Rework Prediction Using Event Logs

Bareman, R.

Award date:
2020

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Architecture of Information Systems Research Group

Rework Prediction Using Event Logs

Master Thesis

R. Bareman

Supervisors:
prof.dr.ir. B.F. van Dongen
dr.ir. R.M. Dijkman
ir. M. Althuizen

Final version

Eindhoven, February 2020

Abstract

Technical developments result in increased mechanization of manufacturing processes. These developments improve the quality of the used machines, meaning that they are controlled independently and they all generate data. This data can be used to provide insights or to control the process automatically. The Smart Industry 4.0 Assistant (SIA)[1], which is being developed by Bright Cape, is a tool that aids in the usage of such data. SIA provides tools to improve both product or process quality. One of its tools predicts product quality. Quality predictions are currently made done without the use of event sequences. However, since event sequences result from the process, it is expected that this data would aid in the prediction of quality.

In this thesis, we investigate five different methods to translate process sequences into attributes for classification. One of those methods also includes additional information that comes available with every event. One problem that arises is that rework is also an activity that is part of the event log. The evaluation method is adapted to take this problem into account. For evaluation, we defined a naive classifier, which is used as a baseline to assess the new prediction models.

The five different event log translation methods are tested using both an artificial event log and a real event log retrieved from an automated production environment. After the translation of the event log, random forests are trained on the resulting attributes. For the simulated data, this resulted in the expected results: the method which also includes event attributes performed best, as it translates most information. The case study, however, did not show any promising results: all resulting random forests achieved the same accuracy or a lower accuracy than the defined baseline. Since the model proved to be working with the simulated data and the results of the case study did not show any useful predictions, it is concluded that the data from the case study did not correlate with rework.

Preface

The finalization of this thesis marks the end of an exciting period. In 2013 I started with a bachelor industrial engineering at TU Eindhoven. Now, after more than six exciting years, I'm going to finish my master Business Information Systems.

Finishing my studies would not have been possible without the help of others. For this reason, I would like to thank Boudewijn van Dongen for his supervision and ideas during this thesis. I also want to thank Bright Cape for providing the opportunity to collaborate on this thesis and the resulting enriching experience. In particular, I thank Jochem Hilbrink, Mart Althuizen and Thijs van de Weijer for their support.

Aside from this thesis, numerous people supported me during the entirety of my studies. I want to thank my parents, my family, my friends and my girlfriend for their support during both my thesis and during my studies.

Rowan Bareman
Breda,
February, 2020

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data availability grows at a high rate, which increases the availability of data in all kinds of environments. The manufacturing industry is no exception. Modern machines keep track of every processed product and record information about that product, such as the time when it is processed. The goals of using such information are to either improve the process or the manufactured product. The research reported in this thesis is conducted at Bright Cape, a company that identified an opportunity with the use of such data.

1.1 Context

Bright Cape is a company located in the Netherlands that helps customers retrieve knowledge from data to optimize their processes, increase revenue, or to decrease costs. It does so by providing services for Data-Driven User Experience (DDEX), Process Mining, and Applied Data Science. Aside from providing insights into the use of data, the company also works the development of innovative products. one of these products machines is called the Smart Industry 4.0 Assistant[1], or SIA for short.

Industry 4.0 is a name for the anticipated fourth industrial revolution[16]. The first being mechanization, the second industrial revolution refers to the intensive use of electrical energy, and the third revolution, which resulted from digitization. The current ongoing revolution aims to create systems that automatically control the manufacturing process while being flexible and able to adapt to each type of product. The need for a fourth revolution arises due to the reduced development periods of new products and the requirement of products to be adapted to the customers' needs. Technological innovations fitting these requirements are possible due to increased mechanization and automation of manufacturing processes. The quality of these machines also increases, meaning that they can be controlled independently in each step of the process. These machines also record several actor- and sensor data, which provides input to guide the process automatically.

The Smart Industry Assistant looks to aid organizations in the realization of Industry 4.0. It collects all generated data from the factory for analysis and provides multiple tools for the user in order to improve both product or process quality. It does so by automatic reporting and visualization of data, predicting the product quality, detect anomalies, provide smart maintenance, or by automatically adapting the process to optimize the resulting products. The main goals of these tools are to increase the revenue, increase the efficiency of resources and to reduce the costs.

1.2 Problem statement

One of the other modules that SIA aims to offer includes process mining. There are concerns regarding the general usability of process mining within largely automated production lines, where there is little to no variation in the production of each product. The limited variety in the processes results in doubts in the usefulness of process mining. These doubts are likely to come from the misconception that process mining is limited only to control-flow discovery[30]. The time perspective and resource perspective provides useful insights into bottlenecks and throughput times.

First, the scope of the study needs to be clarified. This study focuses on an automated sequential production process, meaning that the value-adding production steps are performed in sequence by machines. The only deviations in the production process should be the result of human intervention. These interventions are the result of products not reaching a quality threshold, after which they are either reworked or scrapped.

The goal of this thesis is to investigate the usefulness of process mining in the targeted environment. We attempt to find links between data collected throughout the process and product quality performance. In order to test the value of process data, we train models to predict the presence of a rework activity for one product. Decision making can be improved if quality information becomes available earlier, providing the opportunity to reduce costs and required resources. However, event data needs to be translated into features before it is ready to be used for prediction.

RQ1. How can event logs be translated into features for prediction?

After the data is made prediction-ready, there is a decision to be made on which prediction algorithm to use. Thus, it requires an evaluation of prediction models to decide which fits best for this research. Even though the method for prediction is not the main focus of this thesis, it is crucial to evaluate the different options since it is used to test the different methods to translate event logs into features.

RQ2. What prediction method is most useful for predictions with event logs?

After acquiring a model to assess the usability of data, we need to assess which data is most valuable. We identify three main types of data as output from an automated manufacturing process. First, we identify data that is present before the process starts, which holds information about what is manufactured and when. The second type is the data about the process. The usability of this data is assessed next.

RQ3. Do predictions become more accurate when including event log data in prediction models?

For this research question, we investigate models trained on either type of data. First, we investigate the difference in accuracy between models trained on data of the product specifications and adding the sequence data. Next, we combine both process and product information to train models.

1.3 Outline

The next chapter covers the preliminaries of this thesis. In the third chapter, we discuss previous related research on the matter. The fourth chapter discusses the different methods used to translate an event log into features for classification. Subsequently, we discuss the second research question, where we evaluate different prediction models. Next, the implemented method to predict rework is discussed. In the subsequent chapter, this model is validated using an artificial event log. In the eighth chapter, the same method is used on a real dataset that originates from an actual automated

production environment. Lastly, the thesis concludes with our main findings and conclusions as well as providing suggestions for future research.

Chapter 2

Preliminaries

In this chapter, we explain the basic concepts and notations relating to process mining and prediction. These notations are used in the remainder of this thesis.

2.1 Process Mining

Process mining techniques seek to extract knowledge from event logs [30], meaning that event logs are critical in process mining. Process mining is not a single technique but combines both data mining and the modelling and analysis of business processes. The different techniques are used for the discovery, monitoring, and improvement of real processes from event logs.

Event logs are datasets containing the recorded data of *events*. These events refer to executions of activities for a single *case* and the recorded data for this activity. A *case* refers to a single process instance. For example, one case in a hospital can be one patient, where a case in a production process can refer to one product. By listing the events of each case in sequential order, it becomes possible to extract the sequence of activities of a single case. The sequence of activities for one case is also called a *trace*.

Definition 2.1.1. (Case)
Let C be the set of all cases. Where $c \in C$ refers to a single process instance. For manufacturing, a single case refers to one product.

Definition 2.1.2. (Event log, Event)
Let $\mathcal{E}$ be the set of all events and $e \in \mathcal{E}$ refers to a single event. then let $\mathcal{E} \times \mathcal{E}$ be the total ordening on $\mathcal{E}$ and $\mathcal{E}^* \subseteq \mathcal{E} \times \mathcal{E}$ is the set of ordered events, i.e. $e_1 \geq e_2$. An event log L contains event data of all past events of the process of all cases.

Definition 2.1.3. (Trace)
Each event is related to one case, and one case only. Where a trace σ_c is the ordered set of events for case c, i.e. $\sigma_c = \langle e_1, \ldots, e_n \rangle \in \mathcal{E}^*$. γ is the function that maps each event to one case, i.e. $\gamma : \mathcal{E} \to c$. Then $\forall_{c \in C} \forall_{e \in \sigma_c} \gamma(e) = c \wedge \forall_{e \in \mathcal{E}} \forall_{c \in C} \gamma(c) = \sigma_c$, which states that each event in a trace maps onto the same case, and each case maps onto one trace.

Definition 2.1.4. (Prefix)
A prefix of a trace, is the subset of chronological consecutive events of a trace, starting with the first event. Let $\sigma_c = \langle e_1, \ldots, e_n \rangle$ be a trace of size n. then the prefix of σ_c with length m is $\sigma_c^m = \langle e_1, \ldots, e_i \rangle$ *with* $\sigma_c = \langle e_1, \ldots, e_n \rangle$ *and where* $1 \leq m \leq n$

Events contain a minimum of two types of attributes, namely the activity name and case ID, which are the minimum requirements for process mining[29]. Events often contain other attributes besides the activity name and case ID, such as timestamps or the used resource. *Timestamps*

indicate the start- or end time and date of an activity. An event can have two timestamps, one for the start of the activity and one when it finishes. The *resource* provides information on the one performing the activity. For example, the identification number of an employee or machine. Aside from these components, additional information can be present contained in the event log. For example, the event log can contain information about the quantity of a batch, the weight of a product or quality information.

Definition 2.1.5. (Event Attribute)
Let A be a set of event attribute names, then $\forall_{a \in A}\ \#_a : \mathcal{E} \rightarrow V_a$. Where $\#_a$ is the function that maps each event to attribute name a with V_a possible values. For example, $\#_{activity}(e_1)$ provides the activity of the first event and $V_{activity}$ is the set of activities, then $|V_{activity}|$ is the total number of possible activities.

Similar to event attributes, there are also *case attributes*. Case attributes are related to a single case and do not change over time. For example, in the treatment process of patients in hospitals, each patient can be seen as a case and each treatment as an event. Aside from the treatment information for each patient, there is also a collection of patient information. This patient information can, for example, contain the name, date of birth, and place of birth.

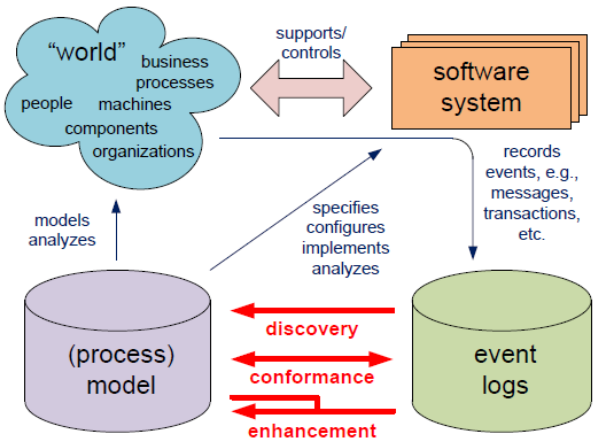

Figure 2.1: Links between Software, Event Logs, Models and the enviornment [29]

Figure 2.1 shows the relations between the real world, software systems, event logs, and models. There are three connections between 'event logs' and '(process) model'. Each connection refers to a different type of process mining. Process discovery is the process of discovering a process model based on an event log. Conformance checking is the process of comparing the actual process with a process model, which can be done either to assess the quality of the model or to detect anomalies in traces. The last type of process mining that Figure 2.1 displays is model enhancement. Here an existing model is enhanced with information from an event log.

Another type of process mining that is not included in Figure 2.1 is case prediction. In case prediction, prediction models are trained based on historical data. Historic data comprises the event logs of *complete cases*, which are cases for which no events will occur in the future. These models are used for predictions for a single case. These predictions can be made for other complete cases or *running cases*. Where running cases are cases that are currently in the system and for which events will occur in the future.

A common misconception of process mining is that it is limited to control-flow discovery [30]. The process discovery segment is an important part of process mining to gain understanding in

the real-life process. However, process mining is used for other principles as well. Aside from the control-flow perspective, there are also the organizational perspective; the time perspective and the case perspective.

2.1.1 Petri nets

One method for depicting process flows are Petri nets. In this section, we provide the basic concepts of a Petri net. There are four common components in a Petri net, namely: Places, Transitions, Arcs, and Tokens. Places carry tokens, and each place can have any amount of tokens. Arcs connect places and transitions. A common method of representing them is shown in Figure 2.2.

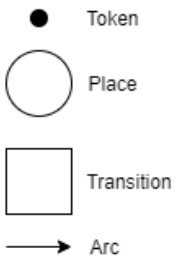

Figure 2.2: Representations of Places, Transitions, Arcs and Tokens

These components can be combined to depict a process. Four often reoccurring combinations are splits and joins, which are depicted in Figures 2.3 to 2.6. These figures are also used to describe the basics of Petri nets. The firing of transitions is critical in understanding Petri nets, since all transitions that are fired from 1 token in the start place up until there is one token in the end place make up a trace. Both transitions and places have arcs entering and outgoing arcs, there are two exceptions. The starting place has no arcs going in, and the end-place has no outgoing arcs. A transition is enabled when each place that is entering the transition carries a token. The amount of tokens produced is equal to the number of outgoing arcs, where one token is created for each place connected to the exiting arcs of the transition.

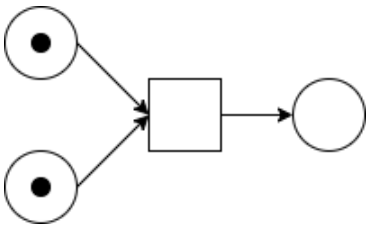

Figure 2.3: AND-Join

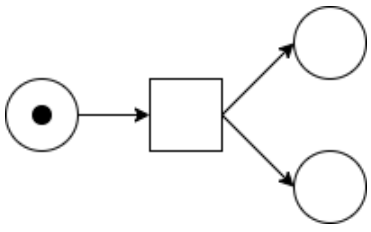

Figure 2.4: AND-Split

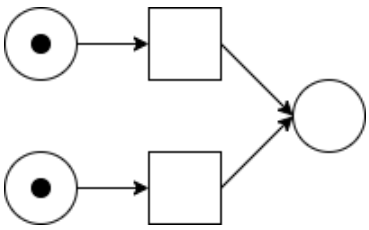

Figure 2.5: XOR-Join

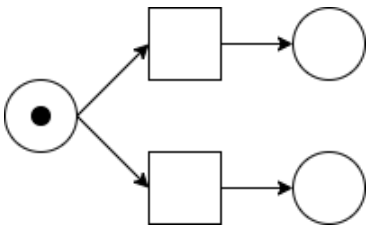

Figure 2.6: XOR-Split

For example, Figure 2.3 shows two entering arcs to the transition and one exiting arc. The places connected to the entering arcs each contain a token, which makes it possible to fire the transition. If this transition fires, then those two tokens are consumed and removed from the place. As a result, one token is added to the place connected to the exiting arc. The AND-Split, shown in Figure 2.4, is the opposite of the AND-Join. When this transition fires, one token is consumed from the place connected to the entering arc. Next, two tokens are produced, one for each place connected to the exiting arcs. Figure 2.5 shows two enabled transitions. These transitions can be fired in any order and each produce one token in the place connected to the exiting arc. The last example is shown in Figure2.6, which shows two enabled transitions but only one token. One of these transitions can be fired, after which the token from the starting place is consumed and a token is produced in the place connected to the exiting arc. After firing one transition, there are no tokens left in the starting place, which are required in order to fire a transition. In the example, the transitions are not named. However, with named transitions, recording the ordered list of fired transitions results in a trace. The names of the transitions would then be the event names or activities.

2.2 Prediction

Data Mining is defined as the extraction of knowledge from large amounts of data. This definition is similar to the definition of process mining. However, process mining focuses on a specific type of data, namely event logs[30]. Machine learning focuses on the creation of a model using training data to either predict or find patterns in data[3]. This thesis focuses on supervised learning. For supervised learning methods, each data point contains a target label l. This label is the targeted variable for prediction and is known for the training data. Labels can be categorical, binary, or a metric variable. In contrast to supervised learning methods, unsupervised learning methods do not require labels for each data point.

In short, the goal of supervised learning is to derive a model from training data to predict the labels of data points where the label is unknown. This model is trained on data with a known label l with input attributes I. The set of possible labels is L, which can be discrete or continuous. If the set of output labels L is continuous, then the prediction task is a regression problem[14]. If the set of output labels L is discrete, then the prediction task is a classification problem, where each value in L is a class.

Definition 2.2.1. (Prediction Model)
Given the input variables of one data point I, and given output labels $l \in L$ a function can be derived to make predictions: $l^* \in L$. This function is written as $P : I \rightarrow l^*$. The prediction model is the function that predicts the output label based on a set of input variables.

2.2.1 Evaluation

The evaluation of prediction models depends on the prediction task, i.e., the evaluation methods are different for regression and classification. In this section, we only discuss the most common evaluation methods classification.

Two-class classification models are generally evaluated using a set of data where the class is already known. By comparing the actual classes to the predicted classes, it is possible to identify four measurements. These measurements are: the number of True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN). These concepts are explained as follows:

TP: Total number of data points that were classified as true which actual class is true
FP: Total number of data points that were classified as true which actual class is false
TN: Total number of data points that were classified as false which actual class is false

FN: Total number of data points that were classified as false which actual class is true

A common way to represent the number of TP, TN, FP, and FN is to use a confusion matrix. Here the predicted classes are compared to the actual class in a tabular form, as shown in Table 2.1.

Table 2.1: Confusion Matrix

		Predicted Class	
		True	False
Actual Class	True	TP	FN
	False	FP	TN

The four previously described measurements are input for several evaluation measures. The two most-basic evaluation measures are the error rate(2.1) and the accuracy rate (2.2). These are the rate of wrongly classified data points and the rate of correctly classified data points, respectively.

$$Error\ rate = \frac{FP + FN}{TP + TN + FP + FN} \tag{2.1}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$

Three other methods are: Recall(2.3), Specificity(2.4) and Precision(2.5). Precision provides a ratio between the data points that are correctly classified positive and the total amount of positive data points. Recall, which is also called sensitivity, provides a ration between the data points correctly classified positive and the data points incorrectly classified as negative. Specificity is similar to recall, but it centres around negative classifications and data points that are negative. These methods might be more relevant for evaluation due to a difference in interpretation. For example, in a hospital, recall might be more important when using classification to predict the required treatments. Here false positives are less of a problem than false negatives, since the consequences of a false negative are larger.

