Pontifícia Universidade Católica
do Rio de Janeiro

Daniel Duque Guimarães Saraiva

Dealing with decision points in process mining

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática .

Advisor: Prof. Hélio Côrtes Vieira Lopes

Rio de Janeiro
September 2018

Pontifícia Universidade Católica
do Rio de Janeiro

Daniel Duque Guimarães Saraiva

Dealing with decision points in process mining

Dissertation presented to the Programa de Pós–graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática . Approved by the undersigned Examination Committee.

Prof. Hélio Côrtes Vieira Lopes
Advisor
Departamento de Informática – PUC-Rio

Prof. Gustavo Robichez de Carvalho
Departamento de Informática – PUC-Rio

Prof. Rafael Barbosa Nasser
Departamento de Informática – PUC-Rio

Prof. Márcio de Silveira Carvalho
Vice Dean of Graduate Studies
Centro Técnico Científico – PUC-Rio

Rio de Janeiro, September 5th, 2018

Daniel Duque Guimarães Saraiva

Graduated in Computer Engineering at Pontifícia Universidade Católica (PUC-Rio) in 2015. While studying there worked at Instituto Tecgraf from June 2012 until February 2013. During the masters degree program was working at Bahia Asset Managment.

Bibliographic data

Duque Guimarães Saraiva, Daniel

Dealing with decision points in process mining / Daniel Duque Guimarães Saraiva; advisor: Hélio Côrtes Vieira Lopes. – Rio de janeiro: PUC-Rio , Departamento de Informática, 2018.

v., 33 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Engenharia de Computação – Teses. 3. Mineração de Decisão;. 4. Pontos de Decisão;. 5. Mineração de Processos;. 6. Redes de Petri;. 7. Árvore de Decisão;. I. Côrtes Vieira Lopes, Hélio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

I would like to first thank my advisor Hélio Lopes for all the things that I've learned from him in the past two years.

I'd like to thank my parents Guilherme and Maria Clara for always believing in my potential and always being there for me.

I also would like to thank the rest of my family for all the love and support, and a special thanks to my older brother Bernardo, for always being by my side, and to my younger sister Alessandra, for being such an incredible person.

There are many friends that I'd like to quote but I'll sum it all up and thank all of them.

Last, but no least, I'd like to thank the Bahia Asset team for being so comprehensive and stimulative in the completion of this Master.

This study was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Duque Guimarães Saraiva, Daniel; Côrtes Vieira Lopes, Hélio (Advisor). **Dealing with decision points in process mining**. Rio de Janeiro, 2018. 33p. Dissertação de mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Due to the increasing competitiveness and demand for higher performance, many companies realized that it is necessary to rethink and enhance their business processes. In order to achieve this goal, companies have been turning to computational techniques that are capable of extracting new information and insights from their, ever-increasing, datasets. Business processes, normally, have many places where a decision has to be made. It is reasonable to expect that similar inputs have the same decisions made to them during the process. The goal of this dissertation is to create a decision miner that automates the decision-making inside a process. First, we will identify decision points in a Petri net model. Then, we will transform the decision-making problem into a classification one, where each of the possible decisions becomes a class. In order to automate the decision-making, a decision tree is trained using data attributes from the event logs. A real world case study is used to validate that the decision miner is reliable when using real world data.

Keywords

Decision Mining; Decision Points; Process Mining; Petri nets; Decision Tree;

Resumo

Duque Guimarães Saraiva, Daniel; Côrtes Vieira Lopes, Hélio. **Tratando pontos de decisão em mineração de processos**. Rio de Janeiro, 2018. 33p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Devido ao grande aumento da competitividade e da, cada vez maior, demanda por eficiência, muitas empresas perceberam que é necessário repensar e melhorar seus processos. Para atingir este objetivo, elas têm cada vez mais buscado técnicas computacionais que sejam capazes de extrair novas informações e conhecimentos de suas grandes bases de dados. Os processos das empresas, normalmente, possuem momentos em que uma decisão deve ser tomada. É razoável esperar que casos similares tenham decisões parecidas sendo tomadas ao longo do processo. O objetivo desta dissertação é criar um minerador de decisão que seja capaz the automatizar a tomada de decisão dentro de um processo. A primeira parte do trabalho consiste na identificação dos pontos de decisão em uma rede de Petri. Em seguida, transformamos a tomada de decisão em um problema de classificação no qual cada possibilidade da decisão se torna uma classe. Para fazer a automatização, é utilizada uma árvore de decisão treinada com os atributos dos dados que estão presentes nos logs dos eventos. Um estudo de caso real é utilizado para validar que o minerador de decisão é confiável para processos reais.

Palavras-chave

Mineração de Decisão; Pontos de Decisão; Mineração de Processos; Redes de Petri; Árvore de Decisão;

Table of contents

List of figures

List of tables

1
Introduction

In order to survive and prosper in today's competitive world, companies have realized that it is necessary to rethink and enhance business processes. Since the 80´s, many companies have invested heavily on process automation technologies. Such initiatives have resulted in tremendous gains, both in terms of productivity and process efficiency.

However, there is still room for further improvements. One way of achieving this goal is to use data that is available or that can be obtained from the systems that manage a company's processes. Such data can be stored in event logs and can be treated by new computational techniques such as machine learning algorithms.

The process mining field was created from this necessity. It combines techniques from process analysis and data science into a single field of study. These techniques have proven to be valuable tools to gain insights into how business processes are handled within an organization and are usually employed in *process discovery* (3, 4) and *conformance checking* (7, 8).

Process discovery can be used to automatically construct a process model that reflects the real business process, derived from real data observations that were stored in event logs. Conformance checking can be used to verify if the process behaviors indicated from data logs follows a given model. It can also validate if business rules are being followed. Together, process discovery and conformance checking, may be used as inputs to improve business processes, for example: process discovery can be a starting point to model complex systems and conformance checking can be used to find problems in existing processes.

An important aspect of processes is the decisions made during their execution. Depending on the choices made, the result may differ, hence, making it one of the key aspects of a process. Our goal is to show that there is information in the event logs that can help us identify which decision should be made throughout our process. In order to achieve this, we have to first identify the decision points (1, 9) in a model. Once they have been identified, we have to understand why the choice was made. One way to approach this problem is thinking of it as a *classification* problem, where each of the possible choices is a different class, thus, making it possible to transform the decision-making

problem into a machine learning one.

There are several classification algorithms, and the *decision tree* was chosen due to its flexibility, in terms of handling data. It is capable of handling discrete and continuous numeric data, non-numeric data and missing data, which is a major concern when dealing with real world data. In order to train the decision tree, we will match the outcomes of the decisions points with the information present in the event logs up to that point. This way we will try to extract which data attribute, or group of data attributes, contains information that can help us to correctly identify the outcome of a decision-making process.

