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
April 16, 2022

Abstract

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

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 CITE . 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 CITE . For instance, in the medical domain, models have been shown to predict the outcome or trajectory of a patient's condition CITE . 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, 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 CITE Moravec Paradox . 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 CITE . Figure 1.1 is a visual representation of the desired goal.

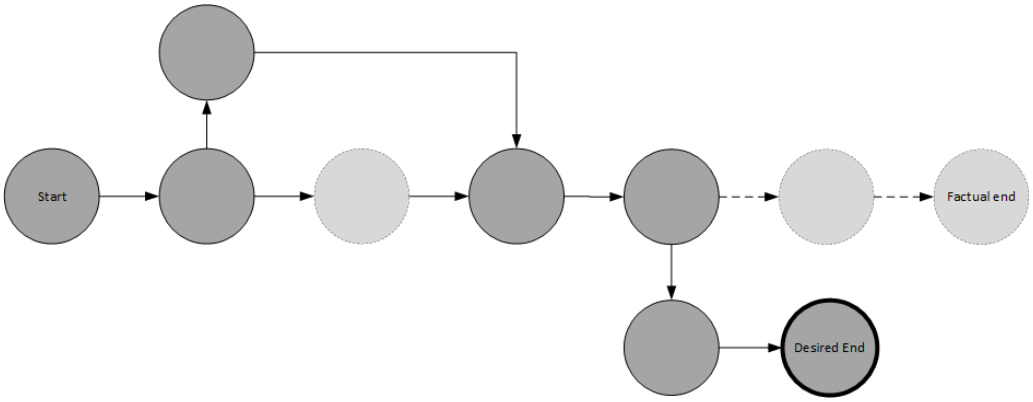

Figure 1.1: This figure illustrates a model, that predicts a certain trajectory of the process. However, we want to change the process steps in such a way, that it changes the outcome.

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 **[examples]** CITE . but some are more comprehensible for humans than others.

A prominent and human-friendly approach are counterfactuals CITE . Counterfactuals within the AI framework help us to answer hypothetical "what-if" questions. Basically, if we know *what* would happen *if* we changed the 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]. A major reason, 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. Second, sequential data is highly variable in length, but processes steps have complicated factors, too CITE . 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 CITE Check process discovery literature . Just these challenges, makes the field in which we can contribute a vast endeavor.

1.3 Related Literature

Many researchers have worked on counterfactuals and process mining. 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˙CounterfactualExplanationsOpening˙2017** in **wachter˙CounterfactualExpla**
The authors defined a loss function which incorporates the criteria to generate a counterfactual which maximizes the likelihood for a predefined outcome

[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 time as a major component, which is also inherently sequential

and minimizes the distance to the original instance. However, the solution of **wachter˙CounterfactualExplanationsOpening˙2017** 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˙MultiObjectiveCounterfactualExplanations˙2020** incoporates four main criteria for counterfactuals (see section 2.3) by applying a genetic algorithm with a multi-objective fitness function[**dandl˙MultiObjectiveCounterfactu** This approach strongly differs from gradient-based methods, as it does not require a differentiable differentiable objective function. However, their solution was only tested on structured data.

1.3.2 Generating Counterfactual Sequential

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˙ExplainingDatadrivenDocument˙2014** and later **krause˙InteractingPredictions** are early examples of counterfactual NLP research[**martens˙ExplainingDatadrivenDocument** **krause˙InteractingPredictionsVisual˙2016**]. Their approach strongly focuses on the manipulation of sentences to achieve the desired outcome. However, as **robeer˙GeneratingRealisticNatural˙2021** puts it, their counterfactuals do not comply with *realisticness*[**robeer˙GeneratingRealisticNatural˙2021**].