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

$$Specificity = \frac{TN}{TN + FP} \tag{2.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

Another evaluation method is the ROC curve, which stands for Receiver Operating Characteristic. When creating a classification model, the model assigns scores or probabilities to a data point for it to belong to one class. With this probability, it is possible to change the threshold for assigning it to a class. The ROC curve plots the false-positive rate (1-Specificity) against the true positive rate (Sensitivity) for all thresholds. For example, at a threshold of 0, every data point is classified as positive or true. This threshold results in a Specificity of 0 and a Sensitivity of 1. A model performs best when there is a point for which the sensitivity is high and the Specificity is high. Figure 2.7 shows an example of a ROC curve. The displayed curve shows that the underlying model performs well. The closer the curve gets to the upper left corner, the better the model. Where a perfect model has a threshold for which the false positive percentage equals 0 and the true positive percentage equals 100. In the example, there are several thresholds for which the false positive percentage is low, but the true positive percentage is high, indicating a well-performing model. From the ROC curve, it is possible to calculate the AUC, which stands for Area Under Curve. A higher value for AUC is better since this indicates a higher true positive rate with a lower false positive rate. The AUC value for the ROC curve shown in Figure 2.7 equals

97.16%.

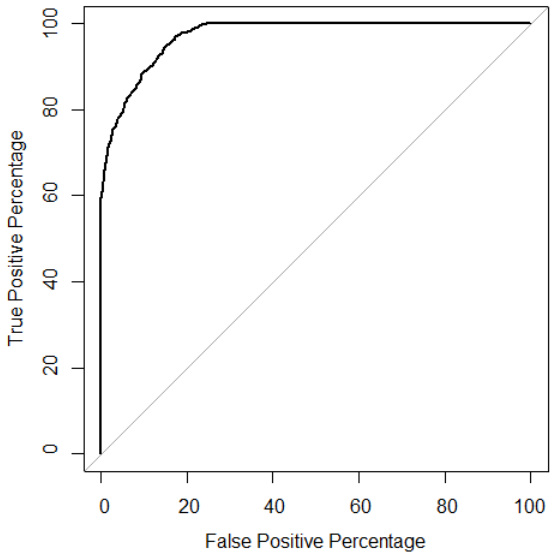

Figure 2.7: ROC Curve Example

Chapter 3

Related Work

Data mining in manufacturing is not new. There is numerous research that tries to predict outcomes such as the total throughput time and the quality of finalized products. The field of process-mining, however, is relatively new; some uses of process data have been described before, but were not classified as a different field of data mining. Making predictions in process mining has been mentioned before, where it is used to find correlations in business process characteristics [9]. However, similar to other literature [6][7][11][15][18], it selects attributes from a complete process rather than translating the process sequence or event log into attributes that can be used for classification. There also exists literature on the predictions of prefixes of processes [21]. In these articles, the attributes are predefined rather than derived from the event-log. Other literature that is irrelevant for this thesis focuses on continuous measurements [12][26], or does not use data on case level[17][26]. In this chapter, we discuss the previous efforts to make predictions using event logs, which are on case level and translate the event-log into features in order to make predictions.

The first part of literature discussed first is not related to manufacturing, but provides an overview of ways to translate sequences into features. Dong and Pei[10] distinguish between basic types of sequence features. It distinguishes between two simple patterns, namely k-grams and k-gapped pairs. K-grams are a subsequence of length k, where k-gapped pairs are parts of a sequence with a fixed distance between positions. Besides the k-gram and k-gapped pairs there are more complex features, where the amount of sequence elements and distance between positions is flexible.

The values of these features are of two types: presence and count. Where a presence type feature is a Boolean feature, which means the feature equals true if the sequence element is present, or false if it is not. The second type is a numeric feature, which counts how many times the feature occurs.

There is one article that includes the most basic forms of translating sequences from event logs into features. The most basic form would be a 1-gram[19]. Where only one sequence element, here an activity, is used as a pattern. The article covers both types of feature values, namely presence and count. In this article, they are named as Boolean encoding and frequency-based encoding.

The next step is a 2-gram, which is covered in one article in an assembly environment[8]. In this article, the method is referred to as a Bi-gram. Da Cunha, Agard and Kusiak use a randomly generated dataset in order to improve the quality of an assembly process. They try to capture patterns from sequences that result in faulty products. They do so by translating the sequences into features for the extraction association rules. The features they extract are a subsequence of length two.

This research was then picked up and continued by Rokach[24], who described the use of the

Teiresias algorithm [23]. This algorithm results in more complex sequences than k-gram and k-gapped pairs[10]. The extracted features are patterns that are restricted by three variables: the maximum length of a pattern, the maximum number of gaps, and the minimum number of occurrences. This algorithm is tested and results in similar results as the Bi-gram, the main difference being that the resulting decision trees are less complicated when features are created using the Teiresias algorithm.

Another algorithm that tries to extract matching patterns is the FAST algorithm [27]. However, Different from the other articles, it does not use the resulting patterns as features. Ceci, Fumarola, Lanotte and Malerba[5] solely use the resulting frequent subsequence to decide on features that can be used for a given pattern, since these features result from parts of the subsequence. Although this is different from the previous methods to create features from event logs, it is capable of including features that become available during the process.

The last described method to translate a trace into features is to use trace indexes as features instead of activities [8][19]. Each activity can be seen as a sequence element which, together which its position, is recorded as a feature [10]. However, instead of creating multiple Boolean variables, it is compressed into one categorical variable, which is called the simple-index encoding[19]. This encoding results in the same number of features as the number of activities in the prefix.

The simple-index encoding can be used to create features from the so-called payload, which is the information that is included in the event log aside from the activity. Translating payload can be done by using only the payload of the latest activity, combined with the simple-index encoding, resulting in the latest-payload encoding[19]. Instead of using only the latest payload, it is also possible to use the payload of each event included in the prefix. Here, each activity, as well as each feature resulting from the payload, is numbered. The number relates to the index of each activity[19]. This encoding is similar to the one described by Maki and Teranishi[22]; in their notation, it is assumed that each activity sequence is the same. This means that the event sequence is not present in the dataset that was used for classification, as it would not add value.

3.1 Conclusion

Only little research is focused on manufacturing sequences. Most other sequence encodings that are described are used in a different environment. The sequences that are used in a manufacturing environment[8][24] focus on sequences that can happen in any order. This thesis focuses on automated processes, where the amount of paths and the complexity of sequences is limited. Since these processes are less complex than other processes, mining large sequences from the process does not result in useful features. Differences in quality are more likely to be the result of small differences in the process, rather than a great combination of activities. Besides that, the results from the discussed research show that complex methods provide close to equal results as a more complex sequence encoding [10].

Chapter 4

Translating Traces to Attributes

One crucial step before making predictions using data from an event log is the translation from traces to features. The related work section describes the literature used as inspiration for doing this translation step. In this chapter, we cover different methods of translating process data into features for prediction into more detail.

For this chapter, we use a running example. In this example, there are four activities named after the first four letters of the alphabet. Which can be translated to: $V_{activity} = \{A, B, C, D\}$, with an example trace of length $n = 5$ with activity sequence: $\#_{activity}(\sigma) = \langle A, B, A, B, D \rangle$. This trace is used to elaborate on the introduced translations in this section.

4.1 Translating Sequences

We consider four methods for the translation of event logs. This section describes those methods. In the next section, one of the translation methods is extended to include event attributes.

4.1.1 1-Gram

The first type of attribute translates the presence or absence of a sequence of length 1 into one attribute. This means that the number of created attributes is equal to the number of activities in the event-log, i.e., $|Attributes| = |V_{activity}|$. Both the Boolean feature type and frequency feature type are similar. However, the first only indicates the presence of the sequence and the second provides the count of occurrences. First, we define a function $\alpha(e)$ that checks if the activity of an event matches the target activity.

$$f(e, target) = \begin{cases} 1, & \text{if } \#_{activity}(e) = target. \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

$$\text{where } e \in \mathcal{E} \text{ and } target \in V_{activity}$$

Equations 4.2 and 4.3 use Equation 4.1 as input. In order to get an attribute value of a Boolean type, the Equation 4.1 needs to be checked for each event in the trace and the maximum value is picked. This results in a 0 if none of the activities matches the target activity and a 1 if one or more activities match the target activity. The previously described steps are combined in Equation 4.2. When this equation is used for all activities in $V_activity$ of the previously introduced example, it results in Table 4.1.

$$g(\sigma, target) = \max_{i=1}^{n} f(e_i, target) \tag{4.2}$$

Table 4.1: Boolean 1-Gram Example

A	B	C	D
1	1	0	1

In order to get an attribute of the frequency type, the sum of all values from Equation 4.1 is required. This is performed using Equation 4.3. If Equation 4.3 is used for each of the activities in $V_{activity}$ on the events of the trace of the previously mentioned example, it results in Table 4.2

$$g(\sigma, target) = \sum_{i=1}^{n} f(e, target) \tag{4.3}$$

Table 4.2: Frequency 1-Gram Example

A	B	C	D
2	2	0	1

4.1.2 Bi-Gram

Another method to translate an event-log into attributes is to use a combination of two consecutive activities. When following the literature of Dong and Pei[10], the combination of two consecutive activities is referred to as a 2-gram or a 0-gapped pair. However, Da Cunha, Agard and Kusiak name this method a Bi-gram[8]. For the remainder of this thesis, this method is referred to as Bi-gram. The setup is similar to the setup of the 1-gram method, but the function f now requires two subsequent events as input and two target activities, instead of just one. This results in Equation 4.4.

$$f(e_i, e_{i+1}, target_1, target_2) = \begin{cases} 1, & \text{if } \#_{activity}(e_i) = target_1 \ \ \& \ \ \#_{activity}(e_{i+1}) = target_2. \\ 0, & \text{otherwise.} \end{cases} \tag{4.4}$$

$$\text{where } e \in \mathcal{E} \text{ and } target \in V_{activity}$$

Using Equation 4.4 two new functions for g can be defined. This function can be used for any combination of two activities from $V_{activity}$, and thus results in a larger number of attributes than the previously mentioned 1-Gram. Since any two combinations of activities can be an attribute, the number of attributes is the number of activities to the power of 2, i.e. $|Attributes| = |V_{activity}|^2$. The Boolean attribute type results from the maximum value of all values for f. This is shown in Equation 4.5. An example of a table for this type of feature is shown in Table 4.3, where there are 16 attributes, since $|V_{activity}| = 4$.

$$g(\sigma, target_1, target_2) = \max_{i=1}^{n-1} f(e_i, e_{i+1}, target_1, target_2) \tag{4.5}$$

Table 4.3: Boolean Bi-gram Example

AA	AB	AC	AD	BA	BB	BC	BD	CA	CB	CC	CD	DA	DB	DC	DD
0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0

The same steps can be performed to get a frequency type attribute. Similar to the 1-gram, the resulting values from f need to be summed. This method is not used in this thesis, since this method is not discussed in previous literature.

The main advantage of using the Bi-gram over a 1-gram is that it translates a part of the sequence. However, it results in a large number of attributes, which can result in higher calculation times or problems with classification algorithms.

4.1.3 simple-index

The simple-index method is different from the previous two methods, since it is not a binary variable nor a numeric variable. The resulting attributes that are translated using the event log are categorical variables. For each event, the activity is stored as an attribute value. Therefore, the attributes are translated using a function that was introduced before but is mentioned again for completeness. The function is defined in Definition 2.1.5, where the attributes are retrieved, where the simple index translation method focuses on the activity of each event.

$$\#_{activity}(e) = Activity \tag{4.6}$$

$$\text{where } Activity \in V_{activity}$$

Each event maps onto one attribute; this means that the number of attributes is the same as the length of a trace, i.e. $|Attributes| = n$. Given the trace σ that was given as an example earlier in this section, the translation to attributes results in five attributes. The five attributes resulting from the example trace are shown in Table 4.4.

Table 4.4: Simple-Index Example

Activity 1	Activity 2	Activity 3	Activity 4	Activity 5
A	B	A	B	D

The simple-index method may be the simplest method to translate a trace of events to a set of attributes. An algorithm can find combinations of activities or patterns that have a relation to the label of a case. The most considerable difference between the previously described methods is that the index of each activity is relevant as well. If a pattern that is related to the outcome occurs in different traces, but these patterns start at different indexes, then the pattern might not be recognized by an algorithm or model. Although this problem might occur, a previous study has shown that it has a better performance than the previously described methods[19].

4.2 Translating Payload

Activities are not the only attributes contained in the event log. Aside from the Case ID and activity of each event, the log often contains other data as well. The most common types of additional attributes are resources and timestamps. Timestamps can indicate the start or end times of an activity, or two timestamps are present to indicate both the start and end times of activities. The event attributes besides the activity, case ID and timestamps can be referred to as payload [19]. The payload that is available in an event log can be used to create additional payload, which is called event log enrichment [28]. Payload cannot be used for classification or regression if it is not translated into attributes. Therefore, this chapter also provides a method to include payload into the attributes used for classification.

Literature introduces only one generic method to translate payload to activities. This method is similar to the simple-index method described in the previous section. Other literature which describes the use of activity sequences to attributes that do include payload are not generic; they

describe a method to select attributes for matching prefixes [27].

The only method that was covered in literature to translate payload is an extension of the simple-index method that was described before. Instead of using the function described in Definition 2.1.5 only for the activity attribute, the function is used for the entire payload.

Then again, there is the decision to decide which events and their payload to translate into attributes. Two methods described in literature [19] involve the translation of either only the last event or the translation of all events.

As an example, consider the sequence σ which was introduced as an example in the previous section. In the previous section only the function$\#_{activity}$ was considered. Now we can include other event attributes, such as: employee, machine, day and shift. Some possible values for these attributes are $V_{say} = \{Monday, Tuesday, Wednesday, Thursday, Friday\}$, $V_{employee} = \{Frank, Eva, John\}$, $V_{machine} = \{1, 2, 3\}$ and $V_{shift} = \{Morning, Day, Night\}$. Then a possible translation of the payload of two events of a sequence is shown in Table 4.5.

Table 4.5: Payload translation Example

Employee 1	Machine 1	Shift 1	Day 1	Employee 2	Machine 2	Shift 2	Day 2
John	1	Night	Monday	Eva	3	Night	Monday

Chapter 5

Comparison of Classification Techniques

A prediction model type has to be chosen before making predictions using an event-log. Since this thesis focuses on the added value of two different types of data, it was chosen to only use one algorithm for classification.

Before comparing the different algorithms, we make a list of the desired characteristics of the algorithm. First, it should be possible to identify which of the attributes used for classification are most important. This information makes it possible to identify relations with attributes to a particular type of rework, which would make it possible to improve the process. Second, the algorithm should be able to take numeric as well as categorical variables into account, since these are common types of variables to find in an event log. For example, the employee that performed an activity would be a discrete attribute and the duration of an activity is a continuous attribute.

Additionally, it would be preferred for the algorithm to be easy to adjust, since one model is trained for each prefix. As a result, numerous models have to be trained, for each of which the parameters can be adjusted to obtain optimal results. If the number of input parameters of the algorithm is low, then the optimization of each model takes considerably less effort and time.

Other algorithm characteristics, such as calculation time are less important, since there is no time restriction as of now. Calculation times might become more critical if it were to be implemented and used on a day-to-day basis.

A review of classification techniques from 2007[13] identifies five main types of classification techniques. The techniques are divided into logic-based techniques, perceptron-based techniques, statistical learning methods, instance-based learning and support vector machines. Next, we evaluate which techniques are most in line with the requirements. This evaluation is supported by Table 5.1, which summarizes different characteristics of the different types of techniques.

Table 5.1 displays a summary of characteristics of six different classification techniques. In this table, logic-based techniques are separated into decision trees and rule-learners. Based on the formulated criteria, a technique is selected that fits these requirements best. The two main criteria being the transparency of the technique and the ability to deal with different attribute types.

Let us first consider the first criteria: The technique should be able to explain which attributes are most important in classifying a data point. As shown in Table 5.1, two techniques have a low performance on the transparency of knowledge, namely neural networks and support vector machines(SVM). Notably, these two techniques are among the highest performing techniques in

Table 5.1: Comparison of Classification Techniques (* indicates lowest performance and **** highest)[13]

	Decision Trees	Neural Networks	Naive Bayes	kNN	SVM	Rule-learners
Accuracy in general	**	***	*	**	****	**
Speed of learning with respect to number of attributes and the number of instances	***	*	****	****	*	**
Speed of classification	****	****	****	*	****	****
Tolerance to missing values	***	*	****	*	**	**
Tolerance to irrelevant attributes	***	*	**	**	****	**
Tolerance to redundant attributes	**	**	*	**	***	**
Tolerance to highly independent attributes	**	***	*	*	***	**
Dealing with different attribute types	****	*** (not discrete)	*** (not continuous)	*** (not directly discrete)	** (not discrete)	*** (not directly continuous)
Tolerance to noise	**	**	***	*	**	*
Dealing with danger of overfitting	**	*	***	***	**	**
Attempts for incremental learning	**	***	****	****	**	*
Transparency of knowledge/classifications	****	*	****	**	*	****
Model parameter handling	***	*	****	***	*	***

terms of accuracy in general. Less accurate algorithms like decision trees, rule-learners and Naive Bayes seem to be a better fit for classification of rework, due to the importance of transparency of the techniques.

The second criteria for the selection of a classification technique is the ability to deal with different attribute types. Table 5.1 shows a clear favorite, namely decision trees. All other algorithms have a problem in dealing with either continuous attributes or discrete attributes. From the algorithms that are under consideration, the only other classification technique that might be interesting are rule-based techniques. Rule-learners can be adjusted to be able to deal with both continuous and discrete attribute types. However, it is beneficial to discretize the continuous attributes in order to increase accuracy and to reduce the time required for training [2]. The referenced table is retrieved from an article that was written 13 years ago. This provides a possible explanation of why it fails to mention that attributes can be adjusted in order to work with Neural Networks, Naive Bayes or SVM. However, decision trees are still more suitable than the other, since they do not require any adaptation of attributes and provide transparency.

In short, logic-based classification methods seem to be the best fit for the first requirement since they are the easiest to interpret [13]. From the logic-based classification methods, decision trees seem to to have the best performance on the chosen criteria. Decision trees do not require any modification on the attributes before they can be used and are therefore the best technique to deal with different attribute types. Notably, this is in line with the previously discussed related work. These articles all focus on logic-based classification methods, where most are focused on decision trees[24][19], while another is focused on more rule-based techniques[8]. The choice of the latter article for using a rule-based technique might be the result of the data only having discrete attributes.

5.1 Decision Trees

Decision trees assign a data point to a class using their feature values. These rules are applied hierarchically. After applying all the rules, the data point is assigned to a class. Figure 5.1 shows an example of a decision tree. Here there are four different attributes: 'at1', 'at2', 'at3' and 'at4. There are two classes in which each data point can be classified: 'yes' and 'no'. at1 and at2 have three possible values, while at3 and at4 only have two possible values. A data point starts from the top, if its value for at1 is a1, it goes on to the next node. Otherwise, if its value for at1 is b1 or c1, then the data point is classified as no. The same decision is made for the different attributes until a data point a class is assigned.

