UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

**Musabir Musabayli**

# Explainable Predictive Process Monitoring

**Master's Thesis (30 EAP)**

Supervisors:

Fabrizio Maria Maggi

Williams Rizzi

Chiara Di Francescomarino

Tartu 2020

# Explainable Predictive Process Monitoring

## Abstract:

The main goal of predictive process monitoring is predicting a possible outcome, execution time, and the cost of a business process by using historical data. The predictions are given at runtime, and historical data is provided in terms of an event log. Each predictive monitoring system contains predictive models which are the main part of it. Predictive models are used to make predictions and are built using information contained in the event logs. However, it is not enough just to show the prediction without giving an explanation since users want to know the rationale behind a prediction. If a person wants to take any action based on the prediction, it is definitely needed to explain the prediction in an understandable way otherwise it would be difficult to trust it. Therefore, in this thesis, we will show why explainable predictive monitoring is useful. We will do this by implementing different predictive model explanation methodologies and by investigating their application in real-life scenarios.

**Keywords:**

Predictive Process Monitoring, Explainable Predictive Process Monitoring, Process Analysis Tool

**CERCS:** P170 - Computer Science, Numerical Analysis, Systems, Control

## Seletatav Ennustav Protsesside Jälgimine

**Lühikokkuvõte:**

Prognoosiva seire põhieesmärk on ennustada võimalik tulemus, selleni jõudmise aeg ning äriprotsessi hind, kasutades ajaloolist andmestikku. Ennustused antakse programmi jooksutamise ajal ja ajalooline andmestik on leitav sündmuste logist. Iga ennustava seire süsteem sisaldab prognoosivaid mudeleid ja koosnebki põhiliselt nendest. Prognoosivad mudelid pannakse kokku sündmuste logis sisalduva informatsiooni abil ja kasutatakse ennustuste tegemiseks. Sellest hoolimata pole ainult võimaliku prognoosi andmine ilma selgitusteta piisav, sest kasutajad soovivad näha ennustustega koos ka põhjendusi. Prognooside põhjal tegutsemiseks on kindlasti vaja ka ennustust arusaadavalt selgitada, vastasel juhul pole see inimeste arvates usaldusväärne. Käesolevas väitekirjas näidatakse, miks seletatud prognoosiv seire on kasulik. Selle jaoks kasutatakse erinevaid prognoosivate mudelite selgitamise meetodeid ning jälgitakse nende rakendamist päris elu stsenaariumites.

**Võtmesõnad:**

Prognoositav Protsesside Jälgimine, Seletatav Ennustav Protsesside Jälgimine, Protsesside Analüüsimise Tööriist

**CERCS:** P170 - Arvutiteadus, arvanalüüs, süsteemid, juhti

# Table of Contents

# 1 Introduction

A predictive business process monitoring system builds predictions over the running process executions to minimize the violation of business constraints and recommend what activities to do to improve the process execution outcomes. There are several techniques for predictive process monitoring. One of them is process intelligence which takes as input the already completed process cases and produces performance analysis [1]. Reactive process monitoring takes an event stream as an input and outputs process performance at runtime. Although these techniques can show the performance of a process either at runtime or not, they find the problems or issues only after they happened. However, it is needed to know the possible problems beforehand and handle them. Predictive process monitoring helps to solve this problem by predicting the future performance of process execution. Predictive process monitoring methods use predictive models that are generated based on information contained in event logs of completed tasks [31]. For predicting future performance, the models are applied to ongoing cases and based on the result, they may alert if the given output does not match with the expected one. However, the main disadvantage of black-box predictive models is that the user does not know the reason behind a prediction and may have difficulties trusting it. There are two different definitions of trust in terms of predictive process monitoring which are trusting an individual prediction and trusting the model. They are directly related to how much a person understands the model's behavior [31]. When a user takes an action relying on individual predictions, trusting is very important. For example, it is important to trust a prediction when the treatment for a specific patient needs to be chosen by a doctor in a hospital. Besides an individual prediction, it is necessary to trust the model altogether. Indeed, users want to be confident about any predictions coming from that model besides being confident that the model will perform very well on real-world data.

Although there are enough papers on process monitoring and predictive process monitoring, there are only a few studies about explainable predictive process monitoring systems. This is due to the fact that explainable predictive process monitoring is a new approach in Business Process Management (BPM), and the studies about this topic mainly focus on the general concepts like why explanation is important and on different metrics

to measure interpretability. In the Machine Learning field, there are frameworks and algorithms for generating explanations for a predictive model. The aim of the thesis is to apply this research in the BPM context and integrate and compare the results of different algorithms, frameworks, and techniques for explainable predictive business process monitoring in terms of accuracy and comprehensibility.

# 2 Related work

Although process monitoring is a well-established concept that mainly focuses on tracking performance changes of a system, providing an explanation about a future prediction is a new term. There are plenty of techniques and frameworks that elicit predictions about future performance based on historical data for ongoing cases. However, sometimes, it is not sufficient to precisely predict that a running process will finish with a certain outcome because users want to have an idea about why a certain prediction has been returned by the algorithm and a way to prevent the undesired outcomes [33].

Prediction models are the core component of all predictive process monitoring systems. These created models are applied to prefixes of running cases, and as a result, future performance predictions are generated [1]. Many machine learning algorithms can build very accurate models, nonetheless, the results they provide are complex to interpret by a user. According to R. Guidotti et al., there are three dimensions for the interpretability of a predictive model [32].

1. **Global and Local interpretability:** In global interpretability, the whole logic of a model is interpretable while in local interpretability, it is possible to understand the reason behind an individual prediction.

2. **Time Limitation:** It is the time that a user spends to understand an individual explanation for a specific prediction or the general explanations for a model. This metric is used for measuring how an explanation is easy to understand.

3. **Nature of User Expertise:** Not all users have the same background. Thus, knowing the user experience beforehand is the key factor for providing good explanations. For example, showing the same kind of explanation to a surgeon or data scientist is meaningless because they have different backgrounds.

All the black-box models are opened by an explanator, and according to R. Guidotti el al., those explanators are categorized based on their types [32].

1. **Decision Tree:** It describes a predictive model like a tree. Each branch shows a possible outcome and the path from the root to the leaves describes classification rules.

2. **Decision Rules:** These rules are used to explain a particular predictive model and outcome. These rules are generated based on a tree.

3. **Feature importance:** It is related to the magnitude and weight of the attributes that are used by the black-box model to make a prediction. It is a kind of metric that is used to determine which values or attributes are more important and have more impact on the decision.

4. **Saliency Mask:** It determines the reason for a certain outcome and visualizes attributes based on the importance level.

5. **Sensitive analysis:** This is used to evaluate the uncertainty of a black-box model.

6. **Partial dependence plot:** It is a plot that shows the connection between black-box output and input.

7. **Prototype selection:** It is a collection of similar instances, and it is used to know which criteria were returned by a prediction.

8. **Activation maximization:** It is used in neural networks to find out which are the fundamental neurons when making a decision. Moreover, it can be used to find the areas or pixels in a black box model which mainly affect a decision.

For real-world systems, it is very necessary to take into account *why* comprehensibility is important. When interpretability is designed, five main factors should be taken into account: transparency, trust, model debugging, learning from the model, and ethics and regulations [33].

1. **Transparency:** It is opposite to opacity which means the inability to see why machine learning decisions are generated from *opaque* models.

2. **Trust**: It is about trusting the model's decision like informing beforehand when it will perform well or when it will not perform well.
3. **Model debugging:** In order to improve the model, it is required to understand its weaknesses, and this is done by model debugging. Improvement can be done in three ways: by using explanations to see where the model fails, by using model interpretability to detect human bias in the data and model and comparing different models.
4. **Learning from the model:** Learning from big data may end up in finding intricate and complex patterns that may not appear to a human by observation.
5. **Ethics and legislation:** Nowadays, the predictions generated by machine learning algorithms are applied in the systems where they have an impact on humans. They may show unwanted *discrimination* in which the input data may contain some correlation that we might not want to capture in the system. There are some characteristics like race, gender, sexuality, and religion that are illegal properties to be used for prediction oftentimes. As a result, machine learning systems may cause some potential issues related to ethics and laws regulating the use of the data.

In order to visualize the output of a black-box model in an interpretable way, different frameworks have been developed. All these techniques have been categorized based on some attributes. Marcel Robeer wrote a paper about why an explanation is needed and show libraries and algorithms that can be used to interpret a black-box model [33]. Moreover, a similar study has been done by Vanessa Buhrmester, David Münch, and Michael Arens [34]. By combining the result of those two studies, we created a table of open source libraries.

| **Column** | **Abbreviations** |
|---|---|
| Name | Technique name |
| Publication year | When the technique was published |
| Explanation scope | Global, Local |

| | |
|---|---|
| Explanation approach | Decompositional, Pedagogical, Transparent |
| Representation | Decision Rules, Decision Tree, Feature Importance, Prototype Selection, Prototype Reconstruction, Saliency (Sensitivity Analysis, Signal Methods, Attribution Methods), Linear model, Annotation, * other |
| Machine Learning Method Type | Unsupervised, Supervised (Classification, Regression), Reinforcement Learning |
| Code | Programming language used |

*Table 1*

Table 1 is the representation of each column's meaning of Table 2. In particular, Table 2 includes technique name, reference, publication year, explanation scope, explanation approach, representation, machine learning method type, and source code language.

| **Name** | **Reference** | **Year** | **Scope** | **Approach** | **Representation** | **ML Type** | **Code** |
|---|---|---|---|---|---|---|---|
| GENESIM | [2] | 2016 | G | D | PR | C | Python |
| DeepVis | [3] | 2015 | G | D | PR | C | Python |
| Nguyen et al. | [4] | 2016 | G | D | PR | C | Python |
| STEL | [5] | 2014 | G | P | DR | C, R | R |
| GA2M | [6] | 2012 | G | P | FI | C, R | Java |
| GoldenEye | [7] | 2014 | G | P | FI | C | R |
| OPIA | [8] | 2015 | G | P | FI | C, R | Python |
| PDPbox | [9] | 2015 | G | P | PDP | C, R | Python |
| Prospector | [10] | 2016 | G | P | PDP | C | Python |
| GFA | [11] | 2018 | G | P | PDP | C | Python |
| RuleFit | [12] | 2008 | G | T | DR | C, R | Python |
| BRL | [13] | 2015 | G | T | DR | C, R | Python |
| FRL | [14] | 2015 | G | T | DR | C | Python |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Bayesian Rule Set | [15] | 2017 | G | T | DR | C | Python |
| SHAP | [16] | 2016 | G, L | P | FI | C, R | Python |
| LIME | [17] | 2016 | G, L | P | FI | C, R | Python |
| Streak | [18] | 2017 | G, L | P | FI | C, R | Python |
| Anchor | [19] | 2018 | G, L | P | DR, PS | C | Python |
| IG | [20] | 2017 | L | D | AM | C | Python |
| Excit. Backprop | [21] | 2017 | L | D | AM | C | Python |
| NeuralTalk | [22] | 2015 | L | D | FI | C | Torch (Lua) |
| LSTMVis | [23] | 2018 | L | D | FI | U, C | Python |
| Deep-Google | [24] | 2015 | L | D | PR | C | Matlab |
| CAM | [25] | 2015 | L | D | SM | C | Python |
| Grad-CAM | [26] | 2015 | L | D | SM | C | Torch (Lua) |
| DeepLIFT | [27] | 2017 | L | D | FI | C | Python |
| RCNN | [28] | 2016 | L | T | FI | C | Python |
| Skater | [29] | 2018 | G | P | FI | C, R | Python |

*Table 2*

While there are studies and documentation about each framework separately, there have not been any studies that implement and compare the result of several different techniques in the same paper.

# 3 Background

In this section, we discuss the main background concepts useful to understand the thesis. We first introduce process mining and predictive process monitoring and continue to show how a model is created in Nirdizati Research, a web application for predictive process monitoring. Then we introduce the main explainability concepts and introduce each of the explanators we have implemented in Nirdizati Research.

## 3.1 Process Mining

In the field of process management, process mining is used in order to analyze the business processes underlying information systems. Process mining is a family of techniques that aim at identifying patterns, trends, and details starting from an event log. An event log contains a set of traces which tracked by an information system, and each trace is composed of a sequence of events that describe a process execution. A trace can have *static* attributes, i.e., attributes that do not change during the trace lifecycle. Events can also have attributes, named *event attributes*. The internal structure of a trace is described in Figure 1 .

```
<trace>
  <string key="concept:name" value="2_100"/>
  <string key="label" value="false"/>
  <event>
    <string key="PClaims" value="No"/>
    <string key="org:resource" value="PR3"/>
    <string key="concept:name" value="Register"/>
    <float key="ClaimValue" value="944.128875155"/>
    <string key="CType" value="Silver"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="ClType" value="H11"/>
    <float key="Age" value="52.0"/>
    <date key="time:timestamp" value="2003-04-06T05:21:00.000+02:00"/>
  </event>
  <event>
    <string key="PClaims" value="No"/>
    <string key="org:resource" value="Mgr1"/>
    <string key="concept:name" value="Create Questionnaire"/>
    <float key="ClaimValue" value="944.128875155"/>
    <string key="CType" value="Silver"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="ClType" value="H11"/>
    <float key="Age" value="52.0"/>
    <date key="time:timestamp" value="2003-04-06T06:20:00.000+02:00"/>
  </event>
</trace>
```

*Figure 1*

The goal of process mining is to understand processes and increase the efficiency of processes. Based on the models created by process mining algorithms, prediction for running instances can be calculated which is done by predictive process monitoring algorithms. The goal of predictive process monitoring is to monitor the process execution continuously and predict the future performance of the system.