2
Petri nets

Petri Nets originated from the work of Carl Adam Petri (5) and, from that point onward, its study has increased considerably. For a review of the history of Petri Nets and bibliography it is recommended to read Murata (6).

The Petri net model has three main components: places, transitions and arcs. Places and transitions are two different types of nodes and arcs are the connections between those nodes. The Petri net is, by definition, a directed bipartite graph, because nodes of the same type are not allowed to be connected and arcs have a direction.

In a model, places are represented by circles, transitions by rectangles and arcs by arrows, as can be seen in figure 2.1. Places may contain zero or more tokens, represented as black dots. A place p is called an input place of a transition t if there exists an arc from p to t and p is called an output place of t if there exists an arc from t to p. A transition t is enabled if all its input places contain at least one token and once it is enabled it can be fired. Firing the transition t consumes one token from each of its input places and creates a new token in all its output places.

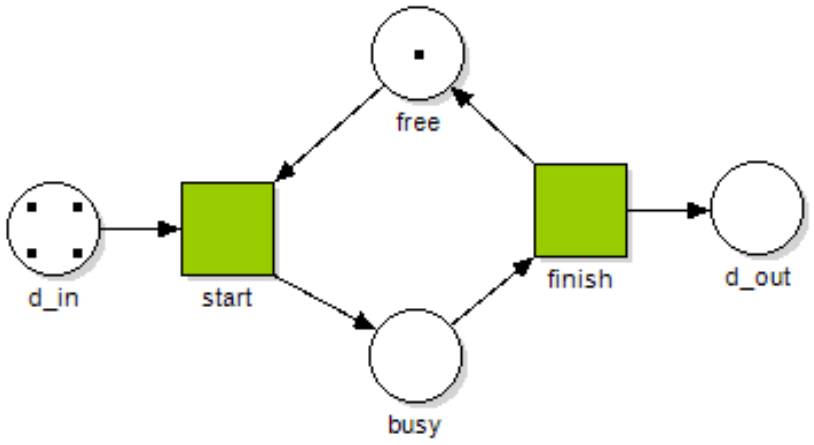

Figure 2.1: Document analysis - initial state.

2.1 Example

To exemplify a Petri net model, we will use the net shown in figure 2.1. This net represents a document analysis by a company's employee and it is composed of four places (*d_in*, *busy*, *free*, *d_out*) and two transitions (*start*, *finish*). In the state shown in the figure 2.1, there are a total of five tokens, four in *place d_in* and one in *place free*. The tokens in *place d_in* represent new documents to be processed by the employee and the token in *place free* indicates that the employee is currently free.

When the *transition start* fires, two tokens will be consumed, one from *place d_in* and one from *place free*, and one token will be created in *place busy*. The new state of the model is represented in figure 2.2 (left). The token in *place busy* indicates that the employee is busy analyzing the document.

Since *place free* has no more tokens, the *transition start* is no longer enabled and because *place busy* has a token the *transition finish* can be fired. When the *transition finish* is fired, the token in *place busy* is removed and new tokens are created in *places d_out* and *free*, representing that one document has already been analyzed and that the employee is again free, as shown in figure 2.2 (right).

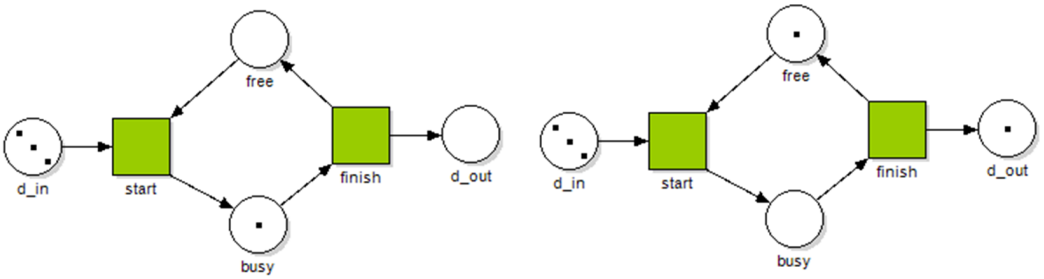

Figure 2.2: Document analysis.

2.2 Mining a Petri net from event logs

Process discovery techniques seek to find a model that best represents a business process. Since there are many algorithms, there can be multiple solutions to a single mining problem. It is not in the scope of this work to delve into much detail about these algorithms. Lijie Wen (4) has already made good work explaining and comparing several of those algorithms and it is

recommended. An example of a mined process model from a set of logs is represented in figure 2.3.

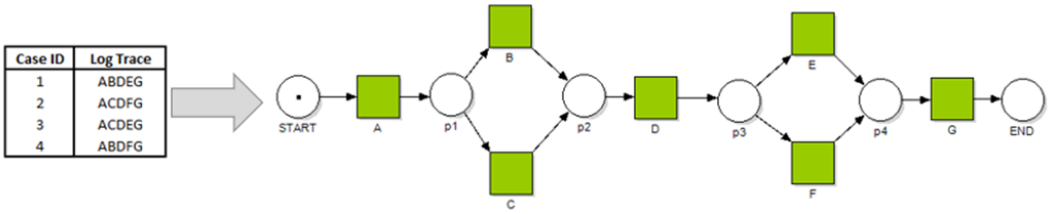

Figure 2.3: Example of Petri net mined from event logs.

It is important to notice that the set of event logs used to mine a model may not comprise all possible routes in a process, hence, it is important to keep in mind that mined models may not represent the whole process. This concept is called completeness. As in many other data science related topics, such as machine learning, one cannot assume to have mapped all possibilities in a training set. Therefore, when we start training decision points, we should always keep in mind that there is a possibility that our model does not contain all possible information.

2.3 Identifying Decision Points in a Petri net

A decision point in any business process is a step of the process where a decision has to be made before the process can follow through one of the many available paths. In a Petri net model, a decision point is any place that is an input place to two or more transitions, or, from another perspective, a place is a decision point if it has more than one outgoing arc. In figure 2.3, for example, places *p1* and *p3* are decision points, because each of them may lead to two possible activities. In *p1* you may choose between transitions B and C, and in *p3* you may choose between transitions E and F.