Instead of training one decision tree, it is possible to train multiple trees. The combination of decision trees and letting them vote for the class of a data point showed significant improvements in classification accuracy[4]. The 'growing' of multiple decision trees and letting them vote for the class of a data point is a procedure called random forests. Aside from its high performance in classification accuracy, random forests have multiple other advantageous features. One important feature is that random forests do not overfit[4]. Another positive feature is that it is still possible to gain insights into the importance of the features used for classification. One of the main reasons to choose decision trees over other classification methods is that it also provides insights into the process when used on an event log. With the combination of multiple decision trees, this is still possible. Just as with decision trees, it is capable of handling both numerical and categorical data. This flexibility avoids the preprocessing of variables before using them to train a model. Lastly, it is also a quick method to train models [4].

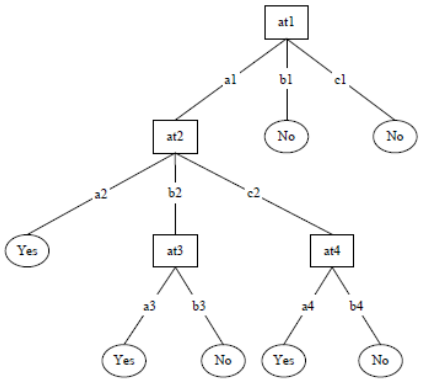

Figure 5.1: Decision Tree Example [13]

Chapter 6

Method Description

In previous chapters, we described five methods to translate event logs into features and which classification model would be most useful. This chapter describes the order of actions which are required to use event logs in order to obtain results in combination with usable models. Besides descriptions, we also provide an example for clarification. The running example is an extension of the running example introduced in Chapter 4, where an activity 'Rework' is added and two other traces are added. This results in the set of activities $V_{activity} = \{A, B, C, D, Rework\}$ and activity sequences $\#_{activity}(\sigma_1) = \langle A, B, A, B, D\rangle$, $\#_{activity}(\sigma_2) = \langle A, B, C, D\rangle$ and $\#_{activity}(\sigma_3) = \langle B, A, Rework, C, D\rangle$.

This thesis highlights two types of data, namely case data and event data. Case data contains information about each case and is not dependent on the process. Event data, or the event log, is data that is stored retrieved from the process and is therefore dependent on the process. Due to this difference, the number of required models are different for both data types. Only one model is required when only considering the case data since no additional data becomes available during the process. Event data is different; each event contains new information. There is a model required for each prefix length, as more information becomes available for each event.

First, we discuss how the label is determined from an event log. In this thesis, we only predict one type of rework. Therefore, the resulting label is a Boolean variable. The value of the label of a case is 'TRUE' or 1 if the rework activity is present in its trace, and 'FALSE' or 0 if none of the activities in the trace match the rework activity. This method is the same as the method used in the Boolean type of the 1-Gram, which is described in Equation 4.2. Equation 4.2 makes use of the input Equation 4.1. The resulting labels for the example traces σ_1, σ_2, and σ_3 are 'FALSE', 'FALSE' and 'TRUE' respectively. This can be derived from Equation 4.2, which results in $g(\sigma_1, Rework) = 0$, $g(\sigma_2, Rework) = 0$ and $g(\sigma_3, Rework) = 1$. The resulting labels are used in combination with the case attributes and the features that were translated from the event log to train models.

When using the case log for prediction, it is the same as most common prediction problems. Each case is a data point with one label. All case attributes can be used as input for training the prediction model. When including the translated event logs, there are more models required since the amount of available data differs for each prefix length. As a result, one model is created for each prefix length.

As the following step, the event log is translated into attributes for classification. This step requires one of the five translation methods described in Chapter 4. The translation of a trace into attributes has to be done for each prefix length m up until the complete trace length n. This translation is repeated for each case. Table 6.1 shows the different attributes for each trace at a prefix length of 5, thus the complete traces, and their corresponding label. As simple as it sounds,

the prediction of a rework activity which is present in the event log results in multiple problems. In the remainder of this chapter, we discuss each of these problems and describe the implemented solution.

Table 6.1: 1-Gram Boolean Attributes and labels of example traces

Trace	A	B	C	D	Rework	Label
σ_1^5	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE
σ_2^5	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
σ_3^5	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE

One can imagine that each case requires a different amount of activities before it finishes. One solution might be to ignore cases with a smaller length than the prefix length on which the model is trained. Ignoring these cases results in another problem, since many data points are then ignored when training a model. In the example, σ_2 would be ignored when training a model for a prefix length of 5. However, in order to include all cases for each model, cases can be made longer by adding 'empty' events. These events do not hold any information. The translation of cases which are shorter than the maximum trace length results in the same attributes when translating the event log using the 1-gram method or the Bi-gram method, i.e. $g(\sigma_2^4, Target) = g(\sigma_2^5, Target)$ for all target activities in $V_{activity}$. In the simple index method, there are additional attributes for each additional event. The attribute values that cannot be filled in are left blank. Classification algorithms can have problems with missing values. These problems can be avoided by filling in a value which is the same for each attribute which should be left blank. Using the simple index translation method to translate σ_2^5 into attributes then results in the attributes shown in Table 6.2. In this table, the fifth activity is an empty activity.

Table 6.2: Simple-Index Example

Activity 1	Activity 2	Activity 3	Activity 4	Activity 5
A	B	C	D	None

Another problem that results from this approach is that both the label and are derived from the event log. This means that the information about the label is also present in the attributes used for classification. These attributes thus contain information which also indicates the label. In this example, the value for rework is the same as the label value. One possible solution is to only consider the section of the trace up until the rework activity, i.e. $\#_{activity}(\sigma_3) = \langle B, A \rangle$. However, this solution results in another problem. If we ignore all activities after and including rework, then other possible signals give away if rework has occurred or not. In the given example, this would be activity 'D'. This activity only occurs at later indexes. If this activity is not present in a trace, then that trace contains 'Rework'. Another possible solution is to ignore the rework activity, i.e. $\#_{activity}(\sigma_3) = \langle B, A, C, D \rangle$. This solution uses less information than is available. The model might still provide wrong classifications even though 'Rework' has already occurred in the trace.

The implemented solution does not adapt the traces, but adapts the evaluation method. Since the translated attributes after certain prefix length contain information about the label, one can expect that the accuracy at one point will reach 100% for running cases. The classification accuracy should become 100% at a certain prefix length. When all rework activities have occurred, all traces with a rework activity are correctly classified. All other traces are correctly classified as false, When an activity which only occurs after rework. In the example traces, this occurs at a prefix length of 5. At this prefix length, all reworks have occurred, since rework only occurs at the fourth event. For the other traces, 'D' occurs as a fourth or fifth event. The presence of 'D'

indicates that no rework will occur in the future. The combination of this information provides the opportunity to achieve a classification accuracy of 100%. Figure **??** provides a section of a decision tree which can be added to ensure that 100% classification accuracy gets achieved for the traces provided in the running example. To ensure that a model can 'know' when a trace has ended, we assume that all cases have the same end activity. In the example, this end activity is activity 'D'. If there is no such activity, then artificial start and end activities can be added to traces.

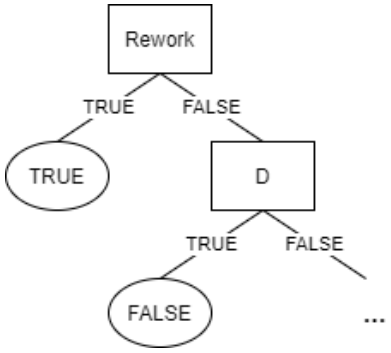

Figure 6.1: Decision Tree which Results in 100% Accuracy for the Given Example

In order to assess the performance of the trained classifications models, a baseline needs to be defined. This baseline is called the *naive classifier*. This classifier classifies all new data points into the same class, which is dominant in the training data. For example, if 30% of the training data was labelled as true and 70% labelled as false, then the naive classifier classifies all of the new data points as false.

The naive classifier has to be extended when including event data. Since the previously described naive classifier only takes the labels of the training data into account, it never reaches an accuracy of 100%. The naive classifier is extended to achieve 100% accuracy as well. The naive classifier predicts all cases as false for all prefixes, up until when a prefix contains the rework activity. When a prefix contains the rework activity, then $l^* = TRUE$. For this naive classifier extension, we assume that rework occurs on less than 50% of the cases. Table 6.3 shows all predictions made using the naive classifier for each prefix length of each example trace. In this example, an accuracy of 100% is reached at a prefix length of 3.

Table 6.3: Label Predictions Resulting from the Naive Classifier

Prefix Length:	1	2	3	4	5	Actual Class
σ_1	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
σ_2	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
σ_3	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE

Given the naive classifier, it would not make sense to evaluate the models based on the ROC curve or AUC. The simple classifier does not provide scores or probabilities for data points to belong to a class, making it impossible to plot the ROC curve or to calculate the AUC. In this thesis, we focus on the accuracy of models. The main concern of using accuracy to assess models is imbalanced data. When the predicted class of all cases is equal to the most abundant class, then a high accuracy can still be achieved. For this reason, models are compared to the naive classifier, which takes away this concern. The naive classifier is used in the next two chapters, both to validate the method described in this section and to assess models.

Chapter 7

Method Validation

The previous chapter described a method to predict future rework in running cases where rework is present in an event log as an activity. This method first needs to be validated before using it in a case study. For the validation of the previously described model we make use of a randomly generated event log. For validation, we describe four phenomena which are possible causes for the presence of a rework. These phenomena are by no means a complete representation of causes for rework, but they provide a sufficient test for the method. These phenomena are explained using the process of a baker.

The first possible cause is the presence of a specific activity. This cause can occur if rework of a product is the result of an addition to the product. For example, in the example of the process of baking, if the glaze has to be redone more often than a product without it, then the activity of adding glaze to the product is correlated to the presence of rework.

Second is the duration of an activity. The duration of an individual activity can be the cause of rework. A logical Explanation from the bakery can be the activity of baking. If a cake sits in the oven for too long, then it is likely that additional action is required to deliver a final product that meets the requirements.

Another possible cause for rework is performing an activity twice. If a pastry needs two be in the oven two times, for example when baking the bottom and again when the top and stuffing are added. Then it is more likely that the pastry burns, since then the bottom is placed in the oven twice.

The last phenomena that we add in our generated event log, is the order of activities. Two activities can happen in parallel, but it might be advantageous for one activity to be started or finished before the other.

For testing the model, we created 10000 artificial cases. Each case has a minimum of 5 activities and a maximum of 9 activities. The process flow is displayed as a Petri net in Figure 7.1. The data is randomly generated using the following constraints:

- Each case has the same start and end event, which are named 'Start' and 'End' respectively. Both these activities have a duration of 0.
- All other activities have a random duration from a uniform distribution between 0 and 100 minutes.
- Each case starts with activities 'A' and 'B', which can happen in any order. Both orders have the same probability of occurring, i.e. 50% and 50%.

- In the following section, the activity 'A' can happen again, or an activity 'C' can occur, or no activity is performed. There is a 25% chance 'A' occurs, 25% chance 'B' occurs and 50% chance that the product continues to the next section. This section is added to include the possibility of the activity A occurring twice.
- In the last section, there are four activities that can happen in any order, but none of which are mandatory. These activities are 'D', 'E', 'F' and 'Rework'. The activities 'D', 'E', 'F' or no activity all have the same chance of occurring and also the order is random.
- The chance of a rework occurring is independent of the activities 'D', 'E' and 'F'. There is a probability of 5% that a rework occurs which increases with the following four phenomena: 'B' happens before 'A', 'B' has a duration longer than 50 minutes, 'A' happens twice, or 'C' occurs. For each of the possible phenomena, the probability that rework is required increases with increments of 25%.

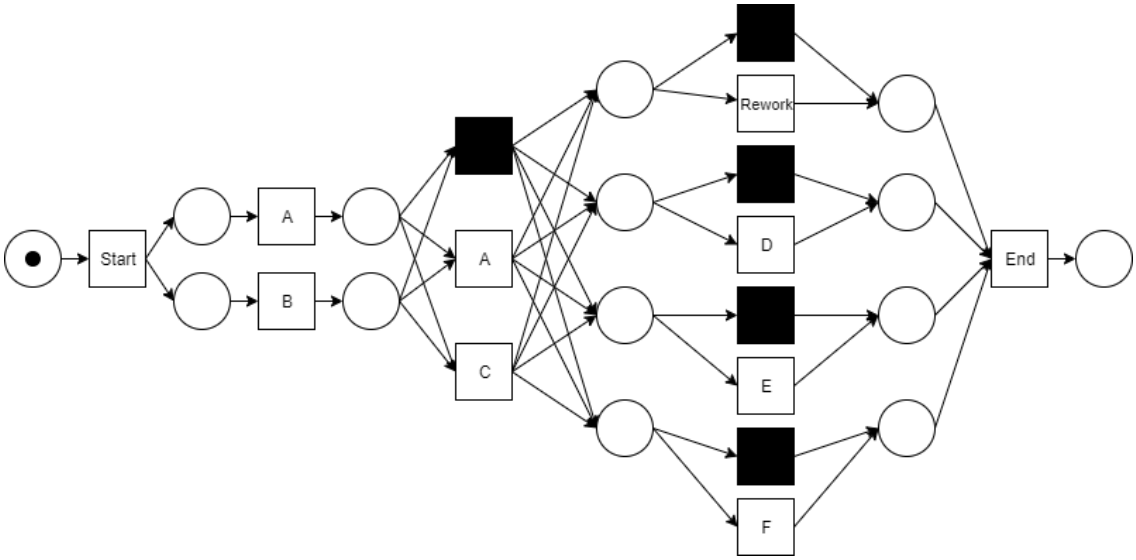

Figure 7.1: Experimental Process visualized as Petri-Net

The expected chance of a rework to occur, with the provided information, equals 42.5%. The expected chance of a rework to occur when 'B' happens before A, or 'B' has a duration longer than 50, goes up to 55%. When both these possible causes occur, this goes up to 67.5% chance. Moreover, if 'A' or 'C' occurs, the chance of rework to be required is 80%. In the next section of this chapter, we discuss the classification results of the random forests that were created using the randomly generated event data and the five previously discussed methods for translating event logs.

7.1 Implementation

In order to implement the different translation methods that were described in Chapter 4 and the model described in Chapter 6, RStudio[25] version 3.6.0. was used. Within RStudio, the package named 'randomForest'[20] version 4.6-14 to train and use random forests.

The implementation is separated into seven different scripts, which are shown in Appendix A. The first script implements the generation of the used event log to validate the described model. Five other scripts are used to translate an event log into attributes for classification. One script for each different translation method described in Chapter 4. The last script implements the naive

classifier, which is used for evaluation. Each of these scripts works with the generated event log, but they can be adjusted to also work with other event logs.

An event log and a case log are the required input for the translation scripts. The minimum requirement of the case log is that it contains the list of case ID's. The generated case log also contains trace length and labels. Including trace length makes it easier to determine the maximum prefix length. Labels of cases are not needed for the translation of event logs. However, they are needed when training and testing models. After translation, the scripts result in one dataset for each prefix length.

The last script that is included in Appendix A implements the naive classifier. It uses an event log to make predictions for each prefix length. Where it results in one table, where each column represents the predictions for that prefix length.

After the translation of the event log using the different methods, the attributes can be used to train models. One model is trained for each prefix length of each different translation method. These models are trained using the randomForest package[20] in RStudio. The number of trained trees equals 200, and the number of used attributes in each tree equals the square root of the total number of attributes. Before the models are trained, the columns with one value are removed. As a result, some translation methods do not result in a model for a particular prefix length. 70% of the data is used for training and 30% as a test set. The scripts are executed using a PC with an AMD Ryzen 5 1600 Six-Core Processor 3.20 GHz and 16GB RAM 3000 MHz. Table 7.1 provides a summary of execution times for the most time-consuming steps. Note that the table shows relatively low execution times, but these execution times grow when there are longer traces and more attributes.

Table 7.1: Execution Time Summary

Translation Method	Total Attributes	Total Models	Execution Time Translation	Execution Time Training Models
Boolean 1-Gram	9	7	16.2 Secs	4.4 Secs
Frequency 1-Gram	9	7	45.3 Secs	4.9 Secs
Bi-Gram	81	8	17.2 Secs	15.5 Secs
Simple Index	9	8	17.1 Secs	4.8 Secs
Simple Index with Payload	18	8	26.2 Secs	11.5 Secs

7.2 Discussion of Results

Table 7.2 shows the classification accuracy of the different event log translation methods and the naive classifier. Note that the table starts with a prefix length of 2. The first index is ignored, due to the first event being the same for each case: a start event with a duration of 0. The same problem occurs for both the 1-gram translation methods at a prefix length of 3. At this length, each case has the same value for each attribute, namely 1 or TRUE for activities 'Start', 'A' and 'B'. The same information is plotted as a graph in Figure 7.2. In the remainder of this section, we discuss the accuracy for each prefix length for each of the different event log translation methods.

Table 7.2: Summary of Random Forest Classification Accuracy on Randomly Generated Data

Prefix Length	1-Gram Boolean Accuracy	1-Gram Frequency Accuracy	Bi-Gram Accuracy	Simple Index Accuracy	Simple Index With Payload Accuracy	Naive Classifier Accuracy
2	0.629	0.629	0.629	0.629	**0.652**	0.573
3			0.629	0.629	**0.652**	0.573
4	0.620	0.646	0.692	0.688	**0.709**	0.587
5	0.704	0.725	0.760	0.766	**0.776**	0.674
6	0.876	0.876	0.888	0.888	**0.894**	0.840
7	**0.974**	**0.974**	0.973	0.973	0.973	0.958
8	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
9	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**

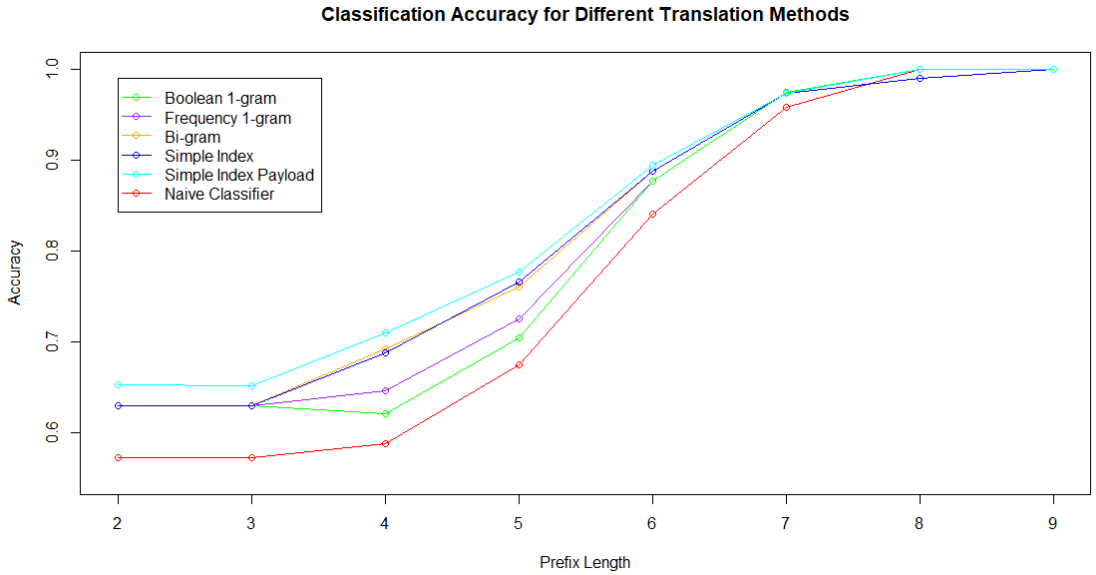

Figure 7.2: Random Forest Classification Accuracy on Randomly Generated Data

At a prefix length of 2, both 1-gram translation methods and the Bi-gram and simple index method have the same accuracy, namely 0.629. Which is higher than the accuracy of the naive classifier, but lower than the simple index method with payload included. These values make sense. At a prefix length of two, each translation method is capable of identifying if 'B' occurs before 'A'. The accuracy of the simple index method with payload has higher accuracy, since it is also able to identify the duration of activity 'B'. Since the naive classifier is incapable of identifying either, it scores lower. When assuming that at a prefix length of 3 the classifications remain the same for the 1-gram method, then the results at a prefix length of 3 are the same.