## 3.2 Nirdizati Research

Nirdizati Research is a web-based application that builds a prediction about the running cases of a process [30]. The application is running at http://research.nirdizati.org/. The back-end side of the application has been developed in Python while the front-end project has been written in React. As seen in Figure 2 , there are 8 different pages in the application. In the **Upload** section, a user can upload either a single file or two different log files for validation and training. The log details can be checked from the **Log Details** section like the number of executed events, employed resources, and new traces per day in a graphical representation. With the help of splitting functionality, a single log input file is divided into two files which are used as training and testing event log files. By default, 80% of an event log is considered for training while the remaining 20% is used for testing. Nonetheless, this percentage ratio can be changed in the **Split** section. The goal of machine learning algorithms is to find the output value, which is a label, and in the **Labeling** section, a model can be labeled in different ways. For a classification method, the label is categorical while for a regression method, it is numerical. One of the core components is **Training** which produces predictive models based on a configuration selected by a user. From the **Task Status** page, the status of models and configuration parameters can be checked. The generated predictive models are used in **Validation** to see the result of the models and to make predictions on a stream of events. The explanation page is described in the Contribution section.

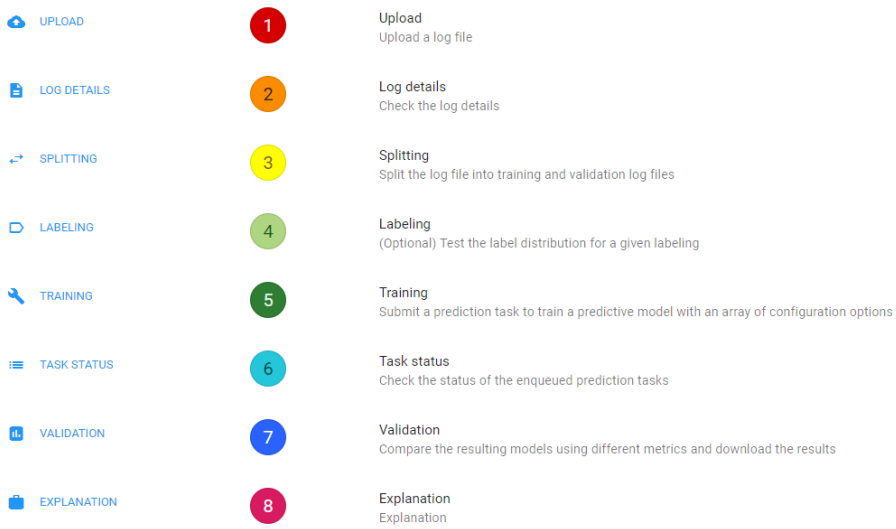


*Figure 2*

In order to create a model, our system provides different configurations. Firstly, it is required to select an event log and prediction method. The event log is split into train and validation data sets by the system when the user uploads an event log to the system. After that, the user is asked to select at least one learning method. The application supports 3 prediction methods which are Regression, Classification, and Time Series Prediction. For each method, the user can select one or more techniques. Regression there are 5 options: Linear, Random Forest, Lasso, XGBoost, and NN. For Classification there are 10 options: KNN, Decision tree, Random forest, XGBoost, Naive Bayes, Hoeffding tree, Adaptive tree, SGD classifier, Perceptron, and NN. For Time Series Prediction, there is only 1 option which is RNN. Moreover, the system supports various encoding types: Simple index, Boolean, Frequency, Complex, and Last payload. In **Simple index** encoding, event attributes are discarded, and each value corresponds to the activity name of the event at a given position in the trace. In **Boolean** encoding, a column corresponding to each activity name is created, and the value represents whether the event has occurred in the case. Event attributes are discarded also in this case. Like Boolean encoding, a column for each activity name is created in the **Frequency** encoding, however, the value shows the frequency of each event. Like Simple and Boolean encoding methods, event attributes are discarded too. In **Complex** encoding, event attributes are included, and each value corresponds to activity name and value in the trace based on a position. The last encoding

type is **Last payload**, in which the values are the last event attributes in the trace. In addition, Encoded log padding and Task generation type should be specified by the user too. There are two types of log paddings which are **No padding** means only traces with higher or equal length to stated prefix length are kept, and **With 0 padding** means all the traces with a length lower than a specified prefix length are padded with 0. Prefix length is a number that can be specified in the **Task generation type** field. The first events of the trace till the specified prefix length are considered for each trace. Finally, it is required to select the labeling type which differs depending on the prediction method. For regression methods, there are only two types: (i) **Remaining time** – i.e., the time requested until the last event in a trace, which depends on the prefix length; (ii) **Trace numerical attributes** – which is the same for all the selected prefix lengths. For classification methods, the labelling types are: (i) Duration; (ii) Next activity; (iii) Trace numerical attributes; and (iv) Trace string attribute. A summary of the different labelling types for the classification method is reported in Table 3. For each option, the classification type is reported together with the information on whether the specific labelling types requires to specify a threshold, a trace attribute as well as whether the labelling type depends on the prefix length. Depending on the labelling type, indeed, it might be required to select a threshold. This threshold is a number that is used to set a value for the label. For example, for numerical values below a certain threshold, the output label is *False* while for the values above or equal to the specified threshold, the output is *True*. However, the threshold can be identified by the system too. In this case, the mean of the values in the attribute column is taken as a default threshold. As a result of these selections, a model is generated, and the model contains encoded data frames of train and test data sets with the specified options. An encoded data frame can be converted to a decoded data frame in which the result contains the actual value for each feature as the original event log.

| **Labeling type** | **Classification type** | **Threshold required** | **Trace attribute required** | **Dependency from prefix length** |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| Duration | Binary | Yes | No | No |
| Next activity | Multiclass | No | No | Yes |
| Trace numerical attributes | Binary | Yes | Yes | No |
| Trace string attribute | Multiclass | No | Yes | No |

*Table 3*

## 3.3 Explainability

The explanation of a prediction is necessary for a company to create a trustable and safe predictive monitoring system. Without having explanations, decisions are based on black-box models. A black-box predictor is a model whose internal behavior is unknown for the user and is not human-understandable. In some cases, explanations are very important such as for decisions related to self-driving cars or where wrong decisions may lead to a life-threatening outcome [32].

We can identify two types of explainability: local and global explainability, for a particular model. In local explainability, an explanation behind an individual decision or prediction is provided, while in global explainability only a single explanation is given. Local explainability can be given, for instance, with respect to the prediction related to the future of a specific trace.

Besides explainability types, all the explanation techniques are grouped under two categories: ante-hoc and post-hoc. In ante-hoc methods, prediction and explanation are given based on the same model, and the explainability is incorporated into the model itself. Linear regression, decision trees, random forests, and Bayesian deep learning are examples of it. On the other hand, in post-hoc techniques, prediction and explanation are given based on different models. Another model is trained, and it is used to provide an explanation for a specific local decision. LIME, SHAP, BETA, Local Gradient Explanation are the most used examples of post-hoc techniques.

There are several techniques and libraries that provide explanations for black-box models. In our project, we have selected those techniques using different criteria: the source code should be publicly available, the source code should be written in Python (the techniques have been integrated into Nirdizati Research, a web application for predictive process monitoring written in Python), tabular data should be accepted because our system works with this kind of data, any machine learning model can be explained especially regression and classification, and the provided output has to be easily understandable by a user. After eliminating the methods based on these specifications, only 4 libraries were implemented in the Nirdizati Research project: LIME, SHAP, PDPbox, and Skater.

### *3.3.1 LIME*

One of the most used libraries for explaining a predictive model is LIME that is an abbreviation of Local Interpretable Model-Agnostic Explanations. It is a library developed in Python. The goal is to explain any models or classifiers in an interpretable and human-understandable manner [17]. LIME can be used to explain an individual prediction for textual, tabular, image, and text classifiers, and it is a post-hoc technique. Besides it, it can provide a global interpretation for a model by combining local explanations.

With the help of LIME, it is possible to present textual explanations or visual plots or artifacts which provide an explanation for decisions.

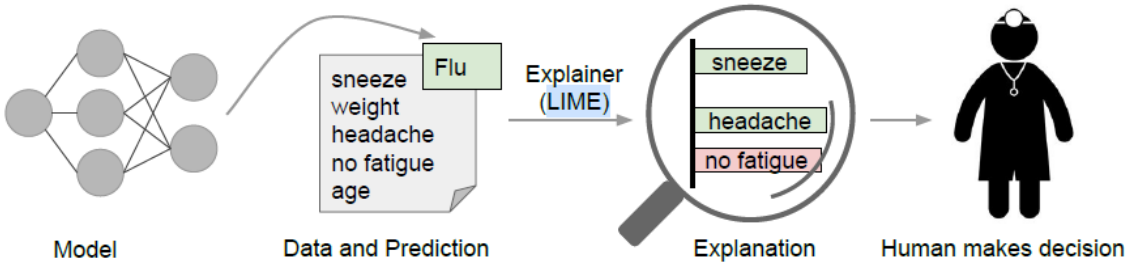


*Figure 3*

Figure 3 is a representation of how LIME works for an individual prediction. The model indicates that the prediction outcome is flu, and the factors are taken into consideration for making the prediction are sneeze, weight, headache, no fatigue, and age. The explainer takes the factors and the decision and classifies the related and unrelated ones. As a result,

sneeze and headache are the signs for flu while no fatigue is a sign for not having flu. Finally, it generates a result by showing the importance of each feature for the prediction.

Checking whether an explanation makes sense or not is important in terms of measuring the trustworthiness of the model because sometimes a model can contain correlation and generalizations which are not accurate. As an example, LIME was applied to a text classifier for exploring how much a text is about "Christianity" and "Atheism". The accuracy of the model was high enough: 94%. Nonetheless, the explanations showed that predictions were not accurate since they were calculated based on several words that are unrelated to both groups like "host" or "posting" (Figure 4). The example shows that although accuracy is high this can be derived from wrong assumptions.

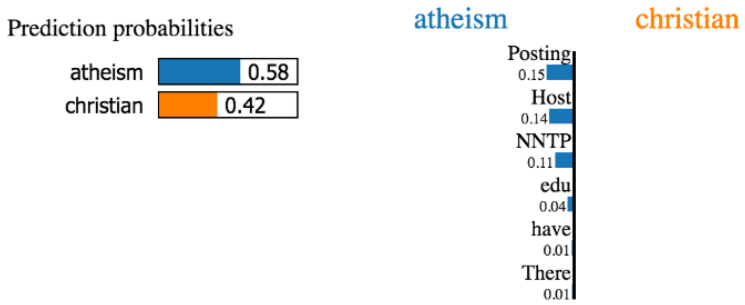


*Figure 4*

### *3.3.2 SHAP*

There are several methods for interpreting a model, however, it is unknown which method is suitable for a particular model. Therefore, one unique framework was developed which combines different techniques that are called SHAP [16]. It stands for "Shapley Additive exPlanation". Like LIME, SHAP is also a Python package, and it is a post-hoc technique. SHAP is working based on SHAP values which take into account all possible predictions from various methods for a single instance by using all different combinations of inputs. As a result, it produces a value that is the measure of feature importance. Therefore, SHAP provides high accuracy and consistency in the output.

SHAP gathers 6 different methods (one of them is LIME). Using different methods makes the framework slow in some cases. In order to solve this problem, an optimization method was developed which helps to decrease the execution time for some models.

SHAP has different implementations that are listed below:

- TreeExplainer: It is a very fast algorithm for calculating SHAP values for trees.
- DeepExplainer: It is an algorithm for calculating SHAP values for deep learning models.
- GradientExplainer: It is a method for the integration of expected gradients to estimate SHAP values for deep learning models.
- LinearExplainer: It computes SHAP values for linear models.
- KernalExplainer: Different from the others, it computes SHAP values for all models since it does not make any assumption about the model type. However, it is slower than others.

The main advantage of using SHAP is a highly accurate outcome and works for all models. Although SHAP is slower than LIME for some models, LIME cannot be used with all models. For example, it is not possible to handle XGBoost predictors.

### *3.3.3 Skater*

Like SHAP, Skater is also a unified framework that provides model interpretation for all kinds of models [32]. It is an open-source framework that was developed in Python. The advantage of Skater is to provide an individual and global explanation for a black-box model.

In the beginning, it was started as a fork of LIME, and for local explanations, and the explanations were generated in the same way as LIME. However, as an improvement, new algorithms were implemented to give both global and local explanations. Currently, a model-agnostic algorithm for global and local explanations is supported by Skater, and feature importance and partial dependence in a tree format are provided. Feature importance means a degree of a feature affecting the predictive model. It is measured by how much it affects a prediction if the value of the feature is changed.