3
Choosing Paths for Decision Points

After identifying decision points in a Petri net, we want to be able to automatically choose routes for new cases in our model. Machine learning algorithms have been widely adopted to extract patterns, similar to the ones we are looking for, from large sets of data. For example, a supervised machine learning algorithm could be taught to identify birds inside a dataset of animals. The training dataset could contain information like: number of legs, the presence of wings, number of eyes, type of coverage (feathers, fur, scales, etc...), and after training the algorithm the "concept" of bird, it would be capable of classifying new animals as birds and non-birds. An analogous approach to the example could be done in the decision point case. If we treat each of the possible activities as one class, we could then teach a machine learning algorithm which route should be taken for a given set of data attributes and then use the trained algorithm to choose routes for new data entries. Supervised learning algorithms normally need a large dataset in order to correctly learn the patterns, in our case, this is not an issue since most of the processes that require analysis have an abundant set of observed data.

When observing data attributes in decision points, we are capable of extracting information on business rules, as can be seen in figure 3.1. In the image, there are 2 highlighted decision points. In the first decision point *p0*, if the client has a premium policy or the request amount is below 500, the next activity is *Check policy only*, if the client has a normal policy and is requesting 500 or more, the next activity is *Check all*. After passing through the activity *Evaluate Claim*, the data attribute *status* is set and in the decision point *p2* the value of the data attribute *status* defines which activity will be the next. If the value is *approved*, it goes to activity *Send Approval*, else it goes to activity *Send Refusal*.

3.1
Choosing an Algorithm

There are many supervised machine learning algorithms in the literature, but most of them have a hard time dealing with non-numeric data, for instance, algorithms like *SVM*, *k-nearest neighbor* and *neural networks* can only handle

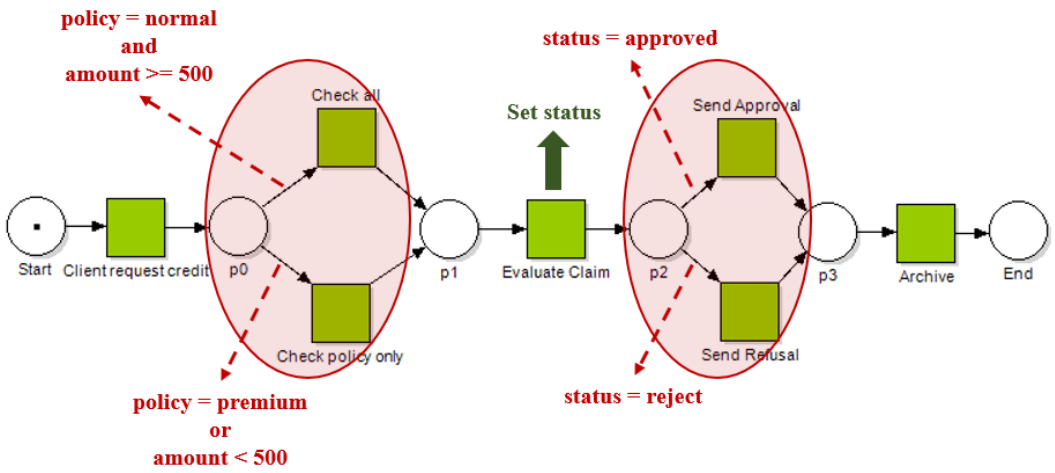

Figure 3.1: Business rules extracted from data attributes.

numeric data. When dealing with binary options, like yes/no, it is simple to make a numeric conversion, option one = 1 and option two = 0. In other cases where there is a clear order of the values, such as good/average/bad, it is also possible to make a numeric conversion, for example, good = 1, average = 0.5 and bad = 0. Unfortunately, there are many cases where there is not a clear order, for example, when dealing with an animal classification, it is not clear how to convert animal classes (mammals, reptiles, amphibians, fish and birds) into a numeric value. Even tough we could create dummy columns for each of the possible classifications, depending on the number of categories we would end up creating lots of columns and this would require more space and processing power to handle, making it not ideal. Besides the non-numeric data limitation in those algorithms, they are also unable to deal with missing data, which is a major concern when dealing with real world data. Because of those issues, these algorithms are not a good suit for the problem at hand.

Another possibility, would be using a *Naive Bayes* approach, but the algorithm is not good at handling continuous numeric data, hence, making it also not an ideal choice for our problem. Decision Tree's algorithm is capable of handling all the issues previously mentioned. It is capable of handling continuous numeric data, as well as, handle non-numeric data, and, most importantly, it is capable of dealing with missing values, making it a viable choice to solve our classification problem. An important thing to be cautious about, when using machine learning algorithms, is *overfitting*. Decision trees have many techniques to avoid such problems, reinforcing it as a good choice for the problem at hand.

One could try to argue that a *random forest* would also be a good choice

for this problem because it is a combination of multiple decision trees, hence, solving all the problems described above. The reason that random forests are not the best choice for our problem, is that we are trying to replicate business rules. When using a single decision tree, we are, intuitively, creating a mapping from the tree's splitting rules into business rules. Using multiple trees would not make much sense, because a process should have a set of defined rules that decides what is the next activity.

Figure 3.2 summarizes all the highlights made in this section. One thing to keep in mind is that some filters that we used to evaluate what is the most fitted algorithm for our task may not be an issue depending on the case. For instance, if representing the business rule is not important, then using the random forest is also a good choice. Similar analogies could be made for the other filters. In our case, we think that the representation of business rules is important because we want to be sure that the algorithm is correctly identifying the patterns.

Algorithm	Non-numeric data	Continuous Values	Missing data	Representation of business rules
SVM	✗	✓	✗	✗
K-nearest neighbor	✗	✓	✗	✗
Neural networks	✗	✓	✗	✗
Naive Bayes	✓	⚠	✓	✗
Decision Trees	✓	✓	✓	✓
Random Forest	✓	✓	✓	✗

Figure 3.2: Algorithms summary.

3.2 Challenges for Decision Mining

There are many challenges when dealing with real-life observations and the main concern is related to the *quality* of the data. One problem that may occur, as was said before, is the absence of data in some logs, in other words, missing data. Another problem is that data could be incorrectly logged in some specific cases, be it by human error or a bug in one of the systems. Hence, we have to keep in mind that the data may contain *noise*. So the mining algorithm

has to be robust[1], in order to deal with this noisy data. Also, when validating which business rules were set, by the algorithm, for each decision point, it is required that a human gets involved, because the system cannot, by itself, reason on the meaning of them nor understand the correctness of the obtained business rules.

When filtering the noisy data, we also have to be careful about some other aspects related to the control-flow, as was described by van der Aalst (1), they are: *invisible activities*, *duplicate activities* and *loops*.

3.2.1 Invisible Activities

An invisible activity is any activity, in a model, that is not present in the event logs, as is defined by van der Aalst (1). Since those activities are not present in the logs, we are unable to train our decision tree using them as a class, because there are no examples to be used as training data. So, instead of using them as the classes in the training model, we should look for the following visible activity. This way, we are able to identify which invisible activity was chosen. For example, in figure 3.3, when training p0, if the first activity observed is D, then we can be sure that the second branch was taken.

