Deliverable 3. Knowledge Discovery results.

Evaluation and interpretation.

Grupo de Sistemas Inteligentes

Departamento de Ingeniería de Sistemas Telemáticos

Universidad Politécnica de Madrid.

Project Report

Madrid, October 2012

Authors:

Adrián Pérez Orozco

Álvaro Carrera Barroso

Carlos A. Iglesias Fernández

Executive Summary

This document describes the main knowledge discovery procedure performed for the *Trainmining* project. After the already described and finished steps of goal definitions, data analysis and preprocessing, this step focuses on the data mining procedures themselves: the actual extraction of the knowledge from the available databases.

All the existing data mining algorithms have been analised and classified in order to find the most appropriate for our purposes. After this survey, we selected the best method to approach our problem: searching for association rules based on frequent patterns.

This method consists on finding frequent patterns in our databases in order to be used as predictive rules. Different algorithms already exist to mine these frequent patterns, of which we will select the most advanced and efficient one: cSPADE. In order to build useful prediction rules, two factors have to be taken into account: temporal adequation (patterns that do not run for too long or inadequate periods) and their actual precision (removing the *noise* introduced by events which are frequent but do not offer good predictive information).

In order to being able to adequately extrapolate our acquired knowledge to periods not covered by our historical data, we must try and remove the specific characteristics of these databases. For this purpose, a special validation method will be used: the *k-fold cross validation method*. This method separates the database into k different learning and result sets and iterates the whole learning/validation procedure in all of them. This way, the result is a system with an average performance for all the different parts of the database, which avoids overfitting of our predictive rules for any specific dataset.

Finally, an analysis of the results is gathered in this document, which allows us to see at first glance the kind of results obtained and their performance rating. The functionality of the acquired rules is also illustrated with examples of operation during selected periods of time.

Contents

List of Figures

List of Tables

1 Introduction

In previous stages of our project we have performed preliminary analysis on our data (mostly statistical) as well as the necessary preprocessing to apply different learning techniques in the following stages. Once we have completed these tasks, it is now time to perform the techniques from which the actual knowledge will be obtained.

This step, usually refered to as *Data Mining* is the most important in the whole *knowledge discovery* procedure. Although data analysis and preprocessing do have a big impact on the quality of the results we will be able to achieve in the end, the choice of an appropriate data mining algorithm is essential for the whole procedure to work.

2 Survey on data mining techniques

In order to find the most adequate techniques for knowledge discovery in our project, we will start making a general survey on all the available techniques. Due to the vast amount of already existing implementations available for each of the different data mining categories, we won't have to make an implementation from scratch but adapt one of the already existing implementations to the characteristics of our problem by setting the necessary constraints.

To begin with, we will describe the categories in which all the different algorithms are usually classified. This classification is usually made as follows:

1. **Classification:** Consists on finding functions to map items into already existing classes according to their parameters or characteristics.

2. **Regression:** Algorithms aimend to learn functions to predict the value of some variables from other variables or previous values of the same one.

3. **Segmentation:** Consists on finding a set of clusters or segments in which the existing items can be categorised.

4. **Summarization:** Algorithms used to find compact description for existing data items.

5. **Association:** Learning significant relations or dependencies between different variables.[8].

6. **Sequence analysis:** Finding frequent sequences or episodes in data [9, 4].

Segmentation and *summarization* algorithms seem to be obviously out of the question, as their functionality differs completely from the objectives we want to achieve in our project. *Regression* algorithms are inadequate as well due to the nature of our data. We do not count on variables whose future value we want to predict.

Classification algorithms might not seem like a good choice at first, as we do not have the need to classificate the events into any existing categories. However, if we define an appropriate set of categories and an appropriate model of items to classify, classification algorithms can be actually useful for our tasks. If we model the current events as the *item* to classify, and the possible categories as the possible events which can happen in the future, we can actually classify the current situation (defined by the events which have already happened) into possible categories each defining what would happen next.

However, *sequence analysis* seems to be the most appropriate category at at first glance: we count on historic data from the past and we want to find event patterns from which we can foresee future events. Patterns which are frequent offer useful information in this direction. If we know a set of several events which often happen in the same sequence, we can expect the later events in the sequence to happen once we have already seen the first ones.

These sequences offer a good starting point, but it is important to realize that *frequency* in a pattern refers to the total of events happening during the observated period, and does not indicate in any way the *probability* of the last events in a sequence to happen once the first ones have been acknowledged.

In other words, we want to obtain predictions with a high *probability* rather than a high *frequency*.

In this direction, we will use the approach of the *association* algorithms. Using frequent sequences as an starting point, we will build *association rules* which will relate events in the form of boolean variables.

Therefore, this is the most appropriate approach, as it addresses our problem directly without the need of transforming it into a different kind of situation.

3 Acquisition of association rules

As we mentioned previously, the most appropriate way to address our problem is the construction of a model based on association rules. This approach consists on building association predictive rules using frequent secuences in our available data as a base.

First of all we must therefore obtain all potential sequential information (patterns) from our database, as mentioned. These sequences will be of the form $\{A, B\} \longrightarrow \{C, D\} \longrightarrow \{E, F\}$ and will serve as a basis to build candidate *association rules*. This step is explained with further details on section 3.1.

Using the frequent sequences obtained from the first step, we can build *candidate association rules*. Candidate association rules are of the form $\{A, B\} \xrightarrow{T} \{C\}$. It is important to notice that in these rules we are putting additional temporal information (a distance of T between terms). This temporal information is not implicit in our previous temporal sequences, but can be inferred from the conditions used on the process to obtain them. This will be explained in detail in section 3.2.

