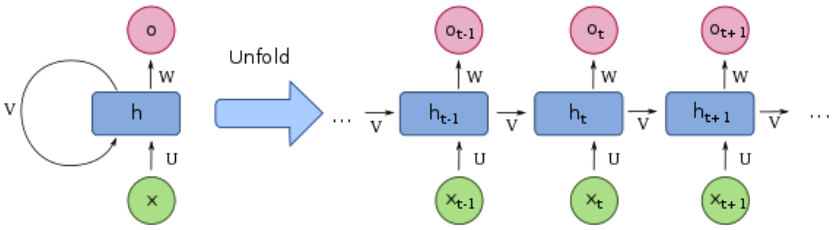


Figure 2.4: A schematic representation of an RNN viewed in compact and unfolded form**??**.

2.7 Damerau-Levenshtein

The Damerau-Levenshstein distance function is a modified version of the Levenshstein distance[31], which is a widely used to compute the edit-distance of two discrete sequences[4, 36]. The most important applications are within the NLP discipline and the Biomedical Sciences. Within these areas, we often use the Levenshtein distance to compute the edit-distance between two words, two sentences or two DNA sequences. Note, that the elements of these sequences are often atomic symbols instead of multidimensional vectors. Generally, the distance accounts for inserts, deletions and substitutions of elements between the two serquences. Damerau modified the distance function to allow for transposition operations. For Process Mining, transpositions are important as one event can transition into two events that are processed in parallel and may have varying processing times. In Figure 2.5, we schematically show two sequences and their distance.

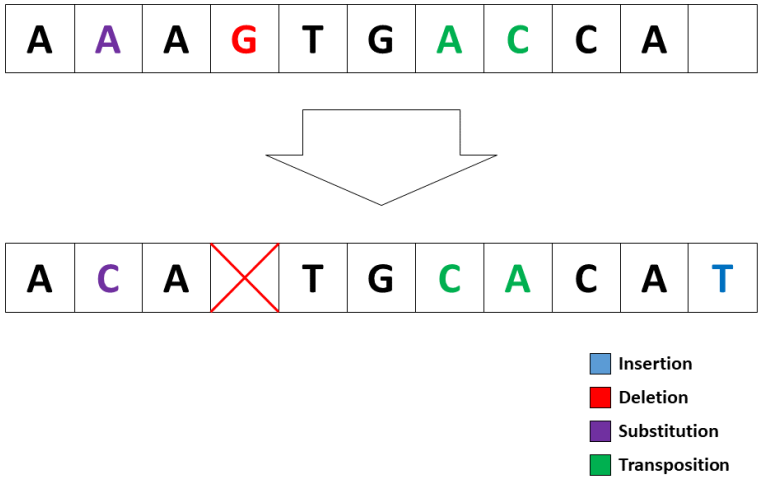


Figure 2.5: Shows two sequences. The edit distance is the sum of multiple operations. Blue shows an insert, red a deletion, purple a substitution and green a transposition. Therefore the edit distance is 4.

Equation 2.7 depicts the recursive formulation of the distance. The distance computes the costs of transforming the sequence a to b, by computing the

minimum of five seperate terms.

$$d_{a,b}(i,j) = \min \begin{cases} d_{a,b}(i-1,j)+1 & \text{if } i > 0 \\ d_{a,b}(i,j-1)+1 & \text{if } j > 0 \\ d_{a,b}(i-1,j-1)+1 & \text{if } i,j > 0 \\ d_{a,b}(i-2,j-2)+1 & \text{if } i,j > 1 \wedge a_i = b_{j-1} \wedge a_{i-1} = b_j \\ 0 & \text{if } i = j = 0 \end{cases} \tag{2.7}$$

The recursive form $d_{a,b}(i,j)$ for sequences a and b with respective elements i and j takes the minimum of each of each allowed edit operation. In particular, no change, deletion, insertion, substitution and transposition. For each operation, the algorithm adds an edit cost of 1.

We cannot use the Damerau-Levenshstein distance for process mining, if the process carries additional information about event attributes. Mainly, because two events may be emitted by the same activity, but they may still carry different event attributes.

To illustrate the issue, we explore a couple of examples. Lets assume, we have two strings $s^1 = aaba$ and $s^2 = acba$. Using the Damerau-Levenshstein distance, the edit distance between both sequences is 1, as we can recognise a substitution at the second position in both strings. However, this representation is insufficient for process instances. Therefore, we now characterise the two sequences as process events rather than strings in Equation 2.8.

$$s^1 = \{a, a, b, a\} \tag{2.8}$$
$$s^2 = \{a, a^*, b, a\} \tag{2.9}$$
$$s^3 = \{a, c, b, a\} \tag{2.10}$$
$$s^4 = \{a, a, b\} \qquad a, b, c \in \mathbb{R}^3 \tag{2.11}$$
$$a = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix} \; a^* = \begin{bmatrix} 3 \\ 3 \\ 4 \end{bmatrix} \; b = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \; c = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix} \tag{2.12}$$

If we do not consider attribute values, it becomes clear that s^2, s^3 and s^4 have an edit-distance to s^1 of 0, 1 and 1. However, with attribute values in mind, s^1 and s^2 display clear differences. Similarly, s^1 and s^3 not only differ in terms of activity but also attribute value. Lastly, s^1 and s^4 are the same in attribute values, but one element still misses entirely. It appears unintuitive that each of these differences are associated with the same cost. The examples show that we can neither disregard attribute values nor events, while computing the edit distance of two process instances.

Instead, we have to define a cost function which takes attribute variables into account. Therefore we modify the Damerau-Levenshstein distance by introducing a cost function instead of a static cost. Here, the cost of each edit-type is determined by a distance-function, which considers the difference between event-attributes. Therefore, we propose an edit-function, which captures structural sequence differences, as well as, content related differences. Going back to our example, if assume our cost function to only count differences in attributes, then the difference between s^1 and s^2 shall be 2 as their activities are the same, but the first two event attributes are different. To illustrate the structural elements, the difference between s^1 and s^3 shall be 3 instead of 2. Even if both a and c have two common event attributes, the activities they represent are still different. For instance, if both s^1 and s^3 were medical processes and a and c represented taking a cancer drug or a placebo, anyone would understand both activities are different even if the patient took the same dosage.

2.8 Evolutionary Algorithms

Many of our generative models are based on Evolutionary Algorithms. This section provides a small overview about this optimization technique.

All evolutionary algorithms use ideas that resemble the process of evolution. There are four broad categories: A Genetic Algorithm (GA) uses bit-string representations of genes, while Genetic Programming (GP) uses binary codes to represent programs or instruction sets. Evolutionary Strategy (ES) require the use of vectors. Lastly, Evolutionary Programming (EP), which closely resembles ES, without imposing a specific data structure type CITE . Our approach falls into the category of GA. The most vital concept in this category is the *gene* representation. For our purposes, the gene of a sequence consists of the sequence of events within a process instance. Hence, if an offspring inherits one gene of a parent, it inherits the activity associated with the event and all event attributes.

Our goal is to generate candidates by evaluating the sequence based on our viability measure. Our measure acts as a fitness function. The cadidates that are deemed fit enough are subsequently selected to reproduce offspring. The offspring is subject to mutations. Then, we evaluate the new population repeat the procedure until a termination condition is reached. It differs from Deep Learning, because it does not require us to use differentiable functions. Hence, we can directly optimise the viability measure established in section 3.3.

For the algorithm, we follow a rigid structure of of operations as outlined

in **??**. As **??** shows, we define 5 fundamental operations. Initiation, Selection, Crossover, Mutation and Recombination.

Algorithm 1 Shows the basic structure of an evolutionary algorithm.

Require: Hyperparameters
Ensure: The result is the final population
  $population \leftarrow$ INIT $population$;
  while not $termination$ **do**
    $parents \leftarrow$ SELECT $population$;
    $offspring \leftarrow$ CROSSOVER $parents$;
    $mutants \leftarrow$ MUTATE $offspring$;
    $survivors \leftarrow$ RECOMBINE $population \cup mutants$;
    $termination \leftarrow$ DETERMINE $termination$
    $population \leftarrow survivors$
  end while

Initiation

The inititiation process refers to the creation of the initial set of candidates for the selection process in the first iteration of the algorithm. Often, this amounts to the random generation of individuals. In this thesis, we call this method the *Random-Initiation*. However, choosing among a subset of the search space can allow for a faster convergence. We chose to implement three different subspaces as a starting point. First, by sampling from the data distribution of the Log (*Sampling-Based-Initiation*). Second, by picking individuals from a subset of the Log (*Case-Based-Initiation*). **And lastly, we can use the factual case itself as a reasonable starting point (*Factual-Initiation*).**

Selection

The selection process chooses a set of individuals among the population according to a selection procedure. These individuals will go on to act as material to generate new individuals. Again, there are multiple ways to accomplish this. In this thesis, we explore three methods. First, the *Roulette-Wheel-Selection*. Here, we compute the fitness of each indivdual in the population and choose a random sample proportionate to their fitness values. Next, the *Tournament-Selection*, which randomly selects pairs of population individuals and uses the individual with the higher fitness value to succeed. Last, we select individuals based on the elitism criterion. In other words, only a top-k

amount of individuals are selected for the next operation (*Elitism-Selection*). This approach is deterministic and therefore subject to getting stuck in local minima.

Crossover

Within the crossover procedure, we select random pairing of individuals to pass on their characteristics. Again allowing a multitude of possible procedures. We can uniformly choose a fraction of genes of one individual (*Parent 1*) and overwrite the respective genes of another individual (*Parent 2*). The result is a new individual. We call that (*Uniform-Crossover*). Figure 2.6 shows a simple schematic example. By repeating this process towards the opposite direction, we create two new offsprings, which share characteristics of both individuals. The amount of inherited genes can be adjusted using a rate-factor. The higher the crossover-rate, the higher the risk of disrupting possible sequences. If we turn to Figure 2.6 again, we see how the second child has 2 repeating genes at the end. If a process does not allow the transition from *activity 8* to another *activity 8*, then the entire process instance becomes infeasible.

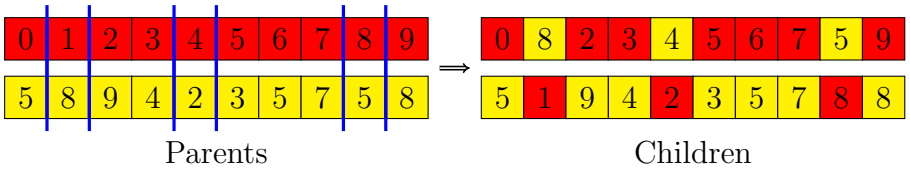


Figure 2.6: A figure showing the process of uniformly applying characteristics of one sequence to another.

The second approach is suituable for sequential data of same lengths. We can choose a point in the sequence and pass on genes of *Parent 1* onto the *Parent 2* from that point onwards and backwards (*One-Point-Crossover*). Thus, creating two new offsprings again as depicted in Figure 2.7.

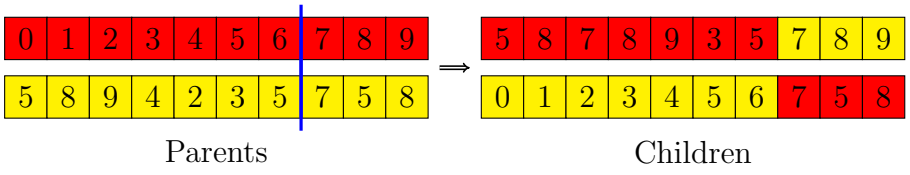


Figure 2.7: A figure showing the process of applying characteristics of one sequence to another using one split point

The last option is called *Two-Point-Crossover* and resembles its single-point counterpart. However, this time, we choose two points in the sequence

and pass on the overlap and the disjoints to generate two new offsprings. Again, Figure 2.8 describes the procedure visually.

Obviously, we can increase the number of crossover points even further. However, this increase comes at the risk of disrupting sequential dependencies.

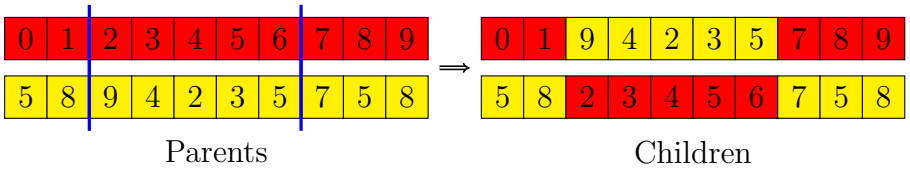


Figure 2.8: A figure showing the process of applying characteristics of one sequence to another using two split points.