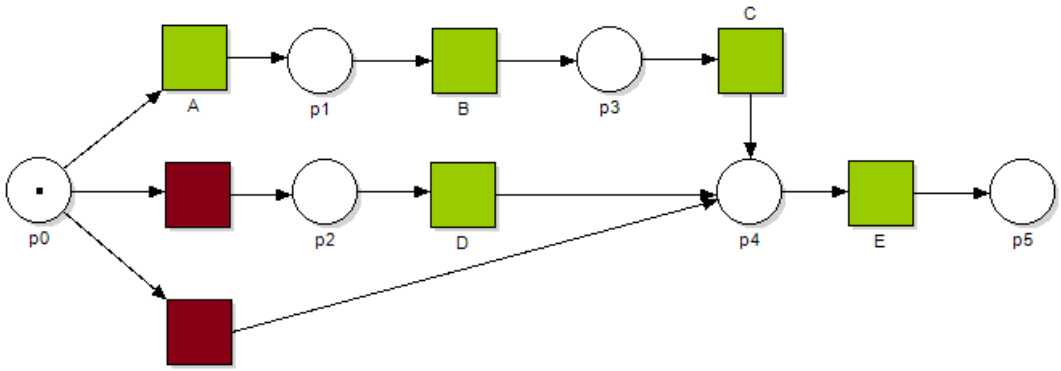

Figure 3.3: Invisible Activity.

Even though we may find a following visible activity, it may not be enough to determine which path was taken. In the example shown in figure 3.3, if the first visible activity is E, we cannot assume that the third branch was taken, because the other branches also reach the same activity, so having it as the first one is not enough to do this assumption. So, when we reach a

[1]Remember that the decision tree has some techniques to avoid overfitting, so when the noise is an outlier or an information that was incorrectly logged a couple of times, it should not have an effect on the tree.

joint construct, we stop tracking those invisible activities and their respective branches are removed from the analysis.

3.2.2 Duplicate Activities

Another problem that needs to be solved is duplicate activities. It is common, in event logs, to have multiple, distinct, activities associated to the same event name, so it may not be clear to which activity they are associated. Moreover, if we are trying to train a decision point similar to *p0*, shown in figure 3.4, it would not be clear what choice was made, because both would indicate class *A*.

The way we are dealing with duplicate activities is doing the same that was done to invisible activities, look for the next visible activity, that is also not duplicate. In the example shown in figure 3.4, if the first branch was chosen, instead of indicating the activity *A*, activity *B* would be the selected one, and if the second branch was selected, the same thing would happen and activity *D* would be the one selected.

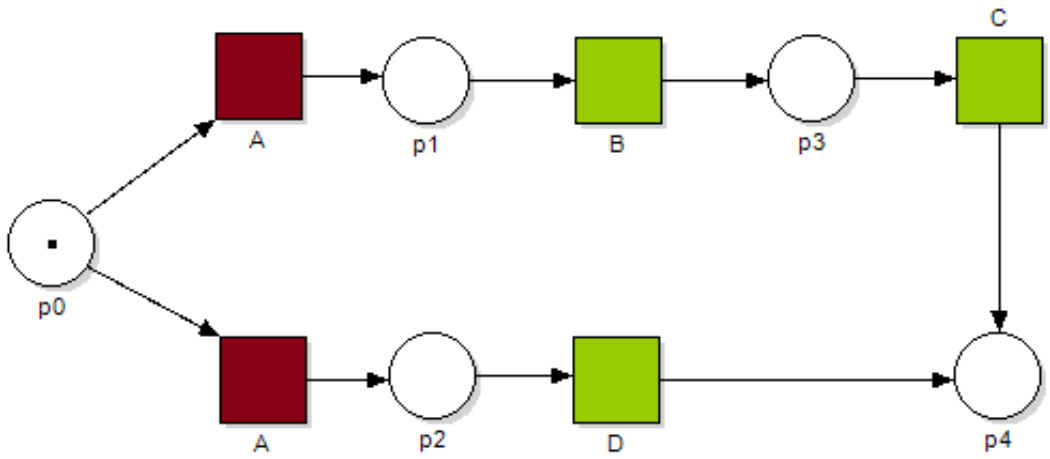

Figure 3.4: Duplicate Activity.

3.2.3 Loops

The last challenge that will be discussed related to decision point training are loops. Figure 3.5 shows a model with two invisible activities (represented in red) and the 3 red circles **A**, **B** and C, that indicate decision points that may have loop related problems.

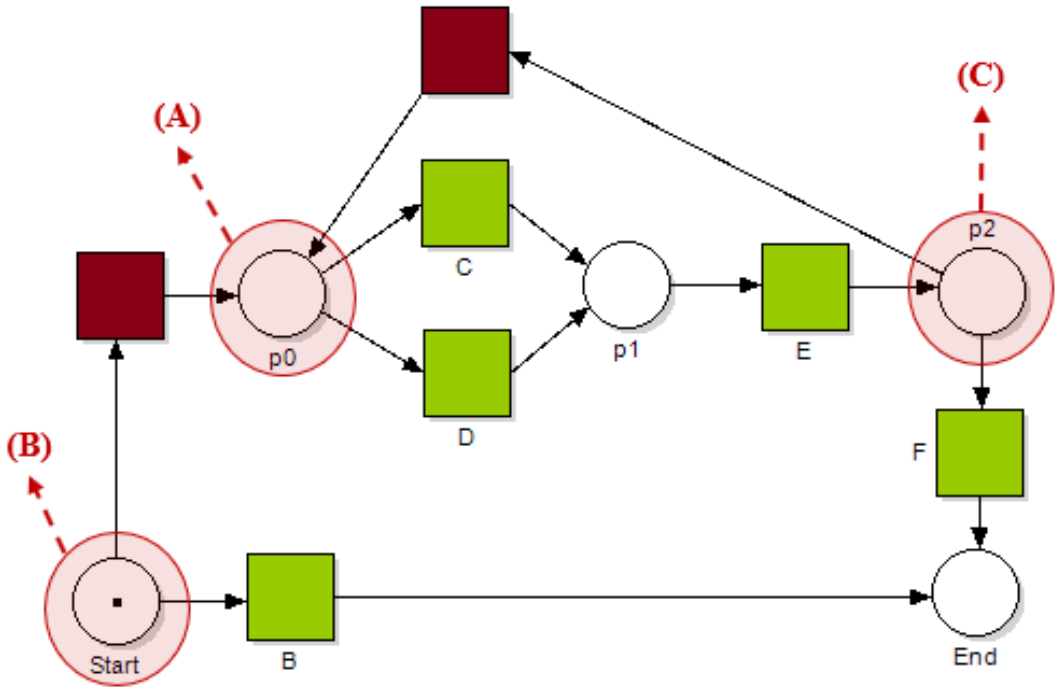

Figure 3.5: Loops.