Finally, we must check which of these *candidate association rules* are actually good predictive rules, and obtain a precise measure of *how good* they are. Specifically, we will measure the *certainty* (precision) of the predictions made by using these rules and the *amount of events* (recall) they are able to predict. This step will be explained in detail in section 3.3.

The whole procedure can be summarized in the following steps:

1. Mining frequent sequences
2. Building candidate association rules
3. Validate the obtained rules

These steps will be further described in the following sections.

3.1 Obtaining frequent sequences

The first step for our chosen approach is to find frequent sequences in our datasets. Frequent sequences will be good candidates from which we can be able to obtain association rules – if there is an unknown causal relation between two events, they will appear together considerably often. Several algorithms have been developed in the past in order to approach this task of finding frequent sequences. Some examples are the *GSB* algorithm and the *SPADE* algorithm, being the later an alternative to the first with better performance and results.

The procedure of finding frequent sequences in a dataset mainly consists on an iterative analysis of all the possible combinations of elements of the database in sequences. For example, the GSB algorithm can be roughly described as follows:

1. All the possible items (events) of the database are counted. These elements can be seen as sequences of length 1, which will be subsequences of any other larger sequence.
2. All the possible length 2 candidates are generated, as combination of length 1 sequences
3. The database is scanned to calculate the support of generated length 2 candidates
4. Length 3 candidates are generated as addition of length 1 sequences to length 2 sequences whose support is higher than a given minimum
5. The process is repeated till no candidates have high enough support

In other words, the candidates are created in a tree fashion, by adding length 1 sequences (possible terms) to elements in a level. If a branch reaches the minimum required support value, it stops growing, as adding more terms to the sequence will make it more specific and necessarily less frequent.

The support of a sequence is calculated as the number of times it happens in our dataset. The support is usually expressed as a percentage of the whole amount of sequences in the database, but it is important to note that this parameter is not related in any way with the confidence or precision of any prediction we might do with the given pattern. A deeper approach on this issue will be described later in this document.

More information on GSB and SPADE algorithms can be found in [6, 9, 2]. Although It is not our priority now to study these algorithms in depth, previous work shows a better performance for SPADE than for GSB, and therefore it will be our algorithm of choice for our work. Furthermore, SPADE implementations are conveniently available in R libraries, which will allow us to easily execute the algorithm on our datasets.

3.1.1 Defining constraints

One of the main problems we find when we look for frequent sequences in our database, is that not any sequence – although frequent – is useful for our purposes. In the end our goal is to make predictions, for which obtaining these frequent patterns is useful. However, our project context – and sometimes common sense – may put additional conditions on *which* kind of predictions are useful; and therefore, *which* kind of patterns we must look for.

For instance, due to the characteristics of our systems, it might not be possible to perform maintenance tasks in short periods of time. Sequences showing us that A always breaks within one hour after B breaking might not be useful even if we can obtain a very high certainty of that prediction. If we need to buy new pieces to fix B, and those pieces are usually delivered in terms of weeks, knowing that B will break one hour before it breaks would not give us any advantage over waiting for it to break and notice without

any prediction.

In other words: we need to define temporal constraints in order to obtain sensible predictions[7]. These constraints are the following:

Observation time. We must define for how long we want to take events into account. For example, our predictions for tomorrow will be most likely be made taking into account today's events, as those from last week are less likely to be related with those happening in the short future.

Minimum gap. This is the minimum amount of time in which we want to predict events. For instance, a gap of 0 would result in predictions for events simultaneous to the observed ones, while a gap of 1 would result in predictions only for the following observation periods.

Maximum gap. The maximum amount of observations between events in our sequences. By fixing it to the same amount as minimum gap we can obtain sequences with a fixed gap between events.

Maximum window. This is the maximum temporal length for our sequences. It is important to stress that this is the length of the whole sequence, while the gap is the separation of events within a sequence.

Given these constraints, we have sequences with the following structure:

$$\{A, B\} \xrightarrow{T_1} \{C, D\} \xrightarrow{T_2} \{E, F\}$$

Where $mingap \leq \{T_1, T_2\} \leq maxgap$, and $T_1 + T_2 \leq maxwin$. It is important to remark that these temporal conditions are not inherent to the sequences obtained by the SPADE algorithm. As we mentioned in section 3.1, sequences are built from all the possible combination of events, and then their support is calculated by checking how many times that sequence appears in the database. It is in support calculation where these constraints apply, but the candidate sequences do not contain any temporal information at all. We will only know that their values will be comprised within the ranges we have defined.

In this sequence we have three terms with two events each. In order to build association rules, it is very convenient to limit the number of terms to two – a single *antecedent* and a single *consequent*. Furthermore, it is very convenient to limit the number of events to one, in order to make individual predictions for each of the events (which may have, for example, different certainties).

Therefore, the previous example sequence can be divided in three subsequences of two terms:

$$\{A, B\} \longrightarrow \{C, D\}$$

$$\{C, D\} \longrightarrow \{E, F\}$$

$$\{A, B\} \longrightarrow \{E, F\}$$

And, furthermore, each of them can be divided into two subsequences with only one item in the last term:

$$\{A, B\} \longrightarrow \{C\}$$

$$\{A, B\} \longrightarrow \{D\}$$

$$\{C, D\} \longrightarrow \{E\}$$

$$\{C, D\} \longrightarrow \{F\}$$

$$\{A, B\} \longrightarrow \{E\}$$

$$\{A, B\} \longrightarrow \{F\}$$

These sequences are in fact subsequences of the original one, and therefore their individual support values will always be higher than the support original one. This means that these subsequences will already have been obtained as frequent sequences by the SPADE algorithm, without the need of performing division on the longer sequences. As a result, we can simply drop the sequences whose length or complexity is inconvenient for our purposes, as their shorter subsequences will be already found by SPADE.