Instead, **robeer˙GeneratingRealisticNatural˙2021** showed that it is possible to generate realistic counterfactuals with a Generative Adversarial Model (GAN)[**robeer˙GeneratingRealisticNatural˙2021**]. 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[**melnyk˙ImprovedNeuralText˙2017**, **wang˙ControllableUnsupervi** By modelling the latent space, we can simply sample from a distribution conditioned on the original instance. **bond-taylor˙DeepGenerativeModelling˙2021** 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*[**leglaive˙RecurrentVariationalAutoencoder˙2020**]. 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˙StructuredInferenceNetworks˙2017**[**krause˙InteractingPredictionsVisual˙201 krishnan˙StructuredInferenceNetworks˙2017**]. 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[**chung˙RecurrentLatentVariable˙2016**, **fraccaro˙SequentialNeuralModels˙201 leglaive˙RecurrentVariationalAutoencoder˙2020**] of **krishnan˙StructuredInferenceNetv** 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[**girin˙DynamicalVariationalAutoencoders**

1.3.3 Generating Counterfactual Time-Series

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

First, **delaney˙InstanceBasedCounterfactualExplanations˙2021** introduces a case-based reasoning to generate counterfactuals[**delaney˙InstanceBasedCounterfa** 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 CITE . 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˙InstanceBasedCounterfactualExplanations˙2021**, as it mainly deals with time series classification CITE . 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˙CounterfactualExplanati** also uses a case-based approach[**ates˙CounterfactualExplanationsMultivariate˙2021**]. However, it contrasts from other approaches, as it does not specify a partic-

ular 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. **The framework will be evaluated in two steps. The first step aims to compare various model types against eachother based on the countefactual viability. The second step scrutinizes the best framework configurations from step one, by presenting its results to a domain expert.**

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[**shook˙AssessmentUseStructural˙2004**, **baker˙ClosingLoopEmpirical˙2017**, **hompes˙DiscoveringCausalFactors˙2017**, **wang˙CounterfactualDataAugmentedSequen**
Among these, **narendra˙CounterfactualReasoningProcess˙2019** approach is one of the first to include counterfactual reasoning for process optimization[**narendra˙Counterf**
oberst˙CounterfactualOffPolicyEvaluation˙2019 use counterfactuals to generate alternative solutions to treatments, which lead to a desired outcome[**oberst˙Counterfac**
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˙CaseLevelCounterfactual˙2021** published the most recent paper on the counterfactual generation of explanations[**qafari˙CaseLevelCounterfactual**
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˙CounterfactualExplanationsSequential˙2021** develop the first explanation method for process data[**tsirtsis˙CounterfactualExplanationsSe**
However, their work closely resembles the work of **oberst˙CounterfactualOffPolicyEvaluation**
and treat the task as Markov Decision Process (MDP)[**oberst˙CounterfactualOffPolicyEvalua**
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 CITE . 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˙CounterfactualExplanationsSequential˙2021** and **oberst˙CounterfactualOf** contribute an important idea. The idea of incrementally generating the counterfactual instead of the full sequence. **hsieh˙DiCE4ELInterpretingProcess˙2021** build on this concept by proposing a system that generates counterfactuals milestone-wise[**hsieh˙DiCE4ELInterpretingProcess˙2021**]. Their work is the closest to our approach. The authors recognised that some processes have critical events, which govern the overall outcome. Hence, by simply avoiding the undesired outcome from milestone to milestone, it is possible to limit the search space and compute viable counterfactuals with counterfactual generation methods, such as the DiCE alogorithm. However, their approach presupposes that the critical event points are known and the outcome is binary. Especially, the first condition makes their approach heavily dependent on the data which is used.

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 generating counterfactual sequences, 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 to viable solutions, while being flexible enough to not just be copies of already existing data instances. 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 process-agnostic method to generate sequential counterfactuals that are viable. In terms of specific research questions we try to answer:

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

- RQ1: Which evaluation metric, reflects the viability of counterfactuals?
- RQ2: To which extend do viable counterfactuals align with domain experts?

We approach these questions, by proposing a schematic framework which allows the exploration of several independent components. section 1.4 shows the conceptual framework of the base approach visually.

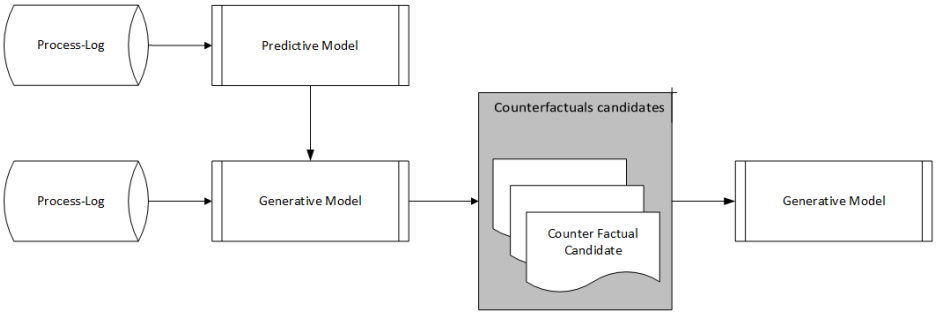

Figure 1.2: This figure shows a simplified schematic representation of the framework which is explored in this thesis.