Figure 5 shows that Age is more important to get the prediction than the other attributes.

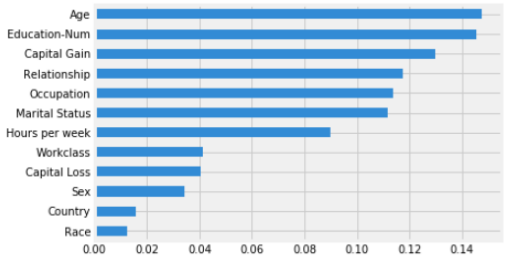


*Figure 5*

In partial dependence, it is shown how a specific feature affects the output by holding other features constant. Figure 6 shows the dependence of the average salary from Age.

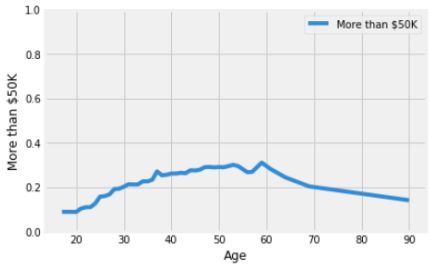


*Figure 6*

Skater supports different kinds of visualization of results. One of the ways to describe an output is a decision tree that shows the importance of features of a model (this type of visualization was implemented in the Nirdizati Research project).

### *3.3.4 PDPbox*

In order to understand the relations between predictors and model results, PDPbox can be used to give an easy interpretation of it [9]. The goal is to show the impact of particular features toward the model prediction, and it works for any supervised learning algorithms. PDPbox is an abbreviation of Python Partial Dependence Plot toolbox. As it is seen from the name, it provides a partial dependence plot which is a representation of dependency of a feature from the model. It is used for global interpretation, and as a predictive model, and it accepts regression and classification models. The source code of the framework is publicly available and was written in Python. PDPbox is inspired by ICEbox which is a library written in R for Individual Conditional Exception plots.

Although it supports all kinds of prediction methods, it only accepts numerical encodings.

# 4 Contribution

In this study, we have implemented explainable predictive process monitoring in the context of Nirdizati Research, a web application for predictive process monitoring. In this part, we will provide information about the new functionalities that have been implemented in the system. In the previous version, Nirdizati Research did not provide any explanation to users. So, the contribution of this thesis is mainly focused on adding a new page to Nirdizati Research that provides explanations for a model and that page is called **Explanation**.

In order to explain the output value of a black-box model, 5 different techniques have been implemented and tested, and based on their results, the best ones have been selected. SHAP(SHapley Additive exPlanation), LIME(Local Interpretable Model-Agnostic Explanation), Skater, PDPbox(Python Partial Dependence Plot Toolbox), Temporal stability, and Confusion matrix feedback are the implemented techniques and algorithms. An explanation is given for a model, and for getting an explanation, initially, it is required to select a model. The general overview of the explanation page is given in
Figure 7. As seen from the image, the page contains several cards, and the goal of each card will be discussed in the following sections.

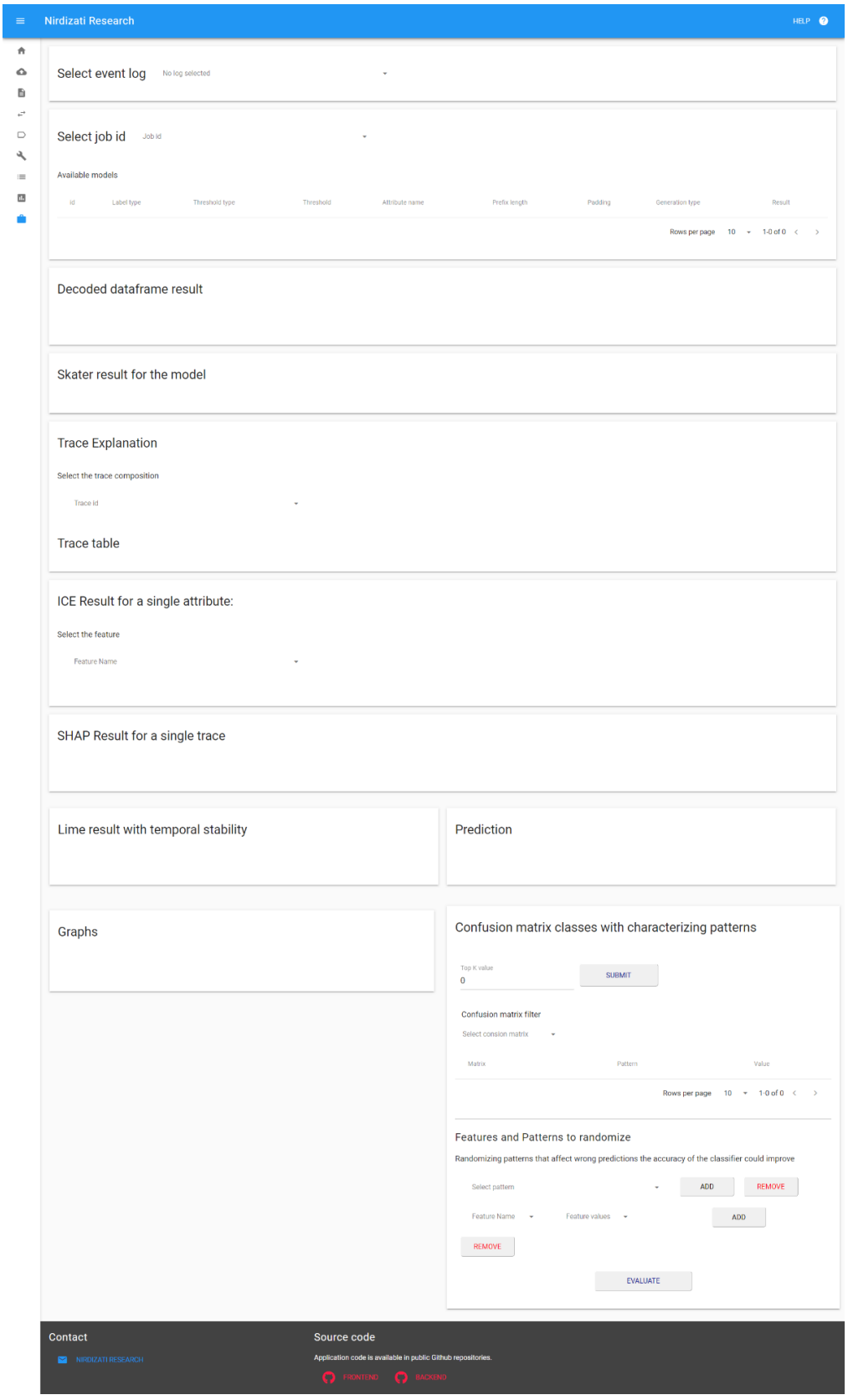
Nirdizati Research
HELP
Select event log
No log selected
Select job id
Job id
Available models
id
Label type
Threshold type
Threshold
Attribute name
Prefix length
Padding
Generation type
Result
Rows per page
10
1-0 of 0
Decoded dataframe result
Skater result for the model
Trace Explanation
Select the trace composition
Trace id
Trace table
ICE Result for a single attribute:
Select the feature
Feature Name
SHAP Result for a single trace
Lime result with temporal stability
Prediction
Graphs
Confusion matrix classes with characterizing patterns
Top K value
0
SUBMIT
Confusion matrix filter
Select consion matrix
Matrix
Pattern
Value
Rows per page
10
1-0 of 0
Features and Patterns to randomize
Randomizing patterns that affect wrong predictions the accuracy of the classifier could improve
Select pattern
ADD
REMOVE
Feature Name
Feature values
ADD
REMOVE
EVALUATE
Contact
NIRDIZATI RESEARCH
Source code
Application code is available in public Github repositories.
FRONTEND
BACKEND

*Figure 7*

The architecture of the Explanation page is described below in Figure 8.

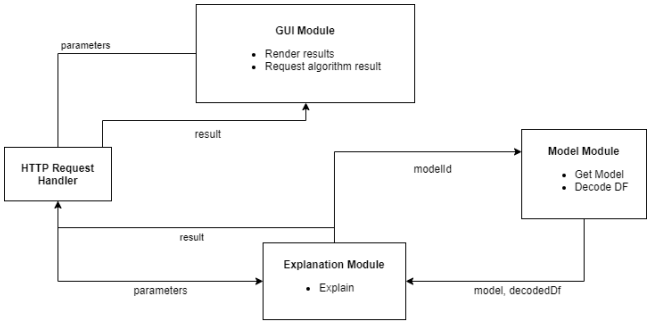


*Figure 8*

As can be seen in Figure 8, HTTP Request Handler module is in charge of sending HTTP requests and handling the response. Explanation and Model modules are the components of the back-end project, and they are two independent services. The Model module is used to handle all the functionalities related to a model while Explanation module handles all explanation techniques' requests. Finally, the GUI module is responsible of rendering the explanation page and showing the results. Figure 10 reports a class diagram that shows the internal structure of the Explanation page – GUI Module.

Furthermore, Figure 10 represents a class diagram for the back-end project which shows how the requests of the Explanation page are handled by the system. All the child classes are connected to a single parent class. Initially, the request is handled in the "Explanation" class, and the model based on a requested model id is found and pass to the requested technique class. For each technique, we have designed a separate class.

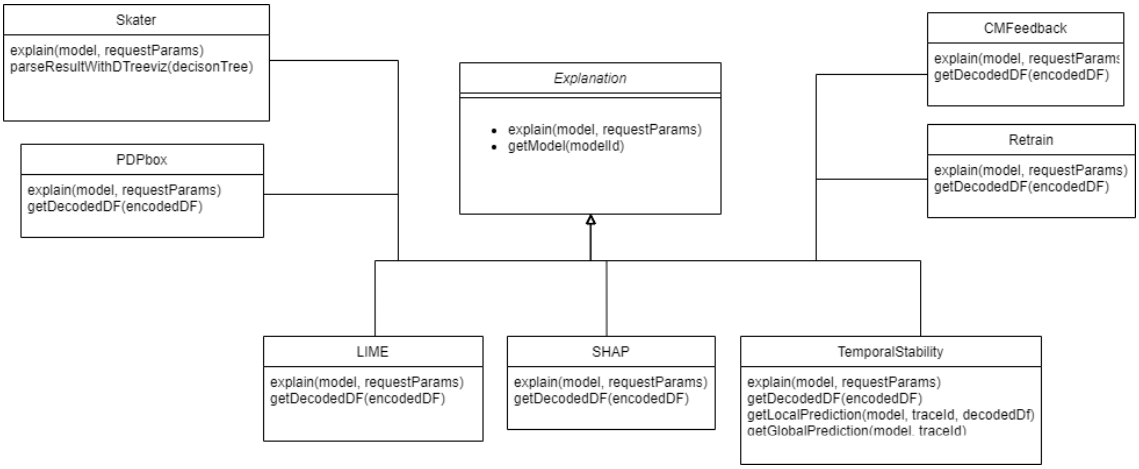


*Figure 9*

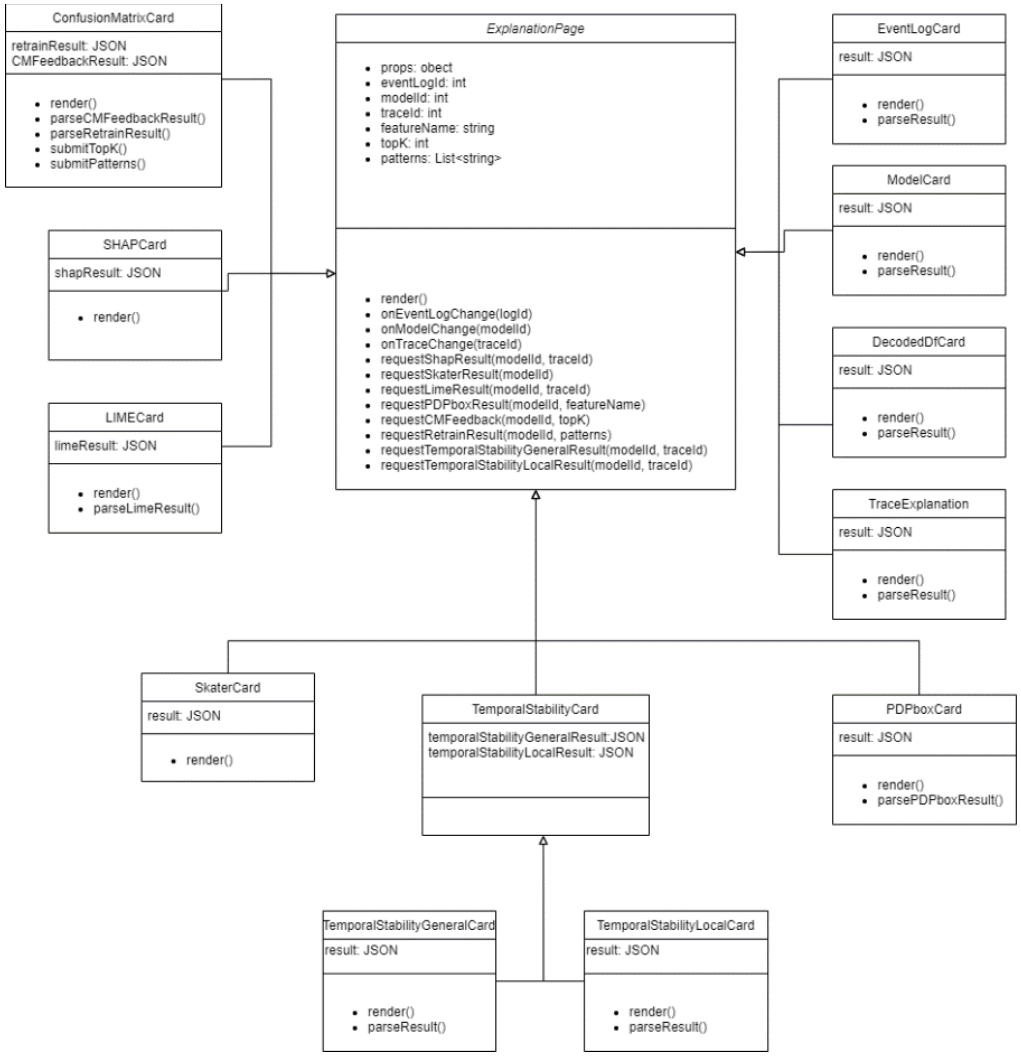


*Figure 10*

It is easy to understand how the cards in the Explanation page are connected. "ExplanationPage" is a parent class, and the rest of the classes are the child classes in which each class represents a card in Figure 10. The request for calling an Application Program Interface (API) is sent in the parent class, nonetheless, the response is handled in the child classes.