This results in additional length constraints:

Maximum terms. The maximum number of terms in the sequence. In our previous example, we should have set it to *2*.

Maximum items per term. This condition defines the maximum amount of items in each of the terms of the sequence. This is *not exactly* what we wanted to achieve with our second division of sequences, as we only want to apply this condition to the last term and not to all of them. In our previous example, we would have to set this limit to 1 but only for the last term of the sequence.

Both these groups of constraints must be applied within the process of the algorithms which will obtain the frequent sequences from our database. The length constraints will limit the construction of *candidate sequences* and the temporal constraints will put conditions to the calculation of *sequence support*. Its implementation must therefore be made into the sequence mining algorithms.

An extended version of the SPADE algorithm has been developed to include some of these constraints which were not contemplated by the original SPADE implementation. The *cSPADE* algorithm[7, 5] provides an implementation taking into account all these mentioned conditions. It is available as an R implementation under the library *arulessequences*[**?**]. The only constraint which we are not directly able to define as a cSPADE condition is the maximum number of items in the last term of the sequence, as the condition imposable as a cSPADE parameter is a maximum number of items for *all* the terms. This will have to be addressed at the time of building the association rules, as we will see in section 3.2.

3.2 Building candidate association rules

The next step in our knowledge discovery process is the construction of potential rules which could be applied to the prediction of events in our system. As we have mentioned several times, a rule is a sentence of the form $A \xrightarrow{T} B[p]$, where A is the antecedent (maybe containing several events), B is the consequent (also maybe containing several events), T is the time period between A and B, and p is the probability of this rule being true.

In order to transform our already available set of frequent sequences into rules of said form, first we must build all the candidate rules which can result from the obtained sequences. As we mentioned in section 3.1, *cSPADE* allows us to define certain constraints in order to obtain suitable sequences to build rules afterwards. Said constraints are:

- Maximum of two terms per sequence. We will set this to *2*, as mentioned in section 3.1.
- Gap between terms (T) comprised between mingap and maxgap. As we are working only with two terms, this defines also the maximum length for the sequence.
- Maximum number of events in a term. We cannot however define independent limits for each of the terms, as we would like

It is important to remember that our data is, at this point, divided into *observations*. Observations are discrete periods of time in which we group events. When we speak of *gaps* and *temporal lengths* we are always speaking in terms of observations, and therefore the real temporal conditions will depend on the length we defined for our observations.

In order to achieve an exact value for T, instead of having to work with ranges, we will set the maximum gap and the minimum gap to the same value. If we want, however, to find rules for a larger range of T values, we can iteratively repeat this process increasing its value. This will provide us with independent rules for each value of T, which will allow us to evaluate and validate them independently, resulting in better results.

We will therefore have sequences of the following form:

$$\{A, B\} \longrightarrow \{C, D\}$$

As we mentioned before, we will divide these into subsequences with a single item on the last term of the sequence. More exactly, we will disregard sequences that do not comply this condition, as their valid subsequences will also have been detected by cSPADE. The result will be the following:

$$\{A, B\} \longrightarrow \{C\}$$

$$\{A, B\} \longrightarrow \{D\}$$

In order to convert these sequences into rules, we simply need to assign them a T value and an associate probability p. The definition of T is quite immediate, as we have defined exactly the gap we want to have in sequences by setting maxgap and mingap to the same value. The probability p will be calculated in the next stage, and will be the factor deciding whether *candidate rules* become actual *prediction rules* or not (as well as a very important performance factor and predictive information).

At this step, we must therefore only gather those sequences which fall into our conditions and give them a temporal value T. Simple as that, the process mainly consists on subsetting tasks performed with simple R scripts, which will give us the following:

$$\{A, B\} \xrightarrow{T} \{C\}$$

$$\{A, B\} \xrightarrow{T} \{D\}$$

At this point we have transformed the initial frequent sequences into candidate association rules. The next step is to check which of these candidate rules are actually useful for predictive purposes. This leads to the last steps in the construction of this model: *evaluation and validation*.

3.3 Evaluation and validation

Once we have obtained a set of candidate rules, we must evaluate them to discern which of them are good enough to make it into the final predictive rule set. For this, we must perform two final tasks: defining evaluation criteria and applying a validation method.

3.3.1 Evaluation criteria

In order to evaluate and validate our rules, we must first define the evaluation criteria. This is, what will make a rule better or worse than others.

The main goal in our project is the implementation of tools which will give us a prediction using current events as its input. As a first thought, we can immediately think of evaluating our predictions by how true they actually are. We can measure the *accuracy* of a prediction rule system easily by checking how often it becomes true and how often it does not. This is an important factor to take into account, but is however not the only significant indicator of the quality of the system. In a limit case in which we only attained a trivial but highly accurate rule which gives valid but trivial predictions all the times, we would have an accuracy of 100%, while the overall quality of the system would be none. We must actually check not only the accuracy of our predictions, but also their relevance against the whole situation.

Therefore, we will need two different evaluation parameters: one related to the accuracy of our predictions, and other related to the fraction of events we are able to predict[3]. In first place, we will define *precision* as the fraction of our predictions which are accurate. In the case of evaluating a rule against a test set, $P_{accurate}$ would be the number of times when both the antecedent and consequent of the given rule have happened within the stipulated time window; while P_{total} would be the number of times when the antecedent of the given rule has happened, whether the consequent has or has not happened. Prediction can be as well calculated for a whole rule set, or for any kind of system which gives a predicted event based on other input events.

$$Prec_i = \frac{P_{i,accurate}}{P_{i,total}} \tag{1}$$

On the other hand, we will define *recall* as the relation between events which have successfully been predicted by our system ($E_{predicted}$) and the total number of events (E_{total}).

$$Rec_i = \frac{E_{i,predicted}}{E_{i,total}} \tag{2}$$

Notice that the number of events which have been predicted ($E_{predicted}$) is, in fact, the number of accurate predictions as calculated in the definition of *precision*, ($P_{accurate}$)

In other words, precision is the ratio between accurate predictions and the total number of predictions; while recall is the ratio between accurate predictions and the total number of events.