Mutation

Mutations introduce random pertubations to the offsprings. Here, we apply only one major operation. However, the extend in which these mutations are applicable can still vary.

Before elaborating on the details, we have to briefly discuss four modification types that we can apply to sequences of data. Reminiscent of edit distances, which were introduced earlier in this thesis, we can either insert, delete or change a step. These edit-types are the fundamental edits we use to modify sequences. For a visual explanation of each edit-type we refer to Figure 2.5 in section 2.7.

However, we can change the rate to which each operation is applied over the sequence. We call these parameters *mutation-rates*. In other words, if the delete-rate equals 1 every individual experiences a modification which results in the deletion of a step. Same applies to other edit types. Further, we modify the amount to which each modification applies to the sequence. We call this rate *edit-rate* and keep it constant accross every edit-type. Meaning, if the edit-rate is 0.5 and the delete-rate is 1, then each individual will have 50% of their sequence deleted.
There are still three noteworthy topics to discuss.

First, these edit-types are disputable. One can argue, that change and transpose are just restricted versions of delete-insert compositions. For instance, if we want to change the activity *Buy-Order* with *Postpone-Order* at timestep 4, we can first, delete *Buy-Order* and insert *Postpone-Order* at the same place. Similar holds for transpositions, albeit more complex. Hence, these operations would naturally occur over repeated iterations in an evolutionary algorithm.

However, these operations follow the structure of established edit-distances like the Damerau-Levenshstein distance. Furthermore, they allow for efficient restrictions with respect to the chosen data encoding. For instance, we can restrict delete operations to steps that are not padding steps. In constras insert operations can be restricted to padding steps only.

Second, we could introduce different edit-rates for each edit-type. However, this adds additional complexity and needlessly increases the search space for hyperparameters.

Third, as we chose the hybrid encoding scheme, we have to define what an insert or a change means for the data. Aside from changing the activity, we also have to choose reasonable data attributes. This necessity requires to define two ways to produce them. We can either choose the features randomly, or choose to sample from a distribution which depends on the previous activities. We name the former approach *Default-Mutation*. We can simplify the latter approach by invoking the markov assumption and sample the feature attributes given the activity in question (*Sample-Based-Mutation*).

Recombination

This operation decides which individuals remain in the population for the next iteration[1]. Here, we introduce three variations.

We name the strict selection of the best individuals among the offsprings and the previous population *Fittest-Survivor-Recombination*. This recombiner strictly optimizes the population and is susceptible to getting stuck in local minima. In contrast, we name the addition of the top-k best offsprings to the initial population *Best-of-Breed-Recombination*. The former will guarantee, that the population size remains the same across all iterations but is prone to local optima. The latter only removes individuals after a population threshold was reached. Afterwards, the worst indivduals are removed to make way for new individuals. Furthermore, we propose one additional recombination operator. The operator selects the new population in a different way than the former recombination operators. Instead of using the viability directly, we sort each individuum by every viability component, seperately. This approach allows us to select individuals regardless of the scales of every individual viability measure. We refer to this method as *Ranked-Recombination*.

[1]We have to point out that in the literature, recombination is often synonymous with crossover. Both steps are similar in their filtering purpose. However, the selector filters potential parents while the recombiner filters the population. However, in this thesis recombination refers to the update process which generates the next population.

Chapter 3

Methods

In this chapter, we describe details of our framework [Name it] and discuss advantages and limitations. Therefore, we we provide a more detailed overview and additionally describe all components. As the framework resembles the work of Hsieh et al., we will also discuss differences and similarities between both solutions.

3.1 Methodological Framework

3.1.1 Architecture

To generate counterfactuals, we will need to establish a conceptual framework, which consists of three main components. The three components are shown in Figure 3.1.

[Change the names of the measures.]

The first component is a predictive model. As we attempt to explain model decisions with counterfactuals, the model needs to be pretrained. We can use any model that can predict the probability of a sequence. This condition holds for models trained for process outcome classification and next-activity prediction. The model used in this thesis is a simple LSTM model using the process log as an input. The model is trained to predict the next action given a sequence.

The second component is a generative model. The generative model produces counterfactuals given a factual sequence. In our approach, each generative model should be able to generate a set of counterfactual candidates given one factual sequence. Hence, we cannot use purely deterministic generators. Therefore, we compare two different generative approaches against a baseline. The baseline uses a simple case based heuristic. The generative

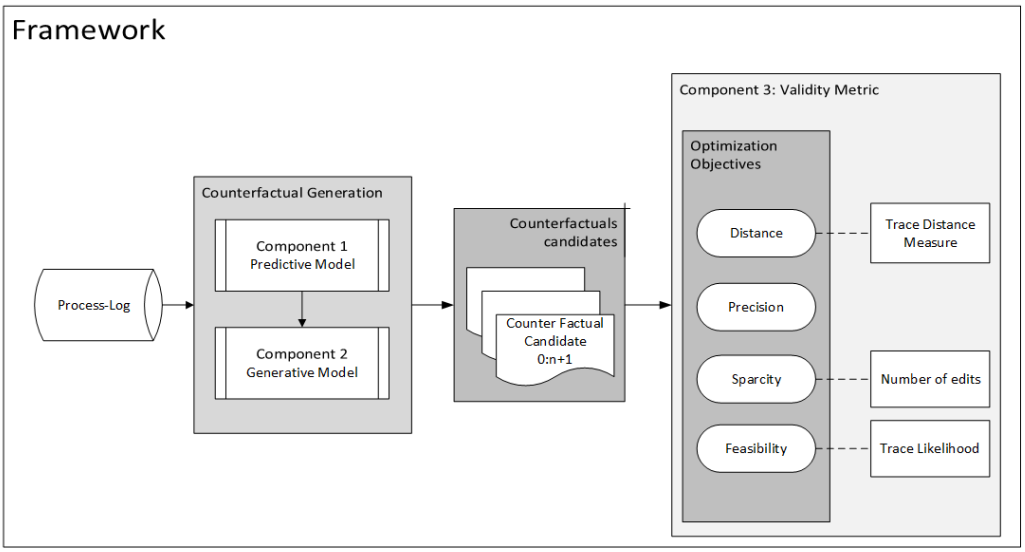


Figure 3.1: Shows the methodoligical framework of this thesis. The input is the process log. The log will be used to train a predictive model (Component 1) and the generative model (Component 2). This process produces a set of candidates which are subject to evaluation via the validity metric (Component 3).

approaches are a sequential a deep generative model and an evolutionary technique. Both methods allow us to use a factual sequence as starting point for the generative production but also generate multiple variations of the final solution.

The generated candidates are subject to the third major component's scrutiny. To select the most *viable* counterfactual candidate, we evaluate their viability score using a custom metric. The metric incorporates all main criterions for viable counterfactuals mentioned in section 2.3. We measure the *similarity* between two sequences using a multivariate sequence distance metric. The *precision* of the prediction will compare the likelihood of the counterfactual outcome without changes to the sequence and the counterfactual candidates. For this purpose, we require the predictive model, as it will compute the likelihoods. We measure *sparcity* by counting the number of changes in the features and computing the edit distance. Lastly, we need to determine the feasibility of a counterfactual. This requires splitting the feasibility into two parts. First, the likelihood of the sequence of each event and second, the likelihood of the features given the event.

3.1.2 Differences to DiCE4EL

Hsieh et al. has recently published a paper on the counterfactual generation of process data. They call their framework DiCE4EL and shares many ideas

with our framework. Therefore, we want to highlight the key differences and similarities.

In their approach they attempt to solve various issues that we have also highlighted in section 1.2. Furthermore, they do so by incrementally generating the model in a sequential order. However, unlike Oberst and Sontag, whose solution creates counterfactuals for every step in the sequence, Hsieh et al. focus on critical decision points they call milestones.

To gain a better understanding, it is important to briefly outline the event log the authors use. It was taken from a Dutch bank which processes loan applications in which customers request a certain amount of money. The activities relate to either application states or manual work activities. The application states consist of tasks generated by a machine and manual work activities produced by humans. Hence, the manual work items occur in reference to the application state. For instance, if the loan application is in a *pre accepted* state, then the next events are often produced by humans who are reviewing the state. Those events are essentially, sub-events of the application state. The human activities do not have to happen sequentially. They can occur in parallel. The moment all manual work items conclude, marks the decision for the next appication state. For instance, from *pre accepted* to *accepted*. Now, to understand why the milestone approach works, requires to know that an application loan process will change to a rejection state, for instance, if all manual work items are completed. There will not be applications that suddenly switch to another state although the work items of a previous state have not concluded, yet. Thus, one can split the entire sequence into subsequences or ignore the sub events entirely, which reduces the search space significantly.

One issue with this approach is the fact that one would first have to identify these milestones. Hence, a crucial distinction to our proposed framework, is their dependence on knowledge about the true process as displayed in this section. Our framework does not leverage structural information about the true process model in question. We believe this is the core contribution in constrast to their approach.

However, similarities between both frameworks do exist. Mainly, our approach also relies on prediction scores of the model we attempt to explain. Similar to Hsieh et al., we incorporate these scores into our quality measure.

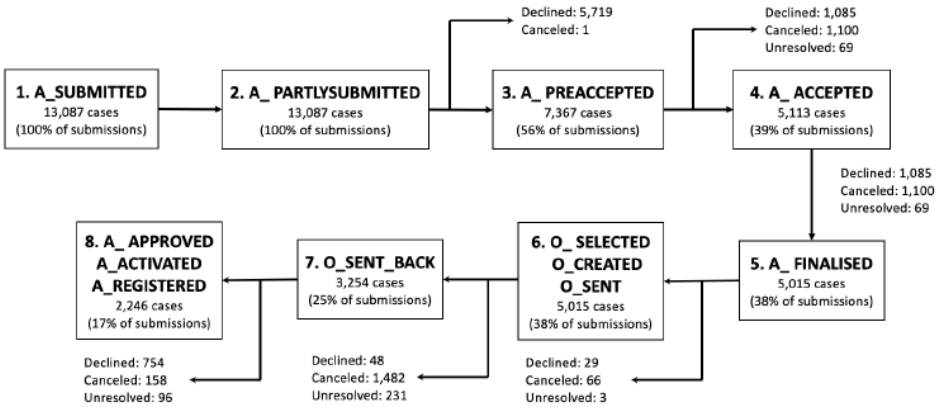


Figure 3.2: Milestones of loan application process captured in BPIC2012 as identified in [7]

3.2 Semi-Structured Damerau-Levenshtein Distance

Before discussing the viability function, we have to introduce an edit-distance for sequences. An edit-distance is used to compute distance between two sequences. Therefore, some distances can take *structural* characteristics of sequences into account. Hence, we show why it is important to take those structures into account and propose a custom edit-distance, which works for business processes.

3.2.1 Semi-Structured Damerau Levenshtein

In order to reflect these differences in attribute values, we introduce a modified version of the Damerau-Levenshstein distance, that not only reflects the difference between two process instances, but also the attribute values. We achieve this by introducing a cost function $cost_{a_i,b_j}$, which applies to a normed vector-space[1]. Concretely, we formulate the modified Damerau-Levenshstein distance as shown in Equation 3.1. For the remainder, we will denote this

edit-distance as Semi-strucured Damerau-Levenshtein distance (SSDLD).

$$d_{a,b}(i,j) = \min \begin{cases} d_{a,b}(i-1,j) + cost(\mathbf{0}, b_j) & \text{if } i > 0 \\ d_{a,b}(i,j-1) + cost(a_i, \mathbf{0}) & \text{if } j > 0 \\ d_{a,b}(i-1,j-1) + cost(a_i, b_j) & \text{if } i,j > 0 \\ & \& \ \overline{a}_i = \overline{b}_j \\ d_{a,b}(i-1,j-1) + cost(a_i, \mathbf{0}) + cost(\mathbf{0}, b_j) & \text{if } i,j > 0 \\ & \& \ \overline{a}_i \neq \overline{b}_j \\ d_{a,b}(i-2,j-2) + cost(a_i, b_{j-1}) + cost(a_{i-1}, b_j) & \text{if } i,j > 1 \\ & \& \ \overline{a}_i = \overline{b}_{j-1} \\ & \& \ \overline{a}_{i-1} = \overline{b}_j \\ 0 & \& \ i = j = 0 \end{cases} \tag{3.1}$$

Here, $d_{a,b}(i,j)$ is the recursive form of the Damerau-Levenshtein-Distance. a and b are sequences and i and j specific elements of the sequence. $cost(a,b)$ is a cost function which takes the attribute values of a and b into account. The first two terms correspond to a deletion and an insertion from a to b. The idea is to compute the maximal cost for that the wrongfully deleted or inserted event. The third term adds the difference between two events with identical activities $\overline{a}_i$ and $\overline{b}_j$. As mentioned earlier, two events that refer to the same activity can still be different due to event attributes. The distance between the event attributes determines *how* different these events are. The fourth term handles the substitution of two events. Here, we compute the substitution cost as the sum of an insertion and a deletion. The fifth term computes the cost after transposing both events. This cost is similar to term 3 only that we now consider the differences between both events after they were aligned. The last term relates to the stopping criterion of the recursive formulation of the Damerau-Levenshstein distance.