All these diagrams might be helpful to understand the sequence diagrams that are described in the following sections.

## 4.1 UI elements

### *4.1.1 Select event log*

The first card allows the user to select an event log. When a user goes to the page, the API for retrieving the list of event logs is requested in the background. If the request is successful, the list of all event log names is shown to the user in the dropdown menu like in Figure 11. The event log name is a concatenation of split id, train, and test log names.

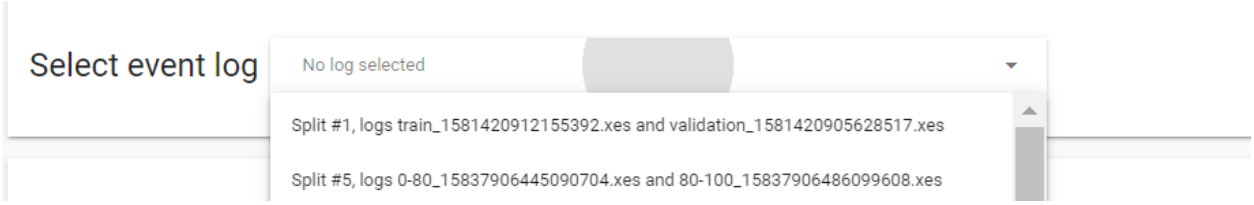


*Figure 11*

### *4.1.2 Select model*

Event logs are input files for the model training. After a user selects a split up event log, all the trained models with the selected event log are visible in the "Available models" table (Figure 12). Each row in the table represents the configuration options of a model. The table contains 9 columns which are model id, label type, threshold type, attribute name, prefix length, padding, generation type and result. The meaning of these columns has been explained in the Background section.

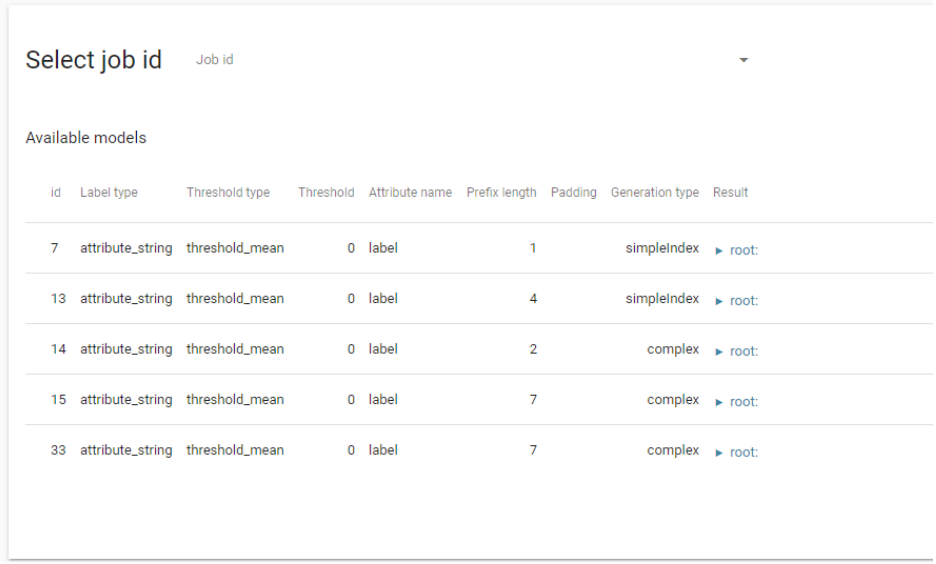


*Figure 12*

The user can select a model id from the dropdown menu above the table.

### *4.1.3 Trace explanation*

Figure 1 shows the structure of a trace which contains two global variables "concept:name" and "label" and two events. The goal of the machine learning techniques that have been implemented is to provide an explanation about how the output label for a given trace is related to the given event attributes and which features are more important for predicting the label.

Among the techniques we have implemented, SHAP, LIME, and Temporal stability require a trace id in order to provide an output since they have been used for local interpretation. Local interpretation is a method to provide an output for a specific trace while global interpretation provides a general output for a whole model (see Background section). The given trace is selected from the testing set. As it can be seen in Figure 13, the "Trace Explanation" card has a dropdown menu that contains a list of the id of the traces in the testing set and when a trace is selected, global and event attributes are shown in the table.

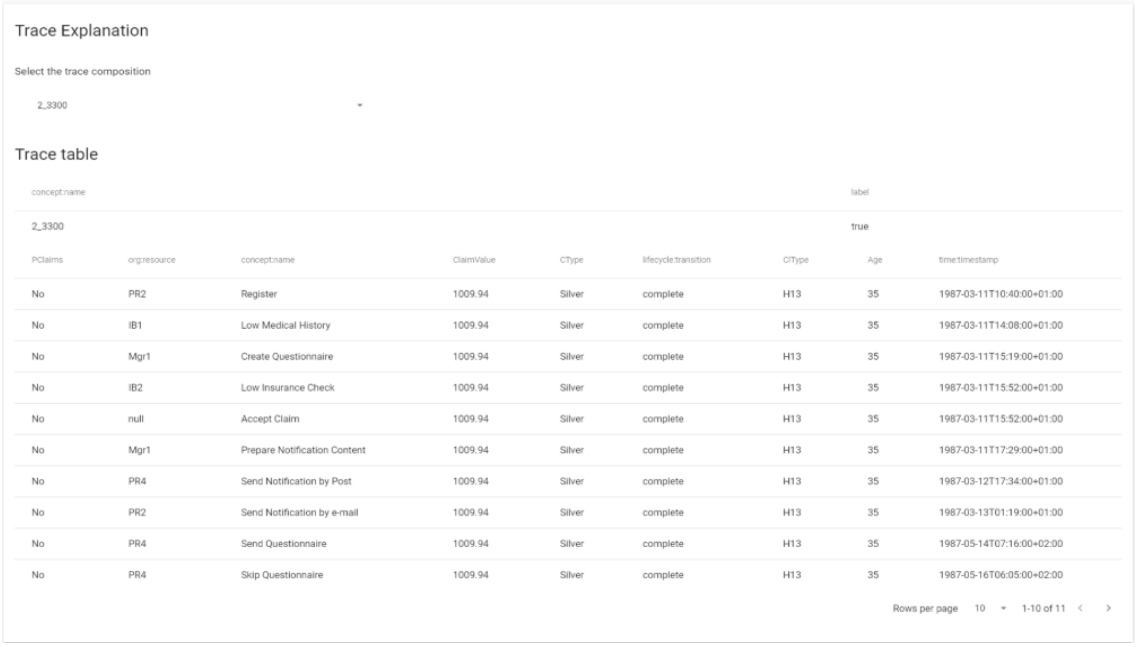


*Figure 13*

Moreover, in order to decrease the number of requests, the API for retrieving test set data is called only once when a model is selected, and the output data list is stored in the

cache memory. With the help of this caching mechanism, a user can see other trace attributes without waiting.

### *4.1.4 Decoded dataframe result*

When the model is trained, the input log file is encoded based on an encoding selected by the user. Some of the explanation techniques require specific encoded data frames as an input because they are working with numerical values only. The results of those techniques contain encoded values which are the combination of feature name and feature value index. An encoded data frame may not be understandable by the user. In order to get the decoded data frame, a request is sent to the backend. In the backend side, if the request is successful, it is handled, and the decoded data frame is provided as a response. For visualizing the internal structure of the response, a table has been added (see Figure 14).

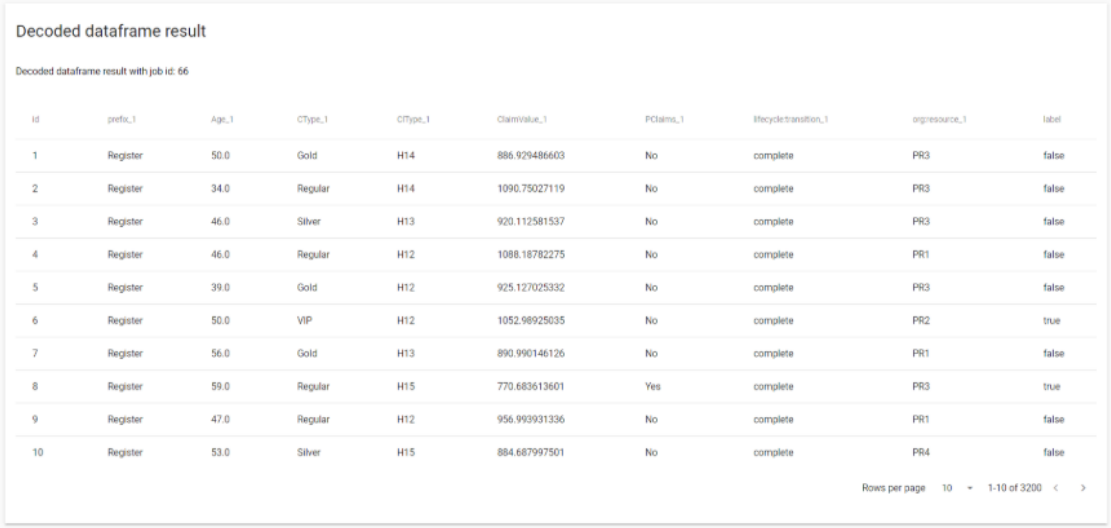


*Figure 14*

The sequences of activities required for the decoded data frame result are shown Figure 15.

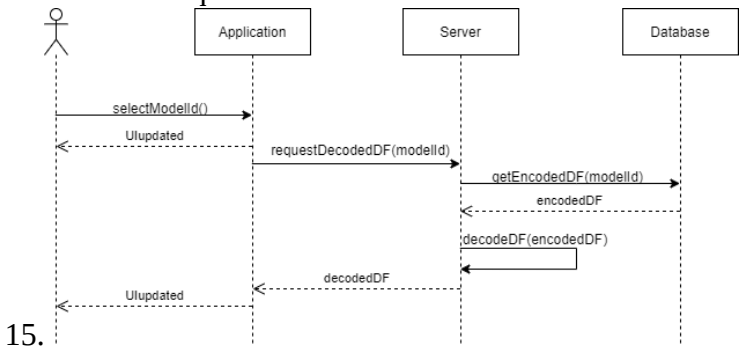


*Figure 15*

### *4.1.5 Skater*

Skater is one of the methods for black-box model interpretation and can be used for local and global interpretation. In the Nirdizati Research project, Skater is used for global interpretation. The card for Skater is added below the decoded model view card because Skater returns encoded outputs. In order to check the real value of an encoded value from the Skater plot, the user can check values from the table. The result of this technique is a decision tree. This model works for regression and classification methods.

When a user selects a model, the API for getting Skater results is requested by including the model id. The request is handled on the back-end side. Initially, it is checked whether the model belongs to a regression or classification prediction method. The sequence of activities that start from selecting a model up to the presentation of the results in the application is described in Figure 16.