Decision points contained in a loop (A): When training the decision tree, it was expected that each process instance generated a single training example. However, having the decision point inside a loop makes it possible to have multiple occurrences of it associated with this single process instance. Since every choice made in a decision point is relevant for our analysis, it is important to use all occurrences in the logs to train the decision tree.

Decision points containing a loop (B): Even though a log may contain multiple occurrences of C and D, only their first occurrence is related to the decision point *Start*. All the others are only related to the decision point *p0*.

Decision points that are loops (C): Since the decision point *p2* is only reached after the process passes once through either C or D, it should not be counted as a training example for this decision point. Only the following occurrences of C or D should be used to train this decision point.

The examples above show that, when there are loops in our model, it is not enough to consider the occurrence of an activity to consider it as a training example for a given decision point. It is necessary to observe the order of the activities to correctly classify them, for example, only occurrences of C and D after occurrences of activity E should be considered as a training example for the decision point *p2*.

4
PUC Decision Miner

A *decision miner* was developed in Python. It requires a *.pnml* file, containing the Petri net model, and a *.xes* file, containing the event logs. As was described before, there are 3 issues that need to be addressed, they are: *Duplicate Activities*, *Loops* and *Invisible Activities*.

Duplicate Activities and *Loops* only need the Petri net model information to be treated whilst *Invisible Activities* need both the model and the event logs. Once those issues have been addressed, the last thing that will be discussed is the training of the decision tree.

4.1
Addressing the Issues

4.1.1
Duplicate Activities

To treat *Duplicate Acitivities* it only requires a loop through the transitions checking whether another one with the same *label* (activity name) has already been visited as can be seen in the algorithm 1.

Algorithm 1 Treating Duplicate Activities

```
1: function TREAT_DUPLICATE_ACTIVITIES(petri_net_model)
2:     for transition in petri_net_model.transitions do
3:         transition.duplicate ← FALSE
4:     end for
5:     visited_labels ← []
6:     for transition in petri_net_model.transitions do
7:         if visited_labels.contains(transition.label) then
8:             transition.duplicate ← TRUE
9:         else
10:            visited_labels.add(transition.label)
11:        end if
12:    end for
13: end function
```

4.1.2 Loops

To treat *Loops* we will identify their starting place and their complete path. This information will later be used when training the tree. In order to find the starting place of the loop, we used a recursive algorithm, as shown in algorithm 2. Instead of using the place's *label* through the recursion, the *key*, which is an identifier of the place, is used. The *key* is a better suit for the recursion because there can be places with the same *label*, thus using the *key* will allow us to distinguish between them.

In the algorithm, when it finds the origin of the loop it creates a *key-value* structure that links the *path* of the loop to the *key* of the starting place. This structure is them added to an internal dictionary[1] of the place and returned.

Algorithm 2 Place's Recursive Function to Find Loops

```
1: function FIND_LOOPS(visited_places_keys)
2:     if self.key in visited_places_keys then
3:         path ← visited_places_keys.from(self.key)
4:         add_loop(self.key, path)
5:         return {self.key : path}
6:     end if
7:     loops_found ← {}
8:     for activity in self.output_arcs do
9:         for place in activity.output_arcs do
10:            loops ← place.set_loops(visited_places_keys.add(self.key))
11:            if length(loops) > 0 then
12:                for loop_key, loop_path in loops do
13:                    if loop_key == self.key then
14:                        continue
15:                    end if
16:                    add_loop(loop_key, loop_path)
17:                    loops_found.add(loop_key, loop_path)
18:                end for
19:            end if
20:        end for
21:    end for
22:    return loops_found
23: end function
```

4.1.3 Invisible Activities

To treat *Invisible Activities*, we will loop through the event logs and find all the distinct activities on it. Once all the distinct activities have been found,

[1]The addition of a new-found path to the dictionary, represented as *add_loop*, was summed up for easier understanding. The *loops_found.add* was also simplified.

we will check if the model's transitions are inside this list. If they are not, they will be marked as invisible. This is represented in algorithm 3.

Algorithm 3 Treating Invisible Activities

```
1: function TREAT_INVISIBLE_ACTIVITIES(petri_net_model, event_logs)
2:     visible_activities ← []
3:     for activity in event_logs.activities do
4:         if !visible_activities.contains(activity.name) then
5:             visible.activities.add(activity.name)
6:         end if
7:     end for
8:     for transition in petri_net_model.transitions do
9:         if visible_activities.contains(transition.label) then
10:            transition.invisible ← FALSE
11:        else
12:            transition.invisible ← TRUE
13:        end if
14:    end for
15: end function
```


4.2 Training the Decision Tree

To train the decision tree consistently, we will always use the id of the activity instead of its name to avoid cases that are *duplicate activities*. Due to this, we will have to later match the id to the activity name or path to know to what branch it refers to.

Even though training the decision tree can be an automatic process, it can create rules that do not represent accurately the real business rule. The tree will always choose the attribute that splits the data most efficiently in spite of its meaning in the process. A person with a deeper understanding of the process and data attributes can enhance the tree by filtering the attributes that are not supposed to be used in the rules.

One example of such case is represented in figure 4.1. It shows an automatically generate tree for place *p0* of the process shown in figure 3.1. As it can be seen, instead of creating a tree with two rules, one for *PolicyType* and one for *Amount*, the algorithm used the attribute *CustomerID* to split the data into two groups. The rule created by the tree is not even close to the actual business rule shown in figure 3.1. This wrong rule is a big issue because once we try to predict the outcome for a new customer, the tree will always choose the *Check policy only* activity, even if the client has a normal policy. In fact, this simple case could also show another similar issue when creating the tree. If each account had an account manager identified with an attribute

AccManagerID and a single manager was responsible for all the premium accounts, the algorithm could choose the *AccManagerID* as one of the rules. Analogous to the *CustomerID* case, once this manager stops handling premium accounts, the tree rule will always choose a single path for all accounts. Hence, having a human intervention when creating the tree can bring benefits to the final outcome.