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.

Chapter 2

Background

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[**˙DefinitionPROCESS˙**]. 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 **vanderaalst˙ProcessMiningManifesto˙2012**[**vanderaals**
Each step, including start and end points, is an process event which was caused by an *activity*[1]. 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*[**vanderaalst˙ProcessMiningManifesto˙2012**].

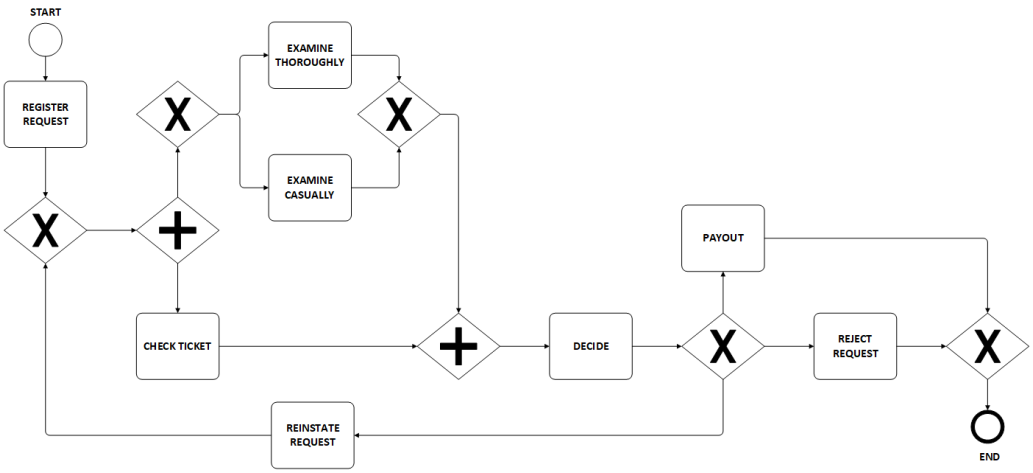

Figure 2.1: This graph shows an example of a BPMN process map.

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 **[EXAMPLE NEEDED]** or continuous processes such as **[EXAMPLE NEEDED]**. There may be valid

[1]In this paper we will use the terms *event* and *activity* interchangebably as their practical difference is quite subtle.

arguments 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)[**vanderaalst˙ProcessMiningManifesto˙2012**]. However, its data-centric approaches originate in Data Mining. The authors **vanderaalst˙ProcessMiningManifesto˙2012** 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"[**vanderaalst˙ProcessMiningManifesto˙2012**]. 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) CITE .

2.1.3 The Challenges of Process Mining

As mentioned in chapter 1, process data modelling and analysis is a challenging task. **vanderaalst˙ProcessMiningManifesto˙2012** mentions a number of issues that arise from processes[**vanderaalst˙ProcessMiningManifesto˙2012**].

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[**vanderaalst˙ProcessMiningManifesto˙201**

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[**vanderaalst˙ProcessMiningManifesto˙2012**].

A third issue which contributes to the datasets' incompleteness and complexity is a phenomenon called *concept drift*. 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 CITE . 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 CITE .

2.2 Multivariate Time-Series Modelling

The temporal and multivariate nature of process instance often turns Process Mining 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 CITE . Hence, we can assume that a human has formulated his sentence in the brain before expressing it in a sequence of words CITE . In contrast to sequences, time series cannot depend on future values. The general understanding of *time* is linear and forward directed CITE . 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 CITE check [**leglaive˙RecurrentVariationalAutoencoder˙2020**] . 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 CITE . 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 CITE see [**anastasiou˙CausalityDistanceMeasures˙2021**] . 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 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"[**˙Counterfactual˙**]. A simpler definition from **starr˙Counterfactuals˙2021** 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[**hitchcock˙CausalModels˙2020**] literature. Here, CITE **Someone** describes a counterfactual as "**[Causal inference definition]**". What binds all of these definitions is the notion of causality within "what if" scenarios.