It is important to notice that in our context, an event can't be *wrongly* predicted. Our prediction can be either true or false, but if we make a prediction of the type $\{A, B\} \longrightarrow \{C\}$ and instead we observe that $\{A, B\} \longrightarrow \{D\}$; it does not mean in any way that we predicted C instead of D, but that our prediction of C was false and we did not predict D. As a result, some other tools generally used to complement values of precision and recall (such as *confusion matrices*) cannot be applied in our case.

Taking a further step, we can merge both indicators in a single one, obtaining a single indicator for a much easier evaluation. Precision and recall are often merged in the called *F-measure*, defined as:

$$F = \frac{(\beta^2 + 1) \cdot Prec \cdot Rec}{\beta^2 \cdot Prec + Rec} \tag{3}$$

where $\beta \in [0, 1]$ balances the importance between recall and precision.

In order to obtain high precision values, we must usually compromise recall and vice versa. Very precise rules will usually require strict conditions, which will reflect situations so specific that there is few probabilities of failure. On the other hand, these strict conditions will only happen a quite limited amount of times, resulting in a low recall value. If we want however to obtain high recall values (predicting a high percentage of the total events) we will be using very general rules, into which most of the situations can fall. More general rules will however result in less precise predictions, as the possibilities that they reflect are much higher, both for situations in which the prediction would be true and those in which the prediction would be false.

In our project, we must look for rules with high precision values, whichever their recall is. In most scenarios, it will be better to count on precise predictions – having the certainty of our predictions being good – than to predict more events but at the compromise of their credibility.

Therefore, we will use precision value as the main evaluation parameter.

3.3.2 Validation method

Once we have defined the evaluation parameters we must calculate the performance of our candidate rules in order to assign precision values to each of them. Precision information is also part of the information we want to give in our predictions, so a proper calculation is of essential importance.

However, if we do this on the same data we have used to obtain this knowledge (our *learning set*) we will obviously obtain extremely good results, as we have already learnt all the patterns happening on that exact data. If we had an ideal, infinite data set with *all* the possible situations that can ever happen in our scenario, we could have learnt absolutely every possible prediction to be made on the system and no future event could be *unexpected* to our new prediction abilities. However, in real systems this is not the case, and it is very likely that patterns and characteristics of the systems vary along time.

Additionally, training our system over a single large set of data can lead to *overfitting*. This happens when our predictive knowledge becomes extremely accurate for the set we have been training on, but performs poorly on any other set of events not contained on our learning set. It is important to avoid overfitting by performing learning procedures in a way that not our whole amount of data available is used at the same time. In this direction, the usage of very large data sets for learning procedures can be very inconvenient. In one hand we might be learning patterns which are exclusive to the specific period we are studying (for instance, we may be trying to obtain knowledge from logs from a specific year which we intend to use for forthcoming years), and when we validate this information, we will obtain unrealistic good performance measures.

In order to make a proper validation of the obtained knowledge, we must separate our data in different sets. One of them will be the *learning set* – over which we will work to obtain our predictive knowledge – and the other will be used as a *testing set* – on which we will test our predictive abilities. This way we will obtain a better validation of our predictive knowledge, as the characteristics of the testing set were not taken into account on the learning process, as would happen for any future set of events.

In order to address this problem, one of the most used methods is the k-fold cross-validation (k-fold CV) method. This method consists on dividing the whole data set in k subsets of equal sizes, using *k-1* of them as the learning set and the kth one as the testing set. Performance results are stored for those specific learning and testing sets and the whole process is repeated a total of k times, until all the possible learning sets/testing sets combinations are obtained.

With this process, we obtain a total of k performance testing results for our model. The important point is that all of them have been tested on sets which were not used for their construction. The overall performance measure is obtained as the arithmetic mean of all the individual performance results.

In some cases, we can even randomize the division of the data into subsets, obtaining different subsets for each process of k-fold CV we perform. In our case, however, we are limited in this direction by the nature of our data, as it is very important to preserve sequential information of our data. Our subsets must therefore be conformed of contiguous observations, and cannot be randomized between different temporal subsamples.

A commonly used value for k is 10. As in our case we will generally work with data sets comprising about a year of historic data, this division will provide learning sets of about 9 months and testing sets of about 1 month, which reasonable when validating predictions in terms of days.

4 Adjusting search parameters

The results of the whole mentioned process will be determined by its execution parameters. In sections 3.1 and 3.2 we already explained all the

possible adjustments we can make in both steps and how they would affect to obtained results.

For this kind of data mining algorithms, what most influences the quality of results is search depth. In other words, they will be determined by how *deep* in our data we are willing (or *able*) to search in order to find our desired information. The deeper we search, the better results we can expect, at the cost of higher need of resources.

In our problem, depth comes defined by two parameters: number of terms and minimum sequence support.

The *maximum number of terms* is simply how long we want our sequence candidates to be. The more items we add to a sequence, the more specific that situation will be, from which we can expect to obtain more precise information. However, the more specific a situation is, the less likely it will be to be extrapolated to usual situations. In terms of computation, the length of the candidates exponentially rises the complexity of the problem and the number of resources needed, and therefore a reasonable limit is to be put on this parameter.