[FOR XIXI: Are there any remarks about the formulas?]

3.2.2 Discussion

There are two important characteristics, which require a discussion.

First, if we assess the first two terms, we use $cost(x,0)$ to denote the maximal distance of inserting and deleting x. $cost(x,0)$ can be read as cost

[1] A normed vector-space is a vector space, in which all vectors have the same dimensionality. For instance, if all vectors have three dimensions, we can call the vector-space *normed*.

between x and a null-vector of the same size. However, it is noteworthy to state that this interpretration does not hold for any arbitrary cost-function. For instance, the cosine-distance does not work with a null vector, as it is impossible to compute the angle between x and a null vector. Here, the maximum distance would just amount to 1. In contrast, the family of Minkowsky distance works well with this notion, because they compute a distance between two points and not two directions.

Second, to explain the intuition behind most terms, it is necessary to establish a common understanding between the relationship of an event with its event attributes. Generally, we can have two notions of this relationship.

For the first relationship, we consider the event and its attributes as seperate entities. This notion is reasonable, as some attributes remain static throughout the whole process run. If we take a loan application process as an example, an applicant's ethnic background will not change regardless of the event. This characteristic can be considered a case attribute, which remains static throughout the process run. This understanding would require us to modify the cost functions, as they treat the activity independently from its attribute values. In other words, if the activities of two events are $\overline{a}$ and $\overline{b}$, but their attribute values are $\binom{2}{3}$ and $\binom{2}{3}$, these events may be seen as more similar than two $\overline{a}$ and $\overline{a}$ with attribute values $\binom{2}{3}$ and $\binom{5}{0}$.

A second notion would treat each event as an independent and atomic point in time. Hence, a and b would be considered completely different even if their event attributes are the same. This understanding is also a valid proposition, as you could argue that an event which occurs at nighttime is not the same event as an event at daytime. Here, the time domain is the main driver of distinction and the content remains a secondary actor.

All the terms described in the SSDLD follow the second notion. There are two reasons for this decision. First, treating event activities and event attributes seperately would further complicate the SSDLD, as we would have to expand the cost structure. Second, the unmodified Damerau-Levenshstein distance applies to discrete sequences, such as textual data with atomic words or characters. By treating each event as an discrete sequence element, we remain faithful to the original function.

3.3 Viability Measure

In this section we define the components of the metric to measure the validity of a sequence.

3.3.1 Similarity-Measure

We use a function to compute the distance between the factual sequence and the counterfactual candidates. Here, a low distance corresponds to a small change. For reasons explained earlier (section 3.2), we want to take the structural distance and the feature distance into account. Henceforth, we will use the previously established SSDLD. The similarity distance uses a cost function as specified in Equation 3.2.

$$\begin{aligned} cost(a_i, b_j) &= L2(a_i, b_j) \\ a_i, b_j &\in \mathbb{R}^d \end{aligned} \tag{3.2}$$

Here, $dist(x, y)$ is an arbitrary distance metric. i and j are the indidices of the sequence elements a and b, respectively.

3.3.2 Sparcity-Measure

Sparsity refers to the number of changes between the factual and counterfactual sequence. We typically want to minimize the number of changes. However, sparcity is hard to measure, as we cannot easily count the changes. There are two reasons, why this is the case: First, the sequences that are compared can have varying lengths. Second, even if they were the same length, the events might not line up in such a way, that we can simply count the changes to a feature. Hence, to solve this issue, we use the previously established SSDLD. The sparcity distance uses a cost function as specified in Equation 3.3.

$$\begin{aligned} cost(a_i, b_j) &= \sum_d \mathbb{I}(a_{id} = b_{jd}) \\ a_i, b_j &\in \mathbb{R}^d \end{aligned} \tag{3.3}$$

Here, $\sum_d \mathbb{I}(a_{id} = b_{jd})$ is an indicator function, that is used to count the number of changes in a vector.

3.3.3 Feasibility-Measure

To determine the feasibility of a counterfactual trace, it is important to recognise two components.

First, we have to compute the probability of the sequence of event transitions. This is a difficult task, given the *Open World assumption*. In theory,

we cannot know whether any event *can* follow after another event or not. However, if the data is representative of the process dynamics, we can make simplifying assumptions. For instance, we can compute the first-order transition probability by counting each transition. However, the issue remains that longer sequences tend to have a zero probability if they have never been seen in the data.

Second, we have to compute the feasibility of the individual feature values given the sequence. We can relax the computation of this probability using the *Markov Assumption*. In other words, we assume that each event vector depends on the current activity, but none of the previous events and features. Meaning, we can model density estimators for every event and use them to determine the likelihood of a set of features. **Hence, we compute the joint probability of a case by using the forward algorithm** **CITE forward algorithm** .

There are many ways to estimate the density of a data set. For our purposes, we incorporate the sequential structure of the log data and make simplifying assumptions. First, we consider every activity as a state in the case. Second, each state is only dependent on its immediate predecessor and neither on future nor on any any states prior to its immediate predecessor. Third, the collection of attributes within an event depend on the activity which emits it. The second assumption is commonly known as *Markov Assumption*. With these assumptions in place, we can model the distribution by knowing the state transition probability and the density to emit a collection of event attributes given the activity.

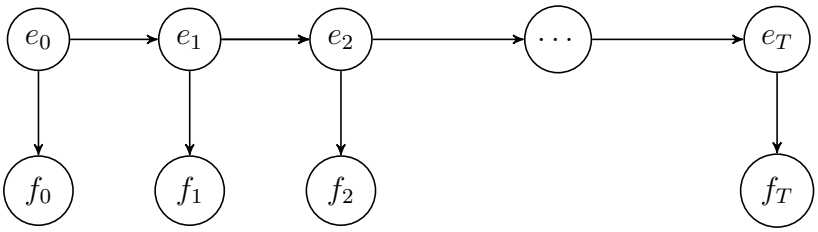


Figure 3.3: The feaibility model in graphical form. e_t represents an event and f_t the features it emits.

Here, e_t represents the transition from one event state to another. Likewise, f represent the emission of the feature attributes. Hence, the probability of a particular sequence is the product of the transition probability multiplied with state emission probability for each step. Note, that this is the same as the feasibility measure as in Equation 3.4.

$$p\left(e_{0:T}, f_{0:T}\right) = p\left(e_0\right) p\left(f_0 \mid e_0\right) \prod_{1}^{T} p\left(e_t \mid e_{t-1}\right) p\left(f_t \mid e_t\right) \tag{3.4}$$

To conclude this section, we have to stress again, that there are many ways to define feasibility. We chose a probablistic approach. There is an issue with this approach. Shorter sequences naturally have higher probabilities. Hence, we introduce a bias into our viability measure towards short sequences. This bias can be beneficial or detrimental depending on the use case. For instance, a medical process model might favor shorter counterfactual explanations. Mainly, because we want to understand how we can effectively reduce the time of illness. However, if we want to explain a highly standardised manufacturing process that went wrong in one instance; then, we would rather keep the counterfactual as close as possible to the factual.

3.3.4 Delta-Measure

[FOR XIXI: Do you think that this belongs to background, too?] For this measure, we evaluate the likelihood of a counterfactual trace by determining whether a counterfactual leads to the desired outcome or not. For this purpose, we use the predictive model, which returns a prediction for each counterfactual sequence. As we are predicting process outcomes, we typically predict a class. However, it is often difficult to force a deterministic model to produce a different class prediction. Therefore, we can relax the condition by maximising the prediction score of the desired counterfactual outcome[37]. If we compare the difference of the counterfactual prediction score with the factual prediction score, we can determine an increase or decrease. Ideally, we want to increase the likelihood of the desired outcome. Therefore, we compute the difference between the two prediction scores. We refer to this value as *delta*, as it reflects the delta between both prediction scores. Equation 3.5 shows the corresponding formula.

$$delta = p(o|s^*) - p(o|s) \tag{3.5}$$

Here, $p(o|s)$ describes the probability of an outcome o given a sequence of events s. s^* denotes a counterfactual sequence.

3.3.5 Discussion

Given the current viability function we can already determine the optimal counterfactual:

> The optimal counterfactual flips the strongly expected factual outcome of a model to a desired outcome, maintaining the same trajectory as the factual in terms of events, with minimal changes its event attributes, while remaining feasible according to the data.

The elements that fulfill these criteria make up the pareto surface of this multi-valued viability function. If each of the values are scaled a range between 0 and 1, the theoretical ceiling is 4. This value is only possible if we can flip the outcome of a factual sequence without chaning it. As this is naturally impossible for deterministic model predictions, the viability has to be lower than 4.

Furthermore, we can already postulate, that a viability of 2 is an important threshold. If we score the viability of a factual against itself, a normalised sparcity and similarity value have to at its maximal value of 1. In contrast, the improvement has to be 0. The feasibility is 0 depending on whether the factual was used to estimate the data distribution or not. **With these observations in mind, we determine that any counterfactual with a viability of at least to is a considerable counterfactual.**

3.3.6 Differences to DiCE4EL

Hsieh et al. follow a very similar pattern of assessing the quality of their counterfactuals. The authors also focus on the aspects similarity, sparcity, feasibility and likelihood improvement. However, they incorporate and operationalize them differently. Their approach is mostly apparent in their loss function.

[Maybe write the losses in title-case]

Similarity: Similar to our approach, the authors use a distance function and optimize it using gradient descent. They evaluate the quality of their counterfactuals using the same function[2]. However, we use a modified Damerau-Levenshstein distance algorithm to also incorporate structural differences such as the sequence lengths or transposed events.

Sparcity: The authors do not optimize towards sparcity, but assess it during their evaluation.

Feasibility: This quality criterion is is embodied by two loss functions: The category loss and scenario Loss. The category loss ensures that categorical variables remain categorical after generation. The scenario loss adds

[2]They call it proximity during evaluation

emphasis on only generating counterfactuals that are in the event log. Unlike our probablistic interpretation, they treat the existence of feasible counterfactuals as a binary criterion[3].

Likelihood: Similar to the authors' scenario loss, they treat the improvement of a class as a binary state. Either the counterfactual changes the model's prediction to the desired class or it does not.

The details of each criterion's operationalisation, are explained in [25]. By assessing their interpretation of quality criterions, we see the clear distinction between our approach and the approach of Hsieh et al.

First, their viability measure decisively discourages the generation of counterfactuals that are not present in the dataset. In constast, our approach treats this aspect as a soft constraint.

Second, while our approach also acknowledges general improvements in likelihoods, DiCE4EL treats all counterfactuals that do not lead to better desired as dentrimental solutions. However, one can argue that improving the likelihood of a desired outcome just slightly is already beneficial.

Third, [25] do not optimize sparcity, while we include it within our framework. One can argue that similarity automatically incorporates aspects of sparcity, but we disagree with this notion. We can see this by employing a simple example: **[This example can be improved, either by referencing a former example or explaining A, B, C and the event attributes at the begiining.]** Let factual A have features signifying the biological sex (binary), the income (normalized) and the age (normalized) $\begin{pmatrix}1\\1\\1\end{pmatrix}$ as event attributes. Let counterfactual B have the same event attributes with $\begin{pmatrix}0\\1\\1\end{pmatrix}$. Let's assume the distance measure uses the L1-norm. Then, a counterfactual C with event attributes $\begin{pmatrix}1\\0.5\\0.5\end{pmatrix}$, would have the same distance to factual A than B has. However, C requires the change of two event attributes, while B only requires 1 change. In a scenario, in which we seek to reduce the number of edits, B is more preferable than C, regardless of the distance to A.

The last difference stems from the fact that Hsieh et al. do not include structural sequence characteristics in their similarity measure. A sequence **XXZXX** might be more similar to **XXXZX**, than **XXXXZ**. The former

[3]They call it plausibility during evaluation

requires only a transposition, while the latter requires two changes. Both have two positions that are not correct.

3.4 Prediction Model: LSTM

The architecture of the prediction model is shown in Figure 3.4. The model architecture is inspired by Hsieh et al. However, we do not seperate the input into dynamic features and and static features.

One input consists of an 2-dimensional event tensor containing integers. The second input is a 3-dimensional tensor containing the remaining feature attributes. The first dimension in each layer represents the variable batch size and *None* acts as a placeholder.