However, for this paper, we will 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[3, ch.9.3]. 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[2]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**FG***. Then a counterfactual *ABCDE**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[3, ch.9.3]. According to Molnar *Valid* counterfactuals satisfy **four** criteria[3, ch.9.3]: **[FIX Indentation!]**

Similarity: A counterfactual should be similar to the original instance. If the counterfactual to sequence 1 was *A**A**CDE**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.

Precision: 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**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.

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**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.

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

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 ***V****BCDEF****H*** and *ABCDE****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[3, 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 CITE . Although, the criteria can support us with the decision, it remains an open question *how* to evaluate counterfactuals. Every automated measure comes with implicit assumptions and they cannot guarantee a realistic explanations. We still need domain experts to assess their *viability*.

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 CITE . 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 CITE .

Chapter 3

Methods

3.1 Methodological Framework

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.

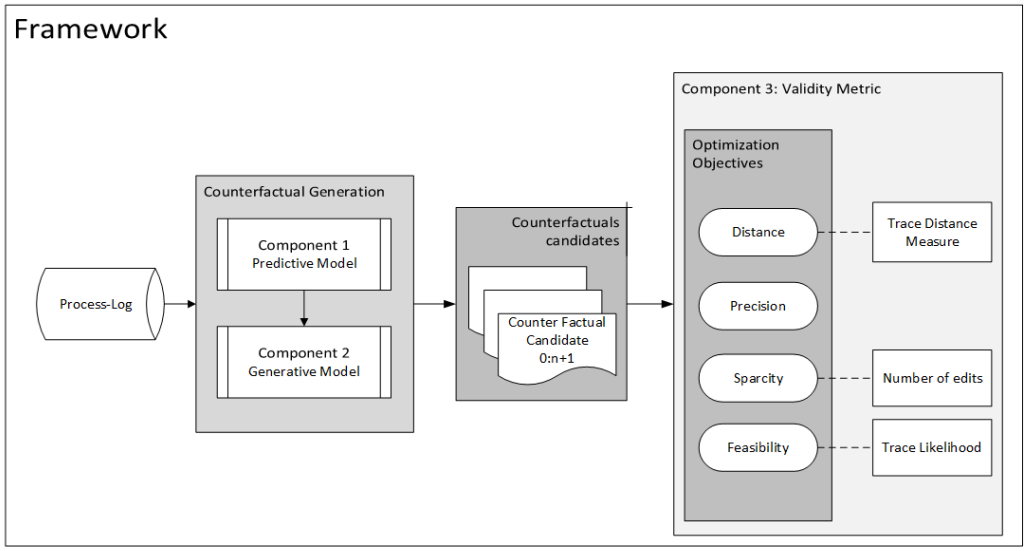

Figure 3.1: XXX.

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 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.

3.2 Formal Definitions

Before diving into the remainder 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.

3.2.1 Process Logs, Cases and Instance Sequences

We start by formalising the log and its elements. Let $\mathcal{E}$ be the universe of event identifiers and $E \subseteq \mathcal{E}$ a set of events. 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 case. For a set of events $E \subseteq \mathcal{E}$, we use a shorthand $s \in C$ being a particular sequence $s^c = \langle e_1, e_2, \ldots, e_t \rangle$ as a case of

length t.

Furthermore, let $\mathcal{T}$ be the time domain and $\pi_t : E \mapsto \mathcal{T}$ a surjective linking function which strictly orders a set of events.

We assume that every e contains a set of data attributes. Hence, let $A = a_1, \ldots, a_i$ be a set of attribute names (e.g., timestamp, resource, action, etc.) and $\mathcal{F}$ be the universe of each attribute's representations $F \in \mathcal{F}$. For each element in A, we have a surjective function $\pi_{a_i} : A \mapsto F_i$ which links each event attribute a_i to its value $f_i \in F_i$. Each attribute value is represented with a vector space of varying dimensions $f_i \in R^d$.

We represent each event as a concatenated vector of its attribute values as $e_t^c = [f_1^c; f_2^c; \ldots; f_i^c]$, in which c specifies a particular case, a specific event within the case at time t and i its event attributes.

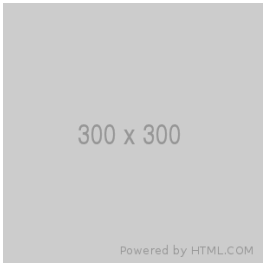

Figure 3.2: This figure show the representation of a particular event e and a case c.