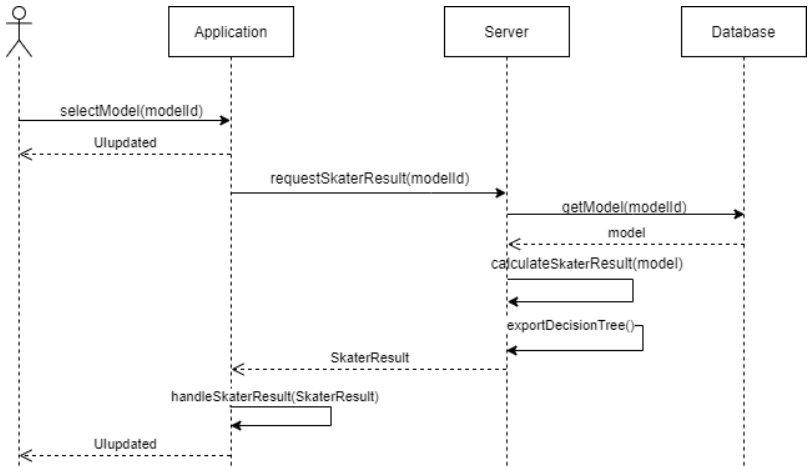


*Figure 16*

The resulting decision tree that Skater provides is not interpretable for large results. Dtreeviz is used to get a more interpretable decision tree [36]. The decision tree contains only the variables that have a role in the output label. Dtreeviz is a Python library that takes the model as input, and as a result, it returns a new decision tree in a more understandable structure. The resulting tree for the classification method is shown in Figure 17.

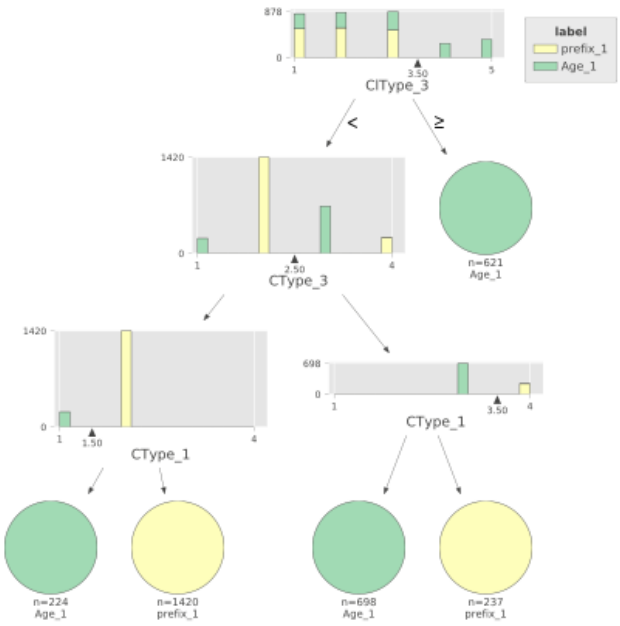


*Figure 17*

The green color represents the *True* label, while the yellow color represents the *False* label which is named incorrectly as "prefix_1" and "Age_1" by the framework. The result shows the number of occurrences of labels depending on "ClType" and "CType" attributes. A threshold for identifying left and right branches is defined in all nodes. For instance, in the first node the threshold is 3.50. The explanation of Figure 17 is given in section 3.3.3 in a detailed form.

### *4.1.6 SHAP*

As a second technique, SHapley Additive exPlanation – SHAP has been implemented. SHAP assigns an importance value for each feature based on a prediction. SHAP provides local and global interpretations while we use it only for local interpretation.

Local interpretation explains a specific prediction. Therefore, it is required to select a trace in order to get the SHAP result. A request for getting the output is called after a trace is selected. Thus, in the request, the model and trace id are specified. In order to calculate SHAP values, the method requires an encoded data frame as an input. In addition, a decoded data frame is passed as an argument for generating the resulting graph. The sequence of activities for getting the SHAP result is shown in Figure 18.

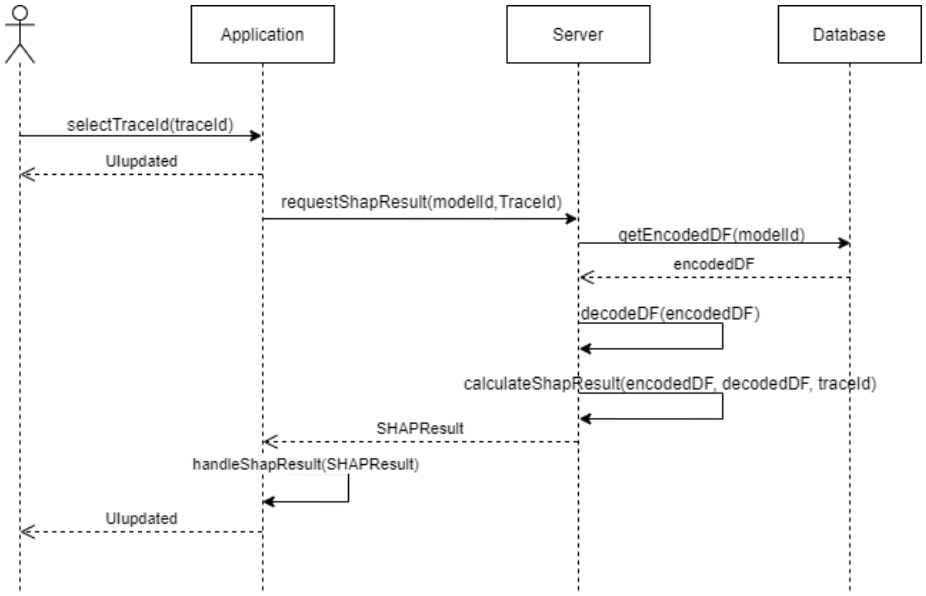


*Figure 18*

When the application gets a successful result from the server, it shows the resulting graph in the SHAP result card. Furthermore, the selected model and trace id are shown inside the card.

SHAP provides different kinds of visualizations for local interpretation which are scatter, bar, dependence, and violin plots. Nonetheless, the graph we show in the application is called "Force plot" (Figure 19). The SHAP values are divided into two parts: the ones that lead the output label to get value *False* (colored in red) and the ones that lead the output label to get value *True* (colored in blue).

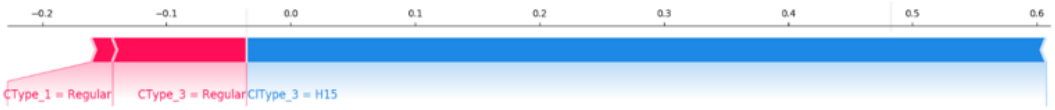


*Figure 19*

### *4.1.7 PDPbox*

Python partial dependence plot toolbox is another global explanation technique to visualize the impact of a feature on a prediction. It works for all supervised machine learning methods. The goal of this technique is to provide an easily understandable

explanation. Like SHAP, this library also provides different kinds of results, and in the application, the result is shown as an information plot.

Differently from other techniques, in this method, the result is generated based on the feature name. The feature name is selected from a dropdown menu which is the list of all the features available. When a user selects a feature, a request is sent to the server by including model id and feature name. The order of the activities for getting PDPbox results is shown in Figure 20.

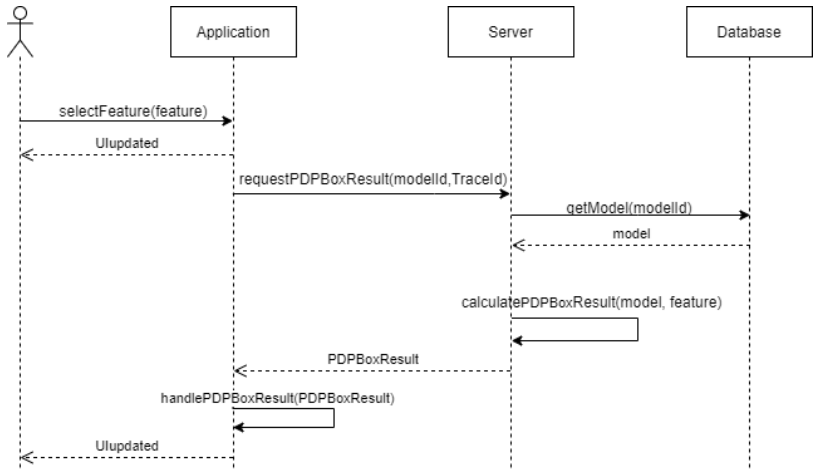


*Figure 20*

The response is sent in JSON format which contains a list of objects with three variables: feature value, label, and count. For classification methods, a label is between 1 and 2 where 1 indicates false while 2 indicates true. The output is shown in Figure 21.

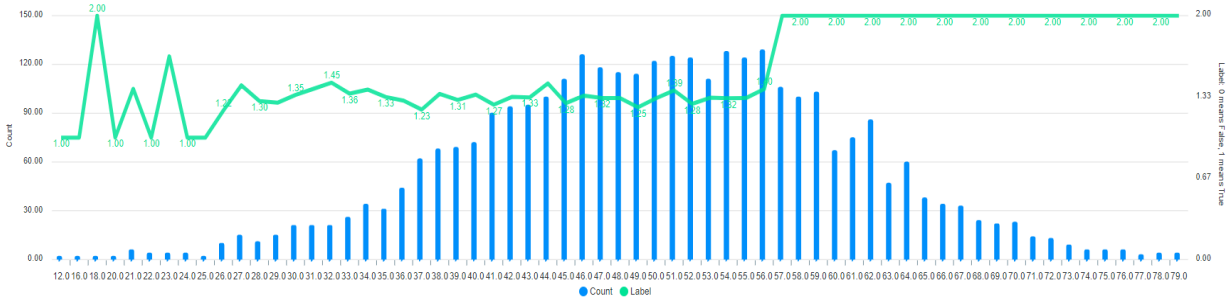


*Figure 21*

The graph is a combination of line and bar plots where the line chart shows the dependence of the label from a feature value while the bar plot shows the number of occurrences of a

feature value in the data frame. If the number of objects in the response is more than 70, we show the list in a table format (Figure 22). The table has three columns: label which represents the predictions, value which represents the values of the selected attribute, and count which shows the frequency of those values in the input data frame. Since for a long list, the width of a bar is so small and makes it difficult to read.

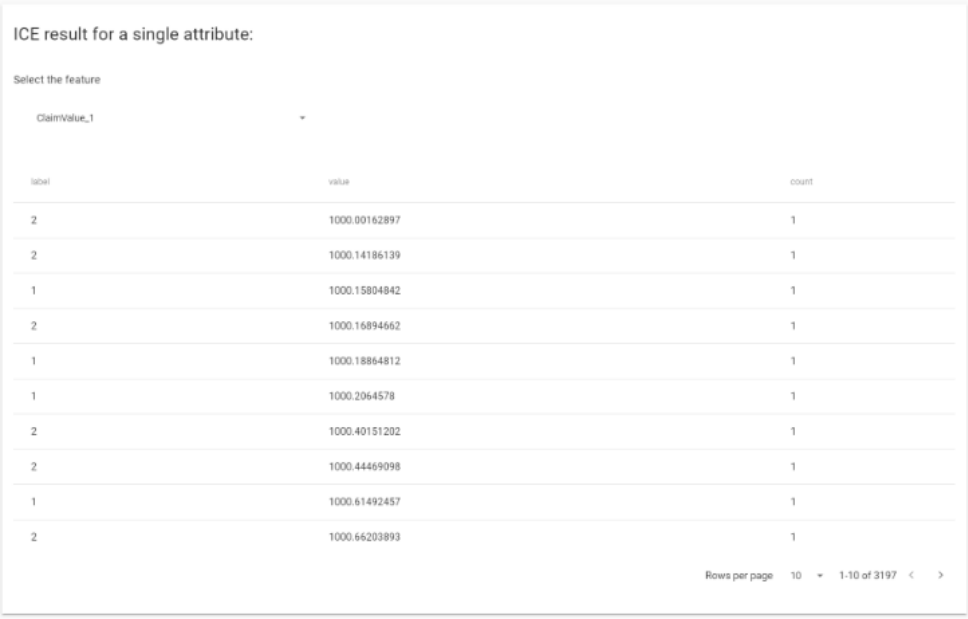


*Figure 22*

### *4.1.8 LIME*

In order to provide another local interpretation method, LIME was implemented. It is a very fast and popular framework for interpreting a black-box model.

As SHAP, in order to run LIME, a model and a trace have to be selected. When a user selects a trace id, a request for a LIME result is sent automatically. The response is provided in JSON format as a list of decoded features and their importance values. An importance value can be positive or negative. The LIME card shows the identifier of the requested trace and model. When a trace is changed, the previous chart is removed from UI, and the chart related to the new trace is shown. The sequence of activities is similar to SHAP (Figure 23). In SHAP, the explainer accepts a decoded data frame and returns a decoded result whereas in LIME the explainer works with an encoded data set, and in the

response, feature names are encoded values. Therefore, after LIME provides a result, feature names are decoded.