The next layer is primarily concerned with preparing the full vector representation. We encode each activity in the sequence into a vector-space. We chose a dense-vector representation instead of a one-hot representation. We also create positional embeddings. Then we concat the activity embedding, positional embedding and the event attribute representation to a final vector representation for the event that occured.

Afterwards, we pass the tensor through a LSTM module. We use the output of the last step to predict the outcome of a sequence using a fully connected neural network layer with a sigmoid activation as this is a binary classification task.

3.5 Counterfactual Generators

3.5.1 Baseline Model: Random Generator

This model acts as one of the baseline methods. Here, we generate a random sequence of events. Afterwards we generate event attributes, randomly. This approach is reasonably fast, but expected to perform poorly.

As explained earlier, a randomly sampling a possible sequence of events becomes more and more unlikely the longer the sequence is and the more events are possible. One generally has a chance of $\frac{1}{A^T}$ to randomly find an event, that is generally possible given the process model. The chances decrease even more if one also generates event attributes randomly. Therefore, we expect most models to perform better on average.

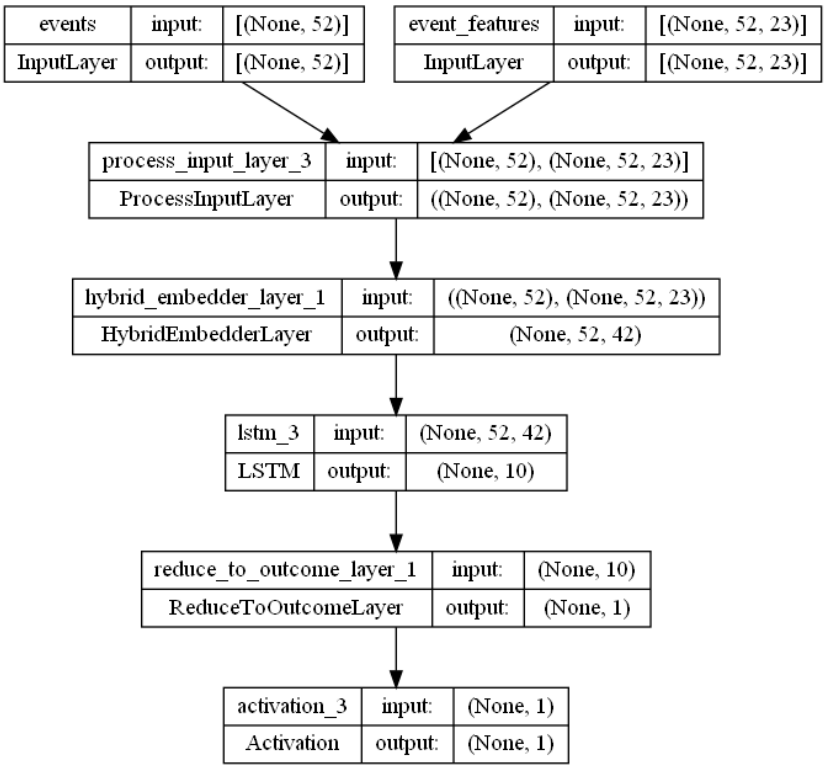


Figure 3.4: Shows the different components of the LSTM architecture. Each elements contains information about the input and output of a layer. None is a placeholder for the batch size.

3.5.2 Baseline Model: Sample-Based Generator

The last baseline resembles resembles the random baseline. However, we use the feasibility model to guide the random search for the generation of counterfactuals. We refer to the model specified in Equation 3.4. The sampling procedure utilises the model structure for the sampling process. We first generate a random seed of possible starting events ($p(e_0)$). Afterwards, we randonly sample subsequent events by iteratively sampling new activities according to the transition probabilities we gathered from the data ($\prod_1^T p(e_t \mid e_{t-1})$). Given the sequence, we simply sample the features per event from $p(f_t \mid e_t)$.

3.5.3 Baseline Model: Case-Based Generator

Case-based techniques leverage the data by using example instances. The idea is to find suitable candidates that fulfill the counterfactual criterions the best. We treat this model as a baseline. Therefore we keep this approach simple. We find candidates by searching by randomly sampling cases from the Log and then, evaluating them using the viability measure.

Inherently, this approach is restricted by the *representativeness* of the data. It is not possible to generate counterfactuals that have not been seen before. This method works for cases, in which the data holds enough information about the process. If this condition is not met, it is impossible to produce suitable candidates.

Note, that this approach will automatically fulfill the criterion of being feasible, as the counterfactuals are drawn from the Log directly. Hence, we expect their feasibility to often be higher than for other methods.

[FOR XIXI: Should I include a baseline which samples from the data distribution?]

3.5.4 Generative Model: Evolutionary Algorithm

We introduced most of the operators in section 2.8. In this section, describe the operators in detail and select a subset that we want to explore further. **[FOR XIXI: Are implementation details important?]**

Operators

We implemented a number of different evolutionary operators. Each one belongs to one of five categories. The categories are inititiation, selection, crossing, mutation and recombination.

Inititation

DI: The Default-Initiator generates an initial population entirely randomly.

SBI: The *Sampling-Based-Initiation* generates an initial population using a distribution estimated from the data.

CBI: *Case-Based-Initiation* uses examples of the data as initial population.

FI: *CFactual-Initiation* Uses the factual itself.

Selection

RWI: *Roulette-Wheel-Selection* Selects individuals randomly, but proportionate to their fitness score.

TS: *Tournament-Selection* Compares two or more individuals and selects a winner among them.

ES: *Elitism-Selection* selects each individual solely on their fitness

Crossing

OPC: *One-Point-Crossing* Chooses on point in the sequence and creates offspring by taking everything from or after that point from another individual.

TPC: *Two-Point-Crossing* Chooses two points in the sequence and creates offspring by taking everything between or outside these points from another individual.

UCx: *Uniform-Crossing* Uniformly selects positions in the sequence to take from another individual. The amount of selected positions is determined by a crossing-rate between 0 and 1.

Mutation

RM: *Random-Mutation* creates entirely random features for inserts and substitution.

SBM: *Sampling-Based-Mutation* creates sampled features based on data distribution for inserts and substitution.

Recombination

FSR: *Fittest-Survivor-Recombination* Determines the survivor among the mutated offsprings and the population.

BBR: *Best-of-Breed-Recombination* Determines better than average survivors among the mutated offsprings and adds them to the population.

RR: *Ranked-Recombination* Determines survivors based on ranking

We use abbreviations to refer to them in figure, tables and so one. For instance, *CBI-RWI-OPC-RM-PR* refers to an evolutionary operator configuration, that samples its initial population from the data, probablistically samples parents based on their fitness, crosses them on one point and so on. For the *Uniform-Crossing* operater we additionally indicate its crossing rate using a number. For instance, *CBI-RWI-UC3-RM-PR* is a model using the *Uniform-Crossing* with a child receiving roughly 30% of the genom of one parent and 70% of another parent.

Model Selection

Chapter 4

Evaluation

In this chapter, we we are going to establish most of the methods, the results section will cover. In detail, we discuss the datasets, we use, the preprocessing pipeline and the final representation for each of the algorithms.

4.1 Datasets

In this thesis, we use a multitude of datasets for generating the counterfactuals. All of the data sets where taken from Teinemaa et al. Each dataset consists of log data and contains labels which signify the outcome of a process. They were introduced by [**some author**]. We focus on binary outcome predictions. Hence, each dataset will provide information about one of two possible outcomes associated with the case. For instance, a medical process might be deemed a success if the patient is cured or a failure if the patient remains ill. A loan application process might deem granting the loan a success or the rejection as failure. The determination of the outcome depends on the use-case and the stakeholders involved. A insurence provider might deem a successful claim as a failure, while the client deems it as a success.

BPIC12 The first dataset is the popular BPIC12 dataset. This dataset was originally published for the Business Process Intelligence Conference and contains events for a loan application process. Each indivdual case relates to one loan application process and can be accepted (regular) or cancelled (deviant).

Sepsis The next dataset is the Sepsis-Dataset. It is a medical dataset, which records of patients with life-threatening sepsis conditions. The outcome describes whether the patient returns to the emergency room within 28 days from initial discharge.

TrafficFines Third, we apply our approach to the Traffic-Fines-Dataset. This dataset contains events related to notifications sent related to a fine. The dataset originates in a log from an Italian local police force.

Dice4EL Lastly, we include a variation of the BPIC dataset. It is the dataset which was used by Hsieh et al. The difference between this dataset and the original dataset is two-fold. First, Hsieh et al. omit most variables except two. Second it is primarily designed for next-activity prediction and not outcome prediction. We modified the dataset, to fit the outcome prediction model.

Table 4.1: All datasets used within the evaluation. Dice4EL is used for the qualitative evaluation and the remaining are used for quantitative evaluation purposes.

Dataset	#Cases	Min Len	Max Len	% Unique Traces	#Unique Ev.	#Data Columns	#Event Attr	#Regular	#Deviant
Dice4EL	3 051	12	25	0.000328	23	9	7	1 853	1 198
BPIC12-25	866	15	25	0.001155	32	23	21	682	184
BPIC12-50	3 728	15	50	0.000268	36	25	23	2 111	1 617
BPIC12-75	4 461	15	75	0.000224	36	25	23	2 379	2 082
BPIC12-100	4 628	15	100	0.000216	36	25	23	2 420	2 208
Sepsis25	707	5	25	0.001414	15	75	73	610	97
Sepsis50	770	5	47	0.001299	15	76	74	662	108
Sepsis75	777	5	66	0.001287	15	76	74	667	110
Sepsis100	779	5	88	0.001284	15	76	74	669	110
TrafficFines	129 615	2	20	0.000008	10	40	38	70 602	59 013

For more information about these datasets we refer to Teinemaa et al.'s comparative study[46]. We list all the important descriptive statistics in Table **??**.**[FOR XIXI: How should I cite. Is mentioning the authors name enough?]**

4.2 Preprocessing

To prepare the data for our experiments, we employed basic tactics for preprocessing. First, we split the log into a training and a test set. The test set will act as our primary source for evaluating factuals, that are completely unknown to the model. We further split the training set into a training set and validation set. This procedure is a common tactic to employ model seletection techniques. In other words, Each dataset is split into 25% Test and 75 remaining and from the remaining we take 25 val and 75 train.

First, we filter out every case, whose' sequence length exceeds 25. We keep this maximum threshold for most of the experiments that forucs on the evolutionary algorithm. The reason is . Furthermore, two components of the proposed viability measure have a run time complexity of at least 2. Hence, limiting the sequence length saves a substantial amount of ressources.

Next, we extract time variables if they are not provided in the log in the first place. Then, we convert all binary columns to the values 1 and 0. **[In cases know time relevant information is avaible, we will XXX...]**

Each categorical variable is converted using binary encoding. Binary encoding is very similar to onehot encoding. However, it is still distinct. Binary encoding uses a binary representation for each class encoded. This representation saves a lot of space as binary encoded variables are less sparse, than one-hot encoded variables. **[from different from onehot encoding as we, encode each categorical as binary value rather than .]**

We also add an offset of 1 to binary and categorical columns to introduce a symbol which represents padding in the sequence. All numerical columns are standardized to have a zero mean and a standard deviation of 1.

We omit the case id, the case activity and label column from this preprocessing procedure, for reasons explained in section 2.5. The case activity are label-encoded. In other words, every category is assigned to a unique integer. The label column is binary as we focus on outcome prediction.

The entire pipeline is visualized in Figure 4.1.

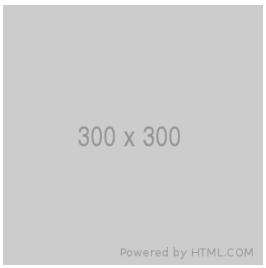


Figure 4.1: shows the entire preprocessing pipeline based on an example case.

4.3 Approach

As mentioned in section 2.3, counterfactual generation is notorious for their lack of a standardised evaluation procedure. Nonetheless, we attempt to address our research questions with the following experiments.

Experiment 1: Model Selection

Before comparing models, it is important to reduce the number of possible models that *can* be compared. Especially, the evolutionary generator has a number of free parameters. These range from structural configurations to general hyperparameters. In terms of operators, we introduced 4 initiators, 3 selectors, 3 crossers, 2 mutators and 3 recombiners. Hence, comparing all

possible eolutionary operator combinations requires to test a total of 216 different models. Furthermore, each model has hyperparameters, we have to define, too. Therefore, the first set of experiments are dedicated to choose among a subset of operator combinations and subsequently select appropriate hyperparameters.