3.2.2 State-Space Models

Generally speaking, every time-series can be represented as a state-space model[**kalman˙NewApproachLinear˙1960a**]. 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 3.1.

$$\begin{aligned} \mathbf{z}_{t+1} &= h(t, \mathbf{z}_t, \mathbf{u}_t) \\ \mathbf{e}_t &= g(t, \mathbf{z}_t, \mathbf{u}_t) \\ \mathbf{z}_{t+1} &:= \frac{d}{dt}\mathbf{z}_t \end{aligned} \tag{3.1}$$

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*[1] states, from *observable* system outputs. Figure 3.3 shows a graphical representation of these equations.

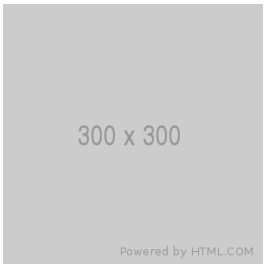

Figure 3.3: This figure shows a simplified graphical representation of a state-space model. Each arrow represents the flow of information.

The body of literature for state-space models is too vast to discuss them in detail[2]. 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. 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.** 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 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 3.1 probablistically as shown in Equation 3.2.

[1] 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.

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

$$\mathbb{E}[p\left(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h\right)] = \int z_{t+1} \cdot p\left(z_{t+1} \mid t, z_{1:T}, u_{1:T}, x_{1:T}, \theta_h\right) \tag{3.2}$$

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

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*. The stochastic process is shown in Figure 3.4.

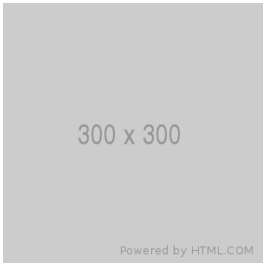

Figure 3.4: Figure shows a graphical representation of the stochastic process.

These characteristics change the probabilities in Equation 3.2 to Equation 3.3:

$$p\left(z_{t+1} \mid z_{1:t}, u_{1:t}, \theta_h\right) = \prod_{1}^{t} p\left(z_t \mid z_{1:t}, u_t, \theta_h\right) \tag{3.3}$$

$$p\left(x_t \mid z_{1:t}, \theta_g\right) = \prod_{1}^{t} p\left(x_{t-1} \mid z_{1:t}, \theta_f\right) \tag{3.4}$$

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 3.3. We then use the samples to reconstruct the most-likely a counterfactual $e^*_{1:t}$. Hence, our goal is to maximize both likelihoods.

[A number of AI techniques where developed to model this representation bla bla bla (HMM, Kalman, etc – Has further formalisation).]

3.3 Viability Measure

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

3.3.1 Damerau-Levenshtein-Distance

Before discussing some of the similarity measures it is important to briefly introduce the Damerau-Levenshtein-Distance. This distance function is a modified version of the Levenshstein-Distance[2], which is a widely used to compute the edit-distance of two discrete sequences. The most important domains of applications being the NLP discipline and Biomedical Sciences. In other words, we often use the Levenshtein-Distance to compute the edit-distance between two words, two sentences or two DNA sequences. Generally, the distance accounts for inserts, deletions and substitutions of elements between the two serquences. Damerau modified the distance function to allow for transpositions. 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. Equation 3.5 depicts the mathematical formulation of the distance.

$$d_{a,b}(i,j) = \min \begin{cases} d_{a,b}(i-1,j)+1 & \text{if } i > 0 \\ d_{a,b}(i,j-1)+1 & \text{if } j > 0 \\ d_{a,b}(i-1,j-1)+1 & \text{if } i,j > 0 \\ d_{a,b}(i-2,j-2)+1 & \text{if } i,j > 1 \wedge a_i = b_{j-1} \wedge a_{i-1} = b_j \\ 0 & \text{if } i = j = 0 \end{cases} \tag{3.5}$$

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, insertion, deletion, substitution and transposition. For each operation, the algorithm adds an edit cost of 1. For Process Mining, it becomes necessary to modify the distance further.