At a prefix length of 4 there is more difference. The 1-gram Boolean method scores lower than at a prefix-length of 2, this can be explained since there is no way of identifying if 'B' occurred before 'A'. The same problem is present in the 1-gram frequency method, but this method is capable of tracking if 'A' occurred twice, thus explaining an increased accuracy. Notably, the classification accuracy of the simple index method is slightly lower than the Bi-gram accuracy. This difference is difficult to explain, as both methods hold the same information. A possible explanation is the presence of multiple variables with multiple categories, instead of just two, which makes classification more complex. The method where payload is included still performs best, as expected,

since it stores most information. The accuracy of the naive classifier also increases. This increase means that the first occurrences of rework happen at the fourth activity, which is in line with our expectations.

At a prefix length of 5, the accuracy increases for all translation methods. Which is likely due to the increase in occurrences of rework. With this in mind, the explanation for the differences in accuracy stays the same. However, the simple index translation method has higher accuracy than the Bi-gram translation method for this prefix length. This difference is likely due to some small errors.

For the remaining prefix lengths, the difference decreases and the accuracy converges to 1. The only notable difference is the slightly lower accuracy of the more complex methods; this difference amounts to 0.001. The complexity of the methods explains this slight difference.

In conclusion, all of the translation methods perform as expected. Overall the simple index method where payload is included performs best, since it also takes event attributes into account. The Bi-gram method and simple index method without payload perform similarly, which makes sense since both methods store the same amount of information. The 1-gram methods have a worse accuracy than the others, which is because the attributes that result from these translation methods do not store the order in which activities occur. The Boolean 1-gram method scores the worst of all event log translation methods, since its resulting attributes do not store the order nor the frequency of activities. It must be noted that there are scenarios in which the frequency 1-gram translation method can outperform the Bi-gram translation method. If a combination of the same two activities happen more than one time and the frequency of these activities correlates with rework, then the frequency 1-gram method performs better. Lastly, there is the naive classifier, which only reacts to occurrences of rework. The naive classifier has worse accuracy than the event log translation methods, since it does not take any process aspects into account.

Chapter 8

Case Study

In order to test the previously introduced method, we use the different translation methods on a data set that is retrieved from a real automated manufacturing environment. This data provides insights into the usefulness of using event logs to predict rework. In this case study, we use an event log and case data that was retrieved from an actual automated manufacturing environment. The data description is shallow, since the received data was requested to remain anonymous.

The remainder of this chapter contains a short data description and a description. Next, the results of the described actions are presented, and finally, these results are evaluated.

8.1 Data Preparation

In order to implement the previously discussed method, the same implementation as described in Chapter 7 is used. We have used RStudio[25] version 3.6.0. Within RStudio a package was used to implement the random forests, named 'randomForest'[20] version 4.6-14. The usage of this package resulted in an issue with categorical variables, since it had trouble dealing with that had over 52 categories. This issue is solved with data preparation, which is discussed next.

Before the data is ready to be used, it first requires cleaning and preparation. After data cleaning, there are a total of 39960 usable complete cases. As mentioned before, some of the attributes have too many categories to be used for growing random forests using the randomForest package. These attributes are each individually evaluated. Some have 200 or more categories; these categories are filtered. The other remaining categorical variables that are above the threshold were adjusted. The top 49 most occurring categories are kept, and the other remaining categories are changed to the same class, named 'OTHER'. After preparation and cleaning, both the event log and case log are ready to be used. The case log features 39960 cases, each with 35 attributes which are usable for classification. The event log contains over 1.3 million events, each containing 7 event attributes. Besides the payload, there are two timestamps, an activity, and a case identifier. Among the activities, there are a total of 63 unique activities.

Before splitting the data, there was a consideration of how the data is split in reality. If this method for predicting rework is implemented, then the models are trained on a specific date while using historical data. To pick this date, we first look at a dotted chart of the process to gain insights. This dotted chart is created using ProM 6.7[31]. A subsection of the dotted chart is shown in Figure 8.1. Note that the timestamps are anonymized, which is why the values on the x-axis do not make sense. In this dotted chart, there are clusters of events, separated with spaces that are left blank. These blank spaces are weekends, which indicates that there are no activities during the weekend. The absence of activities provides the opportunity to train new models during the weekend. For splitting the data into training and test data, a multiple of weeks for which the

ratio between test and training data equals 70/30 is determined. The dotted chart from Figure 8.1 also provides a visualization of how the data is divided. When ignoring the incomplete cases, this results in a training dataset of 25194 cases and a test dataset of 11043 cases. Cross-validation is not required, since random forests train each tree on a bootstrapped sample from the original training set[4]

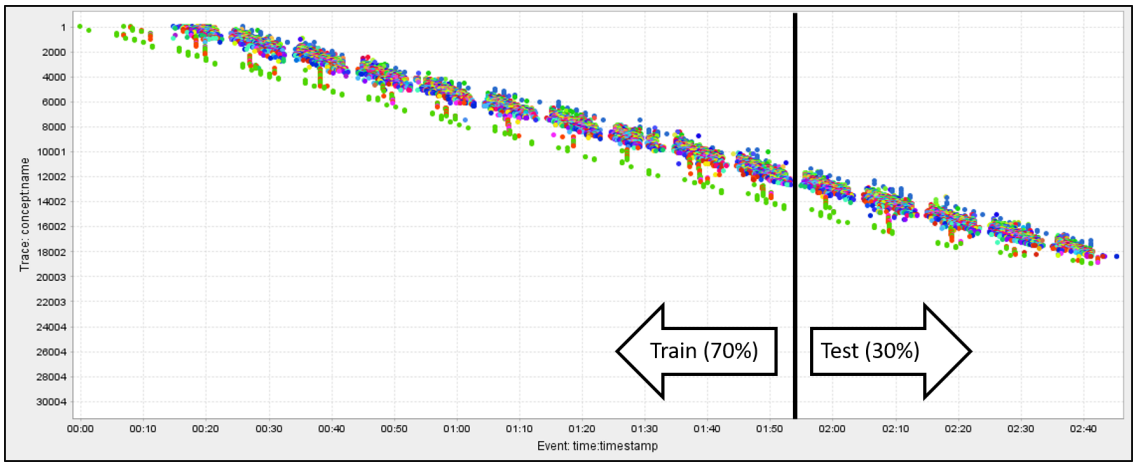

Figure 8.1: Subsection of the Used Dotted Chart

Each of the five described event log translation methods are used to create additional attributes for each case. These attributes are used in combination with the case log, or just by themselves in order to train classification models. From the resulting attributes, a selection is made; attributes for which each case has the same value are ignored. In the next section, we present the results for the different types of data and the different attributes resulting from the five different translation methods.

8.2 Results

This section presents the classification results of this case study. The presented results are interpreted and discussed in the subsequent section. The results are divided into three subsections. The first subsection presents the classification accuracy of a random forest model which was trained using only the case log. The next two subsections provide the accuracy of random forests of all five event log translation methods. Each method results in a total of 54 models, one for each prefix length. In some cases, there are no models for a prefix length of 1, since all attribute values are the same for each case. In the two subsections, there is a difference; one subsection includes the case log data while the other subsection presents the classification accuracy of models that only use the attributes which result from the five different translation methods.

8.2.1 Classification Accuracy of Random Forest trained on Case Data

Table 8.1 shows the confusion matrix of the resulting classifications of the test set. There are a total of 258 true positives, 20405 true negatives, 1138 false positives and 3394 false negatives. These results amount to an accuracy of 0.820.

$$Accuracy = \frac{258 + 20405}{258 + 3394 + 1138 + 20405} = 0.820$$

Table 8.1: Confusion Matrix Case Log Random Forest

		Predicted Class	
		True	False
Actual Class	True	258	3394
	False	1138	20405

To avoid the misinterpretation of accuracy in an unbalanced dataset, we also calculate the accuracy when each case is classified to the largest class. The largest class is 'False', which contains a total of 25153 cases. The remaining cases, amounting to a number of 3651, are classified as 'True'. This results in the confusion matrix shown in Table 8.2. The ratio between these two classes results in an accuracy of 0.855.

Table 8.2: Confusion Matrix Case Log Naive Classifier

		Predicted Class	
		True	False
Actual Class	True	0	3651
	False	0	21543

$$Accuracy = \frac{0 + 21543}{0 + 3651 + 0 + 21543} = 0.855$$

Results show that the naive classifier has higher accuracy than the random forest which is trained on only the case data. However, the classifications using the trained random forest have a higher number of true positives.

8.2.2 Classification Accuracy of Different Translation Methods with Case Data

Next, we present the classification accuracy of the random forests that were trained on attributes resulting from the five different methods in combination with the case data. For the Boolean and frequency 1-gram method, this results in 63 attributes, one for each activity. Of those 63 attributes, there is a maximum of 55 attributes where there is a difference in attribute values. For the frequency 1-gram method, the maximum number of attributes containing differing values is equal to 63; this number is higher since activities that occur multiple times result in a different attribute value. In combination with the case data, these methods result in a maximum of 80 and 98 attributes for the Boolean and frequency 1-gram method respectively.

The Bi-gram translation method can result in a total of 4096 attributes, considering all combinations of all 63 activities. When ignoring the combinations that do not occur in the process, this results in a maximum of 339 attributes. When combined with the 35 attributes from the case log, this results in a maximum of 374 attributes.

Attributes of the simple index method can amount up to a total of 54, since this is the maximum length of a trace. However, the first activity for each trace is the same. This translation method results in a maximum of 53 attributes. When combined with the case data, it results in a total of 88 attributes. When extending this method with event attributes, the so-called payload increases the number of attributes with 378. This number is calculated by multiplying the number of event attributes with the maximum length of a trace. After filtering the attributes where each case has the same value, it results in a total of 429 attributes and 464 attributes when including the case log data. This total means that there are two additional attributes which have the same value for all cases.

The classification accuracy of all trained random forests is shown in Table 8.3. The last six prefixes are not shown in this table, since these show the same results as the prior prefix. The same information is visualized in Figure 8.2. Some prefix lengths are omitted in this figure, since this resulted clearer plots. All confusion matrices and the corresponding accuracy, error rate, precision, recall and specificity are shown in Appendix B in Tables B.1 to B.6.

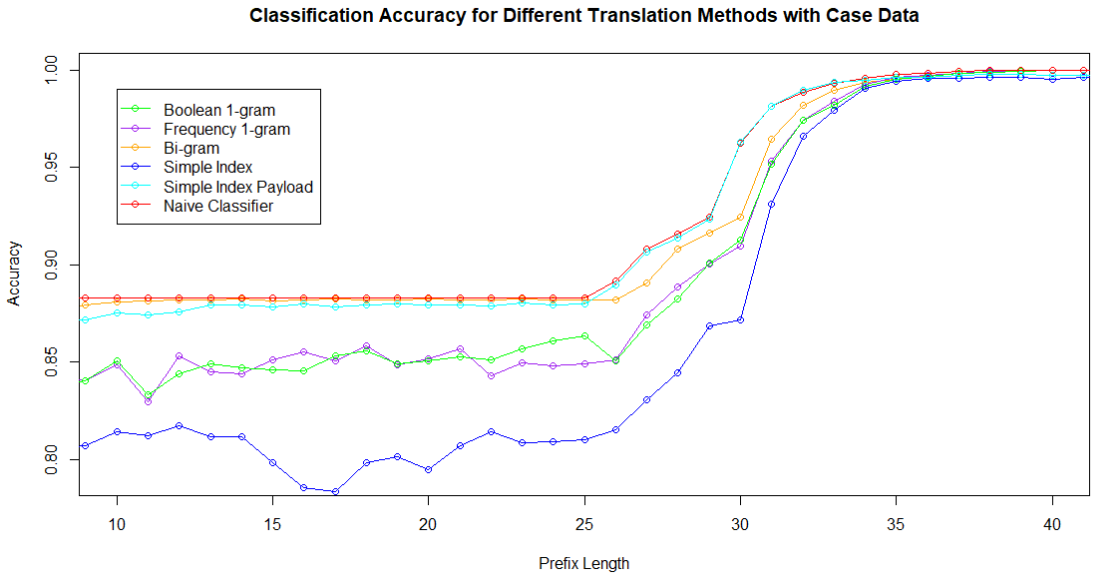

Figure 8.2: Classification Accuracy for Different Translation Methods with Case Data

Results show that the naive classifier has the highest accuracy overall. The only exceptions occur at prefixes 30 to 33, where the simple index method with payload performs slightly better or the same as the naive classifier in terms of classification accuracy. All translation methods converge to an accuracy of 1, however the simple index method and simple index method with payload never reach the upper limit of an accuracy of 1. Up until a prefix length of 30, the simple index method has a considerably lower accuracy than all other methods.

8.2.3 Classification Accuracy of the Different Translation Methods Without Case Data

This subsection presents the results of all random forests which trained on attributes that result from the five different methods to translate an event log into attributes. These random forests are only trained and tested on the attributes that are derived from the event log, meaning that case data is not included. Given the description in the previous section, the Boolean 1-gram method results in a maximum of 55 attributes. The frequency 1-gram method results in a maximum of 63 attributes which are used for the training and testing of models. When using the Bi-gram method, of the 4096 possible combinations, only 339 of those combinations occur in the event log. The last two methods for translating event logs into attributes involve the simple index method. This method is divided into two types, one which only includes activities and one which also includes payload. Without payload, the translated attributes add up to a maximum of 53 attributes. When including payload with the simple index method, this results in a maximum of 429 attributes.

When using the attributes as data for training, this results in one random forest for each prefix and each translation method. The classification accuracy of those random forests on the test set

is summarized in Table 8.4. Prefix lengths of 49 or more are omitted in this table, since the prior nine prefix lengths show similar results. The classification accuracy of the models is also plotted in Figure 8.3. In this figure Some prefix lengths are omitted, since the values did not change much in those ranges. The results are shown more elaborately in Tables C.1 to C.5, in Appendix C. The last column refers to the naive classifier that is implemented. The results of the naive classifier are the same as in the previous section, since the naive classifier only makes use of the event log and ignores the case log. The confusion matrix and corresponding evaluation measures can be found in Table B.1 in Appendix B.

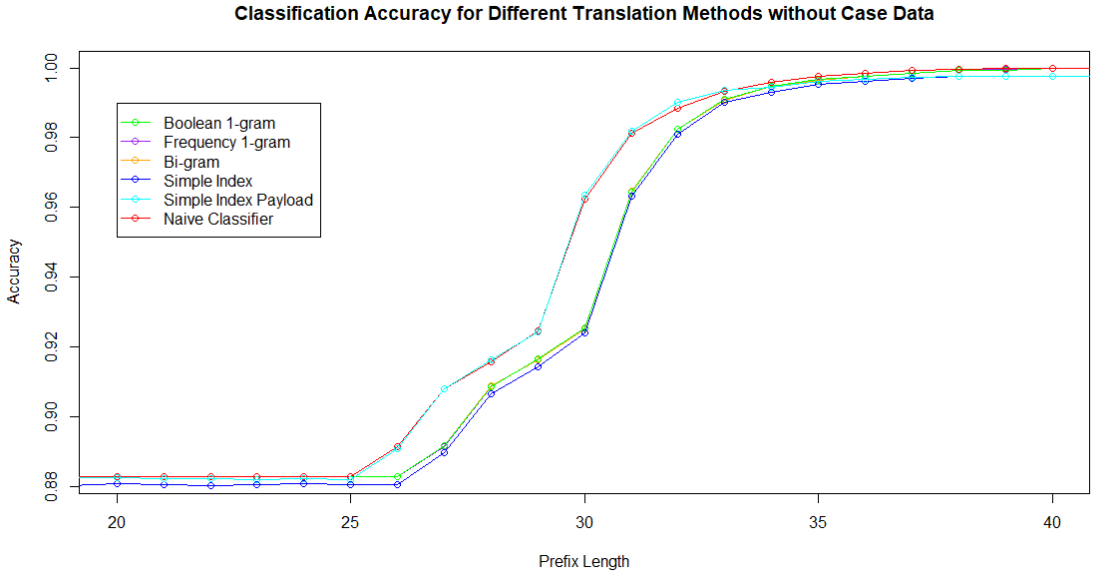

Figure 8.3: Classification Accuracy for Different Translation Methods without Case Data

The resulting classification accuracy of random forests is different when excluding case data. All translation methods have an accuracy which is close to equal of the classification achieved accuracy using the naive classifier. For prefix lengths 2 to 25 and 34 to 54, the maximum difference in accuracy amounts to 0.003. This difference in accuracy amounts to a difference of about 33 classification errors over a total of 11043 cases. There are slight deviations from this statement; there are more substantial differences in the range of prefix lengths 26 to 33. Here, there are cases where the random forest is trained on the attributes resulting from the simple index method with payload outperform the naive classifier. Also, in this range the simple index method without the inclusion of payload has the overall worst accuracy. Both 1-gram methods and the Bi-gram method perform similarly at the prefix lengths of 26 to 33, with a maximum difference in accuracy of 0.001. Note that in this range the reworks occur, since the naive classifiers' accuracy increases from prefix lengths 26 to 38.