First, we compute all possible configurations, without changing any hyperparameter. To avoid confusion, we refer to each unique operator combination as a model-configuration. For instance, one model-configuration would consist of **[a SamplingBasedInitiator, an ElitismSelector, a OnePointCrosser, SamplingBasedMutator and a FittestSurvivorRecombiner]**. We refer to a specific model-configuration in terms of its abbreviated operators. For instance, the earlier example is denoted as **[SBI-ES-OPC-SBM-FSR]**.

Afterwards, we explore the hyperparameters of the model. We start with the termination point. Hence, we want to explore the effects of the iterative cycles that each evolutionary alogirthm will run for. The goal is to find a stopping criterion which yields reasonably good counterfactuals, while reducing the computation time. We will only consider the number of iterative cycles as a stopping criterion. We refer to each different criterion as termination point. Hence, a termination point at 5 means the algorithm, will not proceeed to optimize its results, further after reaching the fifth iteration. We can choose the termination point by inspecting how the average population viablity evolves across each cycle. We keep every other experimental setting as established beforehand.

We determine an appropriate number of individuals we generate in every iterative cycle and a population size. We test both together, as they are dependent on each other. We keep every other experimental setting as before and only experiment on the model-configurations selected prior. Our goal is to find the optimal ratio between children generated and population size.

For determining the mutation rate for every mutation type, we choose the best evolutionary algorithm and run the configuration with 6 rates from 0 to 0.5 in steps of 0.1. We omit everything beyond 0.5 to preserve information about the parent. For instsnce, if we use a change rate of 0.9, we mutate 90% of the genes the child inherited. This would defeat the purpose of evolving better counterfactuals through breeding. We use the termination point established in the prior experiment. We keep every other experimental setting as established beforehand.

After, executing all preliminary experiments we choose the evolutionary generators and compare them with all baseline models in all subsequent experiments.

Experiment 2: Model Comparison

First, we assess the viability of **[a number of]** models. For this purpose, we sample **[10]** factuals and use the models to generate **[50]** counterfactuals. We determine the **[mean]** viability across the counterfactuals. With this experiment, we show that a model which optimizes quality criteria of counterfactuals produces better results than models, which do not. Hence, we expect the evolutionary algorithm to perform best, as it can directly optimize multiple viability criterions. In the following we list all models, we are going to compare:

RNG A Random-Search Generator , which generates random values and acts as a baseline.

CBG A Casebased-Search Generator , which samples from process instances within the training set

EVO A SBI-ES-OPC-SBM-FSR Generator , which optimizes viability using principles of evolution.

In accordance with *RQ1-H1* and *RQ1-H2* we expect the SBI-ES-OPC-SBM-FSR Generator to perform best among these baselines, when it comes to viability.

Experiment 3: Comparing with alternative Literature

The model comparison is not enough to establish the validity of our solution, as defined proposed the viability measure ourselves. Therefore, we also assess each model based on the evaluation criterions of an alternative work. More precisely, we quantify the viability of our models using the metrics employed by Hsieh et al. Hence, we measure the sparsity by computing the average Levenshtein difference and proximity using the L2-Norm. Furthermore, we compute the average intra-list-diversity and plausibility as well as the models capability of changing the prediction to a desired one.

Similar to Hsieh et al., we only focus on the *activities* that are generated by each model and its accompaniying *resource* event-attribute. For diversity and plausibility we remain close to the original evaluation protocol by Hsieh et al. as we will also treat each counterfactual trace sequence as a symbol. Hence, a sequeunce *ABC* is treated as a completely different symbol than *ABCD*.

The goal is to show that models, which optimise viability criterions, perform better, even if viability is assessed differently as stated in *RQ2-H1* of our research question (section 1.4).

Experiment 4: Qualitative Assessment

For the last assessment, we follow Hsieh et al.'s procedure of assessing the models qualitatively. We use the dataset as the authors do. **[FOR XIXI: Should I use the exact same examples?]** However, as we focus on outcome prediction, we attempt to answer one of two questions:

1. *what would I have had to change to prevent the cancellation/rejection of the loan application process*
2. *what would I have had to change to get cancelled/rejected of the loan application process*

The goal is to show, that the results are viable despite not having a standardized protocol to measure their viability.

Chapter 5

Results

This chapter presents the results of each evaluation step. Furthermore, we analyse the results.

5.1 Experiment 1: Model Selection

5.1.1 Model Configuration

Results

As there are many ways to combine each configuration, we select a few configurations by examining them trough simulations.

The model-configuration set contains [144] elements. We choose to run each model-configuration for [50] evolution cycles. For all model-configurations, we use the same set of [4] factual process instances, which are randomly sampled from the test set. We ensure, that the outcomes of these factuals are evenly divided. We decide to return a maximum of [1000] counterfactuals for each factual case. Within each evolutionary cycle, we generate [100] new offsprings. We keep the mutation rate at [0.1] for each mutation type. Hence, across all cases that are mutated, the algorithm deletes, inserts, and changes [1%] of events. We collect the means viability and its components across the iterative cycles of the model.
Figure 5.1 shows the bottom and top-[k] model-configurations based on the viability after the final iterative cycle. We also show how the viability evolves for each iteration.[change evolutionary cycle to iterative cycle] The results reveal a couple of patterns. First, all of the top-[k] algorithms use [either CaseBasedInitiator or SampleBasedInitiator] as initiation operation. In contrast, the bottom-[k] all use [RandomInitiator] as initialisation. Second, we see that most of the top-[k] algorithms use the [Elitism-

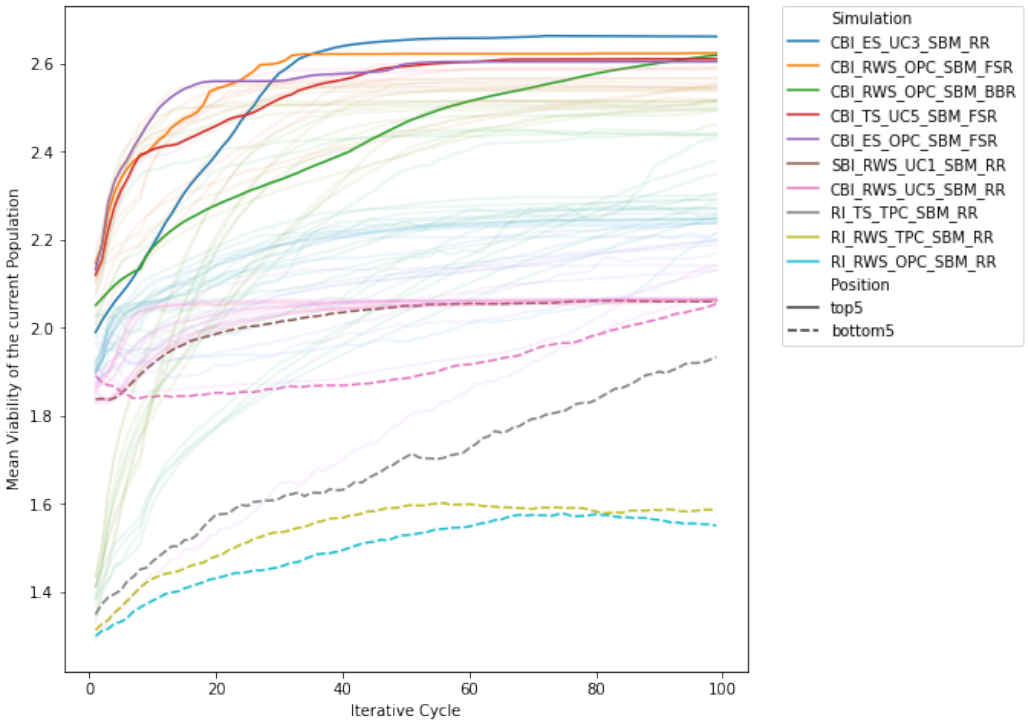


Figure 5.1: This figure shows the average viability of the [10] best and worst model-configurations. The x-axis shows how the viability evolves for each evolutionary cycle.

Selector].

The complete table of results are in **??**.

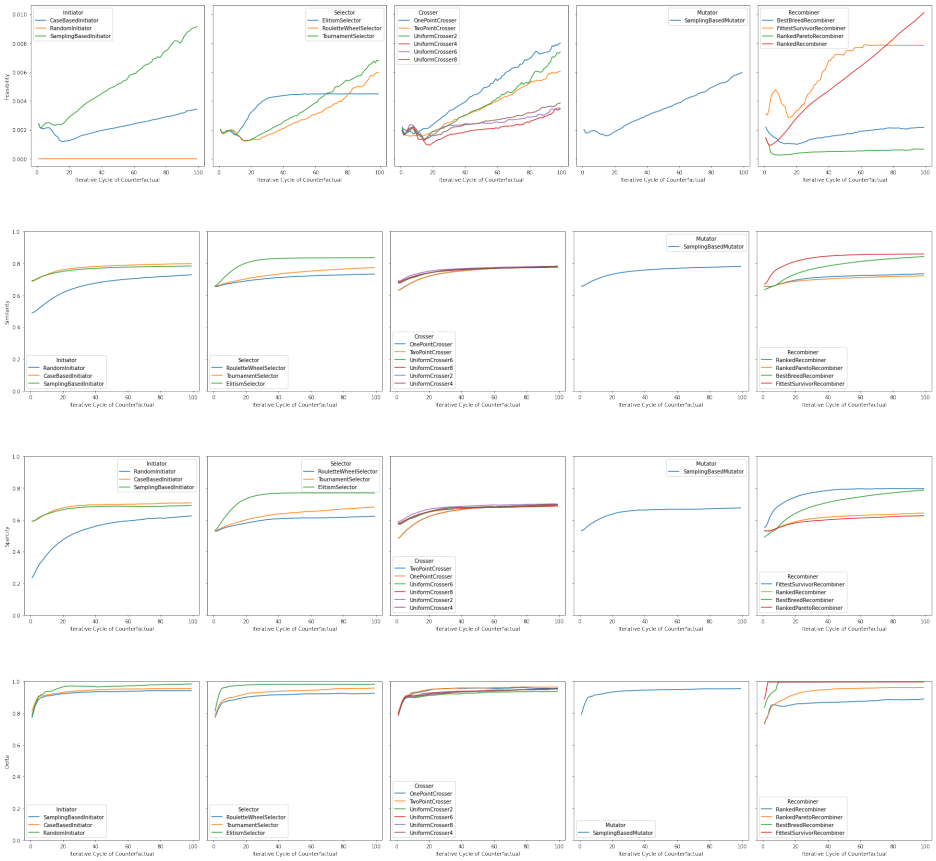


Figure 5.2: Shows the evolution of each viability measure over the entire span of iterative cycles. Each figure adjust the respective operator type by taking the average over all other operator types. **[Make sure, that the legends are aligned in color.]**

[FOR XIXI: Shall I discuss the big figure or just viability in this section?] In Figure 5.2, we see the effects of each operator type, except the mutation operation.

Starting with some commonalities across operator-type and measure, the figure shows that the initiator heavily determines the starting point for each measure. For instance, the **[RandomInitiator]** starts well below the other initiator operations when it comes to sparcity and similarity. Another noteworthy general observation is the delta measure**[Change some of the names to align with Dice4EL]**. Here, for each operator type we see a movement towards the highest possible delta value. Hence, most configurations are capable of changing the source class to the desired class.

In terms of viability Figure **??**, shows an increase only for cases, in which the [**SampleBasedInitiator**] is used. Similar holds for recombination with [**FittestSurvivorRecombiner**].

The results for the selection operator type are undeniably in favor of [**ElitismSelector**] for all viability measures. The same holds for the recombination operation. Here, the [**FittestSurvivorRecombiner**] yields better results.

When it comes to the crossing operation, the results indicate, the differences between [**OnePointCrosser**] and [**TwoPointCrosser**] are inconclusive for all viability measures except feasibility. One can explain that by noting, that both operations are very similar in nature. However, cutting the sequence only once retains produces less impossible sequences for the child sequences.

Discussion

Moving forward, we have to choose a set of operators. We consider the following operators: We choose the [**SampleBasedInitiator**] as it might increase our chances to generate feasible variables.

For selection, we use the [**ElitismSelector**], as it generally appears to return better results.

Furthermore, we choose to move forward with the [**OnePointCrosser**]. This crossing operation is slightly better in yielding feasible results.

For selection and recombination, we use the [**ElitismSelector**] and the [**FittestSurvivorRecombiner**], respectively. They all outcompete their alternatives for all similarity, sparcity and feasibility.

In the next experiment we vary mutation rates, using SBI-ES-OPC-SBM-FSR Generator .