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-Levenshtein-Distance, the edit distance between both sequences is zero, as we can recognise one substitution at the second character in both strings. However, this representation is insufficient for process instances as they may also contain attibute values. Therefore, we characterise the sequences as process events in Equation 3.6.

$$s^1 = aaba \tag{3.6}$$
$$s^2 = aa^*ba \tag{3.7}$$
$$s^3 = acba \tag{3.8}$$
$$s^4 = aba \tag{3.9}$$
$$a, b, c \in \mathbb{R}^3 \tag{3.10}$$
$$a = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix} \; a^* = \begin{bmatrix} 3 \\ 3 \\ 4 \end{bmatrix} \; b = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \; c = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix} \tag{3.11}$$

If we do not consider attribute values, it becomes clear that s^2, s^3 and s^4 have an edit distance of 0, 1 and 1 to s^1. However, with attribute values 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. These examples show that we can neither disregard attribute values nor events, while computing the edit distance of two process instances. We show this in Figure 3.5. In other words, we cannot simply assume a static cost of 1 for each necessary edit operation.

Instead, we have to define a cost function which takes attribute variables into account. In the following sections, we will establish distances which use a modified Damerau-Levenshtein-Distance approach. Here, the cost of each edit-operation will be weighted with an appropriate distance function which includes both events and event attributes. Concretely, we can say that s^1 and s^2 only differ in terms of events only by 1 change. However, taking attribute values into account, s^1 and s^2 differ on two 2 values.

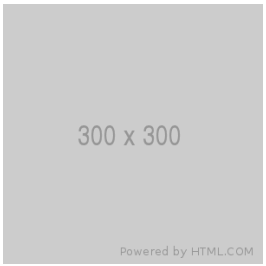

Figure 3.5: This figure exemplifies the differeneces between the normal DL-distance and this one used.

Therefore, we will introduce a modified version of the Damerau-Levenshtein-Distance, that not only reflects the difference between two process instances but also the attribute values by introducing an appropriate cost function $cost_{a_i,b_j}$. Concretely, we define a modified Damerau-Levenshtein-Distance as shown in Equation 3.12.

$$d_{a,b}(i,j) = \min \begin{cases} d_{a,b}(i-1,j) + cost(0,b_j) & \text{if } i > 0 \\ d_{a,b}(i,j-1) + cost(a_i,0) & \text{if } j > 0 \\ d_{a,b}(i-1,j-1) + cost(a_i,b_j) & \text{if } i,j > 0 \\ d_{a,b}(i-2,j-2) + cost(a_i,b_j) & \text{if } i,j > 1 \wedge a_i = b_{j-1} \wedge a_{i-1} = b_j \\ 0 & \text{if } i = j = 0 \end{cases} \tag{3.12}$$

3.3.2 Likelihood-Measure

We can evaluate the precision of a counterfactual trace by determining whether a counterfactual leads to the desired outcome. For this purpose, we use the predictive model which will out put a prediction base on the counterfactual sequence. However, it is often difficult to force a deterministic model to produce a different result. We can relax the condition by maximising the likelihood of the counterfactual outcome. If we compare the likelihood

of the desired outcome under the factual sequence with the counterfactual sequence, we can determine an increase or decrease. Ideally, we want to increase the likelihood. We can compute a odds or the difference between the two likelihoods. We choose to use the odds[**Needs some thought or testing. Odds may be too aggressive.**]. In Equation 3.13, we define the function.

$$odds = \frac{p(o^*|e^*)}{p(o^*|e)} \tag{3.13}$$

Here, $p(o|e)$ describes the probability of an outcome, given a sequence of events.

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 events themselves. This is a difficult task, given the *open world assumption*. In theory, we cannot know whether any event *can* follow after a nother event or not. However, we can assume that the data is representative of the process dynamics. Hence, we can simply 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. We use the Kneser-Ney Smoothing [CITE] approach to ensure that unseen sequences are accounted for. Second, we have 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 determines its feature values. Meaning, we can model density estimators for every event and use them to determine the likelihood of a set of features. In Equation 3.14, we define both parts of the function.

$$feasibility_e = p(a_n|a_1 \dots e_{n-1}) \approx p(a_n|a_{n-1}) \tag{3.14}$$

$$feasibility_f = p(f|a_n) \tag{3.15}$$

Here, a and f are the activity and features of a particular event. The first equation shows the approximation based on the markov assumption.