Table 8.3: Summary of Classification Accuracy of the Different Translation Methods With Case Data

Prefix Length	1-Gram Boolean Accuracy	1-Gram Frequency Accuracy	Bi-Gram Accuracy	Simple Index Accuracy	Simple Index With Payload Accuracy	Naive Classifier Accuracy
1	0.821	0.821	0.820	0.821	0.821	**0.883**
2	0.824	0.833	0.834	0.795	0.840	**0.883**
3	0.833	0.840	0.850	0.782	0.843	**0.883**
4	0.828	0.827	0.876	0.791	0.849	**0.883**
5	0.834	0.840	0.876	0.806	0.857	**0.883**
6	0.836	0.837	0.874	0.813	0.867	**0.883**
7	0.829	0.829	0.878	0.809	0.869	**0.883**
8	0.837	0.843	0.875	0.806	0.871	**0.883**
9	0.840	0.840	0.879	0.807	0.872	**0.883**
10	0.850	0.849	0.881	0.814	0.875	**0.883**
11	0.833	0.830	0.881	0.812	0.874	**0.883**
12	0.844	0.853	0.882	0.817	0.876	**0.883**
13	0.849	0.845	0.882	0.812	0.879	**0.883**
14	0.847	0.844	0.882	0.811	0.879	**0.883**
15	0.846	0.851	0.881	0.798	0.878	**0.883**
16	0.846	0.855	0.882	0.785	0.880	**0.883**
17	0.853	0.850	0.882	0.783	0.878	**0.883**
18	0.856	0.858	0.882	0.798	0.880	**0.883**
19	0.849	0.848	0.882	0.801	0.880	**0.883**
20	0.851	0.852	0.882	0.795	0.879	**0.883**
21	0.853	0.857	0.882	0.807	0.879	**0.883**
22	0.851	0.843	0.882	0.814	0.879	**0.883**
23	0.857	0.850	0.883	0.809	0.880	**0.883**
24	0.861	0.848	0.882	0.809	0.879	**0.883**
25	0.864	0.849	0.882	0.810	0.880	**0.883**
26	0.850	0.851	0.882	0.815	0.889	**0.891**
27	0.869	0.874	0.891	0.831	0.906	**0.908**
28	0.882	0.889	0.908	0.845	0.914	**0.916**
29	0.901	0.900	0.916	0.869	0.923	**0.925**
30	0.913	0.910	0.925	0.872	**0.963**	0.962
31	0.952	0.953	0.964	0.931	**0.981**	**0.981**
32	0.974	0.974	0.982	0.966	**0.990**	0.988
33	0.982	0.984	0.990	0.979	**0.993**	**0.993**
34	0.991	0.992	0.994	0.991	0.995	**0.996**
35	0.995	0.996	0.995	0.994	0.996	**0.998**
36	0.996	0.997	0.997	0.996	0.996	**0.998**
37	0.998	0.998	0.998	0.996	0.997	**0.999**
38	0.999	0.999	0.999	0.996	0.998	**1.000**
39	0.999	**1.000**	0.999	0.996	0.998	**1.000**
40	**1.000**	**1.000**	**1.000**	0.995	0.997	**1.000**
41	**1.000**	**1.000**	**1.000**	0.996	0.997	**1.000**
42	**1.000**	**1.000**	0.999	0.996	0.998	**1.000**
43	**1.000**	**1.000**	**1.000**	0.996	0.997	**1.000**
44	**1.000**	**1.000**	**1.000**	0.996	0.997	**1.000**
45	**1.000**	**1.000**	**1.000**	0.996	0.997	**1.000**
46	**1.000**	**1.000**	**1.000**	0.996	0.997	**1.000**
47	**1.000**	**1.000**	0.999	0.996	0.998	**1.000**
48	**1.000**	**1.000**	**1.000**	0.997	0.997	**1.000**

Table 8.4: Summary of Classification Accuracy of the Different Translation Methods Without Case Data

Prefix Length	1-Gram Boolean Accuracy	1-Gram Frequency Accuracy	Bi-Gram Accuracy	Simple Index Accuracy	Simple Index With Payload Accuracy	Naive Classifier Accuracy
1					0.883	0.883
2	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**
3	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**
4	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**
5	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**
6	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**
7	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**
8	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**
9	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**
10	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**	**0.883**
11	**0.883**	**0.883**	**0.883**	0.882	**0.883**	**0.883**
12	**0.883**	**0.883**	**0.883**	0.882	**0.883**	**0.883**
13	**0.883**	**0.883**	**0.883**	**0.883**	0.882	**0.883**
14	**0.883**	**0.883**	**0.883**	0.882	**0.883**	**0.883**
15	**0.883**	**0.883**	**0.883**	0.882	0.882	**0.883**
16	**0.883**	0.882	**0.883**	0.882	0.882	**0.883**
17	**0.883**	**0.883**	**0.883**	0.881	0.882	**0.883**
18	**0.883**	**0.883**	**0.883**	0.880	0.882	**0.883**
19	**0.883**	**0.883**	**0.883**	0.880	0.882	**0.883**
20	**0.883**	**0.883**	**0.883**	0.881	0.882	**0.883**
21	**0.883**	**0.883**	**0.883**	0.880	0.882	**0.883**
22	**0.883**	**0.883**	**0.883**	0.880	0.882	**0.883**
23	**0.883**	**0.883**	**0.883**	0.880	0.882	**0.883**
24	**0.883**	**0.883**	**0.883**	0.881	0.882	**0.883**
25	**0.883**	**0.883**	**0.883**	0.880	0.882	**0.883**
26	0.883	0.883	0.883	0.880	**0.891**	**0.891**
27	0.892	0.892	0.892	0.890	**0.908**	**0.908**
28	0.909	0.909	0.909	0.906	**0.916**	**0.916**
29	0.917	0.917	0.916	0.914	0.924	**0.925**
30	0.925	0.925	0.925	0.924	**0.964**	0.962
31	0.964	0.964	0.965	0.963	**0.982**	0.981
32	0.983	0.982	0.982	0.981	**0.990**	0.988
33	0.991	0.991	0.991	0.990	**0.994**	0.993
34	0.995	0.995	0.995	0.993	0.995	**0.996**
35	0.997	0.997	0.996	0.995	0.996	**0.998**
36	**0.998**	**0.998**	**0.998**	0.996	0.997	**0.998**
37	0.998	0.998	**0.999**	0.997	0.997	**0.999**
38	0.999	0.999	0.999	0.997	0.998	**1.000**
39	0.999	**1.000**	0.999	0.998	0.998	**1.000**
40	**1.000**	**1.000**	**1.000**	0.998	0.998	**1.000**
41	**1.000**	**1.000**	**1.000**	0.998	0.998	**1.000**
42	**1.000**	**1.000**	**1.000**	0.998	0.998	**1.000**
43	**1.000**	**1.000**	**1.000**	0.997	0.998	**1.000**
44	**1.000**	**1.000**	**1.000**	0.998	0.997	**1.000**
45	**1.000**	**1.000**	**1.000**	0.998	0.998	**1.000**
46	**1.000**	**1.000**	**1.000**	0.997	0.997	**1.000**
47	**1.000**	**1.000**	**1.000**	0.997	0.997	**1.000**
48	**1.000**	**1.000**	**1.000**	0.998	0.998	**1.000**

8.3 Discussion

In this section we discuss the results that are presented in the previous section. Here, the results of each translation method are discussed separately and are each compared to the naive classifier. An important result from the naive classifier is that reworks occur at indexes 26 to 38, for the other indexes there is no rework. First we discuss the 1-gram methods, starting with the Boolean method. Next, the Bi-gram translation method results are discussed, and lastly, the simple index methods are discussed. Finally, the discussion section concludes with a general comparison between the use of the case log and the event log.

Translating event logs into attributes using the Boolean 1-gram method is the simplest method. It results in the lowest amount of attributes and the lowest values of those attributes. Despite its simplicity, this method does not have the lowest overall accuracy. In most cases, the simple index method without payload performs worse than the Boolean 1-gram method. This method performs similarly to the frequency 1-gram. When including the case log into the training of random forests, the classification accuracy sometimes both outperform the other without any clear structure. Without case data, both methods perform much more similar. In Table 8.4, there are only two lengths of prefixes for which the frequency method has an accuracy which is only 0.001 higher. Indicating that, with this data, both methods have a similar performance.

The Bi-gram translation method does not perform similar to any of the other translation methods. When including case data, the classification accuracy of the random forests is close to that of the naive classifier when using the Bi-gram method. Without case data, this method performs almost equal to the naive classifier in terms of accuracy. The only exception is the range of indexes in which rework occurs, where the simple index method with payload and the naive classifier perform better. From this, it can be concluded that the Bi-gram method produces more attributes which are related to the process and thus making it less sensitive to attributes that are not related to rework.

When translating the event log with the simple-index method, it results in the worst models in terms of classification accuracy. It scores lower than all other methods, including the naive classifier for most prefixes. There are some cases in which it performs similar to the other methods. This similarity occurs for the first 25 prefixes where the case data is excluded, where the most substantial difference in accuracy equals 0.003. When using the same method, but including payload, the results are different. This method has a range in which it slightly outperforms the naive classifier. The difference can be explained. One of the event attributes holds information on the quality of the product. With a specific value, it indicates that rework is performed next. Outside of this range, it performs similar to the other translation methods. With the inclusion of case data, the simple index method with payload results in one of the better random forests. Only the models that are trained on attributes from the Bi-gram method have a higher accuracy up until the prefixes that contain the rework activity.

Overall it shows that the naive classifier outperforms the models that are trained on the attributes which result from the different translation methods. The only exception being the simple index method where payload is included. This difference is explained by one of the event attributes, which holds information about the next activity. The inclusion of event data does increase accuracy. However, even higher accuracy can be achieved by ignoring the case data. This indicates that the information in the case log does not contain a relation with rework.

8.4 Conclusion

From the differences in terms of accuracy between all of the models, it becomes clear that the trained random forest models experience issues with categorical attributes with a large number of categories. This conclusion can be drawn, since the inclusion of case data results in lower accuracy and since the simple index methods never reach a point in which the classifications are perfect. The models resulting from the other three translation methods and the naive classifier do converge to an accuracy of 1. Another conclusion that can be drawn from the decrease in accuracy when including case data is the insignificance of the case data. Since these attributes result in lower accuracy, it is concluded that these attributes have no relation with rework. The only positive side of the inclusion of case data results in more true positives, which is displayed in Appendix B. However, these true positives are much lower than the number of false positives which indicates that these classifications are not the result of a relation between case attributes and rework.

Another important notice is the difference of accuracy between the naive classifier and most other trained models. It has the highest accuracy in most cases, the only exception being the models resulting from the simple index method translation method were payload is included. This difference can be explained by one of the event attributes containing information on the next activity. It can be concluded that there is no relation between all but one of the attributes, both from the case log and the event log. Continuing this conclusion, the used data was not useful for proving the usefulness of including event data in the quality prediction process.

Chapter 9

Conclusion & Future research

In this thesis, we investigated the possibility of adding information contained in event logs as input for classification models. This investigation resulted in three research questions, namely: "How can event logs be translated into features for prediction?", "What prediction method is most useful for predictions with event logs?" and "Do predictions become more accurate when including event log data in your prediction models?". In this final chapter, we state the main findings and provide suggestions for future research.

There are five methods for translating event logs into features for classification, which are described in this thesis. The 1-gram Boolean method records for all activities in a process, if the activity is present in a trace. The frequency 1-gram extends this method by counting how often each activity occurs. Another translation combines the presence of two subsequent activities into a 2-gram which is also called a Bi-gram [8]. This method determines for each combination of activities if it is present in a trace. The last method, called the simple index method, uses each event as a distinct set of attributes. Each activity maps onto one attribute. If event attributes are included in the translation, then each event attribute is mapped onto a separate attribute.

The best prediction method for this research is decision trees. Decision trees can process both categorical as numerical variables. This method is also the most transparent one; decision trees are simple to understand and interpret. In this thesis, we used random forests which are a set of decision trees. Random forests achieve higher accuracy and are immune to overfitting while preserving transparency.

Next, both the translation methods and prediction methods are used to predict rework in a trace. The resulting method provides a quick method to include event-logs in the prediction of rework. With the inclusion of event logs, there is more information available. This additional information improves predictions when process aspects and rework correlate.

A problem that arises when predicting the presence of a future event using event logs is that the activity to predict is present in the event log. Instead of working around this problem, it was chosen to adapt the evaluation method. A naive classifier was defined, which assumes that the rework does not occur. This classification changes as soon as the event log provides information that the rework happened. For this to work, rework should only be required for less than 50% of the cases. By defining this naive classifier, we established a baseline which can be used to evaluate classification models that result from translated event logs.

While validating the model, it showed that the results were as expected. The simple index with payload method performs best, since it can translate the structure, occurrence and frequency of activities into event logs. It also includes event attributes, where if there is a correlation with an attribute and the probability of rework, models trained on its resulting attributes are able to

find this relation.

Despite the promising results found during the model validation, only a few findings resulted from the case study. It was found that none of the data, in the event log nor in the case log, contained information which was relevant for the prediction of rework. The simple index method with payload only performed slightly better for a couple of prefix lengths, but it never resulted in perfect classifications. Perfect classifications are expected, since the translated attributes at some point state if the rework has already occurred. More straightforward methods, such as the 1-gram, did reach the perfect accuracy of 1. The choice of translation methods is dependent on the actual results. If none of the event attributes correlates with rework, then a simpler method should be chosen.

9.1 Future Research

The results of this thesis show that the translation of event logs has yet to prove its value in automated production environments. In previous literature, there was only one article [19] which used real event logs to prove its usefulness. Other articles only used artificially created event logs to test their methods of translation [8][24]. Even though this proves its possible usefulness, it would be more promising if its results were shown with an event log originating from an actual process. A possible cause is the lack of structured event logs. There are only a few event logs made available publicly, of which none of these are sequential processes.

Only a few articles translate event sequences, which is also done in this thesis. Besides, this thesis introduces a new problem where running cases are used to predict the occurrence of an event. It is attempted to prove usefulness with a case study, but the data did not contain any relation with rework, which opposed this attempt. This problem could prove itself in other environments as well. Additional case studies could prove the added value of the event log data in predicting activity occurrences. When the value of event logs in predictions is proven, a comparison can be made between the different translation methods. Each method has its advantages and disadvantages; thus an article describing these differences is useful.

One other point which was not mentioned in this thesis, is the assumed independence of cases. Assuming cases are independent makes it easier to clean data, since each case can be removed without having a consequence. During the case study, there were multiple cases which are not used due to incompleteness, which makes it challenging to investigate the dependence of cases. In reality, cases are likely to be dependent, especially in a sequential automated production environment. For example, it is likely that if a machine malfunctions, there are multiple consecutive flawed products. Due to the lack of clean event logs from production processes, it is challenging to analyze this expectation.

Bibliography

[1] Smart industry assistant. https://brightcape.nl/smart-industry-assistant. Accessed: 7-2-2020. iii, 1

[2] Aijun An and Nick Cercone. Discretization of continuous attributes for learning classification rules. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 509–514. Springer, 1999. 19

[3] Christopher M Bishop. *Pattern recognition and machine learning*. Springer Science+ Business Media, 2006. 8

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 19, 32

[5] Michelangelo Ceci, Pasqua Fabiana Lanotte, Fabio Fumarola, Dario Pietro Cavallo, and Donato Malerba. Completion time and next activity prediction of processes using sequential pattern mining. In *International Conference on Discovery Science*, pages 49–61. Springer, 2014. 12

[6] RS Chen, Yeh-Cheng Chen, and CC Chen. Using data mining technology to deign an quality control system for manufacturing industry. In *Proceedings of European conference on Computer science*, pages 272–276, 2010. 11

[7] Wei-Chou Chen, Shian-Shyong Tseng, and Ching-Yao Wang. A novel manufacturing defect detection method using association rule mining techniques. *Expert systems with applications*, 29(4):807–815, 2005. 11

[8] Catherine Da Cunha, Bruno Agard, and Andrew Kusiak. Data mining for improvement of product quality. *International journal of production research*, 44(18-19):4027–4041, 2006. 11, 12, 14, 19, 41, 42

[9] Massimiliano De Leoni, Wil MP Van der Aalst, and Marcus Dees. A general framework for correlating business process characteristics. In *International Conference on Business Process Management*, pages 250–266. Springer, 2014. 11

[10] Guozhu Dong and Jian Pei. Classification, clustering, features and distances of sequence data. In *Sequence data mining*, volume 33, chapter 10, pages 47–65. Springer Science & Business Media, West Lafayette, 2007. 11, 12, 14

[11] Chang-Xue Jack Feng and Xian-Feng Wang. Data mining techniques applied to predictive modeling of the knurling process. *Iie Transactions*, 36(3):253–263, 2004. 11

[12] Moises Goldszmidt. Finding soon-to-fail disks in a haystack. In *HotStorage*, 2012. 11

[13] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007. ix, xi, 17, 18, 19, 20

[14] Miroslav Kubat. *An introduction to machine learning*, volume 2. Springer, 2017. 8

[15] Andrew Kusiak and Christian Kurasek. Data mining of printed-circuit board defects. *IEEE transactions on robotics and automation*, 17(2):191–196, 2001. 11

[16] Heiner Lasi, Hans-Georg Kemper, Peter Fettke, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & information systems engineering*, 6(4):239–242, 2014. 1

[17] Mark Last and Abraham Kandel. Data mining for process and quality control in the semiconductor industry. In *Data mining for design and manufacturing*, pages 207–234. Springer, 2001. 11

[18] CKH Lee, GTS Ho, KL Choy, and GKH Pang. A rfid-based recursive process mining system for quality assurance in the garment industry. *International Journal of Production Research*, 52(14):4216–4238, 2014. 11

[19] Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maria Maggi. Complex symbolic sequence encodings for predictive monitoring of business processes. In *International Conference on Business Process Management*, pages 297–313. Springer, 2016. 11, 12, 15, 16, 19, 42

[20] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002. 26, 27, 31

[21] Fabrizio Maria Maggi, Chiara Di Francescomarino, Marlon Dumas, and Chiara Ghidini. Predictive monitoring of business processes. In *International conference on advanced information systems engineering*, pages 457–472. Springer, 2014. 11

[22] Hideyuki Maki and Yuko Teranishi. Development of automated data mining system for quality control in manufacturing. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 93–100. Springer, 2001. 12

[23] Isidore Rigoutsos and Aris Floratos. Combinatorial pattern discovery in biological sequences: The teiresias algorithm. *Bioinformatics (Oxford, England)*, 14(1):55–67, 1998. 12

[24] Lior Rokach, Roni Romano, and Oded Maimon. Mining manufacturing databases to discover the effect of operation sequence on the product quality. *Journal of Intelligent manufacturing*, 19(3):313–325, 2008. 11, 12, 19, 42

[25] RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA, 2015. 26, 31

[26] SA Russell, P Kesavan, JayHyung Lee, and BA Ogunnaike. Recursive data-based prediction and control of batch product quality. *AIChE Journal*, 44(11):2442–2458, 1998. 11

[27] Eliana Salvemini, Fabio Fumarola, Donato Malerba, and Jiawei Han. Fast sequence mining based on sparse id-lists. In *International Symposium on Methodologies for Intelligent Systems*, pages 316–325. Springer, 2011. 12, 16

[28] Suriadi Suriadi, Chun Ouyang, Wil MP van der Aalst, and Arthur HM ter Hofstede. Root cause analysis with enriched process logs. In *International Conference on Business Process Management*, pages 174–186. Springer, 2012. 15

[29] Wil Van Der Aalst. *Process mining: discovery, conformance and enhancement of business processes*, volume 2. Springer, 2011. ix, 5, 6

[30] Wil Van Der Aalst, Arya Adriansyah, Ana Karla Alves De Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter Van Den Brand, Ronald Brandtjen, Joos Buijs, Andrea Burattin, Josep Carmona, Malu Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano De Leoni, Pavlos Delias, Boudewijn F. Van Dongen, Marlon Dumas, Schahram Dustdar, Dirk Fahland, Diogo R.