5.1.2 Model Termination Point

Results

In Figure 5.3, we see a general increase in viability for each termination point. It shows, that increasing the termination point also yields better results at the end of the generation process. We see that [**CBI-ES-UC3-SBM-RR**] returns the best results in the shortest time span. The model converges after roughly 50 iterative cycles. [**CBI-RWS-OPC-SBM-BBR**] appears to have not reached convergence. The randomly initiated models have not reached convergence as well. However, they remain far below models that use a more sophisticated method to initialize their population.

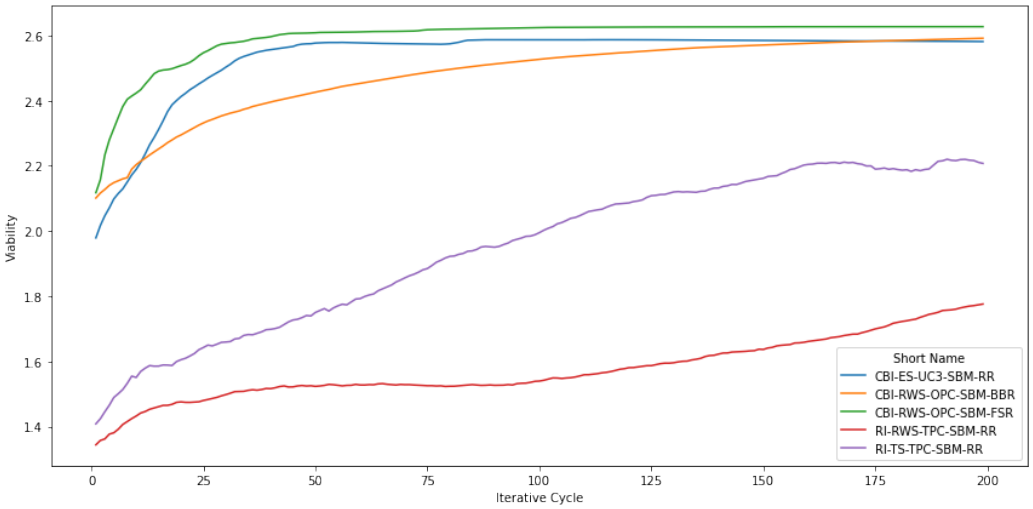


Figure 5.3: This figure shows the viability of across the iteration cycles.

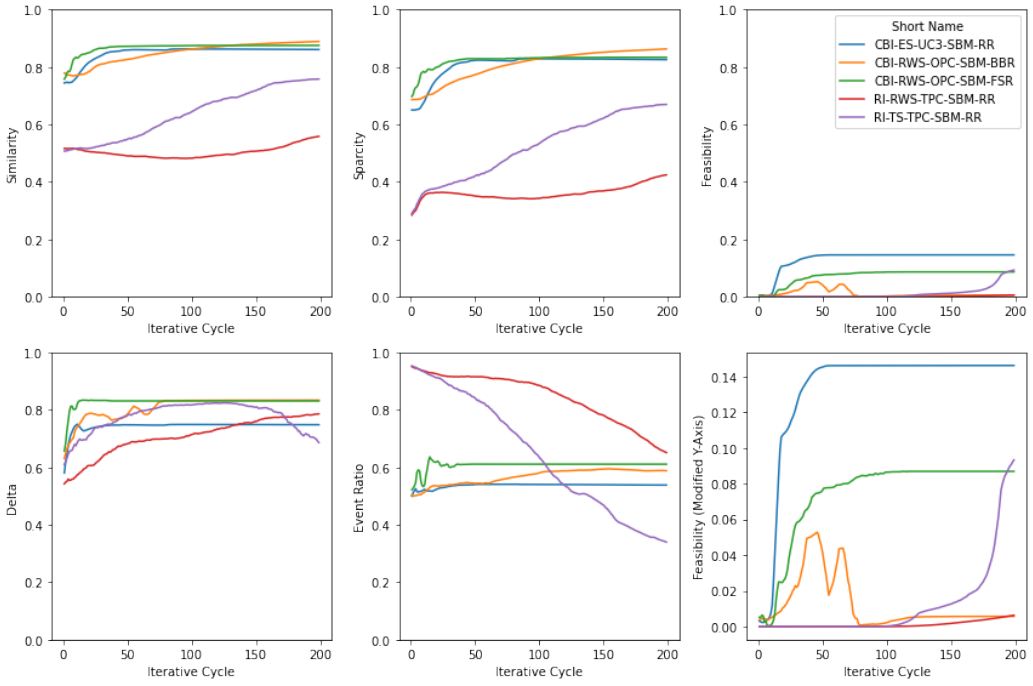


Figure 5.4: This figure shows the remaining measure components. Additionally, we show the ratio of events within the population. We also show a magnified version of the feasibility measure.

Figure 5.4 shows a decomposed view on how the viability measure evolves. Furthermore, we show the average amount of events within a generated counterfactual. In terms of similarity and sparcity all models behave similar. This is no surprise as both measures are ineherently interlinked. We see that the randomly initiated models (RI-x) decrease the amount of events they generate. Case-based initiated models appear to slightly gain more events. Although, **[CBI-RWS-OPC-SBN-BBR]** appears that reaches its saturation point significantly later (**[100]**th cycle). Interestingly, the **[CBI-RWS-OPC-SBM-BBR]** model struggles to maintain feasibility and collapses to near 0 after the **[100th]** iterative cycle. Another surprise is the steep ascention of only model that uses tournament selection (RI-TS-TPC-SBM-RR) towards the end of the generation process. The model even overtakes the model that leads the model-configurations in terms of **[viability]**. Furthermore, we see that CBI-ES-UC-SBM-RR has the highest feasibility among all models. However, it also quickly converges after 50 iterative cycles.

Discussion

The results are not surprising. The longer the algorithm runs the closer it gets to a local minimum. We expect every evolutionary algorithm to converge at some point, as only the best within the population are chosen for the next iteration. If the model does not include enough non-deterministic components the results collapse to one optimal case interms of structure. Hence, the counterfactual activities remain unchanged for the rest of the generation process. The events ratio should optimally approach a number around 0.5 if the factuals are evenly distributed in length. All model-configurations seeminly follow this trajectory. However, models (**[RI-TS-TPC-SBM-RR]**) falls below this level. This coincides with its sharp rise in feasibility. We assume this behavior relates to a bias of the feasibility measure towards shorter sequences. The rise and decline of **[CBI-RWS-OPC-SBM-BBR]** shortly before overtaking all other models in terms of similarity and sparcity indicate a trade-off between how close the counterfactual is to the factual and how feasible it is.

For the next experiments we are going to use **[50]** as a termination point. It appears to be a reasonable point in which most models reach their highest viability yield and have not converged yet. We do not seek convergence, as it we want to maintain the diversity of our counterfactuals.

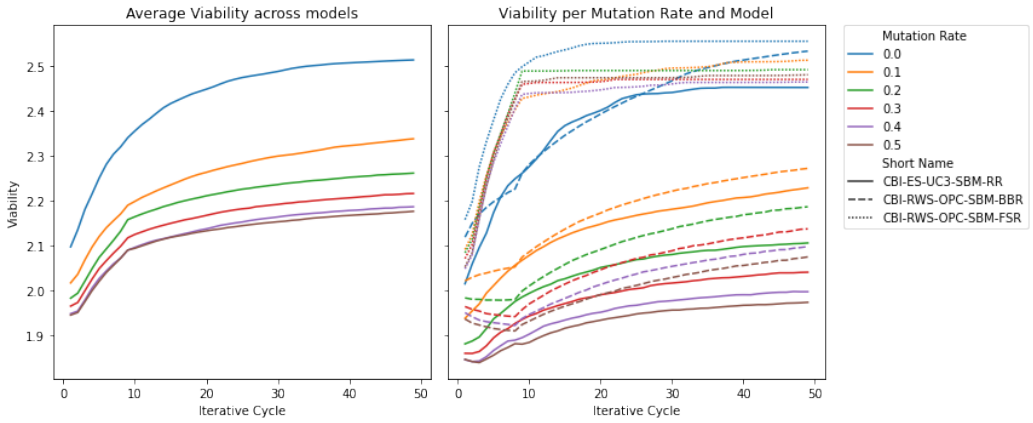


Figure 5.5: This figure shows the viability for each model and mutation rate per iterative cycle. The first plot shows the average across models. The second figure shows the same information per model. The x-axis shows how the viability evolves for each evolutionary cycle. The color indicates the mutation rate. The line-type marks each model tested.

5.1.3 Model Parameters

Results

As we can see in Figure 5.5, that a mutation rate of 0 yields better results on average. Suggesting that mutating the children might impede the model. For model-configurations that use the Fittest-Survivor-Recombination we observe a sharp apattern of convergence before the 10th iterative cycle.

Figure 5.6 reveals the reason for this behavior. In all plots of a shapr change right before the 10th iterative cycle. However, the feasibility measure also displays a sudden stop of improvement for all mutation rates except 0.0 and 0.4. These exceptions also change their rate of growth, but improve shortly after the 30th iterative cycle. The figure also shows that a mutation rate of 0.2 reaches the highest feasibility among the other edit rates. However, after 48 cycles the mutation rate 0.0 overtakes 0.2.

Discussion

While it is expected, that every rate-configuration eventually converges towards an optimal value, it remains surprising that most rate-configurations suddenly converge around the 10 iteration. There are a multitude of possible reasons for this phenomenon. As the viability measure incorporates structural information and event-related information, we assume that the algorithm focuses on finding a structural optimum first.

Hence, the algorithm first prioritizes finding the best sequence in terms of activities. After finding a activity sequence, the model mostly focuses

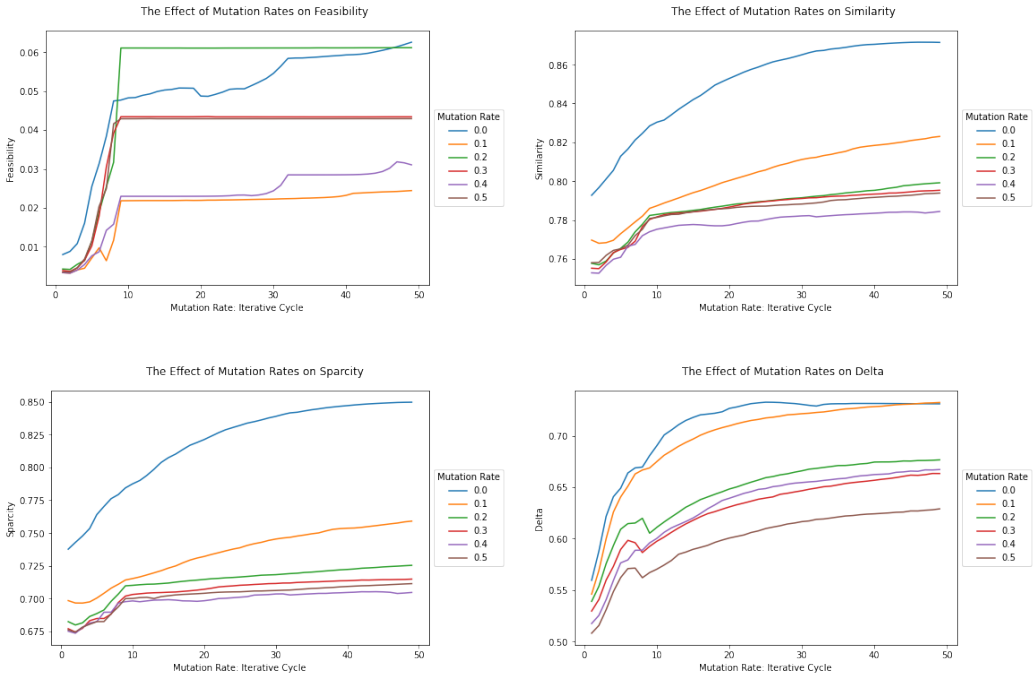


Figure 5.6: Shows all components of the viability measure.

on improving the event attributes. Another explanation could be the ratio between the number of generated children and the population threshold. In this experiment, we generated 200 new children while limiting the population size to 1000.

With these observations in mind, we choose to set the mutation rate to 0.01. This decision implicates that mutations occur very rarely. Therefore, the main driving force for finding the best counterfactual is now the crossing operation. With this setting, we maintain the models ability to improve beyond [50]th iterative cycle.

5.1.4 Model Parameters

Discussion

To concude this section, we summarize the model selection, by choosing the models and their respective hyperparameters. Furthermore, we provide a quick overview of their characteristics. All models use the same mutator. Namely, the *Sample-Based-Mutator*.

CBI-ES-UC3-SBM-RR This model initializes the first population actual using process instances.

For each iterative cycle the individuals with the highest viability will go on to cross over their genom. Every child will receive 30% and 70% of its parents respectively. After the mutation phase, the generator reranks the full population and discards all individuals that have not meet reached the threshold. We choose this model as it promises to return the most feasible counterfactuals. However, the model most likely does not return the highest viability compared to other generators.