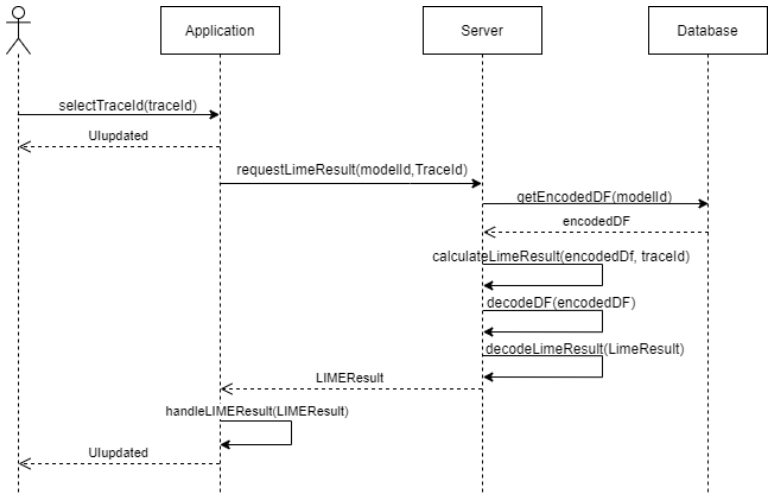


*Figure 23*

The result is shown in a horizontal bar chart (Figure 24). The Y-axis shows a concatenation of feature name and value while in the X-axis feature the importance is shown. Each bar in the chart is colored in a different color, which eases the graph readability. The value range is between [-1; 1]. Positive value leads to get the output label *True* while negative label leads to get the output *False*.

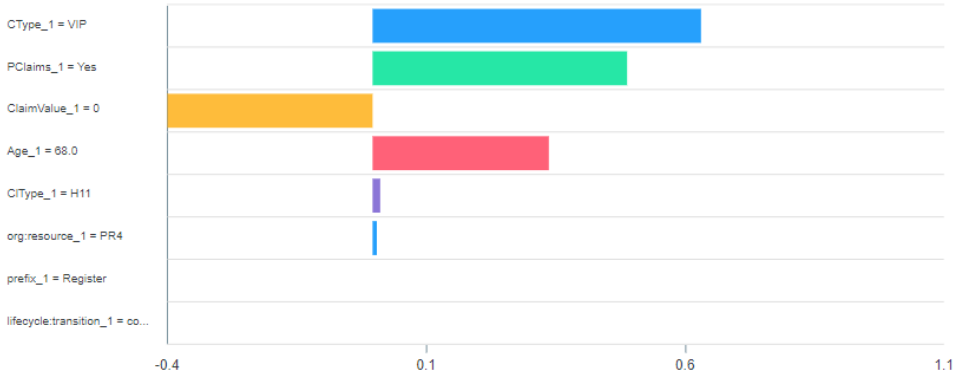


*Figure 24*

### *4.1.9 Temporal Stability*

LIME and SHAP provide feature importance values for a prediction for a particular trace but they do not show if at runtime the prediction changes when new data is available. In

order to see if a prediction is stable (does not change) when the data is updated, we have implemented an algorithm to identify each feature's importance in terms of stability [35].

To run this algorithm, model id and trace should be provided. We have implemented two different API in order to get the prediction for the trace and stability degree of each feature in the data. The result of a single prediction for the whole model is described with a scatter plot (Figure 25). The output is either 2 or 1 which corresponds to *True* and *False*. If the output is 2, it means the predictor is stable, otherwise, it is not stable.

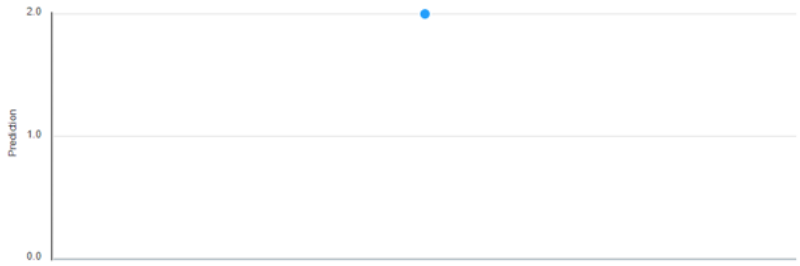


*Figure 25*

Furthermore, stability for each feature value is described in a scatter plot (Figure 26). Y-axis shows the stability degree of values. Each value is colored in a different color, and at the bottom of the graph, feature names and values with corresponding colors are shown. When the mouse goes over a dot on the plot, the feature attribute, values, and importance degree for all points are shown.

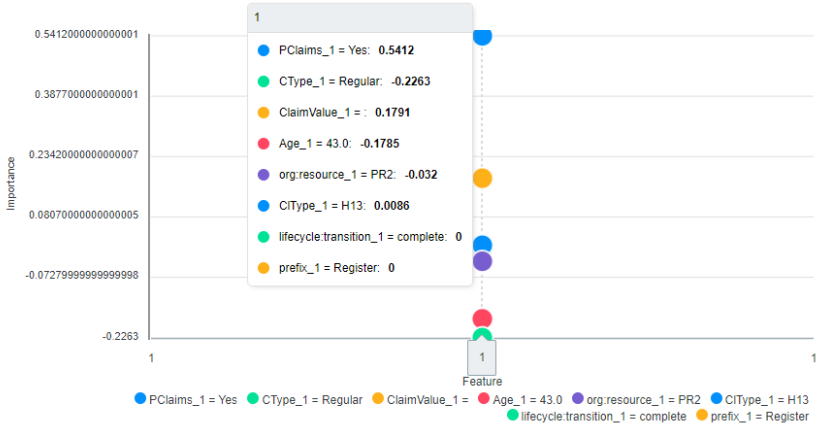


*Figure 26*

The sequence of activities is the same as for LIME in this algorithm except for calculating stability values. When a user selects a trace, two requests are sent for getting the stability level for each feature in the input data. The result is returned in JSON format. The response API to get the stability degree contains arrays of attributes with corresponding values and the degrees while for the single prediction API, the result is a single value.

Besides the plots, a requested model and trace identifiers are shown to the user in the cards. When a new trace is selected, the user interface (UI) is updated.

### *4.1.10 Confusion matrix feedback*

The main goal of this section of the tool is to identify patterns given by combinations of feature names and values that characterize the different outcomes of a confusion matrix [True Positive, True Negative, False Positive, False Negative].

In order to run this algorithm, it is required to identify a binary classification model and a "top k" value. "top k" value is a positive number that is used to retrieve the "top k" relevant patterns characterizing each class of the confusion matrix. The request for feedback results is called when the user enters the "top k" value and press the "Submit" button. The response is returned in JSON format and handled on the front-end side. The response is visualized in a tabular format as shown in Figure 27.

| Class | Pattern | Frequency |
|---|---|---|
| True Positive | Pattern_0:prefix_2_Low Insurance Check | 9 |
| True Positive | Pattern_1:prefix_2_Low Insurance Check; prefix_3_Low Medical History | 7 |
| True Positive | Pattern_2:prefix_2_Low Medical History | 14 |
| True Positive | Pattern_3:prefix_2_Low Medical History; prefix_3_Low Insurance Check | 13 |
| True Positive | Pattern_4:prefix_3_Low Medical History | 20 |
| True Positive | Pattern_5:prefix_3_Low Insurance Check | 29 |
| False Negative | Pattern_6:prefix_2_Create Questionnaire | 11 |
| False Negative | Pattern_7:prefix_2_Create Questionnaire; prefix_3_Contact Hospital | 3 |
| False Negative | Pattern_8:prefix_2_Create Questionnaire; prefix_3_High Insurance Check | 5 |
| False Negative | Pattern_9:prefix_2_Create Questionnaire; prefix_3_High Medical History | 2 |

Rows per page 10 1-10 of 13

*Figure 27*

The table has three columns: Class, Pattern, and Frequency. There are 4 possible values for the class column such as True Positive, True Negative, False Positive, and False Negative. A pattern represents a combination of feature name and value. For example "Pattern_6:prefix_2:Create Questionnaire" is a combination of three strings: Pattern_6 means 6th pattern in the table, prefix_2 means feature name created by simple index encoding, and Create Questionnaire means feature value for that attribute. The second row in Figure 28 represents a complex pattern in which the number of elements is more than 1. It means that in the true positive cases it is possible to find 7 instances in which at prefix_2 "Low Insurance Check" happens and at prefix_3 "Low Medical History" happens. The third column represents the frequency of the pattern in the input data set.

| Class | Pattern | Frequency |
|---|---|---|
| True Positive | Pattern_0:prefix_2_Low Insurance Check | 9 |
| True Positive | Pattern_1:prefix_2_Low Insurance Check; prefix_3_Low Medical History | 7 |

*Figure 28*

In order to run this algorithm, LIME is used to give explanations about all traces belonging to each class in the confusion matrix. For each class in the confusion matrix, we get different sets of explanations. By applying an algorithm for the discovery of frequent item sets, for each class in the confusion matrix, we identify the sets of pairs feature-value that occur more frequently for that class.

Based on the feedback that the algorithm provides, we implemented a new algorithm to randomize the values of a feature or a pattern. The goal is to randomize patterns that affect wrong predictions, and as a result, the accuracy of the classifier could improve. In order to do it, a user can select multiple features or patterns based on confusion matrix feedback results. Based on the selected elements, a request is sent to the server by identifying the model id and selected feature names and feature values. A new model is created after randomizing the feature values for selected patterns. The format of the response is JSON, in a successful case, a table with values of estimated time, precision, recall, accuracy, and f1_score for initial and after retraining cases are shown (Figure 29).

| Matrix | Initial Result | Retrain Result |
|---|---|---|
| elapsed_time | 0.188659 | 0.096728 |
| f1_score | 0.7786 | 0.7883 |
| accuracy | 0.78 | 0.79 |
| precision | 0.778 | 0.7874 |
| recall | 0.7817 | 0.7906 |

*Figure 29*

The sequence of activities for both algorithms is described in Figure 30. The main contribution to these two algorithms was the implementation of the APIs and the development of the UI side.

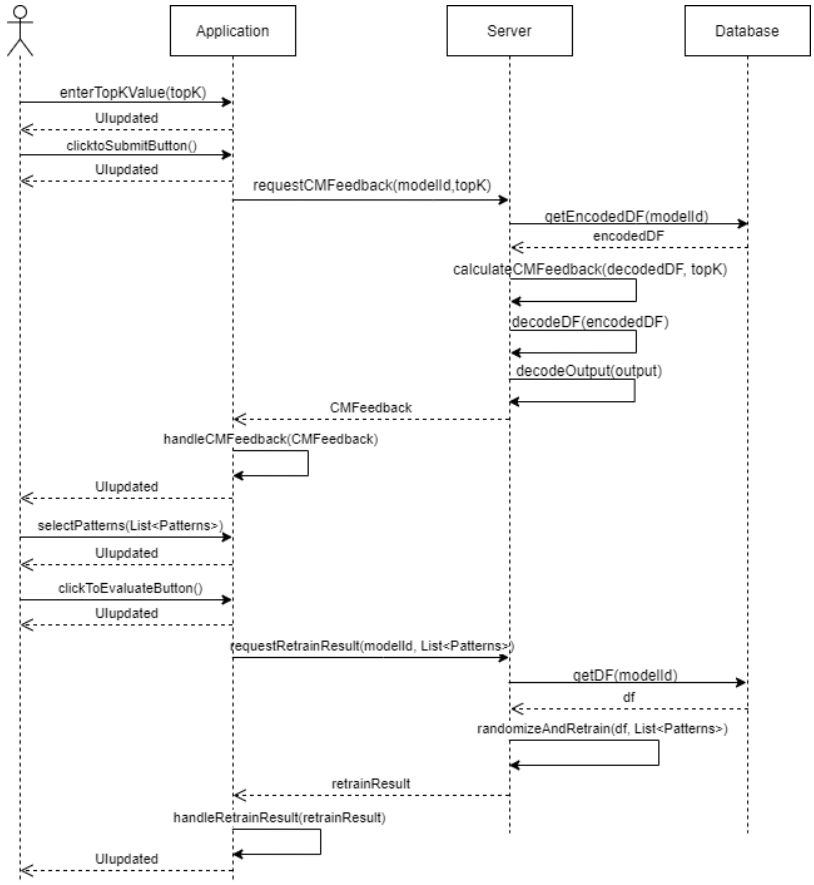


*Figure 30*

# 5 Evaluation

In this section, we evaluate the system by using a dataset (Drift1) distributed in two event logs. The main goal is to check how the different implemented frameworks work for different prediction models and encoding types. We have evaluated six frameworks and techniques which are SHAP (SHapley Additive exPlanation), LIME (Local Interpretable Model-Agnostic Explanation), Skater, PDPbox (Python Partial Dependence Plot Toolbox), Temporal stability, and Confusion matrix feedback with retrain.

## 5.1 Dataset

The internal structure of the log files is described in Figure 31. The first event log contains 35358 cases in 3200 events, and the number of activities is 17. On average, each trace contains 11 events. In the second event log, the number of cases is 35377 in 3200 cases with 17 different activity names.

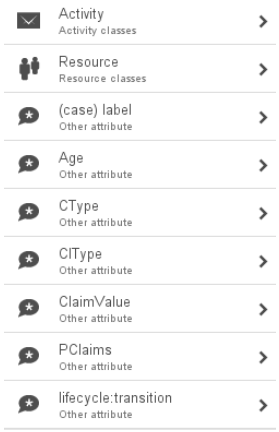


*Figure 31*

The possible values for the label are either *True* or *False*, and we tried to balance the labels by changing their values based on event attributes. We have labeled the first train and test data log files in three different ways which are the followings:

1. If the value of the "Age" attribute is less than or equal to 50, the traces are labeled as *False*, while the remaining ones as *True*. In the train data, with the given formula, the percentage of traces labeled with *False* is 49%, while the percentage of cases labeled with *True*, is 51%.