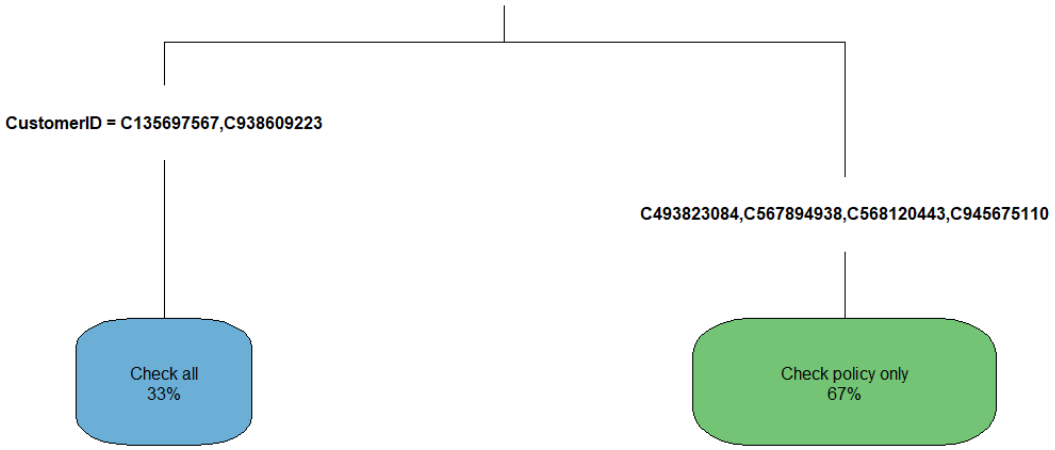

Figure 4.1: Automatically generated tree that uses a data attribute that is not meant to be used.

5
Examples

To finalize this dissertation, we will test the miner with an artificial case and a real world one. As was explained before, in order to create more accurate rules there will be human intervention to filter out some of the data attributes.

5.1
Artificial Case

The artificial case's Petri net is represented in figure 5.1. This case is based on the one present in the paper of Van der Aalst (1) and the dataset used was obtained on the website of the Prom extension (10). In this case there are 3 decision points: *p0*, *p2* and *p3*.

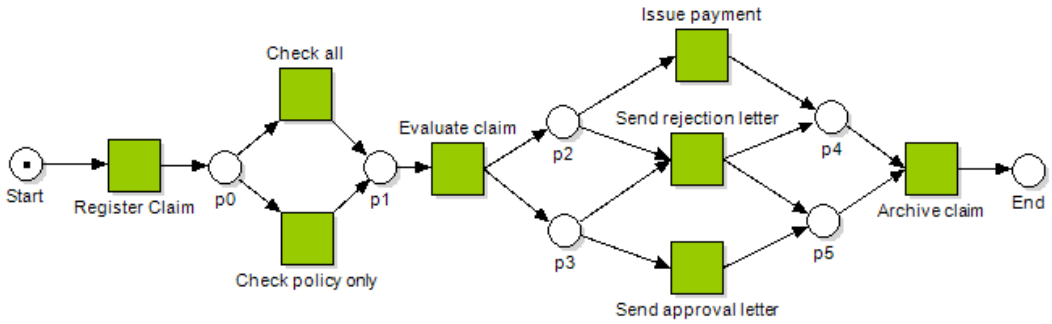

Figure 5.1: Artificial Case - Petri Net

The rules created by the tree of place *p0*, figure 5.2, are similar, but not equal, to the actual rules. The difference is that the actual business rule separates the *Amount* at the value of 500 instead of 525. This difference is negligible and happens due to the lack of data in the training set. For example, if we had only two training samples, one with the *Amount* of 600 and another with 450, the rule that would be created would use the value of 525. This happens due to the fact that the decision tree cannot know where is the exact splitting point between those two values, so the tree makes a guess at the middle of those two values, which is 525, and uses this value in the rule.

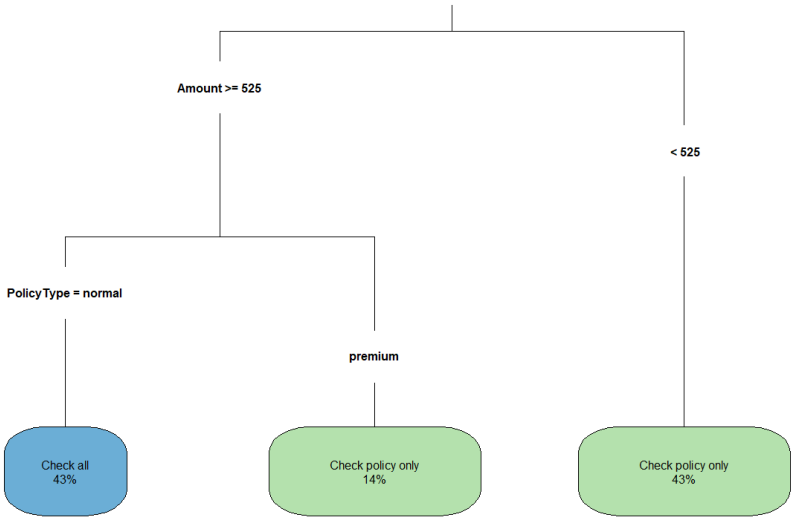

Figure 5.2: Decision Tree for place *p0*

The trees for places *p2* and *p3* are similar, since they are complementary[1], and are shown in figures 5.3 and 5.4, respectively. Both rules are simple and are equal to the actual rule.

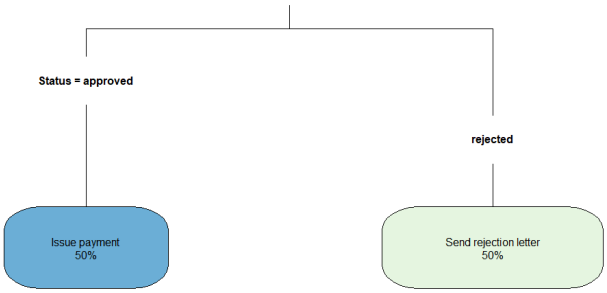

Figure 5.3: Decision Tree for place *p2*

5.2 Real World Case

The real world case that we will be using as the validation case is the one used at the BPI 2017 challenge[2]. Instead of using the complete Petri net, we are using a reduced one that was used by our team in the competition. The simplified Petri net is shown in figure 5.5. As it can be seen, there are many

[1] A decision in *p2* also sets the outcome for *p3* and vice-versa.

[2] The BPI (Business Process Intelligence) Challenge 2017 was a competition sponsored by Minit and Celonis that they provided participants with a real-life event log and asked them to analyze these data using whatever techniques available, focusing on one or more of the process owner's questions or proving other unique insights into the process captured in the event log.