Ferreira, Walid Gaaloul, Frank Van Geffen, Sukriti Goel, Christian Günther, Antonella Guzzo, Paul Harmon, Arthur Ter Hofstede, John Hoogland, Jon Espen Ingvaldsen, Koki Kato, Rudolf Kuhn, Akhil Kumar, Marcello La Rosa, Fabrizio Maggi, Donato Malerba, Ronny S. Mans, Alberto Manuel, Martin McCreesh, Paola Mello, Jan Mendling, Marco Montali, Hamid R. Motahari-Nezhad, Michael Zur Muehlen, Jorge Munoz-Gama, Luigi Pontieri, Joel Ribeiro, Anne Rozinat, Hugo Seguel Pérez, Ricardo Seguel Pérez, Marcos Sepúlveda, Jim Sinur, Pnina Soffer, Minseok Song, Alessandro Sperduti, Giovanni Stilo, Casper Stoel, Keith Swenson, Maurizio Talamo, Wei Tan, Chris Turner, Jan Vanthienen, George Varvaressos, Eric Verbeek, Marc Verdonk, Roberto Vigo, Jianmin Wang, Barbara Weber, Matthias Weidlich, Ton Weijters, Lijie Wen, Michael Westergaard, and Moe Wynn. Process mining manifesto. *Lecture Notes in Business Information Processing*, 99 LNBIP(PART 1):169–194, 2012. 2, 5, 6, 8

[31] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. Dongen, van, and W.M.P. Aalst, van der. Prom 6 : the process mining toolkit. In M. La Rosa, editor, *Proceedings of the Business Process Management 2010 Demonstration Track (Hoboken NJ, USA, September 14-16, 2010)*, CEUR Workshop Proceedings, pages 34–39. CEUR-WS.org, 2010. 31

Appendix A

R Scripts

A.1 Event Log Generation

```
#Define Number of Cases & Set seed
NumberOfCases <- 10000
Name_Activity <- "Rework"

# Generate Event Log ====
# Name Columns
EventLog <- data.frame(matrix(ncol = 4))
colnames(EventLog) <- c("CaseID", "Activity", "Index", "Duration")

# Fill in rows
Case <- 1
Row <- 1

while(Case < NumberOfCases + 1){
  # Reset Parameters
  Phenom1 <- FALSE
  Phenom2 <- FALSE
  Phenom3 <- FALSE
  Phenom4 <- FALSE
  Index <- 1

  # Start Activity
  EventLog[Row,] <- c(Case, "Start", Index, 0)
  Row <- Row + 1
  Index <- Index + 1

  # Section 1
  Section1 <- sample(c("A","B"), 2, replace = FALSE)
  if ((Section1[1] == "B")&(Section1[2] == "A")){
    Phenom1 <- TRUE
  }

  for(Activity in Section1){
    Time <- runif(1, 0, 100)
    if ((Activity == "B") & (Time > 50)){
      Phenom2 <- TRUE
    }
    EventLog[Row,] <- c(Case, Activity, Index, Time)
    Row <- Row + 1
    Index <- Index + 1
  }

  # Section 2
  prob <- runif(1, 0, 100)
  if (prob < 50){
    if (prob < 25){
```

```
      EventLog[Row,] <- c(Case, "A", Index, Time)
      Phenom3 <- TRUE
      Row <- Row + 1
      Index <- Index + 1
    } else {
      EventLog[Row,] <- c(Case, "C", Index, Time)
      Phenom4 <- TRUE
      Row <- Row + 1
      Index <- Index + 1
    }
  }

  # Section 3
  Section3 <- c()
  prob <- runif(1, 0, 100)
  if (prob < 50){
    Section3 <- c(Section3, "D")
  }
  prob <- runif(1, 0, 100)
  if (prob < 50){
    Section3 <- c(Section3, "E")
  }
  prob <- runif(1, 0, 100)
  if (prob < 50){
    Section3 <- c(Section3, "F")
  }
  prob_Rework <- (5 + (Phenom1 * 25)+ (Phenom2 * 25)+ (Phenom3 * 25)+ (Phenom4 *
      25))
  prob <- runif(1, 0, 100)
  if (prob < prob_Rework){
    Section3 <- c(Section3, "Rework")
  }
  for(Activity in Section3){
    EventLog[Row,] <- c(Case, Activity, Index, runif(1, 0, 100))
    Row <- Row + 1
    Index <- Index + 1
  }
  # End Activity
  EventLog[Row,] <- c(Case, "End", Index, 0)
  Row <- Row + 1
  Index <- Index + 1
  Case <- Case + 1
}
EventLog$Duration <- as.numeric(EventLog$Duration)

# Create CaseLog and Define Label
CaseLog <- data.frame(matrix(ncol = 3, nrow = NumberOfCases))
colnames(CaseLog) <- c("CaseID", "Length", "Label")
CaseLog$Label = FALSE

for (i in 1:NumberOfCases){
  CaseLog$CaseID[i] <- i
  Trace <- EventLog[(EventLog[,1] == CaseLog$CaseID[i]),]
  CaseLog$Length[i] <- NROW(Trace)
  if (any(Trace$Activity == Name_Activity)){
    CaseLog$Label[i] <- TRUE
  }
}

rm(Trace,Activity,Case,i,Index,NumberOfCases,Phenom1,Phenom2,Phenom3,Phenom4,prob,
    prob_Rework,Row,Section1,Section3,Time)
save.image(file = "Data.RData")
```

A.2 Boolean 1-Gram Implementation

```
load("Data.RData")
Prefix_MaxLength <- max(CaseLog$Length)

# Make Empty Datasets (1 Per Prefix Length) #
Activities <- as.character(unique(EventLog$Activity)) #list of different Activities
EmptyDataset <- data.frame(matrix(nrow = NROW(CaseLog),ncol = length(Activities)))
colnames(EmptyDataset) <- Activities
EmptyDataset[,] <- FALSE

for (i in 1:Prefix_MaxLength){
  Name_Dataset <- paste("Data_Prefix",i, sep = "_")
  assign(Name_Dataset, EmptyDataset)
}
rm(EmptyDataset)

# Determine Values #
for(i in 1:NROW(CaseLog)){ #Loop per case
  Trace <- EventLog[(EventLog$CaseID == CaseLog$CaseID[i]),]
  for (ii in 1:NROW(Trace)){ #Loop per index
    Name_Dataset <- paste("Data_Prefix",ii, sep = "_")
    Add_Data <- get(Name_Dataset)
    for (iii in 1:length(Activities)){ #Check per activity
      if (any(as.character(Trace$Activity[1:ii]) == Activities[iii])){
        Add_Data[i,iii] <- TRUE
      }
    }
    assign(Name_Dataset, Add_Data)
  }
  # When a trace length is shorter than the maximum length, the data remains the
      same for those prefixes
  if (NROW(Trace) < Prefix_MaxLength){
    Name_Last <- paste("Data_Prefix",NROW(Trace), sep = "_")
    Data_Last <- get(Name_Last)
    Data_Last <- Data_Last[i,]
    for (ii in ((NROW(Trace)+1):Prefix_MaxLength)){
      Name_Dataset <- paste("Data_Prefix",ii, sep = "_")
      Add_Data <- get(Name_Dataset)
      Add_Data[i,] <- Data_Last
      assign(Name_Dataset, Add_Data)
    }
  }
}

rm(Activities, i, ii, iii, Name_Dataset,Name_Last,Data_Last,Add_Data, Trace, Prefix
    _MaxLength)
save.image("1Gram-Boolean.RData")
```

A.3 Frequency 1-Gram Implementation

```
load("Data.RData")
Prefix_MaxLength <- max(CaseLog$Length)

# Make Empty Datasets (1 Per Prefix Length) #
Activities <- as.character(unique(EventLog$Activity)) #list of different Activities
EmptyDataset <- data.frame(matrix(nrow = NROW(CaseLog),ncol = length(Activities)))
colnames(EmptyDataset) <- Activities
EmptyDataset[,] <- 0

for (i in 1:Prefix_MaxLength){
  Name_Dataset <- paste("Data_Prefix",i, sep = "_")
  assign(Name_Dataset, EmptyDataset)
}
rm(EmptyDataset)

# Determine Values #
for(i in 1:NROW(CaseLog)){ #Loop per case
  Trace <- EventLog[(EventLog$CaseID == CaseLog$CaseID[i]),]
  for (ii in 1:NROW(Trace)){ #Loop per index
    Name_Dataset <- paste("Data_Prefix",ii, sep = "_")
    Add_Data <- get(Name_Dataset)
    for (iii in 1:length(Activities)){ #Check per activity
      if (any(as.character(Trace$Activity[1:ii]) == Activities[iii])){
        Add_Data[i,iii] <- length(which(as.character(Trace$Activity[1:ii]) ==
            Activities[iii]))
      }
    }
    assign(Name_Dataset, Add_Data)
  }
  # When a trace length is shorter than the maximum length, the data remains the
      same for those prefixes
  if (NROW(Trace) < Prefix_MaxLength){
    Name_Last <- paste("Data_Prefix",NROW(Trace), sep = "_")
    Data_Last <- get(Name_Last)
    Data_Last <- Data_Last[i,]
    for (ii in ((NROW(Trace)+1):Prefix_MaxLength)){
      Name_Dataset <- paste("Data_Prefix",ii, sep = "_")
      Add_Data <- get(Name_Dataset)
      Add_Data[i,] <- Data_Last
      assign(Name_Dataset, Add_Data)
    }
  }
}

rm(Activities, i, ii, iii, Name_Dataset,Name_Last,Data_Last,Add_Data, Trace, Prefix
    _MaxLength)
save.image("1Gram-Frequency.RData")
```

A.4 Bi-Gram Implementation

```
load("Data.RData")
Prefix_MaxLength <- max(CaseLog$Length)

Activities <- unique(as.character(EventLog$Activity))
Values <- array(dim = c(NROW(CaseLog),Prefix_MaxLength))
Activities_NO <- length(Activities)

for(i in 1:NROW(CaseLog)){
  Trace <- EventLog[(EventLog[,1] == CaseLog[i,1]),]
  for (ii in 2:NROW(Trace)){
    Activity1 <- which(Activities == as.character(Trace$Activity[ii-1]))
    Activity2 <- which(Activities == as.character(Trace$Activity[ii]))
    Values[i,ii] <- (Activities_NO*(Activity1-1)+Activity2)
  }
}

#Create Empty Dataset
Add_Columns <- data.frame(matrix(nrow = NROW(CaseLog),ncol = Activities_NO*
    Activities_NO))
Add_Columns[] <- FALSE
ColNames <- c()
for (i in 1:length(Activities)){
  for (ii in 1:length(Activities)){
    ColNames <- c(ColNames,paste(Activities[i], Activities[ii], sep = "_"))
  }
}
colnames(Add_Columns) <- ColNames

# Assign Data Per Prefix #
for(i in 2:Prefix_MaxLength){
  for (ii in 1:NROW(CaseLog)){
    x <- Values[ii,i]
    if (is.na(x)){
      next()
    }
    Add_Columns[ii,x] <- TRUE
  }
  Name_Dataset <- paste("Data_Prefix",i, sep = "_")
  assign(Name_Dataset,Add_Columns)
}
rm(x,Prefix_MaxLength,Name_Dataset,i,ii,ColNames,Activity1,Activity2,Activities_NO,
    Activities,Values,Trace,Add_Columns)
save.image(file = "BiGram.RData")
```

A.5 Simple Index Implementation

```
load("Data.RData")
Prefix_MaxLength <- max(CaseLog$Length)

# Create Empty Dataset #
Add_Columns <- data.frame(matrix(nrow = NROW(CaseLog),ncol = max(CaseLog$Length)))
names <- c()
for (i in 1:max(CaseLog$Length)){
  names <- c(names,paste("Event",i,sep = "_"))
}
colnames(Add_Columns) <- names
rm(names)

#Fill In Data Values #
for (i in 1:NROW(CaseLog)){
  Trace <- EventLog[(EventLog$CaseID == CaseLog$CaseID[i]),]
  dif <- NCOL(Add_Columns)-NROW(Trace)
  Add_Columns[i,] <- c(as.character(Trace$Activity),rep("None", dif))
}

# Change Column Type to factor #
for (i in 1:NCOL(Add_Columns)){
  Add_Columns[,i] <- as.factor(Add_Columns[,i])
}
# Divide Complete Set Per prefix
for (i in 1:Prefix_MaxLength){
  Name_Data <- paste("Data_Prefix",i, sep = "_")
  assign(Name_Data,Add_Columns[,(1:i)])
}
rm(Prefix_MaxLength,Name_Data,i,dif,Add_Columns)
save.image("SimpleIndex.RData")
```

A.6 Simple Index With Payload Implementation

```
load("Data.RData")
Prefix_MaxLength <- max(CaseLog$Length)
IgnoreList <- c("CaseID", "Index") #Payload that shouldnt be converted into
    attributes

# Create Empty Dataframe #
Add_Columns <- data.frame(matrix(nrow = NROW(CaseLog), ncol = max(CaseLog$Length) *
     (NCOL(EventLog) - length(IgnoreList))))
nameslist <- c()
names <- names(EventLog)
names <- names[-which(names %in% IgnoreList)]
for (i in 1:max(CaseLog$Length)){
  nameslist <- c(nameslist, paste(names, i, sep = "_"))
}
colnames(Add_Columns) <- nameslist

EventLog_filtered <- EventLog[,-which(names(EventLog) %in% IgnoreList)]

#Create Empty Activity (To fill columns that are finished Earlier)#
EmptyActivity <- EventLog_filtered[1,]
for (i in 1:NCOL(EmptyActivity)){
  if(class(EmptyActivity[,i]) == "character"){
    EmptyActivity[1,i] <- "None"
  } else {
    EmptyActivity[1,i] <- 0
  }
}

for (i in 1:NROW(CaseLog)){
  Trace <- EventLog_filtered[(EventLog$CaseID == CaseLog$CaseID[i]),]
  for (ii in 1:Prefix_MaxLength){
    if (ii < NROW(Trace)+1){
      Add_Columns[i,(((ii-1)*NCOL(EventLog_filtered)+1):(NCOL(EventLog_filtered)*ii
          ))] <- Trace[ii,]
    } else {
      Add_Columns[i,(((ii-1)*NCOL(EventLog_filtered)+1):(NCOL(EventLog_filtered)*ii
          ))] <- EmptyActivity
    }
  }
}
Add_Columns <- as.data.frame(unclass(Add_Columns)) #change Strings to Factors
for (i in 1:max(CaseLog$Length)){
  Name_Data <- paste("Data_Prefix",i, sep = "_")
  Data <- Add_Columns[,1:(i*NCOL(EventLog_filtered))]
  assign(Name_Data,Data)
}
rm(Add_Columns, Data, EmptyActivity, EventLog_filtered, Trace, i, IgnoreList, ii,
    Name_Data,names,nameslist,Prefix_MaxLength)
save.image("SimpIndexPayload.RData")
```

A.7 Naive Classifier Implementation

```
load("Data.RData")
Prefix_MaxLength <- max(CaseLog$Length)
NaiveClassifier <- data.frame(matrix(ncol = (max(CaseLog$Length)),nrow = NROW(
    CaseLog)))
Name_Activity <- "Rework"

# Define Column Names
names <- c()
for (i in 1:max(CaseLog$Length)){
  names <- c(names,paste("Class",i,sep = ""))
}
colnames(NaiveClassifier) <- names

# Determine Values
for (i in 1:NROW(NaiveClassifier)){
  Trace <- EventLog[(EventLog$CaseID == CaseLog$CaseID[i]),]
  if (any(Trace$Activity == Name_Activity)){
    IndexOfRework <- which(Trace$Activity == Name_Activity)
    NaiveClassifier[i,(1:(IndexOfRework[1]-1))] <- FALSE
    NaiveClassifier[i,(IndexOfRework[1]:NCOL(NaiveClassifier))] <- TRUE
  } else {
    NaiveClassifier[i,(1:NCOL(NaiveClassifier))] <- FALSE
  }
}
rm(Trace, i, IndexOfRework,Name_Activity,names,Prefix_MaxLength)
save.image(file = "NaiveClassifier.RData")
```

Appendix B

Classification Results of Random Forests with Case Data

The next pages of this appendix presents six different table. Five of those tables present the confusion matrices and corresponding evaluation measures for each random forest model that was trained using attributes from that translation method. This appendix also includes the confusion matrices for each prefix of the simple classifier, which is shown in table B.1 on page 56.