The *minimum support* defines the number of times a sequence needs to have happened to be considered *frequent*. This parameter has already been mentioned in the explanation of the chosen algorithm in section 3.1. A lower minimum support value will offer a higher number of candidates, and therefore a higher number of association rules. However, this parameter drastically affects the computational costs needed to execute the algorithm.

As we mentioned in section 3.1, candidates are built in a tree fashion, with branches growing larger and larger till the minimum support is reached. Both parameters define when branches of the trees stop growing: once the maximum number of terms has been reached, or once the support is not enough for the sequences to be considered frequent. A compromise must therefore be found between both parameters, in order to be able to search as deep as possible.

Minimum support does not have any effect on the obtained rules. Reducing it will generate a higher number of candidates and rules, but their quality or characteristics won't have any relation with this parameter. The *maxi-*

mum number of terms, however, does affect them. Longer rules will usually be much less adequate than shorter ones. If we could predict *everything* just by knowing one of the events which have happened today, it would be much better than having to wait till ten of them happen in order to be able to predict anything. However, it is very likely that we can make good predictions counting on very little information, so longer rules could be expected to have higher performance values.

The decision criteria is then clear for the minimum support: we want to set this to the minimum value which can be handled by our computation capabilities. In terms of number of terms, however, increasing depth will provide *more* results, some of which will however be too long as to be actually useful.

4.1 Determining optimal values for search parameters

In order to fix the maximum number of terms in sequences, we have executed the whole process in a smaller sample of data with a considerably high maximum value of terms. Setting this value to *10*, we could analyse results for sequences of different lengths. It is important to note that although we could expect longer sequences always to be more precise, they will also be much less frequent, so we will much faster reach the minimum support limit obtaining less candidates and probably worse rules. For this analysis, the selected minimum support was of *0.1*.

The results are shown in table 1.

In this test run, no candidates were found with lengths of over 7 terms and support higher than 0.1. We also see that for values of 6, precision starts to decay as fewer candidates are found, with which not very good rules could be generated. We will therefore generally choose a value of 5 as the maximum size of candidates, and up to 7 in cases in which the minimum support can be significantly reduced (smaller or divided data sets).

The choice for minimum support is obvious. We will choose the minimum value our computation capabilities can handle. As this is difficult to determine beforehand, we will follow a trial and error method. Starting from

Antecedent length	Maximum precision
1	0.43
2	0.51
3	0.76
4	0.83
5	0.83
6	0.32
7	—

Table 1: Possible maximum length parameters. Test execution

Item	Details
Processor	Intel(R) Core(TM) i7 CPU 950 @ 3.07GHz
Number of cores	8
Memory	12 GB
OS	Ubuntu 11.10 Server

Table 2: Server details

values of 0.01, we will iteratively increase it when computation fails to finish in a period of 24 hours.

The details of the server used for these operations can be seen in table 2.

5 Data clustering for complexity reduction

6 Results

In this section we will make a deep insight on the obtained data. Our goal was to generate different rulesets for each of our maintenance stations, each of them covering different time windows. Specifically, we have studied three different time windows: one day, two days and one week. This defines the observation period for our predictive work, which means not only the time span we will use as antecedents for our rules but also the time window within which our prediction can occur.

The results are very different depending on which station and time period we chose. In this section we will analyse the results obtained for each of them from different perspectives.

6.1 Description of obtained rulesets

The result of the previous work described in section 3.2 comes in the form of a set of association rules. These *rule sets* contain a list of all the association rules found, along with their performance value calculed by the *K-fold-CV* procedure as defined in section 3.3. The obtained rulesets and the number of rules they contain can be seen in table 3.

These sets contain all the information obtained from the procedure defined in section 3.2. It is important to note that not all of them will be useful in order to implement a predictive system, as their precision is sometimes as low as 5%. The threshold for a rule to be useful needs to be defined by maintenance workers who know the associated costs of maintenance tasks required to handle raised predictions before knowing if they will actually happen. If some event needs a significantly high amount of money to be avoided we will probably want to be *very* confident about our precisions regarding that kind of event before investing resources on preventing it.

However, in general terms we can consider a prediction is good when it has more chances of being a right guess than a wrong one. We will therefore set p=0.5 as the point where rules *start* to be useful. Rules with lower precisions are still provided and can be useful for other analysis tasks or further research, but from now on we will disregard them and focus on the

Station	Time Window	Rules
Albacete	1 day	27522
Albacete	7 days	50281
Antequera	1 day	39214
Segovia	1 day	113
Segovia	2 days	72
Segovia	7 days	133
Sevilla	1 day	8091

Table 3: Size of obtained rule sets for each station and time window

0.5 set which can be used right away for predictive purposes. These new sets and their size can be seen in table 4. Obviously, the size of the sets reduces drastically when imposing this kind of conditions.

At this point, it is important to remember the decisions taken in terms of search depth and data subsetting as mentioned in sections 4 and 5. In these terms, rule sets for some of the stations and time windows could not be generated with our available computation capabilities. A better server or algorithm optimisation would be required in order to obtain result sets for these cases.

6.2 Number of rules against precision

As we have seen in section 6.1, the amount of rules decreases significantly if we impose strict conditions for their validity. In this section we will perform a deeper analysis on how amount of rules vary when setting different thresholds. For this purpose, we will set different thresholds and check the number of rules complying with this condition. The results are shown in tables 5, 6, 7, 8, 9 and 10. As expected, the number of rules increases exponentially when decreasing the threshold. For better visualization, these amounts for the $> 50\%$ *subsets* are represented in figures 1, 2, 3, 4, 5 and 6.