3.3.4 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. We will use a modified version of the Damerau-Levenshtein distance. This distance computes the costs associated with aligning two sequences by taking changes, inserts, deletes and transpositions of elements into account. Each of these alignment operations is typically associated with a cost of 1. This allows us to directly compute the structural difference between two sequences regardless of their lenghts. Additionally, instead of computing a cost of 1 for every operation, we compute a cost of a distance between the feature vectors of each step. This allows us to take not only the sequential differences into account but the feature differences, too. The similarity edit function is defined in Equation 3.16.

$$cost_{a_i,b_j} = dist(a_i, b_j) \tag{3.16}$$

$$a_i, b_j \in \mathbb{R}^d \tag{3.17}$$

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

3.3.5 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 another modified version of the Damerau-Levenshstein edit distance. The sparcity edit function is defined in Equation 3.18.

$$\tag{3.18}$$

$$cost_{a_i,b_j} = \sum_d \mathbb{I}(a_{id} = b_{jd}) \tag{3.19}$$

$$a_i, b_j \in \mathbb{R}^d \tag{3.20}$$

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.

3.4 Models

3.5 Prediction Model

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.

3.5.1 Long Short-Term Memory Models

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[**hochreiter**]. 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

$$h_{t+1} = \sigma(Vh_t + Ux_t + b) \tag{3.21}$$

$$o_t = \sigma(Wh_t + b) \tag{3.22}$$

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. Equation 3.5.1 shows a schematic representation of an RNN.

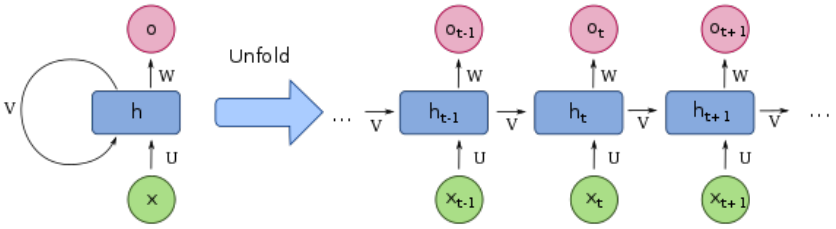

Figure 3.6: A schematic representation of an RNN viewed in compact and unfolded form**??**.

3.5.2 Transformer Model

Transformer model are modern sequnetial models within Deep Learning. They have a multitude of advantages over sequential models such as RNNs or LSTMs CITE . First, they do not need to be computed sequentially. Hence, it is possible to parallelise the training and inference substantially using GPUs. Second, they can take the full sequence as an input using the attention mechanism. This mechanism also allows to inspect which inputs have had a significant role into producing the prediction. However, transformer models are more complicated to implement. The overall setup is shown in **??**[**vaswani˙AttentionAllYou˙2017**].

The transformer model is an Encoder-Decoder model. The encoder takes in the input sequence as a whole and generates a vector which encodes the information. The decoder uses the econding to produce the resulting sequence. The encoder module uses two important concepts. First, in order to preserve the temporal information of the input we encode the position of each sequential input as an additional input vector. We choose to encode every position by jointly learning positional embeddings. The second component is multihead-self-attention. According to **vaswani˙AttentionAllYou˙2017**, we can describe attention as a function which maps a query and a set of key-value pairs to an output. More specifically, self attention allows us to relate every input element in the sequence to be related to any other sequence in the input. The output is a weighted sum of the values. It is possible to stack multiple self-attention modules. This procedure is called multihead-attention. Figure 3.8 shows how to compute self-attention accoring to Equation 3.23[**vaswani˙AttentionAllYou˙2017**].

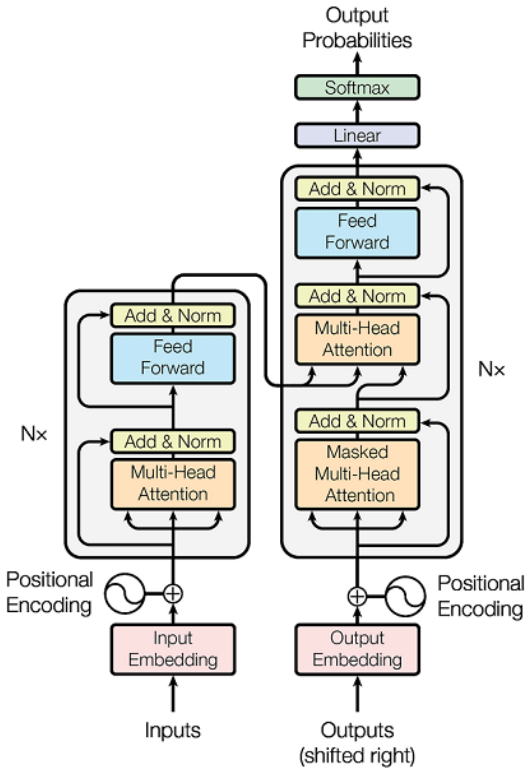

Figure 3.7: A schematic representation of a Transformer model.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.23}$$

Q, K, V are all the same input sequence. d_k refers to the dimension of an input vector. Note, that T is the transpose operation of matrix computations and does not relate to the time step of the final sequence element.