2. If the value of "Age" is less than 60 and the value of "ClType" is either "H12" or "H14" or "H15", the traces are labeled as *True,* while the remaining ones are labeled as *False*. In the train data, with the given formula, the percentage of traces labeled with *True* is 51%, while the percentage of traces labeled with *False,* is 49%.
3. If the value of "Age" is less than or equal to 56, the value of "PClaims" is equal "No", and the value of "CType" is either "Regular" or "Silver" or "Gold", the traces are labeled as *False,* while the remaining ones are labeled as *True*. In the train data, with the given formula, the percentage of traces labeled with False is 48%, while the percentage of traces labeled as *True*, is 52%.

For the second train and test data, we have used different patterns to create labeling. These patterns are described below:

4. If the value of the "PClaims" attribute is equal to "No", the traces are labeled as *False,* while the remaining ones are labeled as *True*. In the train data, with the given formula, the percentage of traces labeled with *False* is 50%, while the percentage of traces labeled as *True*, is 50%.
5. If the value of "CType" attribute is equal either to "Regular" or "VIP", and the value of "ClType" is either "H11" or "H12" or "H13" the traces are labeled as *False,* whilethe remaining ones are labeled as *True*. In the train data, with the given formula, the percentage of traces labeled with *False* is 51%, while the percentage of traces labeled with *True,* is 49%.
6. If the value of "Age" is greater than or equal to 45, and the value of "ClType" is either "H11" or "H12" or "H13", and the value of "CType" is either "Regular" or "Silver", the traces are labeled as *True,* while the remaining ones are labeled as *False*. In the train data, with the given formula, the percentage of traces labeled with *True* is 48%, while the percentage of traces labeled with *False,* is 52%.
7. If the value of activity name equals to "Accept Claim", (acceptance in most of the cases occurs when the claim is of low value, i.e., in the cases in which low insurance check or low medical history occur), the traces are labeled as *True*, while the remaining ones are labeled as *False*. In the train data, with the given formula, the percentage of

traces labeled with *True* is 55%, while the percentage of traces labeled with *False*, is 45%.

## 5.2 Evaluation setting

By using the two event logs labeled with the labeling options described above, we have trained 7 different models with various configurations. Table 4 contains the configuration parameters of the models:

| ID | Input log | Predictive Model | Method | Encoding | Prefix Length | Log padding | Labelling |
|---|---|---|---|---|---|---|---|
| 1 | Labeled input log with option 1 | Classification | Decision Tree | Simple index | 4 | No padding | Trace string attribute-label |
| 2 | Labeled input log with option 3 | Classification | Decision tree | Complex index | 4 | No padding | Trace string attribute-label |
| 3 | Labeled input log with option 3 | Classification | Decision tree | Simple index | 6 | No padding | Trace string attribute-label |
| 4 | Labeled input log with option 5 | Classification | Decision tree | Complex index | 4 | No padding | Trace string attribute-label |
| 5 | Labeled input log with option 2 | Classification | KNN | Complex index | 6 | No padding | Trace string attribute-label |
| 6 | Labeled input log with option 6 | Classification | Decision tree | Complex index | 5 | No padding | Trace string attribute-label |
| 7 | Labeled input log with option 3 | Classification | Decision tree | Complex index | 1 | No padding | Trace string attribute-label |
| 8 | Labeled input log with option 7 | Classification | Decision tree | Simple index | 5 | No padding | Trace string attribute-label |

*Table 4*

## 5.3 Results

In this section we describe the results obtained with each of the different implemented frameworks.

### *5.3.1 SHAP*

SHAP is one of the libraries that have been used to explain the output of any machine learning model. Although SHAP supports different kinds of visualizations for both local and global interpretations, we mainly focus on a local interpretation, which means that we explain the given prediction for a given trace.

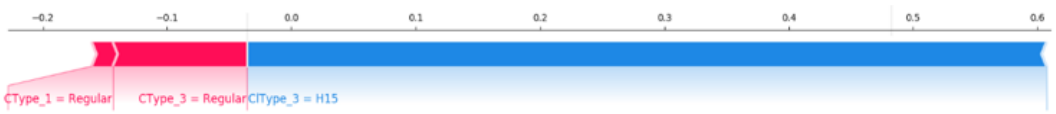


*Figure 32*

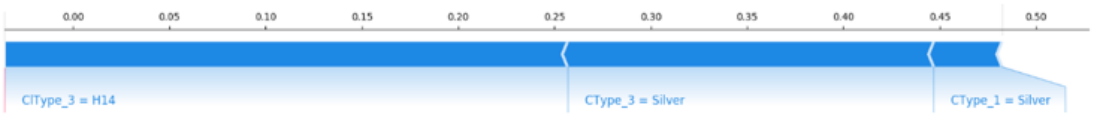


*Figure 33*

In Figure 32 and Figure 33, we show the results for two example traces taken for configuration number 4 . As it can be seen from the patterns that have been used for labeling the input log file of the model (option 5), there are only two attributes “CType" and “ClType” that determine whether the label is *True* or *False*. Indeed, only those variables are shown in the resulting SHAP graph. Although there are 9 event attributes, only the ones which affect the decision are shown in the resulting plot. The values in the plots show how each feature affects the output. In Figure 32, the graph shows that when “CType” has the value “Regular” the label is *False* (the bar is red), while when “ClType” has the value “H15” the label is *True* (the bar is blue), and the importance is -0.2 for “CType” and +0.6 for “ClType”, so the relevance of “ClType” in determining the label *True* is higher than the relevance of “CType” in determining the label *False*. In Figure 33, the importance value for all the variables are positive.

By looking at all the correct predictions returned for the traces in the test set, we can find that the explanation provided by SHAP on each of the correctly predicted traces fits the criteria used for labeling the event log. We have evaluated this technique by model number 3 and 5 too. Also, in these cases the results match with the expected output.

### *5.3.2 Skater*

Skater is one of the libraries we have used to implement in Nirdizati Research a functionality able to provide a global interpretation of a predictive model. It is a unified system to get an interpretation for all model types. In Nirdizati Research, we have used it to explain the model in a decision tree format. In order to make it more understandable, the result tree was visualized in a different, more interpretable way

For classification, the evaluation has been done with 4 different models: two of them based on simple index encoding (model 1 and 3) and two of them based on complex encoding (model 2 and 4). The resulting tree for model 4 is shown in Figure 34.

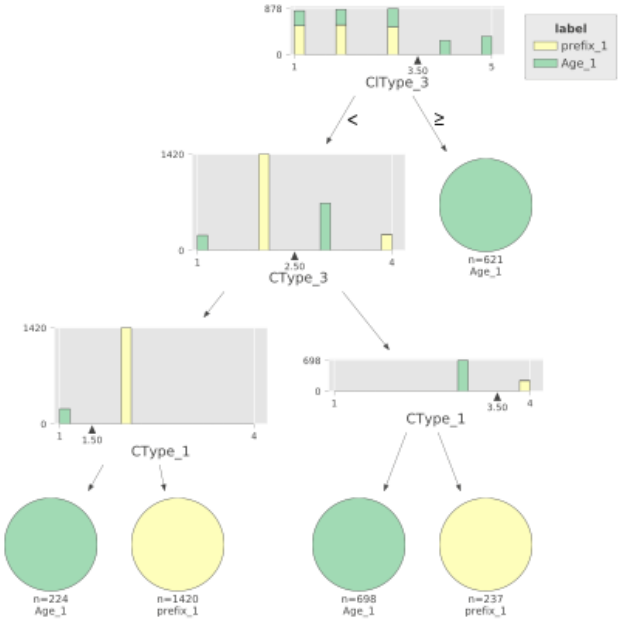


*Figure 34*

As shown in the plot (Figure 34), the decision tree is generated by using two features: “ClType” and “CType” which match with the labeling option in model 4. Green and yellow colors are the representation of *True* and *False* labels correspondingly [36]. In the log input file, there are 5 different values for “ClType” those are: [H11, H12, H13, H14,

H15], and for “CType”, there are 4 values which are: [VIP, Silver, Gold, Regular]. The result contains encoded values that are generated by concatenating the feature name and prefix number. Initially, the decision tree is separated into two branches based on “ClType” value that is the events with values [H11, H12, H13] are categorized in the left branch while [H14, H15] are grouped in the right branch. . The value of the features in the branches are defined by a threshold that, for the first node in the plot in Figure 34 is set to 3.50. The threshold is related to the index of the values in the list of feature values. For instance, in the case of the first node, the values of the “ClType” feature with index one, two and three (i.e., H11, H12, H13) in the list of feature values [H11, H12, H13, H14, H15] are below the threshold, whereas the values with index four and five (i.e., . H14 and H15) are higher. The traces with feature “ClType” equal to H14 and H15 are labelled as *True* in the right branch, and the number of occurrences of these traces is 621. On the left branch, the traces labeled as *True* and *False* are further grouped by leveraging the values of the attributed “CType”. As a result, the first and third values (i.e., VIP and Gold) in the list of CType values are categorized with the green color while Silver and Regular were defined with a yellow color. The distribution of branches and number of frequencies shown in the plot (Figure 34) match with the expected result for the model 4.

### *5.3.3 PDPbox*

The implementation of the PDPbox framework is different from the one of the other frameworks. The result, indeed, is calculated based on a model and a feature name. PDPbox is one of the most used techniques for explaining an output of a model because it clearly explains how possible values of a feature affect the result. It supports different kinds of visualization techniques.

Firstly, we have tested the framework with model 2 (option 3) with the future “Age_1”. The output plot is a combination of a bar plot and a line chart. The bar plot represents the number of occurrences of the value in the model while the line chart is the importance of each feature value in affecting a certain label. The result is shown in Figure 35.

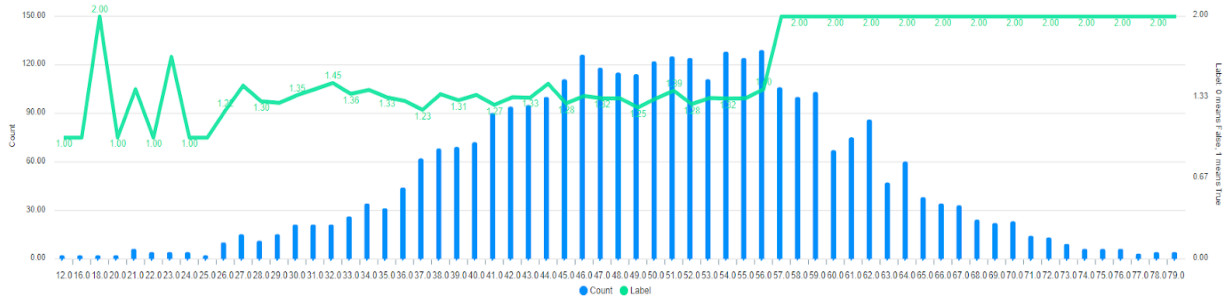


*Figure 35*

A selected attribute represents a combination of feature name and event index in the trace. In our case, Age is a feature name, and 1 indicates the first event in the traces. After value 56, the output of the label is 2 which is an encoded value of *True*. On the other hand, the label value for most of the values below or equal to 56 is less than 1.5 which means they are *False*. This result matches the pattern for the relabeling of the input log file.

### *5.3.4 LIME*

LIME is a post hoc technique that can be used for global and local interpretation. This evaluation has been done for models that have been trained with classification prediction methods.

Figure 36 shows the results for a trace evaluated using model 7 (option 3). It can be seen that "CType" with value VIP, "PClaims" with value Yes, and "Age" with value 68 have more importance in terms of getting a *True* prediction, while "ClaimValue" with value 0 is the only pair of attribute and value affecting the prediction *False*. The three pairs of attribute and value characterizing the *True* label, are indeed the criteria that have been used for labeling traces with option 3.

We have tested the framework with multiple traces, and by checking all the correct prediction, which was returned by LIME, we can say that for the technique predicts the output for the tested traces correctly.

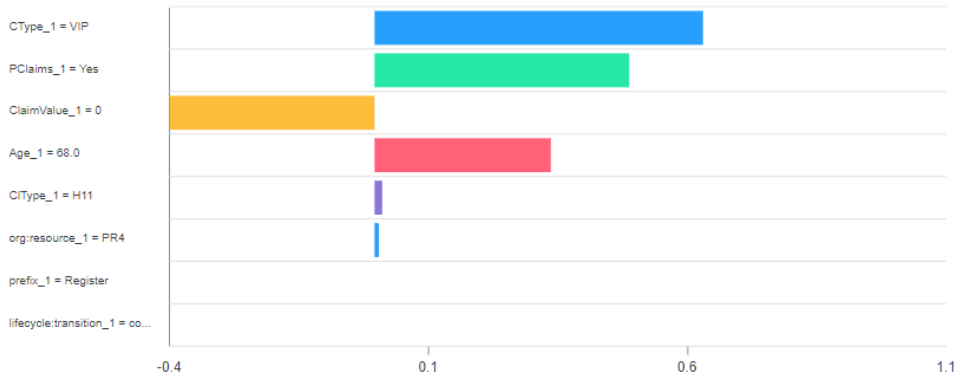


*Figure 36*

### *5.3.5 Temporal Stability*

The concept of temporal stability comes from the stability of a classifier. It shows how a prediction changes when new data is available about the same case. In order to test the algorithm that has been implemented for temporal stability in the Nirdizati Research project, we have evaluated the result with 4 different models.