Station	Time Window	Rules
Albacete	7 days	9
Antequera	1 day	7104
Segovia	1 day	31
Segovia	2 days	30
Segovia	7 days	48
Sevilla	1 day	242

Table 4: Number of rules for each set setting a threshold of 50% precision

Threshold	Number of rules
0.05	44620
0.10	38007
0.20	4060
0.30	30
0.40	27
0.50	9
0.60	8

Table 5: Number of rules for different thresholds in Albacete (7 days)

Threshold	Number of rules
0.05	31846
0.10	28338
0.20	21566
0.30	15829
0.40	10876
0.50	7137
0.60	4928
0.70	891
0.80	47

Table 6: Number of rules for different thresholds in Antequera (1 day)

Threshold	Number of rules
0.05	106
0.10	95
0.20	72
0.30	44
0.40	33
0.50	31
0.60	30
0.70	24
0.80	12

Table 7: Number of rules for different thresholds in Segovia (1 day)

Threshold	Number of rules
0.05	69
0.10	66
0.20	56
0.30	40
0.40	31
0.50	30
0.60	14

Table 8: Number of rules for different thresholds in Segovia (2 days)

Threshold	Number of rules
0.05	128
0.10	125
0.20	92
0.30	75
0.40	64
0.50	48
0.60	35
0.70	30
0.80	25
0.90	4

Table 9: Number of rules for different thresholds in Segovia (7 days)

Threshold	Number of rules
0.05	6730
0.10	2832
0.20	2357
0.30	1799
0.40	642
0.50	246
0.60	78
0.70	2

Table 10: Number of rules for different thresholds in Sevilla (1 day)

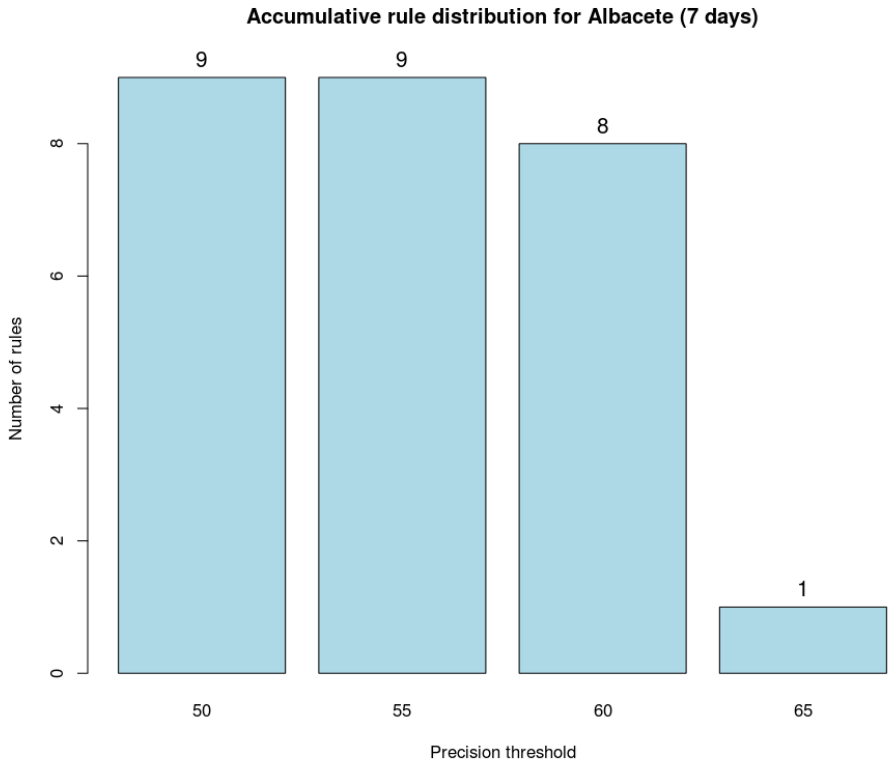

Figure 1: Number of rules for different thresholds in Albacete (7 days)

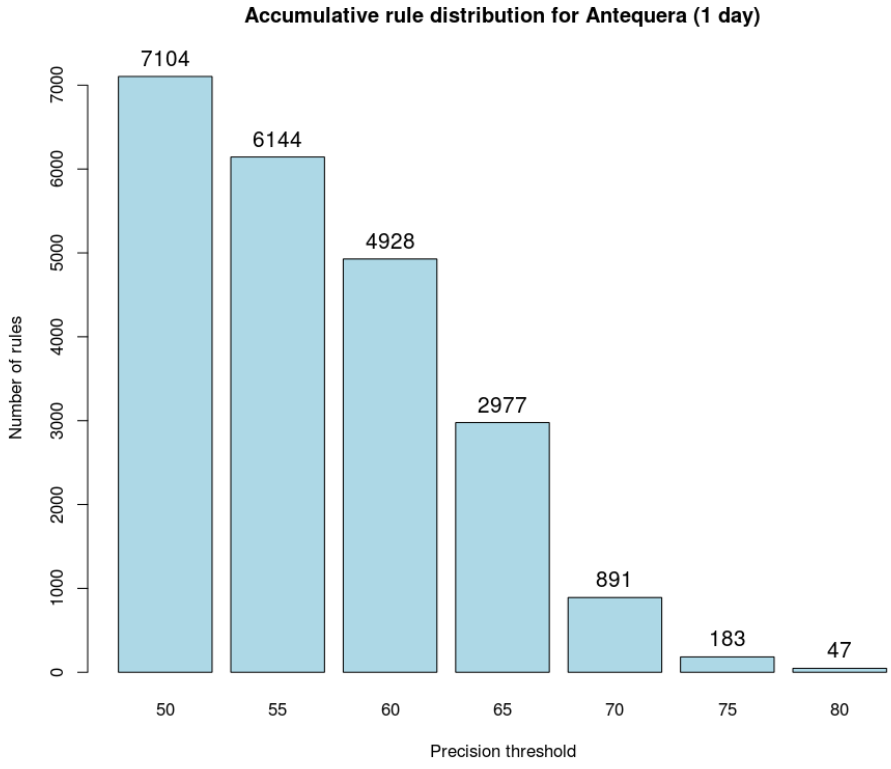

Figure 2: Number of rules for different thresholds in Antequera (1 day)