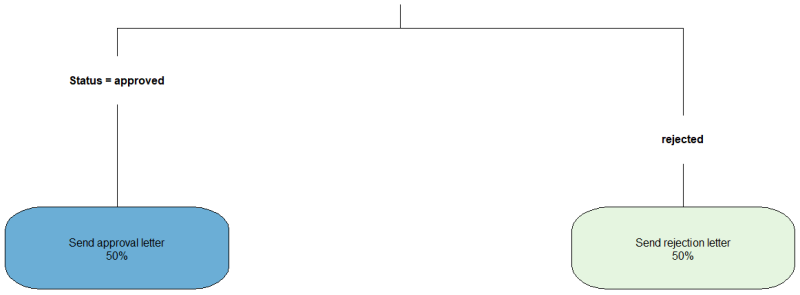

Figure 5.4: Decision Tree for place *p3*

transitions named as *tau*. All of them are a summary of activities that have been simplified for an easier analysis of the net.

In the Petri net, there are 4 decision points: *p4*, *p7*, *p8* and *p9*. Despite the number of decision points, only *p8* can be analyzed in this simplified net. This happens because all *tau* transitions are invisible activities[3] and because the transition *O_Create Offer* is a duplicate activity. Due to this two facts, the decision points *p4*, *p7* and *p9* end up with only one route available, eliminating them from the analysis.

In figure 5.5 it can be seen that there are three possible outcomes for the decision point *p8*: *A_Denied*, *A_Pending* and *W_Call incomplete files*. If the route taken is through *W_Call incomplete files*, it can either go through *p8* again and be used as new training data or it can go through a different route and end up at *A_Cancelled*.

The decision tree created for *p8* is shown in figure 5.6. There are two things that should be pointed out before we start the analysis of the results. First is that all clients that do not have a *CreditScore* are set with the value of zero, so the split in the decision tree reflects a segregation of clients that have a *CreditScore* and clients who don't. Second is that the *A_Pending* actually reflects an approval for the offer made.

Table 5.1 summarizes the results obtained in an in-sample test for our tree. As it is presented, the tree has a great accuracy, over 80%, for both *A_Denied* and *W_Call incomplete files* and a really low accuracy, about 10%, for *A_Pending*. If we take a closer look into the data and the tree we can get some clues to why this is happening. The first split on the tree, using the *CreditScore*, segregates clients who have a *CreditScore* and those who don't. Since there is not another split inside the group of clients that have a *CreditScore*, there can only be two conclusions: either all, or at least most, of the clients go only through one path (*W_Call incomplete files*), or there

[3]Since *tau* transitions are groups of activities they are not available inside the event logs.

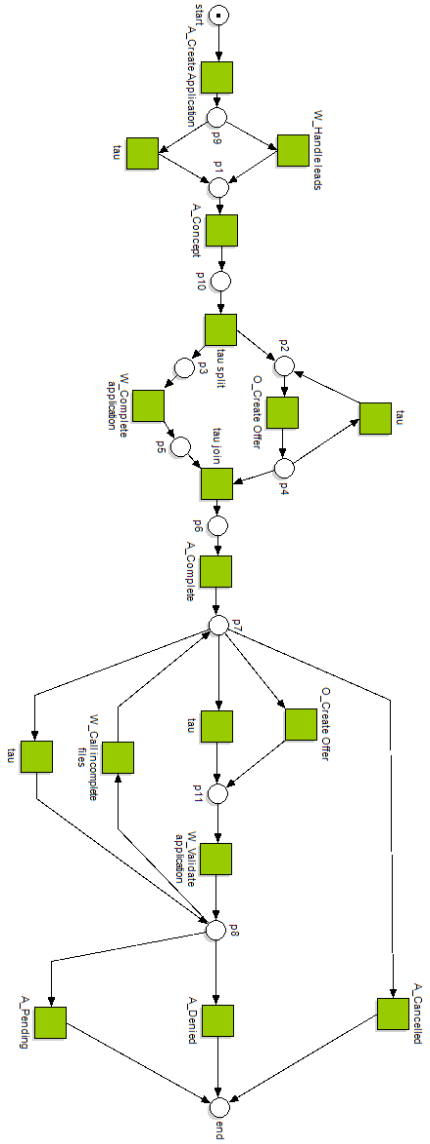

Figure 5.5: Real world case *p0*

is not much difference between groups that go through different paths. Table 5.2 shows the outcome for clients who have a *CreditScore*. It can be seen that about 54% are asked for more data (*W_Call incomplete files*) and about 45% are given the credit (*A_Pending*). Thus, invalidating the first possibility.

As we can see, basically, clients with *CreditScore* cannot be separated in a group that is missing information (*W_Call incomplete files*) and one that gets the credit (*A_Pending*). This implies that there may be some things that are not represented in the event logs that affects the choice to give credit or not. It could either be some missing documentation, such as mailing address, or even some intangible aspect, such as the confidence that the bank has on the client. For instance, a call from the account manager saying good things about the client asking for credit probably enhances his chances of getting credit.

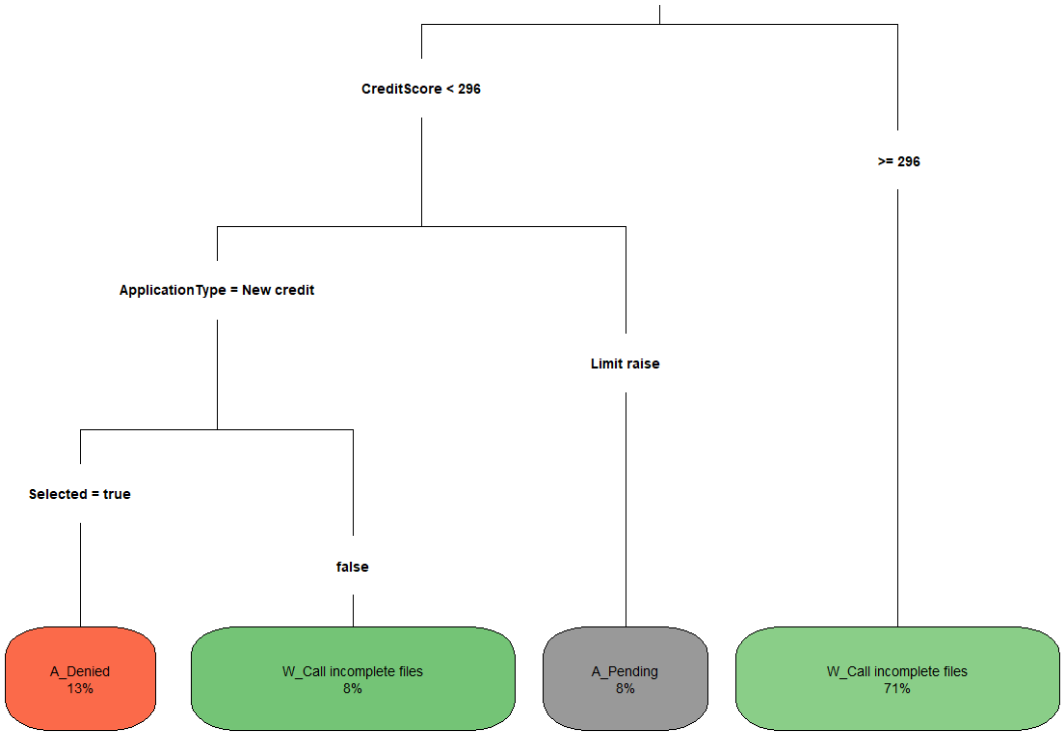

Figure 5.6: Decision Tree for place *p8*

Another possible variant not inside the data is the date that the account was open, which could indicate the clients "fidelity" to the bank. Also, there could be some of the so called social engineering at work that is also a non-tangible aspect of the negotiation.