Table B.1: Classification Results of the Simple Classifier That Was Implemented

Prefix	TN	FP	FN	TP	Accuracy	Error Rate	Precision	Recall	Specificity
1	9750	0	1293	0	0.883	0.117		0.000	1.000
2	9750	0	1293	0	0.883	0.117		0.000	1.000
3	9750	0	1293	0	0.883	0.117		0.000	1.000
4	9750	0	1293	0	0.883	0.117		0.000	1.000
5	9750	0	1293	0	0.883	0.117		0.000	1.000
6	9750	0	1293	0	0.883	0.117		0.000	1.000
7	9750	0	1293	0	0.883	0.117		0.000	1.000
8	9750	0	1293	0	0.883	0.117		0.000	1.000
9	9750	0	1293	0	0.883	0.117		0.000	1.000
10	9750	0	1293	0	0.883	0.117		0.000	1.000
11	9750	0	1293	0	0.883	0.117		0.000	1.000
12	9750	0	1293	0	0.883	0.117		0.000	1.000
13	9750	0	1293	0	0.883	0.117		0.000	1.000
14	9750	0	1293	0	0.883	0.117		0.000	1.000
15	9750	0	1293	0	0.883	0.117		0.000	1.000
16	9750	0	1293	0	0.883	0.117		0.000	1.000
17	9750	0	1293	0	0.883	0.117		0.000	1.000
18	9750	0	1293	0	0.883	0.117		0.000	1.000
19	9750	0	1293	0	0.883	0.117		0.000	1.000
20	9750	0	1293	0	0.883	0.117		0.000	1.000
21	9750	0	1293	0	0.883	0.117		0.000	1.000
22	9750	0	1293	0	0.883	0.117		0.000	1.000
23	9750	0	1293	0	0.883	0.117		0.000	1.000
24	9750	0	1293	0	0.883	0.117		0.000	1.000
25	9750	0	1293	0	0.883	0.117		0.000	1.000
26	9750	0	1199	94	0.891	0.109	1.000	0.073	1.000
27	9750	0	1017	276	0.908	0.092	1.000	0.213	1.000
28	9750	0	932	361	0.916	0.084	1.000	0.279	1.000
29	9750	0	832	461	0.925	0.075	1.000	0.357	1.000
30	9750	0	416	877	0.962	0.038	1.000	0.678	1.000
31	9750	0	208	1085	0.981	0.019	1.000	0.839	1.000
32	9750	0	127	1166	0.988	0.012	1.000	0.902	1.000
33	9750	0	75	1218	0.993	0.007	1.000	0.942	1.000
34	9750	0	47	1246	0.996	0.004	1.000	0.964	1.000
35	9750	0	27	1266	0.998	0.002	1.000	0.979	1.000
36	9750	0	18	1275	0.998	0.002	1.000	0.986	1.000
37	9750	0	7	1286	0.999	0.001	1.000	0.995	1.000
38	9750	0	5	1288	1.000	0.000	1.000	0.996	1.000
39	9750	0	1	1292	1.000	0.000	1.000	0.999	1.000
40	9750	0	1	1292	1.000	0.000	1.000	0.999	1.000
41	9750	0	1	1292	1.000	0.000	1.000	0.999	1.000
42	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
43	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
44	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
45	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
46	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
47	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
48	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
49	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
50	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
51	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
52	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
53	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
54	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000

Table B.2: Classification Results of the Boolean 1-Gram Translation Method In Combination With Case Data

PrefixLength	TN	FP	FN	TP	Accuracy	Error Rate	Precision	Recall	Specificity
1	8935	815	1165	128	0.821	0.179	0.136	0.099	0.916
2	8972	778	1165	128	0.824	0.176	0.141	0.099	0.920
3	9095	655	1187	106	0.833	0.167	0.139	0.082	0.933
4	9013	737	1162	131	0.828	0.172	0.151	0.101	0.924
5	9104	646	1191	102	0.834	0.166	0.136	0.079	0.934
6	9127	623	1185	108	0.836	0.164	0.148	0.084	0.936
7	9038	712	1174	119	0.829	0.171	0.143	0.092	0.927
8	9142	608	1189	104	0.837	0.163	0.146	0.080	0.938
9	9185	565	1199	94	0.840	0.160	0.143	0.073	0.942
10	9318	432	1219	74	0.850	0.150	0.146	0.057	0.956
11	9085	665	1177	116	0.833	0.167	0.149	0.090	0.932
12	9233	517	1205	88	0.844	0.156	0.145	0.068	0.947
13	9296	454	1211	82	0.849	0.151	0.153	0.063	0.953
14	9267	483	1207	86	0.847	0.153	0.151	0.067	0.950
15	9258	492	1206	87	0.846	0.154	0.150	0.067	0.950
16	9261	489	1216	77	0.846	0.154	0.136	0.060	0.950
17	9339	411	1210	83	0.853	0.147	0.168	0.064	0.958
18	9377	373	1221	72	0.856	0.144	0.162	0.056	0.962
19	9308	442	1222	71	0.849	0.151	0.138	0.055	0.955
20	9307	443	1204	89	0.851	0.149	0.167	0.069	0.955
21	9332	418	1210	83	0.853	0.147	0.166	0.064	0.957
22	9324	426	1219	74	0.851	0.149	0.148	0.057	0.956
23	9390	360	1221	72	0.857	0.143	0.167	0.056	0.963
24	9450	300	1236	57	0.861	0.139	0.160	0.044	0.969
25	9482	268	1238	55	0.864	0.136	0.170	0.043	0.973
26	9318	432	1219	74	0.850	0.150	0.146	0.057	0.956
27	9434	316	1130	163	0.869	0.131	0.340	0.126	0.968
28	9416	334	964	329	0.882	0.118	0.496	0.254	0.966
29	9527	223	874	419	0.901	0.099	0.653	0.324	0.977
30	9589	161	802	491	0.913	0.087	0.753	0.380	0.983
31	9578	172	361	932	0.952	0.048	0.844	0.721	0.982
32	9633	117	171	1122	0.974	0.026	0.906	0.868	0.988
33	9630	120	78	1215	0.982	0.018	0.910	0.940	0.988
34	9699	51	44	1249	0.991	0.009	0.961	0.966	0.995
35	9716	34	22	1271	0.995	0.005	0.974	0.983	0.997
36	9727	23	16	1277	0.996	0.004	0.982	0.988	0.998
37	9738	12	10	1283	0.998	0.002	0.991	0.992	0.999
38	9744	6	6	1287	0.999	0.001	0.995	0.995	0.999
39	9747	3	4	1289	0.999	0.001	0.998	0.997	1.000
40	9747	3	1	1292	1.000	0.000	0.998	0.999	1.000
41	9749	1	1	1292	1.000	0.000	0.999	0.999	1.000
42	9749	1	1	1292	1.000	0.000	0.999	0.999	1.000
43	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
44	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
45	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
46	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
47	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
48	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
49	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
50	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
51	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
52	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
53	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
54	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000

Table B.3: Classification Results of the Frequency 1-Gram Translation Method In Combination With Case Data

PrefixLength	TN	FP	FN	TP	Accuracy	Error Rate	Precision	Recall	Specificity
1	8939	811	1165	128	0.821	0.179	0.136	0.099	0.917
2	9080	670	1170	123	0.833	0.167	0.155	0.095	0.931
3	9168	582	1188	105	0.840	0.160	0.153	0.081	0.940
4	9019	731	1177	116	0.827	0.173	0.137	0.090	0.925
5	9170	580	1190	103	0.840	0.160	0.151	0.080	0.941
6	9144	606	1196	97	0.837	0.163	0.138	0.075	0.938
7	9028	722	1171	122	0.829	0.171	0.145	0.094	0.926
8	9208	542	1190	103	0.843	0.157	0.160	0.080	0.944
9	9180	570	1192	101	0.840	0.160	0.151	0.078	0.942
10	9286	464	1209	84	0.849	0.151	0.153	0.065	0.952
11	9047	703	1178	115	0.830	0.170	0.141	0.089	0.928
12	9343	407	1214	79	0.853	0.147	0.163	0.061	0.958
13	9241	509	1202	91	0.845	0.155	0.152	0.070	0.948
14	9231	519	1202	91	0.844	0.156	0.149	0.070	0.947
15	9318	432	1210	83	0.851	0.149	0.161	0.064	0.956
16	9379	371	1229	64	0.855	0.145	0.147	0.049	0.962
17	9310	440	1211	82	0.850	0.150	0.157	0.063	0.955
18	9413	337	1228	65	0.858	0.142	0.162	0.050	0.965
19	9286	464	1211	82	0.848	0.152	0.150	0.063	0.952
20	9330	420	1219	74	0.852	0.148	0.150	0.057	0.957
21	9384	366	1217	76	0.857	0.143	0.172	0.059	0.962
22	9212	538	1194	99	0.843	0.157	0.155	0.077	0.945
23	9297	453	1207	86	0.850	0.150	0.160	0.067	0.954
24	9277	473	1207	86	0.848	0.152	0.154	0.067	0.951
25	9285	465	1203	90	0.849	0.151	0.162	0.070	0.952
26	9321	429	1213	80	0.851	0.149	0.157	0.062	0.956
27	9511	239	1148	145	0.874	0.126	0.378	0.112	0.975
28	9483	267	964	329	0.889	0.111	0.552	0.254	0.973
29	9526	224	878	415	0.900	0.100	0.649	0.321	0.977
30	9543	207	790	503	0.910	0.090	0.708	0.389	0.979
31	9599	151	368	925	0.953	0.047	0.860	0.715	0.985
32	9633	117	171	1122	0.974	0.026	0.906	0.868	0.988
33	9652	98	78	1215	0.984	0.016	0.925	0.940	0.990
34	9712	38	46	1247	0.992	0.008	0.970	0.964	0.996
35	9729	21	23	1270	0.996	0.004	0.984	0.982	0.998
36	9739	11	17	1276	0.997	0.003	0.991	0.987	0.999
37	9740	10	10	1283	0.998	0.002	0.992	0.992	0.999
38	9747	3	6	1287	0.999	0.001	0.998	0.995	1.000
39	9749	1	4	1289	1.000	0.000	0.999	0.997	1.000
40	9750	0	1	1292	1.000	0.000	1.000	0.999	1.000
41	9750	0	1	1292	1.000	0.000	1.000	0.999	1.000
42	9750	0	1	1292	1.000	0.000	1.000	0.999	1.000
43	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
44	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
45	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
46	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
47	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
48	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
49	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
50	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
51	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
52	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
53	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
54	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000

Table B.4: Classification Results of the Bi-gram Translation Method In Combination With Case Data

PrefixLength	TN	FP	FN	TP	Accuracy	Error Rate	Precision	Recall	Specificity
1									
2	9104	646	1182	111	0.834	0.166	0.147	0.086	0.934
3	9308	442	1212	81	0.850	0.150	0.155	0.063	0.955
4	9651	99	1270	23	0.876	0.124	0.189	0.018	0.990
5	9646	104	1264	29	0.876	0.124	0.218	0.022	0.989
6	9620	130	1259	34	0.874	0.126	0.207	0.026	0.987
7	9673	77	1270	23	0.878	0.122	0.230	0.018	0.992
8	9631	119	1261	32	0.875	0.125	0.212	0.025	0.988
9	9700	50	1281	12	0.879	0.121	0.194	0.009	0.995
10	9714	36	1281	12	0.881	0.119	0.250	0.009	0.996
11	9725	25	1287	6	0.881	0.119	0.194	0.005	0.997
12	9734	16	1286	7	0.882	0.118	0.304	0.005	0.998
13	9735	15	1288	5	0.882	0.118	0.250	0.004	0.998
14	9741	9	1291	2	0.882	0.118	0.182	0.002	0.999
15	9728	22	1287	6	0.881	0.119	0.214	0.005	0.998
16	9731	19	1288	5	0.882	0.118	0.208	0.004	0.998
17	9739	11	1290	3	0.882	0.118	0.214	0.002	0.999
18	9736	14	1288	5	0.882	0.118	0.263	0.004	0.999
19	9737	13	1289	4	0.882	0.118	0.235	0.003	0.999
20	9739	11	1288	5	0.882	0.118	0.312	0.004	0.999
21	9739	11	1291	2	0.882	0.118	0.154	0.002	0.999
22	9734	16	1287	6	0.882	0.118	0.273	0.005	0.998
23	9743	7	1289	4	0.883	0.117	0.364	0.003	0.999
24	9736	14	1288	5	0.882	0.118	0.263	0.004	0.999
25	9733	17	1290	3	0.882	0.118	0.150	0.002	0.998
26	9738	12	1290	3	0.882	0.118	0.200	0.002	0.999
27	9737	13	1193	100	0.891	0.109	0.885	0.077	0.999
28	9744	6	1008	285	0.908	0.092	0.979	0.220	0.999
29	9742	8	919	374	0.916	0.084	0.979	0.289	0.999
30	9746	4	828	465	0.925	0.075	0.991	0.360	1.000
31	9747	3	392	901	0.964	0.036	0.997	0.697	1.000
32	9738	12	191	1102	0.982	0.018	0.989	0.852	0.999
33	9732	18	96	1197	0.990	0.010	0.985	0.926	0.998
34	9730	20	50	1243	0.994	0.006	0.984	0.961	0.998
35	9730	20	34	1259	0.995	0.005	0.984	0.974	0.998
36	9741	9	20	1273	0.997	0.003	0.993	0.985	0.999
37	9741	9	11	1282	0.998	0.002	0.993	0.991	0.999
38	9743	7	7	1286	0.999	0.001	0.995	0.995	0.999
39	9745	5	5	1288	0.999	0.001	0.996	0.996	0.999
40	9746	4	1	1292	1.000	0.000	0.997	0.999	1.000
41	9749	1	1	1292	1.000	0.000	0.999	0.999	1.000
42	9745	5	1	1292	0.999	0.001	0.996	0.999	0.999
43	9749	1	0	1293	1.000	0.000	0.999	1.000	1.000
44	9748	2	1	1292	1.000	0.000	0.998	0.999	1.000
45	9747	3	0	1293	1.000	0.000	0.998	1.000	1.000
46	9747	3	0	1293	1.000	0.000	0.998	1.000	1.000
47	9744	6	0	1293	0.999	0.001	0.995	1.000	0.999
48	9747	3	0	1293	1.000	0.000	0.998	1.000	1.000
49	9746	4	1	1292	1.000	0.000	0.997	0.999	1.000
50	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
51	9746	4	0	1293	1.000	0.000	0.997	1.000	1.000
52	9746	4	0	1293	1.000	0.000	0.997	1.000	1.000
53	9749	1	0	1293	1.000	0.000	0.999	1.000	1.000
54	9746	4	0	1293	1.000	0.000	0.997	1.000	1.000

Table B.5: Classification Results of the Simple Index Translation Method without payload in combination with Case Data

PrefixLength	TN	FP	FN	TP	Accuracy	Error Rate	Precision	Recall	Specificity
1	8942	808	1164	129	0.821	0.179	0.138	0.100	0.917
2	8590	1160	1106	187	0.795	0.205	0.139	0.145	0.881
3	8435	1315	1089	204	0.782	0.218	0.134	0.158	0.865
4	8539	1211	1098	195	0.791	0.209	0.139	0.151	0.876
5	8745	1005	1137	156	0.806	0.194	0.134	0.121	0.897
6	8819	931	1130	163	0.813	0.187	0.149	0.126	0.905
7	8773	977	1136	157	0.809	0.191	0.138	0.121	0.900
8	8735	1015	1131	162	0.806	0.194	0.138	0.125	0.896
9	8758	992	1137	156	0.807	0.193	0.136	0.121	0.898
10	8846	904	1150	143	0.814	0.186	0.137	0.111	0.907
11	8806	944	1132	161	0.812	0.188	0.146	0.125	0.903
12	8874	876	1141	152	0.817	0.183	0.148	0.118	0.910
13	8820	930	1150	143	0.812	0.188	0.133	0.111	0.905
14	8817	933	1149	144	0.811	0.189	0.134	0.111	0.904
15	8636	1114	1115	178	0.798	0.202	0.138	0.138	0.886
16	8480	1270	1100	193	0.785	0.215	0.132	0.149	0.870
17	8440	1310	1081	212	0.783	0.217	0.139	0.164	0.866
18	8633	1117	1110	183	0.798	0.202	0.141	0.142	0.885
19	8669	1081	1115	178	0.801	0.199	0.141	0.138	0.889
20	8594	1156	1111	182	0.795	0.205	0.136	0.141	0.881
21	8754	996	1133	160	0.807	0.193	0.138	0.124	0.898
22	8847	903	1149	144	0.814	0.186	0.138	0.111	0.907
23	8774	976	1137	156	0.809	0.191	0.138	0.121	0.900
24	8775	975	1135	158	0.809	0.191	0.139	0.122	0.900
25	8795	955	1140	153	0.810	0.190	0.138	0.118	0.902
26	8860	890	1150	143	0.815	0.185	0.138	0.111	0.909
27	8950	800	1071	222	0.831	0.169	0.217	0.172	0.918
28	8927	823	893	400	0.845	0.155	0.327	0.309	0.916
29	9114	636	814	479	0.869	0.131	0.430	0.370	0.935
30	9042	708	708	585	0.872	0.128	0.452	0.452	0.927
31	9302	448	314	979	0.931	0.069	0.686	0.757	0.954
32	9528	222	156	1137	0.966	0.034	0.837	0.879	0.977
33	9587	163	67	1226	0.979	0.021	0.883	0.948	0.983
34	9687	63	40	1253	0.991	0.009	0.952	0.969	0.994
35	9706	44	19	1274	0.994	0.006	0.967	0.985	0.995
36	9710	40	9	1284	0.996	0.004	0.970	0.993	0.996
37	9708	42	5	1288	0.996	0.004	0.968	0.996	0.996
38	9711	39	5	1288	0.996	0.004	0.971	0.996	0.996
39	9710	40	3	1290	0.996	0.004	0.970	0.998	0.996
40	9704	46	5	1288	0.995	0.005	0.966	0.996	0.995
41	9708	42	2	1291	0.996	0.004	0.968	0.998	0.996
42	9709	41	1	1292	0.996	0.004	0.969	0.999	0.996
43	9709	41	2	1291	0.996	0.004	0.969	0.998	0.996
44	9709	41	3	1290	0.996	0.004	0.969	0.998	0.996
45	9707	43	2	1291	0.996	0.004	0.968	0.998	0.996
46	9708	42	2	1291	0.996	0.004	0.968	0.998	0.996
47	9710	40	3	1290	0.996	0.004	0.970	0.998	0.996
48	9714	36	1	1292	0.997	0.003	0.973	0.999	0.996
49	9710	40	2	1291	0.996	0.004	0.970	0.998	0.996
50	9707	43	2	1291	0.996	0.004	0.968	0.998	0.996
51	9712	38	2	1291	0.996	0.004	0.971	0.998	0.996
52	9708	42	3	1290	0.996	0.004	0.968	0.998	0.996
53	9714	36	2	1291	0.997	0.003	0.973	0.998	0.996
54	9706	44	2	1291	0.996	0.004	0.967	0.998	0.995

Table B.6: Classification Results of the Simple Index Translation Method with payload in combination with Case Data

PrefixLength	TN	FP	FN	TP	Accuracy	Error Rate	Precision	Recall	Specificity
1	8931	819	1159	134	0.821	0.179	0.141	0.104	0.916
2	9173	577	1186	107	0.840	0.160	0.156	0.083	0.941
3	9208	542	1196	97	0.843	0.157	0.152	0.075	0.944
4	9304	446	1218	75	0.849	0.151	0.144	0.058	0.954
5	9409	341	1234	59	0.857	0.143	0.147	0.046	0.965
6	9534	216	1252	41	0.867	0.133	0.160	0.032	0.978
7	9550	200	1248	45	0.869	0.131	0.184	0.035	0.979
8	9597	153	1271	22	0.871	0.129	0.126	0.017	0.984
9	9596	154	1264	29	0.872	0.128	0.158	0.022	0.984
10	9650	100	1278	15	0.875	0.125	0.130	0.012	0.990
11	9637	113	1274	19	0.874	0.126	0.144	0.015	0.988
12	9654	96	1278	15	0.876	0.124	0.135	0.012	0.990
13	9698	52	1283	10	0.879	0.121	0.161	0.008	0.995
14	9698	52	1280	13	0.879	0.121	0.200	0.010	0.995
15	9690	60	1284	9	0.878	0.122	0.130	0.007	0.994
16	9709	41	1288	5	0.880	0.120	0.109	0.004	0.996
17	9696	54	1290	3	0.878	0.122	0.053	0.002	0.994
18	9706	44	1286	7	0.880	0.120	0.137	0.005	0.995
19	9708	42	1287	6	0.880	0.120	0.125	0.005	0.996
20	9708	42	1289	4	0.879	0.121	0.087	0.003	0.996
21	9704	46	1287	6	0.879	0.121	0.115	0.005	0.995
22	9700	50	1288	5	0.879	0.121	0.091	0.004	0.995
23	9716	34	1286	7	0.880	0.120	0.171	0.005	0.997
24	9706	44	1288	5	0.879	0.121	0.102	0.004	0.995
25	9712	38	1290	3	0.880	0.120	0.073	0.002	0.996
26	9717	33	1189	104	0.889	0.111	0.759	0.080	0.997
27	9717	33	1001	292	0.906	0.094	0.898	0.226	0.997
28	9712	38	914	379	0.914	0.086	0.909	0.293	0.996
29	9726	24	822	471	0.923	0.077	0.952	0.364	0.998
30	9724	26	385	908	0.963	0.037	0.972	0.702	0.997
31	9726	24	184	1109	0.981	0.019	0.979	0.858	0.998
32	9721	29	86	1207	0.990	0.010	0.977	0.933	0.997
33	9726	24	49	1244	0.993	0.007	0.981	0.962	0.998
34	9726	24	34	1259	0.995	0.005	0.981	0.974	0.998
35	9727	23	18	1275	0.996	0.004	0.982	0.986	0.998
36	9722	28	12	1281	0.996	0.004	0.979	0.991	0.997
37	9729	21	10	1283	0.997	0.003	0.984	0.992	0.998
38	9728	22	5	1288	0.998	0.002	0.983	0.996	0.998
39	9731	19	6	1287	0.998	0.002	0.985	0.995	0.998
40	9727	23	7	1286	0.997	0.003	0.982	0.995	0.998
41	9724	26	6	1287	0.997	0.003	0.980	0.995	0.997
42	9728	22	4	1289	0.998	0.002	0.983	0.997	0.998
43	9727	23	7	1286	0.997	0.003	0.982	0.995	0.998
44	9727	23	5	1288	0.997	0.003	0.982	0.996	0.998
45	9726	24	6	1287	0.997	0.003	0.982	0.995	0.998
46	9727	23	6	1287	0.997	0.003	0.982	0.995	0.998
47	9727	23	4	1289	0.998	0.002	0.982	0.997	0.998
48	9726	24	4	1289	0.997	0.003	0.982	0.997	0.998
49	9728	22	5	1288	0.998	0.002	0.983	0.996	0.998
50	9727	23	6	1287	0.997	0.003	0.982	0.995	0.998
51	9723	27	5	1288	0.997	0.003	0.979	0.996	0.997
52	9725	25	5	1288	0.997	0.003	0.981	0.996	0.997
53	9723	27	6	1287	0.997	0.003	0.979	0.995	0.997
54	9727	23	6	1287	0.997	0.003	0.982	0.995	0.998

Appendix C

Classification Results of Random Forests without Case Data

The following pages present five tables, one per page. Each table shows the confusion matrices and corresponding evaluation measures of the classification results. Each table shows the classification results of random forests that were trained on attributes resulting from one of the five described translation methods.