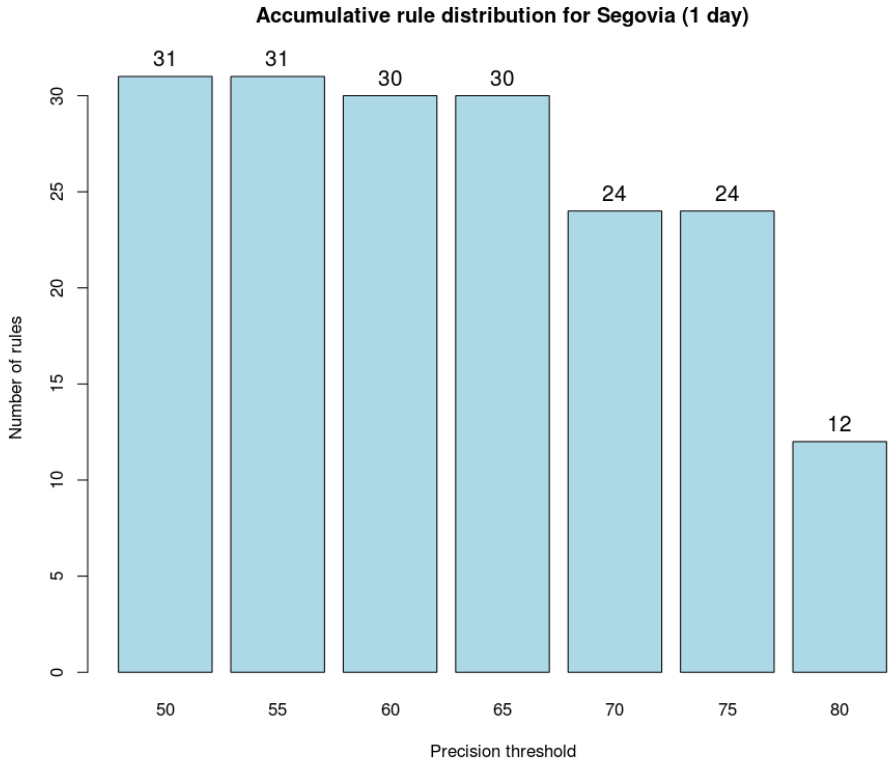

Figure 3: Number of rules for different thresholds in Segovia (1 day)

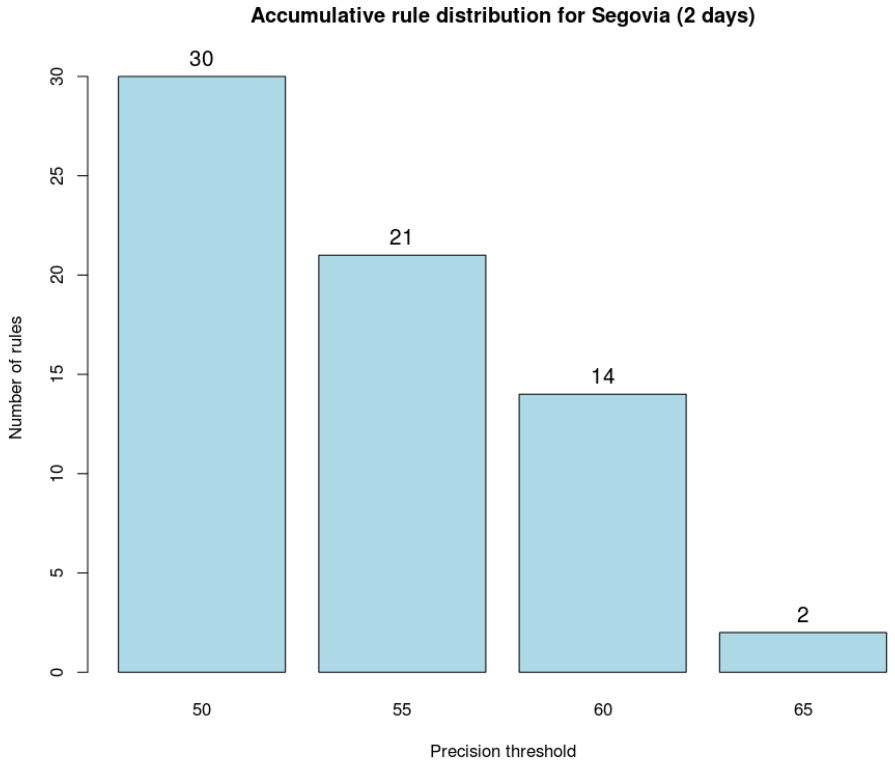

Figure 4: Number of rules for different thresholds in Segovia (2 days)