Even though we could go further with this analysis, it is not in the scope of this work to delve much into detail about this specific real world case, but rather use it to validate our decision miner. The analysis already presented has shown its value and how it could be used to better understand the dataset used and the outcomes. In spite of the tree being unable to make good decision between *W_Call incomplete files* and *A_Pending*, we were able to explain the reason of this behavior. The result observed in this study only reinforces the idea that, still, some sort of human interaction during this part of the process is required.

5.3 Other algorithms

In spite of explaining in section 3.1 why some of the algorithms are not the best choice for this work, we think it is important to test whether some of them have a better performance than the decision tree. Specifically, we are going to test the *random forest* and the *naive bayes*. For our test, we used the

real world case data and did a cross validation test using 5 groups. The results are shown in table 5.3.

As it can be seen, overall, the decision tree had the best performance 54.9%. The results also show that both the random forest and the nayve bayes had a hard time separating *A_Pending* and *W_Call incomplete files* outcomes. This strengths our point that there is probably information not represented in the data that differentiates those two types of outcomes. With this test we may conclude that, for this real world case, the decision tree is the best choice[4] between the three algorithms.

5.4 State of the art

As far as we are aware, the state of the art of decision mining is described in the work of Van der Aalst (1). Since it is the state of the art, we have implemented our miner based on his work (1). It was tested using the same test case as his work, described in section 5.1, and also using a real world case, described in section 5.2.

In our work, besides showing some of the key aspects of the implementation, we have also discussed the results obtained by our miner in the real world case. From our analysis of the results, we can see that, in addition of being a tool that automates the decision-making process, the decision miner's decision tree can be used as a starting point in data analysis[5].

5.5 Future Work

There are lots of things to be done when it comes to decision mining. Just in this paper there are many possibilities of studies, specially if we discard the assumption that the algorithm must represent business rules, like: testing with other algorithms, using enhancers for the algorithms, like Gradient Boost for the Decision Tree, and combining two or more algorithms. Some of these approaches are likely to increase the accuracy obtained in our tests.

Another work that could be done is to solve the challenges that we stated in section 3.2. This would be a great advance in decision mining. Just in our real world test case 3 out of the 4 decision points were eliminated due to one of more of these challenges.

[4]It is the best choice considering the overall result, if we were interested in predicting *A_Denied* and were not worried about representing business rules, the Naive Bayse would be the best one.

[5]The whole analysis done in section 5.2 started and was based in the decision tree.

Table 5.1: In-sample test accuracy.

Output	**Accuracy**
W_Call incomplete files	82.8%
A_Denied	81.3%
A_Pending	11.5%

Table 5.2: Outputs for clients that have *CreditScore*.

Output	**Percentage of cases**
W_Call incomplete files	54.8%
A_Pending	45.2%
A_Denied	0.0032%

Table 5.3: Algorithms performance in a 5 fold Cross Validation test.

Output	**Decision Tree**	**Random Forest**	**Naive Bayes**
W_Call incomplete files	82.5%	67.9%	32.7%
A_Pending	12.1%	30.0%	48.6%
A_Denied	81.3%	65.8%	99.1%
Total	**54.9%**	**52.9%**	**44.6%**

6
Conclusions

In this dissertation, the state of the art of decision mining was reviewed based mostly on the work of van der Aalst (1) through the development of a decision miner in Python.

We've started by presenting the basic concepts of Petri nets, since it is the base model used by the miner. Afterwards we've presented the challenges associated to decision mining (*invisible activities*, *duplicate activities* and *loops*) and ways to solve them. Then, it was shown that between several supervised machine learning algorithms, the decision tree is the best option for this task due to its flexibility, when it comes to data handling, and intuition[1].

To finalize this dissertation, we've tested the decision miner using a real world case. We could see that the miner was capable of creating a decision tree for the decision point and them we've made an analysis of the result. During this analysis we were able to explain why the result was not as good as we expected. This result also reinforced the fact that, still, some level of human intervention is required during the final steps. It is also important to remark that, even though the decision miner was created to only automate the decision-making, the data frame structure used, as well as, the decision tree, can help us get some insights about the data and also help in other types of data analysis just as the ones presented in the study.

[1]The association that can be made between the trees splitting rules and the actual business rules.

Bibliography

[1] A. ROZINAT AND W.M.P. VAN DER AALST. **Decision mining in business processes**, 2007.

[3] W.M.P. VAN DER AALST, A.J.M.M. WEIJTERS, AND L. MARUSTER. **Workflow mining: Discovering process models from event logs**. IEEE Transactions on Knowledge and Data Engineering, 16, Issue 9:1128–1142, 2004.

[4] WEN, L.. **Process mining: Overview and outlook of petri net discovery algorithms**, 01 2009.

[5] PETRI, C.A.. **Kommunikation mit automatenl**. Ph.d. thesis, Institut für Angewaldte Mathematik der Universität Bonn, Bonn, Deutschland, 1962.

[6] T. MURATA. **Petri nets: Properties, analysis and applications**. Proceedings of the IEEE, 77, Issue 4:541–580, 1989.

[7] ROZINAT A., VAN DER AALST W.M.P.. **Conformance testing: Measuring the fit and appropriateness of event logs and process models**. In: BPM'05 PROCEEDINGS OF THE THIRD INTERNATIONAL CONFERENCE ON BUSINESS PROCESS MANAGEMENT, p. 163–176, Springer, Berlin, Heidelberg, 2006.

[8] W.M.P. VAN DER AALST. **Business alignment: Using process mining as a tool for delta analysis and conformance testing**. 10, Issue 3:198–211, 2005.

[9] W.M.P. VAN DER AALST. **Decision mining in prom**. In: BPM'06 PROCEEDINGS OF THE 4TH INTERNATIONAL CONFERENCE ON BUSINESS PROCESS MANAGEMENT, p. 420–425, Viena, Austria, 2006.

[10] A.ROZINAT, W. V. D. A.. **Decision miner**. Prom Decision Miner, 2018. Acessed at: July of 2018.