Table C.1: Classification Results of the Boolean 1-Gram Translation Method

Prefix	TN	FP	FN	TP	Accuracy	Error Rate	Precision	Recall	Specificity
1									
2	9750	0	1293	0	0.883	0.117		0.000	1.000
3	9750	0	1293	0	0.883	0.117		0.000	1.000
4	9750	0	1293	0	0.883	0.117		0.000	1.000
5	9750	0	1293	0	0.883	0.117		0.000	1.000
6	9750	0	1293	0	0.883	0.117		0.000	1.000
7	9750	0	1293	0	0.883	0.117		0.000	1.000
8	9750	0	1293	0	0.883	0.117		0.000	1.000
9	9750	0	1293	0	0.883	0.117		0.000	1.000
10	9750	0	1293	0	0.883	0.117		0.000	1.000
11	9750	0	1293	0	0.883	0.117		0.000	1.000
12	9750	0	1293	0	0.883	0.117		0.000	1.000
13	9750	0	1293	0	0.883	0.117		0.000	1.000
14	9750	0	1293	0	0.883	0.117		0.000	1.000
15	9750	0	1293	0	0.883	0.117		0.000	1.000
16	9750	0	1293	0	0.883	0.117		0.000	1.000
17	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
18	9750	0	1293	0	0.883	0.117		0.000	1.000
19	9750	0	1293	0	0.883	0.117		0.000	1.000
20	9750	0	1293	0	0.883	0.117		0.000	1.000
21	9750	0	1293	0	0.883	0.117		0.000	1.000
22	9750	0	1293	0	0.883	0.117		0.000	1.000
23	9750	0	1293	0	0.883	0.117		0.000	1.000
24	9750	0	1293	0	0.883	0.117		0.000	1.000
25	9750	0	1293	0	0.883	0.117		0.000	1.000
26	9750	0	1293	0	0.883	0.117		0.000	1.000
27	9750	0	1196	97	0.892	0.108	1.000	0.075	1.000
28	9750	0	1009	284	0.909	0.091	1.000	0.220	1.000
29	9749	1	920	373	0.917	0.083	0.997	0.288	1.000
30	9748	2	822	471	0.925	0.075	0.996	0.364	1.000
31	9747	3	390	903	0.964	0.036	0.997	0.698	1.000
32	9744	6	187	1106	0.983	0.017	0.995	0.855	0.999
33	9742	8	90	1203	0.991	0.009	0.993	0.930	0.999
34	9741	9	50	1243	0.995	0.005	0.993	0.961	0.999
35	9740	10	27	1266	0.997	0.003	0.992	0.979	0.999
36	9742	8	19	1274	0.998	0.002	0.994	0.985	0.999
37	9745	5	13	1280	0.998	0.002	0.996	0.990	0.999
38	9748	2	7	1286	0.999	0.001	0.998	0.995	1.000
39	9749	1	5	1288	0.999	0.001	0.999	0.996	1.000
40	9749	1	1	1292	1.000	0.000	0.999	0.999	1.000
41	9750	0	1	1292	1.000	0.000	1.000	0.999	1.000
42	9749	1	1	1292	1.000	0.000	0.999	0.999	1.000
43	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
44	9749	1	0	1293	1.000	0.000	0.999	1.000	1.000
45	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
46	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
47	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
48	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
49	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
50	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
51	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
52	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
53	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
54	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000

Table C.2: Classification Results of the Frequency 1-Gram Translation Method

Prefix	TN	FP	FN	TP	Accuracy	Error Rate	Precision	Recall	Specificity
1									
2	9750	0	1293	0	0.883	0.117		0.000	1.000
3	9750	0	1293	0	0.883	0.117		0.000	1.000
4	9750	0	1293	0	0.883	0.117		0.000	1.000
5	9750	0	1293	0	0.883	0.117		0.000	1.000
6	9750	0	1293	0	0.883	0.117		0.000	1.000
7	9750	0	1293	0	0.883	0.117		0.000	1.000
8	9750	0	1293	0	0.883	0.117		0.000	1.000
9	9750	0	1293	0	0.883	0.117		0.000	1.000
10	9750	0	1293	0	0.883	0.117		0.000	1.000
11	9750	0	1293	0	0.883	0.117		0.000	1.000
12	9750	0	1293	0	0.883	0.117		0.000	1.000
13	9750	0	1293	0	0.883	0.117		0.000	1.000
14	9750	0	1293	0	0.883	0.117		0.000	1.000
15	9750	0	1293	0	0.883	0.117		0.000	1.000
16	9744	6	1293	0	0.882	0.118	0.000	0.000	0.999
17	9750	0	1293	0	0.883	0.117		0.000	1.000
18	9750	0	1293	0	0.883	0.117		0.000	1.000
19	9750	0	1293	0	0.883	0.117		0.000	1.000
20	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
21	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
22	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
23	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
24	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
25	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
26	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
27	9748	2	1196	97	0.892	0.108	0.980	0.075	1.000
28	9749	1	1008	285	0.909	0.091	0.997	0.220	1.000
29	9748	2	919	374	0.917	0.083	0.995	0.289	1.000
30	9747	3	821	472	0.925	0.075	0.994	0.365	1.000
31	9747	3	390	903	0.964	0.036	0.997	0.698	1.000
32	9743	7	187	1106	0.982	0.018	0.994	0.855	0.999
33	9740	10	90	1203	0.991	0.009	0.992	0.930	0.999
34	9740	10	49	1244	0.995	0.005	0.992	0.962	0.999
35	9740	10	27	1266	0.997	0.003	0.992	0.979	0.999
36	9743	7	18	1275	0.998	0.002	0.995	0.986	0.999
37	9746	4	13	1280	0.998	0.002	0.997	0.990	1.000
38	9749	1	7	1286	0.999	0.001	0.999	0.995	1.000
39	9750	0	5	1288	1.000	0.000	1.000	0.996	1.000
40	9750	0	1	1292	1.000	0.000	1.000	0.999	1.000
41	9750	0	1	1292	1.000	0.000	1.000	0.999	1.000
42	9750	0	1	1292	1.000	0.000	1.000	0.999	1.000
43	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
44	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
45	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
46	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
47	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
48	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
49	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
50	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
51	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
52	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
53	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
54	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000

Table C.3: Classification Results of the Bi-Gram Translation Method

Prefix	TN	FP	FN	TP	Accuracy	Error Rate	Precision	Recall	Specificity
1									
2	9750	0	1293	0	0.883	0.117		0.000	1.000
3	9750	0	1293	0	0.883	0.117		0.000	1.000
4	9750	0	1293	0	0.883	0.117		0.000	1.000
5	9750	0	1293	0	0.883	0.117		0.000	1.000
6	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
7	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
8	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
9	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
10	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
11	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
12	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
13	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
14	9748	2	1293	0	0.883	0.117	0.000	0.000	1.000
15	9748	2	1293	0	0.883	0.117	0.000	0.000	1.000
16	9748	2	1293	0	0.883	0.117	0.000	0.000	1.000
17	9748	2	1293	0	0.883	0.117	0.000	0.000	1.000
18	9748	2	1293	0	0.883	0.117	0.000	0.000	1.000
19	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
20	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
21	9748	2	1293	0	0.883	0.117	0.000	0.000	1.000
22	9748	2	1293	0	0.883	0.117	0.000	0.000	1.000
23	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
24	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
25	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
26	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
27	9748	2	1195	98	0.892	0.108	0.980	0.076	1.000
28	9749	1	1007	286	0.909	0.091	0.997	0.221	1.000
29	9748	2	921	372	0.916	0.084	0.995	0.288	1.000
30	9750	0	828	465	0.925	0.075	1.000	0.360	1.000
31	9747	3	387	906	0.965	0.035	0.997	0.701	1.000
32	9745	5	189	1104	0.982	0.018	0.995	0.854	0.999
33	9745	5	96	1197	0.991	0.009	0.996	0.926	0.999
34	9744	6	52	1241	0.995	0.005	0.995	0.960	0.999
35	9744	6	33	1260	0.996	0.004	0.995	0.974	0.999
36	9747	3	22	1271	0.998	0.002	0.998	0.983	1.000
37	9746	4	12	1281	0.999	0.001	0.997	0.991	1.000
38	9747	3	6	1287	0.999	0.001	0.998	0.995	1.000
39	9749	1	5	1288	0.999	0.001	0.999	0.996	1.000
40	9749	1	1	1292	1.000	0.000	0.999	0.999	1.000
41	9750	0	2	1291	1.000	0.000	1.000	0.998	1.000
42	9749	1	1	1292	1.000	0.000	0.999	0.999	1.000
43	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
44	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
45	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
46	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
47	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
48	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
49	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
50	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
51	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
52	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
53	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000
54	9750	0	0	1293	1.000	0.000	1.000	1.000	1.000

Table C.4: Classification Results of the Simple Index Translation Method without payload

Prefix	TN	FP	FN	TP	Accuracy	Error Rate	Precision	Recall	Specificity
1									
2	9750	0	1293	0	0.883	0.117		0.000	1.000
3	9750	0	1293	0	0.883	0.117		0.000	1.000
4	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
5	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
6	9748	2	1293	0	0.883	0.117	0.000	0.000	1.000
7	9747	3	1293	0	0.883	0.117	0.000	0.000	1.000
8	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
9	9747	3	1293	0	0.883	0.117	0.000	0.000	1.000
10	9746	4	1293	0	0.883	0.117	0.000	0.000	1.000
11	9745	5	1293	0	0.882	0.118	0.000	0.000	0.999
12	9744	6	1293	0	0.882	0.118	0.000	0.000	0.999
13	9746	4	1293	0	0.883	0.117	0.000	0.000	1.000
14	9745	5	1293	0	0.882	0.118	0.000	0.000	0.999
15	9742	8	1293	0	0.882	0.118	0.000	0.000	0.999
16	9740	10	1293	0	0.882	0.118	0.000	0.000	0.999
17	9734	16	1293	0	0.881	0.119	0.000	0.000	0.998
18	9721	29	1292	1	0.880	0.120	0.033	0.001	0.997
19	9720	30	1293	0	0.880	0.120	0.000	0.000	0.997
20	9726	24	1292	1	0.881	0.119	0.040	0.001	0.998
21	9721	29	1292	1	0.880	0.120	0.033	0.001	0.997
22	9719	31	1291	2	0.880	0.120	0.061	0.002	0.997
23	9721	29	1291	2	0.880	0.120	0.065	0.002	0.997
24	9724	26	1290	3	0.881	0.119	0.103	0.002	0.997
25	9721	29	1291	2	0.880	0.120	0.065	0.002	0.997
26	9718	32	1289	4	0.880	0.120	0.111	0.003	0.997
27	9725	25	1192	101	0.890	0.110	0.802	0.078	0.997
28	9721	29	1004	289	0.906	0.094	0.909	0.224	0.997
29	9721	29	917	376	0.914	0.086	0.928	0.291	0.997
30	9725	25	815	478	0.924	0.076	0.950	0.370	0.997
31	9725	25	382	911	0.963	0.037	0.973	0.705	0.997
32	9720	30	179	1114	0.981	0.019	0.974	0.862	0.997
33	9721	29	81	1212	0.990	0.010	0.977	0.937	0.997
34	9717	33	43	1250	0.993	0.007	0.974	0.967	0.997
35	9718	32	20	1273	0.995	0.005	0.975	0.985	0.997
36	9721	29	12	1281	0.996	0.004	0.978	0.991	0.997
37	9723	27	7	1286	0.997	0.003	0.979	0.995	0.997
38	9727	23	5	1288	0.997	0.003	0.982	0.996	0.998
39	9727	23	4	1289	0.998	0.002	0.982	0.997	0.998
40	9728	22	4	1289	0.998	0.002	0.983	0.997	0.998
41	9728	22	5	1288	0.998	0.002	0.983	0.996	0.998
42	9726	24	3	1290	0.998	0.002	0.982	0.998	0.998
43	9726	24	4	1289	0.997	0.003	0.982	0.997	0.998
44	9728	22	3	1290	0.998	0.002	0.983	0.998	0.998
45	9727	23	4	1289	0.998	0.002	0.982	0.997	0.998
46	9727	23	5	1288	0.997	0.003	0.982	0.996	0.998
47	9725	25	3	1290	0.997	0.003	0.981	0.998	0.997
48	9727	23	3	1290	0.998	0.002	0.982	0.998	0.998
49	9724	26	5	1288	0.997	0.003	0.980	0.996	0.997
50	9726	24	2	1291	0.998	0.002	0.982	0.998	0.998
51	9726	24	3	1290	0.998	0.002	0.982	0.998	0.998
52	9728	22	5	1288	0.998	0.002	0.983	0.996	0.998
53	9727	23	3	1290	0.998	0.002	0.982	0.998	0.998
54	9725	25	3	1290	0.997	0.003	0.981	0.998	0.997

Table C.5: Classification Results of the Simple Index Translation Method with payload

Prefix	TN	FP	FN	TP	Accuracy	Error Rate	Precision	Recall	Specificity
1	9750	0	1293	0	0.883	0.117		0.000	1.000
2	9750	0	1293	0	0.883	0.117		0.000	1.000
3	9750	0	1293	0	0.883	0.117		0.000	1.000
4	9750	0	1293	0	0.883	0.117		0.000	1.000
5	9748	2	1293	0	0.883	0.117	0.000	0.000	1.000
6	9750	0	1293	0	0.883	0.117		0.000	1.000
7	9750	0	1293	0	0.883	0.117		0.000	1.000
8	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
9	9750	0	1293	0	0.883	0.117		0.000	1.000
10	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
11	9749	1	1293	0	0.883	0.117	0.000	0.000	1.000
12	9747	3	1293	0	0.883	0.117	0.000	0.000	1.000
13	9745	5	1293	0	0.882	0.118	0.000	0.000	0.999
14	9746	4	1292	1	0.883	0.117	0.200	0.001	1.000
15	9744	6	1292	1	0.882	0.118	0.143	0.001	0.999
16	9743	7	1292	1	0.882	0.118	0.125	0.001	0.999
17	9741	9	1292	1	0.882	0.118	0.100	0.001	0.999
18	9739	11	1292	1	0.882	0.118	0.083	0.001	0.999
19	9744	6	1292	1	0.882	0.118	0.143	0.001	0.999
20	9743	7	1292	1	0.882	0.118	0.125	0.001	0.999
21	9741	9	1293	0	0.882	0.118	0.000	0.000	0.999
22	9741	9	1292	1	0.882	0.118	0.100	0.001	0.999
23	9735	15	1291	2	0.882	0.118	0.118	0.002	0.998
24	9741	9	1292	1	0.882	0.118	0.100	0.001	0.999
25	9738	12	1293	0	0.882	0.118	0.000	0.000	0.999
26	9738	12	1194	99	0.891	0.109	0.892	0.077	0.999
27	9738	12	1006	287	0.908	0.092	0.960	0.222	0.999
28	9743	7	919	374	0.916	0.084	0.982	0.289	0.999
29	9739	11	825	468	0.924	0.076	0.977	0.362	0.999
30	9733	17	386	907	0.964	0.036	0.982	0.701	0.998
31	9736	14	186	1107	0.982	0.018	0.988	0.856	0.999
32	9732	18	90	1203	0.990	0.010	0.985	0.930	0.998
33	9729	21	49	1244	0.994	0.006	0.983	0.962	0.998
34	9726	24	36	1257	0.995	0.005	0.981	0.972	0.998
35	9732	18	22	1271	0.996	0.004	0.986	0.983	0.998
36	9728	22	13	1280	0.997	0.003	0.983	0.990	0.998
37	9730	20	11	1282	0.997	0.003	0.985	0.991	0.998
38	9731	19	7	1286	0.998	0.002	0.985	0.995	0.998
39	9731	19	8	1285	0.998	0.002	0.985	0.994	0.998
40	9730	20	6	1287	0.998	0.002	0.985	0.995	0.998
41	9730	20	7	1286	0.998	0.002	0.985	0.995	0.998
42	9728	22	4	1289	0.998	0.002	0.983	0.997	0.998
43	9730	20	6	1287	0.998	0.002	0.985	0.995	0.998
44	9728	22	7	1286	0.997	0.003	0.983	0.995	0.998
45	9730	20	6	1287	0.998	0.002	0.985	0.995	0.998
46	9728	22	7	1286	0.997	0.003	0.983	0.995	0.998
47	9728	22	6	1287	0.997	0.003	0.983	0.995	0.998
48	9729	21	6	1287	0.998	0.002	0.984	0.995	0.998
49	9729	21	7	1286	0.997	0.003	0.984	0.995	0.998
50	9729	21	7	1286	0.997	0.003	0.984	0.995	0.998
51	9729	21	8	1285	0.997	0.003	0.984	0.994	0.998
52	9728	22	7	1286	0.997	0.003	0.983	0.995	0.998
53	9727	23	6	1287	0.997	0.003	0.982	0.995	0.998
54	9729	21	8	1285	0.997	0.003	0.984	0.994	0.998