BI-RWS-OPC-SBM-FSR This model initializes the first population using actual process instances. For each iterative cycle the individuals that pass on their genes are probablistically selected based in proportion to their viability. For every child a crossover point decides how much of a parents genom are inherited. After the mutation phase, the generator selects the most viable individuals as the next population. We choose this model as it promises to return the highest value in terms of viability. However, the model is prone to reaching convergence very quickly.

FI-X-X-X-X **[This model initializes the first population using the factual instance. For each iterative cycle XXX. For every child XXX. After the mutation phase, XXX. We choose this model as it promises to return the highest viability of all models. However, these counterfactuals are likely infeasbile.]**

5.2 Experiment 2: Model Comparison

In this section we examine the results of each model's average viability across all datasets.

5.2.1 Results

Figure 5.7 displays the results of running each algorithm on a set of factuals. The figure shows clear dominance of the evolutionary model ([**Model Name**]) across all datasets.

Here, [**a specific model**] returns an avarage viability of [**some value**]. This result is not surprising, as we optimize with the same metric used to evaluate the results.

A notable observation occurs in the processing time. Although all models require more time to produce their results, if we increase the maximum length of sequences, the evolutionary algorithms require substiantially more time. **??** shows, the processing time is much higher than the baseline models. On

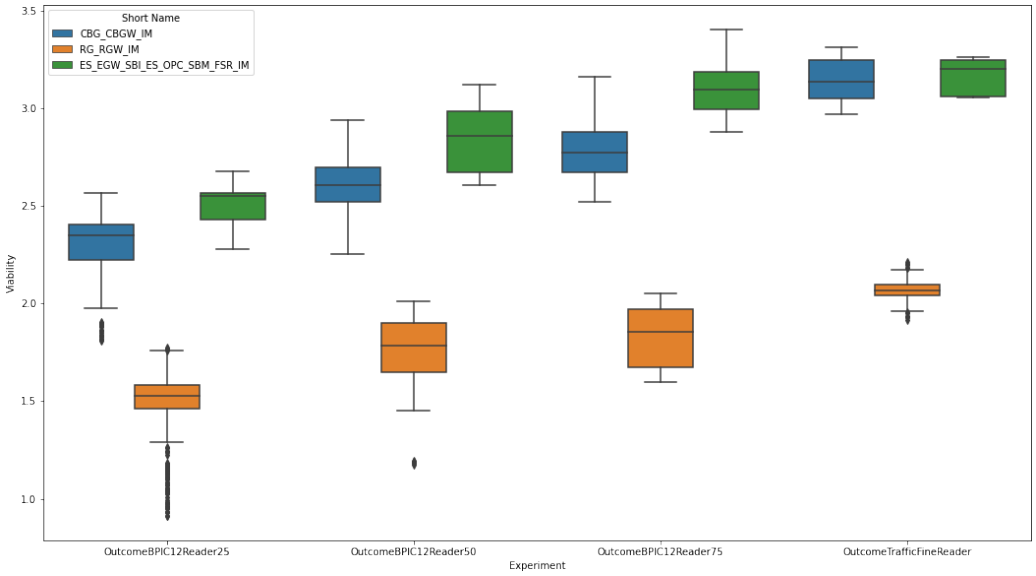


Figure 5.7: This figure shows boxplots of the viability of each models' generated counterfactual.

average, [evo model] require [duration in seconds], while [other models] require [XX], [XX] and [XX], respectively.

5.2.2 Analysis

The results show that [... TBD]. The increase of generation time can be explained by the viability measure. More specifically, the current implementation of the SSDLD within the sparsity and similarity measure has a quadratic time complexity. The time complexity primarily depends on the maximal sequence length. The number of cases that are compared is less of a factor as we use a highly vectorized implementation of the distance using numpy.

5.3 Experiment 3: Evaluation under a different Viability Measure

5.3.1 Results

Table 5.1 shows how each model scores under different operationalisations of viability aspects. They were derived from Hsieh et al.'s custom evaluation protocol and aim to provide a better comparison. Each value reflects the mean across all counterfactual results per model.

Table 5.1: Shows the mean result of each models' result with respect to diversity, plausibility proximity and sparsity.

Model	Property Dimension	Diversity	Plausibility	Proximity	Sparsity
Casebased Generator	Activity	0.994050	0.000000	17.025000	11.915000
	Resource	0.994350	0.000000	19.950000	18.670000
ES-EGW-CBI-ES-OPC-SBM-FSR-IM	Activity	0.750000	0.000000	19.000000	7.250000
	Resource	0.750000	0.000000	19.000000	16.500000
ES-EGW-CBI-ES-OPC-SBM-RR-IM	Activity	0.988850	0.000000	13.225000	6.035000
	Resource	0.992400	0.000000	15.870000	13.125000
Evoluationary: SBI-ES-OPC-SBM-FSR	Activity	0.750000	0.000000	15.750000	10.750000
	Resource	0.750000	0.000000	20.500000	18.500000
Random Generator	Activity	0.995000	0.000000	25.270000	23.340000
	Resource	0.700200	0.000000	24.405000	24.230000

The results show that diversity is the highest for the evolutionary algorithm in terms of activity traces and resource traces. The Random-Search Generator displays low diversity for activities generated and a higher diversity for the resource.

Only the Casebased-Search Generator reaches a maximum score of 1 for plausibility. All the other models are far below or 0.

In terms of proximiny, the Casebased-Search Generator has the lowest actvity prximity. The average distance is 12.55. The SBI-ES-OPC-SBM-FSR Generator takes the second place. Interestingly, the gap between the proximity for activities is larger than the gap between proximities in terms of resources.

Again, the Casebased-Search Generator has the lowest sparcity with 9.34 in terms of activity but only remains slightly better than SBI-ES-OPC-SBM-FSR Generator in terms of resources.

The results sugest that the SBI-ES-OPC-SBM-FSR Generator is capable of producing very diverse counterfactual solutions, but cannot compete with the Casebased-Search Generator in terms of plausibility, proximity and sparcity. Hence, the Casebased-Search Generator is completely plausible given the data, is closer to the factual on average amd displays less changes.

However, this only holds for the activities that are generated. In terms of resources that where generated, the Casebased-Search Generator is just slightly better.

5.3.2 Analysis

Based on these results, we can see that our model does seem to optimize properly for our viability function, but it does not compete under different operationalisations of counterfactual viability.

It is unsurprising, that the Casebased-Search Generator achieves the highest plausibility as all of the counterfactuals were drawn from the data itself. This

In terms of the very similar resource proximity and sparsity scores, we assume that the model is able to identify the correct resource given the activity. For instance, if within a loan approval process only one person or machine executes the activity of checking the identity of an applicant, then SBI-ES-OPC-SBM-FSR Generator seems capable of learning this relationship.

5.4 Experiment 4: Qualitative Assessment

5.4.1 Results

In the result tables you can see 4 factuals with the best counterfactual the model produced.

Table 5.2: Shows a factual and the corresponding counterfactual generated. This counterfactual was generated by the evolutionary algorithm. It is the result which appears to have the highest viability score.

Factual Sequence				Counterfactual Sequence			
Activity	Amount	Resource	Outcome	Activity	Amount	Resource	Outcome
¡/s¿	15 214	nan	1	A-APPROVED	15 000	138	0

Table 5.3: Shows a factual and the corresponding counterfactual generated. This counterfactuals was generated by the case-based model. The counterfactual seems far more viable than the one generated by the evolutionary algorithm.

Factual Sequence				Counterfactual Sequence			
Activity	Amount	Resource	Outcome	Activity	Amount	Resource	Outcome
¡s¿	15 214	nan	0				
A-SUBMITTED	15 000	112	0				
A-PARTLYSUBMITTED	15 000	112	0				
A-PREACCEPTED	15 000	112	0				
A-ACCEPTED	15 000	9	0	¡s¿	15 214	nan	1
O-SELECTED	15 000	9	0	A-SUBMITTED	8 000	112	1
A-FINALIZED	15 000	9	0	A-PARTLYSUBMITTED	8 000	112	1
O-CREATED	15 000	9	0	A-PREACCEPTED	8 000	112	1
O-SENT	15 000	9	0	A-ACCEPTED	8 000	1	1
W-Completeren aanvraag	15 000	9	0	O-SELECTED	8 000	1	1
W-Nabellen offertes	15 000	103	0	A-FINALIZED	8 000	1	1
W-Nabellen offertes	15 000	11181	0	O-CREATED	8 000	1	1
W-Nabellen offertes	15 000	982	0	O-SENT	8 000	1	1
O-SENT-BACK	15 000	11259	0	W-Completeren aanvraag	8 000	1	1
W-Nabellen offertes	15 000	11259	0	W-Nabellen offertes	8 000	11121	1
W-Valideren aanvraag	15 000	138	0	W-Nabellen offertes	8 000	1	1
O-ACCEPTED	15 000	138	0	O-SENT-BACK	8 000	899	1
A-REGISTERED	15 000	138	0	W-Nabellen offertes	8 000	899	1
A-APPROVED	15 000	138	0	O-DECLINED	8 000	9	1
A-ACTIVATED	15 000	138	0	A-DECLINED	8 000	9	1
W-Valideren aanvraag	15 000	138	0	W-Valideren aanvraag	8 000	9	1
¡/s¿	15 214	nan	0	¡/s¿	15 214	nan	1

Across all examples, we see the bias of the viability measure. Every counterfactual is shorter in length than their factual counterpart. We also see, that the value for *Amount* fluctuates heavily for the evolutionary generator. **[FOR XIXI: Should I just average the amount?]** Similar, holds for

the resource field. All models manage to capture the first three activities and its resource. Also, the counterfactual outcome is the opposite of the factual outcome in all cases. Hence, the each generator successfully inverts the model prediction. Furthermore, each model captures the starting events, quite well.

We see this in Table 5.2 and Table 5.3. Furthermore, the model with the highest feasibility among the Evolutionary Algorithm Generator generated counterfactuals, is also the one with the highest viability.

Table 5.3 displays a counterfactual generated by the Casebased-Search Generator . We see that its result appears to be more viable upon inspection. Here, all Amount variables are below the factual Amount. Hsieh et al. interprets this as a reasonable result. The authors state, that we can interpret the result as having better chances of a successful loan application process if we request a lower loan. The results for the Casebased-Search Generator tells us, that the viability measure *does* capture a notion of viability. However, it is not enough to generated realistic counterfactuals for models that optimize it.

5.4.2 Analysis

Some of the results are not surprising. We expected the feasibility to favor smaller sequences. This characteristic can be an advantag for use cases, such as medicine. The changing loan amount was expected, as well. We did not implement any safeguard to prevent this. Throughout a process sequence, the amount acts more similar to a constant or categorical variable than a numerical one if we look at the factual examples.

We are also not surprised, all models manage to capture the first few activities. These are exactly the same accross all cases. Hence, there no variation. A model not recognizing this characteristic of the proce cannot be deemed viable.

All models successfully manage to flip the out come of the prediction model, despite seeming unviable at first glance. This observation tells us that these counterfactuals tell us more about the model rather than the true process.

The Casebased-Search Generator does seem to generate viable counterfactuals. Hence, its results are not only close to the original, but also reasonable.

All in all, we cannot claim that the generator model tells us anything about the true process. The results are far too inconsistent to see any pattern.

Chapter 6

Discussion

In this chapter, we are going to reexamine many of the past decisions we made. We critically assess the results of experiments and how we interpret them. We also propose possible improvements and opportunities for future reasearch.

6.1 Interpretation of Results

6.1.1 Model Framework

No reason to believe the framework doesn't work. Mainly because the case based model has shown that it works.

6.1.2 Viability Measure

Feasibility modelling could have been better. Second, diversity should have been introduced. For instance by using similarity as a proxy.

6.1.3 Evolutionary Algorithm

Worked fine, but there are most likely better approaches out there.

6.1.4 Implications

We also have to mention that these results reflect the behaviour of the model itself. Whether the model reflects the real process realisticly requires a more extensive research approach. First, we have to ensure the model predicts the real outcome reliably. We showed in this thesis that the test scores are generally high. However, this does not mean that this is the case on real

world data. Next, although we our framework does not require any domain knowledge, it is important to evaluate the results with domain experts.

6.2 Proposed Improvements

6.2.1 Better Viability Measure Composition

The experiment mostly assumed a similar importance of each viability composite. Furthermore, crucial parts such as diversity where missing.

6.2.2 Employ Modern Evolutionary Algorithm Techniques like CMA-ES

6.3 Future Work

6.3.1 The Effects of using other Viability Measures

It would be interesting to see the out put of using Hsieh et al.'s way to measure viability.