Initially, we have evaluated the algorithm with model 7. The result in Figure 37 shows how the feature "PClaims" affects the stability of the predictor's output more than other features. This feature is the most important one affecting the output, hence, in case it changes its value, the result may be unstable.

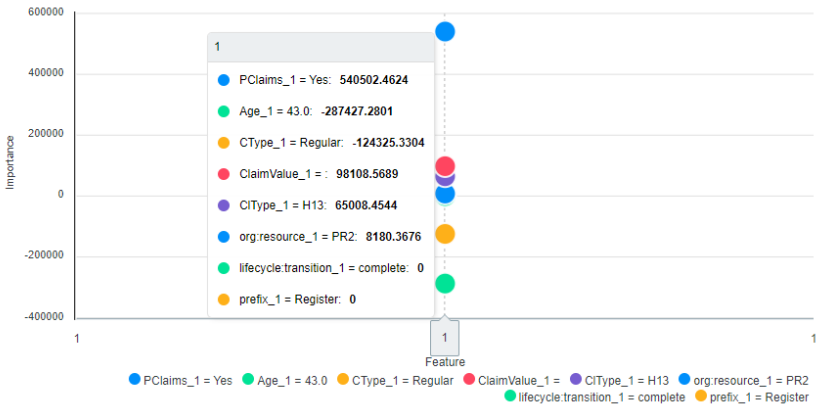


*Figure 37*

The final prediction for the trace is shown in Figure 38. The output is 2 which means *True* which means the predictor is stable.

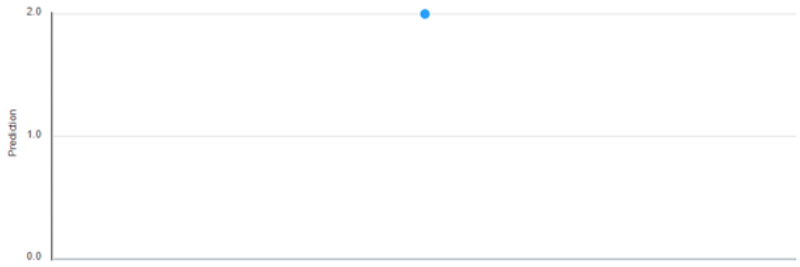


*Figure 38*

We have tested this technique with multiple different traces. All tests that we did are in line with the criteria that we used for the event log labeling for this technique.

### *5.3.6 Confusion matrix feedback*

A confusion matrix is used in the application to show which patterns or features affect the model output positively and negatively. After that, the classifier can be trained again by randomizing those features in order to increase the accuracy of the predictor. This algorithm only works for classification.

We have evaluated this technique with model 8. As a "top k" value, we have entered 6. The result of the evaluation is described in Figure 39.

| Class | Pattern | Frequency |
|---|---|---|
| True Positive | Pattern_0:prefix_2_Low Insurance Check | 9 |
| True Positive | Pattern_1:prefix_2_Low Insurance Check; prefix_3_Low Medical History | 7 |
| True Positive | Pattern_2:prefix_2_Low Medical History | 14 |
| True Positive | Pattern_3:prefix_2_Low Medical History; prefix_3_Low Insurance Check | 13 |
| True Positive | Pattern_4:prefix_3_Low Medical History | 20 |
| True Positive | Pattern_5:prefix_3_Low Insurance Check | 29 |
| False Negative | Pattern_6:prefix_2_Create Questionnaire | 11 |
| False Negative | Pattern_7:prefix_2_Create Questionnaire; prefix_3_Contact Hospital | 3 |
| False Negative | Pattern_8:prefix_2_Create Questionnaire; prefix_3_High Insurance Check | 5 |
| False Negative | Pattern_9:prefix_2_Create Questionnaire; prefix_3_High Medical History | 2 |

Rows per page 10 1-10 of 13

*Figure 39*

The explanation of each column is described in section 4.1.10. The result contains the top 6 patterns for each class. Top values are selected based on frequency. In order to increase the accuracy of the model, we have selected a single pattern which includes to "False

Negative" class which is "Pattern_0:prefix_2:Create Questionnaire". When a pattern is selected from the "False Negative" matrix class to randomize its values, the model is trained again with the new values and evaluated. In some cases, the randomization of the noisy features allows for increasing the accuracy of the predictions. Figure 40 shows how the f1_score, accuracy, prediction, and recall are changed after randomizing values for "Pattern_6:prefix_2_Create Questionnaire".

| Matrix | Initial Result | Retrain Result |
|---|---|---|
| elapsed_time | 0.188659 | 0.096728 |
| f1_score | 0.7786 | 0.7883 |
| accuracy | 0.78 | 0.79 |
| precision | 0.778 | 0.7874 |
| recall | 0.7817 | 0.7906 |

*Figure 40*

Moreover, we have selected "Pattern_0:prefix_2_Low Insurance Check" which is included in the "True Positive" class, the result for f1_score, accuracy, prediction, and recall did not change.

The result for this technique is in line with the labelling.

# 6 Conclusion

A predictive business process monitoring system shows predictions about future values of performance indicators of ongoing process executions. Based on historical traces, the system estimates the future performance. Nonetheless, it is not enough just to show predictions for the users in order to take action to improve the future performance of an ongoing case. Explanations are necessary to trust the predictions and decide what to do next. Therefore, the users are mainly searching for explainable predictive process monitoring systems that explain predictions in a human-understandable way.

Nirdizati Research is a system to create predictive models based on different configurations and validate them for predictive process monitoring. The contribution of this thesis was to integrate new algorithms and frameworks to show local and global explanations with ante-hoc and post-hoc approaches with different kinds of visualizations and for different types of predictive models.

As a future improvement, different techniques can be checked and added to the application. Moreover, in the current version of the project, there are only 5 versions of encoding methodologies, and it does not support one-hot encoding. However, several frameworks and algorithms work only with that encoding type, and it was not possible to implement them for the present release. Thus, it is required to add one-hot encoding too. A final improvement for the Explanation page can be implementing all the currently integrated frameworks for all prediction method types (regression, classification, and time series prediction) because some of the techniques do not work with time series prediction models.

# 7 Bibliography

[1] I. Verenich, "A general framework for predictive business process monitoring," Proceedings of CAiSE 2016 Doctoral Consortium, 2016.

[2] G. Vandewiele, O. Janssens, F. Ongenae, F. De Turck, and S. Van Hoecke, "GENESIM: Genetic Extraction of a Single, Interpretable Model," NIPS 2016 Workshop on Interpretable Machine Learning in Complex Systems, 2016.

[3] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding Neural Networks Through Deep Visualization," 2015. [Online]. Available: http://arxiv.org/abs/1506.06579.

[4] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune, "Synthesizing the Preferred Inputs for Neurons in Neural Networks via Deep Generator Networks," vol. 29, Advances in Neural Information Processing Systems (NIPS), 2016.

[5] H. Deng, "Interpreting Tree Ensembles with inTrees," 2014. [Online]. Available: http://arxiv.org/abs/1408.5456.

[6] Y. Lou, R. Caruana, J. Gehrke, and G. Hooker, "Accurate Intelligible Models with Pairwise Interactions," 19th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD'13), Proceedings, 2013, p. 623–631.

[7] A. Henelius, K. Puolamäki, H. Boström, L. Asker, and P. Papapetrou, "A Peek into the Black Box: Exploring Classifiers by Randomization," vol. 28, Data Mining and Knowledge Discovery, 2014, p. 1503–1529.

[8] J. Adebayo and L. Kagal, "Iterative Orthogonal Feature Projection for Diagnosing Bias in Black-Box," 2015. [Online]. Available: http://arxiv.org/abs/1611.04967.

[9] A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin, "Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation," vol. 24, Journal of Computational and Graphical Statistics, 2015, p. 44–65.

[10] J. Krause, A. Perer, and K. Ng, "Interacting with Predictions: Visual Inspection of Black-box Machine Learning Models," ACM Conference on Human Factors in Computing Systems, 2016, p. 5686–5697.

[11] P. Adler, C. Falk, S. A. Friedler, T. Nix, G. Rybeck, C. Scheidegger, B. Smith, and S. Venkatasubramanian, "Auditing Black-Box Models for Indirect Influence," vol. 54, Knowledge and Information Systems, 2018, p. 95–122.

[12] J. H. Friedman and B. E. Popescu, "Predictive Learning via Rule Ensembles," vol. 2, Annals of Applied Statistics, 2008, p. 916–954.

[13] B. Letham, C. Rudin, T. H. McCormick, and D. Madigan, "Interpretable Classifiers Using Rules and Bayesian Analysis: Building a Better Stroke Prediction Model," vol. 9, Annals of Applied Statistics, 2015, p. 1350–1371.

[14] F. Wang and C. Rudin, "Falling Rule Lists," Artificial Intelligence and Statistics (AISTATS), 2015.

[15] T. Wang, C. Rudin, F. Velez-Doshi, Y. Liu, E. Klampfl, and P. Macneille, "Bayesian Rule Sets for Interpretable Classification," IEEE International Conference on Data Mining (ICDM), Proceedings. IEEE, 2017, p. 1269–1274.

[16] S. Lundberg and S.-I. Lee, "An Unexpected Unity Among Methods for Interpreting Model Predictions," 29th Conference on Neural Information Processing Systems (NIPS 2016), 2016.

[17] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You?": Explaining the Predictions of Any Classifier," 22nd ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD'16), Proceedings, 2016.

[18] E. R. Elenberg, A. G. Dimakis, M. Feldman, and A. Karbasi, "Streaming Weak Submodularity: Interpreting Neural Networks on the Fly," 2017. [Online]. Available: http://arxiv.org/abs/1703.02647.

[19] M. T. Ribeiro, S. Singh, C. Guestrin, "Anchors: High-Precision Model-Agnostic Explanations," AAAI Conference on Artificial Intelligence, 2018.

[20] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic Attribution for Deep Networks," 34th International Conference on Machine Learning (ICML), Proceedings, 2017.

[21] J. Zhang, S. A. Bargal, Z. Lin, J. Brandt, X. Shen, and S. Sclaroff, "Top-Down Neural Attention by Excitation Backprop," International Journal of Computer Vision, 2017, p. 1–19.

[22] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," 2015. [Online]. Available: http://arxiv.org/abs/1502.03044.

[23] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush, "LSTMVis: A Tool for Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks," IEEE Transactions on Visualization and Computer Graphics, 2018, p. 667–676.

[24] A. Mahendran and A. Vedaldi, "Understanding Deep Image Representations by Inverting Them," IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Proceedings, 2015, p. 5188–5196.

[25] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning Deep Features for Discriminative," Computer Vision and Pattern Recognition (CVPR), 2016, p. 2921–2929.

[26] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization," NIPS 2016 Workshop on Interpretable Machine Learning in Complex Systems, 2016.

[27] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning Important Features Through Propagating Activation Differences," arXiv preprint, 2017.

[28] T. Lei, R. Barzilay, and T. Jaakkola, "Rationalizing Neural Predictions," 2016. [Online]. Available: http://arxiv.org/abs/1606.04155.

[29] "Skater," 2017. [Online]. Available: https://github.com/oracle/Skater.

[30] K. Jorbina,A. Rozumnyi,I. Verenich, C. Di Francescomarino, M. Dumas, " Nirdizati: A web-based tool for predictive process monitoring," BPM Demo Track and BPM Dissertation Award, 2017.

[31] I. Verenich, "Explainable predictive monitoring of temporal," 2018.

[32] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi., "A Survey of Methods for Explaining Black Box Models," ACM computing surveys, 2019.

[33] M. Jurriaan Robeer, "Contrastive Explanation for Machine Learning," 2018.

[34] V. Buhrmester, D. Münch, and M. Arens, "Analysis of Explainers of Black Box Deep Neural Networks for Computer Vision: A Survey," 2019. [Online]. Available: https://arxiv.org/pdf/1911.12116.

[35] I. Teinemaa , M. Dumas, A. Leontjeva, F. M. Maggi, "Temporal Stability in Predictive Process Monitoring," 2018, pp. 1306-1338 .

[36] T. Parr, P. Grover, and T. Lapusan, "Dtreeviz," 2018. [Online]. Available: https://github.com/parrt/dtreeviz.

# 8 License

**Non-exclusive license to reproduce thesis and make thesis public**

I, **Musabir Musabayli**

1. herewith grant the University of Tartu a free permit (non-exclusive license) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

**Explainable Predictive Process Monitoring**

supervised by Fabrizio Maria Maggi, Williams Rizzi, Chiara Di Francescomarino

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons license CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive license does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Musabir Musabayli

15/05/2020