3.6 Generative Models

This section dives into the generative models that we will explore in this thesis. They cover fundamentally different approaches to the counterfactual generation of sequences. The first approach acts as a simple baseline using existing instances in the dataset. Hence, the counterfactual candidates are not generated probablistically but chosen among the data itself. Next, we explore a deep generative approach. Here, we attempt to capture the latent

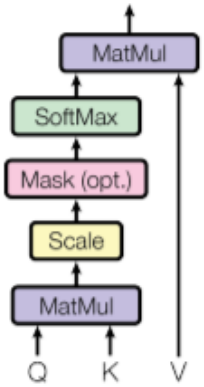

Figure 3.8: Shows the computational graph of self-attention. Q, K and V are all the same input sequence. Q stands for query, K for key and V for value.

state-space of a model and use this space to sample counterfactual candidates. Last, we explore a technique which does not require to optimise a differentiable objective function. Instead we use the viability measure as a fitness function and maximise the fitness of each counterfactual candidate.

3.6.1 Generative Model: Case-Based Approach

Case-based techniques leverage the information that the data provides. The idea is to find suitable candidates the fulfill the counterfactual criterion of precision. In this thesis we find the candidates by searching for alternative process instances that closely resemble the factual instance and still lead to the desired outcome. For this purpose, we use a distance function to compute the distance of every alternative to the factual instance and declare the top-K most-similar insances as counterfactual candidates. 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. We apply the viability metric established in section 3.3 on the subset of of instances which yield the desired outcome. The results of this approach are considered as a suitable baseline for the other methods explored in this thesis. Note, that all case-based approaches automaticall fulfill the criterion of being feasible, as they are drawn from data, that has been *seen*

already. Hence, we expect their feasibility to always be higher than the counterfactuals generated by other methods.

3.6.2 Generative Model: Deep Generative Approach

The generative approach assumes, it is possible to capture a latent state z and use this state to generate suitable counterfactual candidates. We condition the generation procedure on the factual instance to generate counterfactuals that show sparse differences to the original sequence. The core idea is to sample randomly $e^* \sim p(z|e)$ to generate counterfactual candidates. We can sort each candidate by their *viability* and choose top-K contenders as viable couunterfactuals. There are a multitude of approaches to generate the counterfactuals. However, we will limit our exploration to sequential Variational Autoencoders (VAEs) and sequential GANs. Both technieques allow us to sample from a smooth latent space conditioned on the factual sequence. VAEs approximate $p(z|e)$ by trying to reconstruct the input using Monte-Carlo methods. GANs require a generator model and a distriminatior model. The generator model attempts to fool the discriminator model by generating, results that closely resemble true process instances. In contrast, the discriminator tries to distinguish generated instances from real instances.

3.6.3 Generative Model: Evolutionary Approach

Evolutionary approaches use ideas that resemble the process of evolution CITE . Here, we randomly generate candidates by modifying the factual sequence and evaluate their fitness based on a fitness function. Those, cadidates that are deemed as fit enough are subsequently modified to produce offspring. Afterwards, the procedure will repeat until a fitness convergence was reached. This procedure automatically maximises the fitness function. It differs from , because it does not require to use differiantible functions. Hence, we can directly optimise the viability metric established in section 3.3.

3.7 Datasets

3.7.1 Datasets

3.7.2 Preprocessing

Chapter 4

Results

4.1 Evaluation Procedure

4.1.1

Chapter 5

Discussion

Chapter 6

Conclusion

Bibliography

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

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

Molnar, C. (2019). *Interpretable machine learning. a guide for making black box models explainable*. ZSCC: 0001930 tex.ids= molnar_InterpretableMachineLearning_. Retrieved from https://christophm.github.io/interpretable-ml-book/