6.3.2 Inclusion of a better Measure for Sequential Feasibility

If the measure is probabilistic one could try to compute the expected probability given the sequence instead of just the probability. Furthermore, one could measure the feasibility by employing perplexity or another NLP evaluation metric.

Chapter 7

Conclusion

Bibliography

Agrawal, K. (2010). To study the phenomenon of the Moravec's Paradox. *ArXiv*.

Anastasiou, A., Hatzopoulos, P., Karagrigoriou, A., & Mavridoglou, G. (2021). Causality Distance Measures for Multivariate Time Series with Applications. *Mathematics*, *9*(21), 2708. doi:10.3390/math9212708

Anderson, E. (1936). The Species Problem in Iris. *Annals of the Missouri Botanical Garden*, *23*(3), 457–509. doi:10.2307/2394164. JSTOR: 2394164

Apostolico, A., & Giancarlo, R. (1998). Sequence Alignment in Molecular Biology. *Journal of Computational Biology*, *5*(2), 173–196. doi:10.1089/cmb.1998.5.173

Ates, E., Aksar, B., Leung, V. J., & Coskun, A. K. (2021, May 19). Counterfactual Explanations for Multivariate Time Series. In *2021 International Conference on Applied Artificial Intelligence (ICAPAI)* (pp. 1–8). 2021 International Conference on Applied Artificial Intelligence (ICAPAI). doi:10.1109/ICAPAI49758.2021.9462056

Baker, J., Song, J., & Jones, D. R. (2017). Closing the Loop: An Empirical Investigation of Causality in IT Business Value. *undefined*. Retrieved March 1, 2022, from https://www.semanticscholar.org/paper/Closing-the-Loop%3A-An-Empirical-Investigation-of-in-Baker-Song/df210060211bdc598f2d3382c68c615319287f71

Bautista, A. D., Wangikar, L., & Akbar, S. (2012). Process Mining-Driven Optimization of a Consumer Loan Approvals Process - The BPIC 2012 Challenge Case Study. In *Business Process Management Workshops*. doi:10.1007/978-3-642-36285-9_24

Bond-Taylor, S., Leach, A., Long, Y., & Willcocks, C. G. (2021, April 14). *Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models*. arXiv: 2103.04922 [cs, stat]. Retrieved October 1, 2021, from http://arxiv.org/abs/2103.04922

Carvalho, D. V., Pereira, E. M., & Cardoso, J. S. (2019). Machine Learning Interpretability: A Survey on Methods and Metrics. *Electronics*, *8*(8), 832. doi:10.3390/electronics8080832

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A., & Bengio, Y. (2016, April 6). A Recurrent Latent Variable Model for Sequential Data. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2* (pp. 2980–2988). Cambridge, MA, USA: MIT Press. Retrieved February 3, 2022, from http://arxiv.org/abs/1506.02216

Counterfactual. (n.d.). doi:10.1093/oi/authority.20110803095642948

Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, *7*(3), 171–176. doi:10.1145/363958.363994

Dandl, S., Molnar, C., Binder, M., & Bischl, B. (2020). Multi-Objective Counterfactual Explanations. In T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. Emmerich, & H. Trautmann (Eds.), *Parallel Problem Solving from Nature – PPSN XVI* (pp. 448–469). doi:10.1007/978-3-030-58112-1_31

Definition of PROCESS. (n.d.). Retrieved February 17, 2022, from https://www.merriam-webster.com/dictionary/process

Delaney, E., Greene, D., & Keane, M. T. (2021). Instance-Based Counterfactual Explanations for Time Series Classification. In A. A. Sánchez-Ruiz & M. W. Floyd (Eds.), *Case-Based Reasoning Research and Development* (pp. 32–47). doi:10.1007/978-3-030-86957-1_3

Deng, L. (2012). The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine*, *29*(6), 141–142. doi:10.1109/MSP.2012.2211477

Fisher, R. A. (1936). The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, *7*(2), 179–188. doi:10.1111/j.1469-1809.1936.tb02137.x

Fraccaro, M., Sønderby, S. K., Paquet, U., & Winther, O. (2016, December 5). Sequential neural models with stochastic layers. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (pp. 2207–2215). Red Hook, NY, USA: Curran Associates Inc.

Francis, W. N., & Kucera, H. (1979). *Brown corpus manual*. Department of Linguistics, Brown University, Providence, Rhode Island, US. Retrieved from http://icame.uib.no/brown/bcm.html

Frank, S. L., & Christiansen, M. H. (2018). Hierarchical and sequential processing of language. *Language, Cognition and Neuroscience*, *33*(9), 1213–1218. doi:10.1080/23273798.2018.1424347

Girin, L., Leglaive, S., Bie, X., Diard, J., Hueber, T., & Alameda-Pineda, X. (2021). Dynamical Variational Autoencoders: A Comprehensive Review. *Foundations and Trends® in Machine Learning*, *15*(1-2), 1–175. doi:10.1561/2200000089. arXiv: 2008.12595

Hitchcock, C. (2020). Causal Models. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Summer 2020). Metaphysics Research Lab, Stanford University. Retrieved February 10, 2022, from https://plato.stanford.edu/archives/sum2020/entries/causal-models/

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. doi:10.1162/neco.1997.9.8.1735

Hompes, B. F. A., Maaradji, A., La Rosa, M., Dumas, M., Buijs, J. C. A. M., & van der Aalst, W. M. P. (2017). Discovering Causal Factors Explaining Business Process Performance Variation. In E. Dubois & K. Pohl (Eds.), *Advanced Information Systems Engineering* (pp. 177–192). doi:10.1007/978-3-319-59536-8_12

Hsieh, C., Moreira, C., & Ouyang, C. (2021, October 31). DiCE4EL: Interpreting Process Predictions using a Milestone-Aware Counterfactual Approach. In *2021 3rd International Conference on Process Mining (ICPM)* (pp. 88–95). 2021 3rd International Conference on Process Mining (ICPM). doi:10.1109/ICPM53251.2021.9576881

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, *82*, 35–45.

Klimek, J., Klimek, J., Kraskiewicz, W., & Topolewski, M. (2021, August 17). *Long-term series forecasting with Query Selector – efficient model of sparse attention*. arXiv: 2107.08687 [cs]. Retrieved November 9, 2021, from http://arxiv.org/abs/2107.08687

Krause, J., Perer, A., & Ng, K. (2016, May 7). Interacting with Predictions: Visual Inspection of Black-box Machine Learning Models. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (pp. 5686–5697). doi:10.1145/2858036.2858529

Krishnan, R., Shalit, U., & Sontag, D. (2017). Structured Inference Networks for Nonlinear State Space Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, *31*(1). Retrieved February 22, 2022, from https://ojs.aaai.org/index.php/AAAI/article/view/10779

Leglaive, S., Alameda-Pineda, X., Girin, L., & Horaud, R. (2020, February 10). *A Recurrent Variational Autoencoder for Speech Enhancement*. arXiv: 1910.10942 [cs, eess]. Retrieved February 7, 2022, from http://arxiv.org/abs/1910.10942

Levenshtein, V. (1965). Binary codes capable of correcting deletions, insertions, and reversals. *undefined*. Retrieved April 15, 2022, from https://

www.semanticscholar.org/paper/Binary-codes-capable-of-correcting-deletions%2C-and-Levenshtein/b2f8876482c97e804bb50a5e2433881ae31d0cdd

Mannhardt, F., & Blinde, D. (2017). Analyzing the trajectories of patients with sepsis using process mining: RADAR + EMISA 2017. *RADAR+EMISA 2017, Essen, Germany, June 12-13, 2017*, 72–80. Retrieved April 22, 2022, from http://www.scopus.com/inward/record.url?scp=85022001209&partnerID=8YFLogxK

Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a large annotated corpus of English: The penn treebank. *Computational Linguistics*, *19*(2), 313–330.

Martens, D., & Provost, F. (2014). Explaining data-driven document classifications. *MIS Quarterly*, *38*(1), 73–100. doi:10.25300/MISQ/2014/38.1.04

Melnyk, I., Santos, C. N. dos, Wadhawan, K., Padhi, I., & Kumar, A. (2017, December 4). *Improved Neural Text Attribute Transfer with Non-parallel Data*. arXiv: 1711.09395 [cs]. Retrieved February 28, 2022, from http://arxiv.org/abs/1711.09395

Mitton, R. (2010). Fifty years of spellchecking. *Writing Systems Research*, *2*(1), 1–7. doi:10.1093/wsr/wsq004

Molnar, C. (2019). *Interpretable machine learning. A Guide for Making Black Box Models Explainable*. Retrieved from https://christophm.github.io/interpretable-ml-book/

Narendra, T., Agarwal, P., Gupta, M., & Dechu, S. (2019). Counterfactual Reasoning for Process Optimization Using Structural Causal Models. In T. Hildebrandt, B. F. van Dongen, M. Röglinger, & J. Mendling (Eds.), *Business Process Management Forum* (pp. 91–106). doi:10.1007/978-3-030-26643-1_6

Oberst, M., & Sontag, D. (2019, June 6). *Counterfactual Off-Policy Evaluation with Gumbel-Max Structural Causal Models*. arXiv: 1905.05824 [cs, stat]. Retrieved September 22, 2021, from http://arxiv.org/abs/1905.05824

Pearl, J., Glymour, M., & Jewell, N. P. (2016). *Causal inference in statistics: A primer*. Chichester, West Sussex: Wiley.

Qafari, M. S., & van der Aalst, W. M. P. (2021). Case Level Counterfactual Reasoning in Process Mining. In S. Nurcan & A. Korthaus (Eds.), *Intelligent Information Systems* (pp. 55–63). doi:10.1007/978-3-030-79108-7_7

Robeer, M., Bex, F., & Feelders, A. (2021, November). Generating Realistic Natural Language Counterfactuals. In *Findings of the Association for Computational Linguistics: EMNLP 2021* (pp. 3611–3625). EMNLP-Findings 2021. doi:10.18653/v1/2021.findings-emnlp.306

Shook, C. L., Ketchen Jr., D. J., Hult, G. T. M., & Kacmar, K. M. (2004). An assessment of the use of structural equation modeling in strategic management research. *Strategic Management Journal*, *25*(4), 397–404. doi:10.1002/smj.385

Starr, W. (2021). Counterfactuals. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Summer 2021). Metaphysics Research Lab, Stanford University. Retrieved February 9, 2022, from https://plato.stanford.edu/archives/sum2021/entries/counterfactuals/

Tax, N., Verenich, I., La Rosa, M., & Dumas, M. (2017). Predictive Business Process Monitoring with LSTM Neural Networks. In E. Dubois & K. Pohl (Eds.), *Advanced Information Systems Engineering* (pp. 477–492). doi:10.1007/978-3-319-59536-8_30

Teinemaa, I., Dumas, M., La Rosa, M., & Maggi, F. M. (2018, October 23). *Outcome-Oriented Predictive Process Monitoring: Review and Benchmark*. arXiv: 1707.06766 [cs]. Retrieved May 7, 2022, from http://arxiv.org/abs/1707.06766

Tsirtsis, S., De, A., & Gomez-Rodriguez, M. (2021, July 6). *Counterfactual Explanations in Sequential Decision Making Under Uncertainty*. arXiv: 2107.02776 [cs, stat]. Retrieved September 9, 2021, from http://arxiv.org/abs/2107.02776

van der Aalst, W., Adriansyah, A., de Medeiros, A. K. A., Arcieri, F., Baier, T., Blickle, T., ... Wynn, M. (2012). Process Mining Manifesto. In F. Daniel, K. Barkaoui, & S. Dustdar (Eds.), *Business Process Management Workshops* (pp. 169–194). doi:10.1007/978-3-642-28108-2_19

Wachter, S., Mittelstadt, B., & Russell, C. (2017). Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR. *ArXiv*. doi:10.2139/ssrn.3063289

Wang, J., Song, S., Zhu, X., & Lin, X. (2013). Efficient recovery of missing events. *Proceedings of the VLDB Endowment*, *6*(10), 841–852. doi:10.14778/2536206.2536212

Wang, K., Hua, H., & Wan, X. (2019, December 12). *Controllable Unsupervised Text Attribute Transfer via Editing Entangled Latent Representation*. arXiv: 1905.12926 [cs]. Retrieved February 28, 2022, from http://arxiv.org/abs/1905.12926

Wang, Z., Zhang, J., Xu, H., Chen, X., Zhang, Y., Zhao, W. X., & Wen, J.-R. (2021, July 11). Counterfactual Data-Augmented Sequential Recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 347–356). doi:10.1145/3404835.3462855