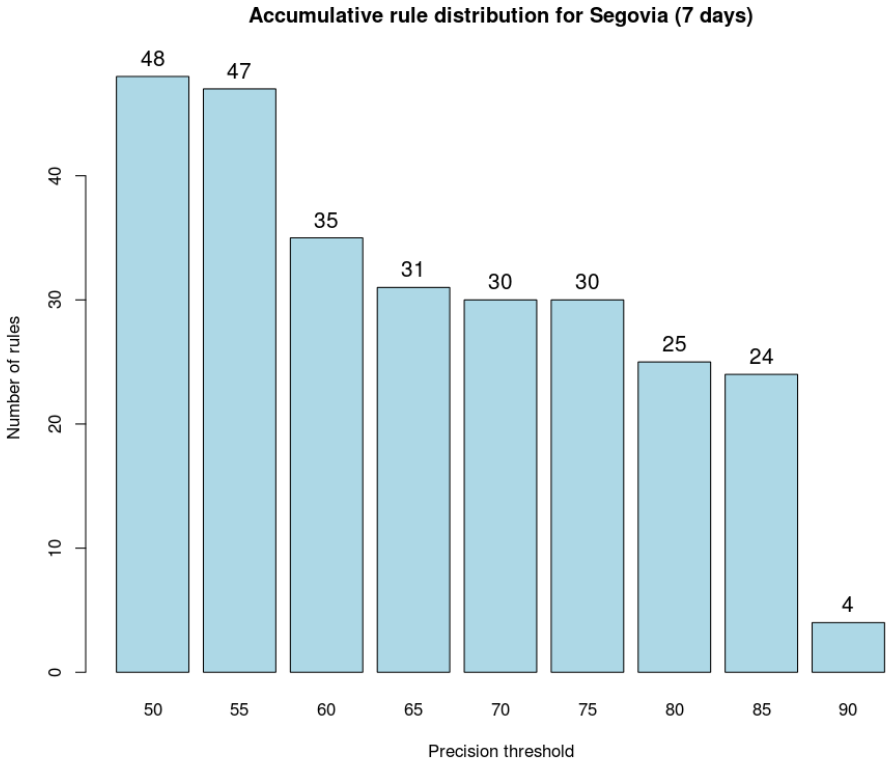

Figure 5: Number of rules for different thresholds in Segovia (7 days)

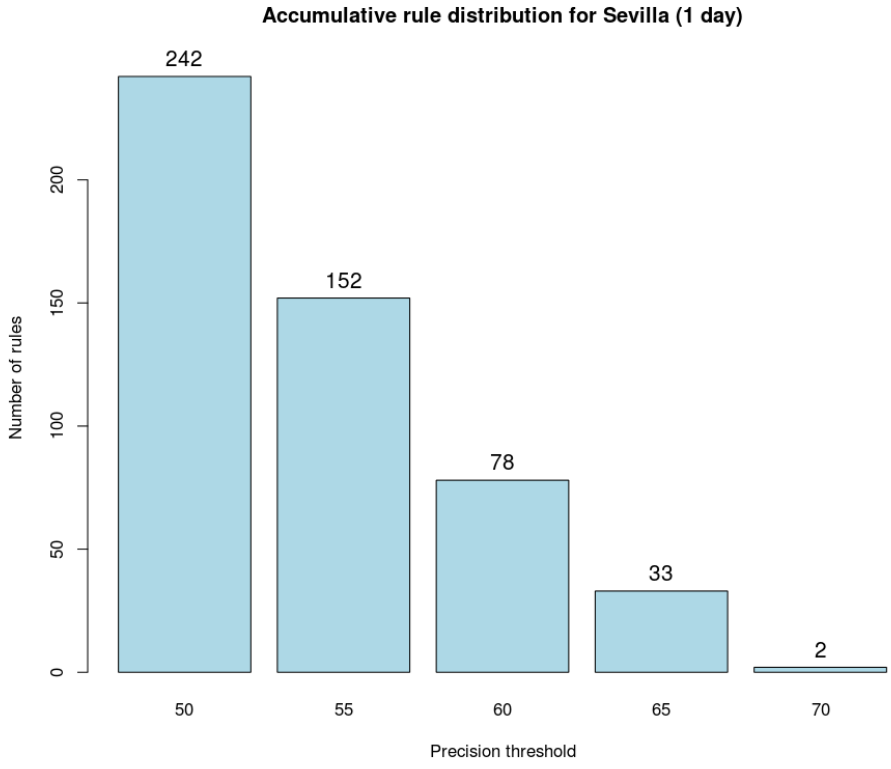

Figure 6: Number of rules for different thresholds in Sevilla (1 day)

6.3 Precision distribution

In section 6.2 we analysed the number of rules we would obtain if we set different precision levels as thresholds to generate different subsets. According to our expectations, we observed a decay in these numbers as we set higher thresholds. However, this decay, although apparently exponential, shows flat zones and other irregularities which would be unexpected at first.

In order to better visualize these anomalities we will graphically represent the precision distribution in form of histograms for the previous sets. These distributions can be seen in figures 7, 8, 9, 10, 11 and 12.

In these histograms we can see that the distribution does not grow exponentially as we lower the precision, as we would expect and as we apparently saw in the analysis from section 6.2. Instead, there are some accumulation points around which precision tends to take values.

For example, looking at the distribution for Antequera (figure 8) we see that there are more rules with precisions between 0.65 and 0.70 than between 0.50 and 0.55.

As we do not have large sets of rules for all the stations, performing a deeper analysis of the distributions is not possible. As these kind of irregularities appear for all the studied cases, it is very unlikely that they are caused just by chance. However, the causes and possible implications of these distributions cannot be infered at this point and would require further research.

7 Example scenario

In this section we will analyse the output of our predictive model during a short period of time. We will illustrate the kind of information an operator could obtain from an average period of time. Specifically, we will take a random period of 50 consecutive days for the station of *Antequera*, over which we will perform predictions for the next days.

It is important to note that this example scenario has been obtained using the rule set with precisions higher than 0.50. In a real scenario, an

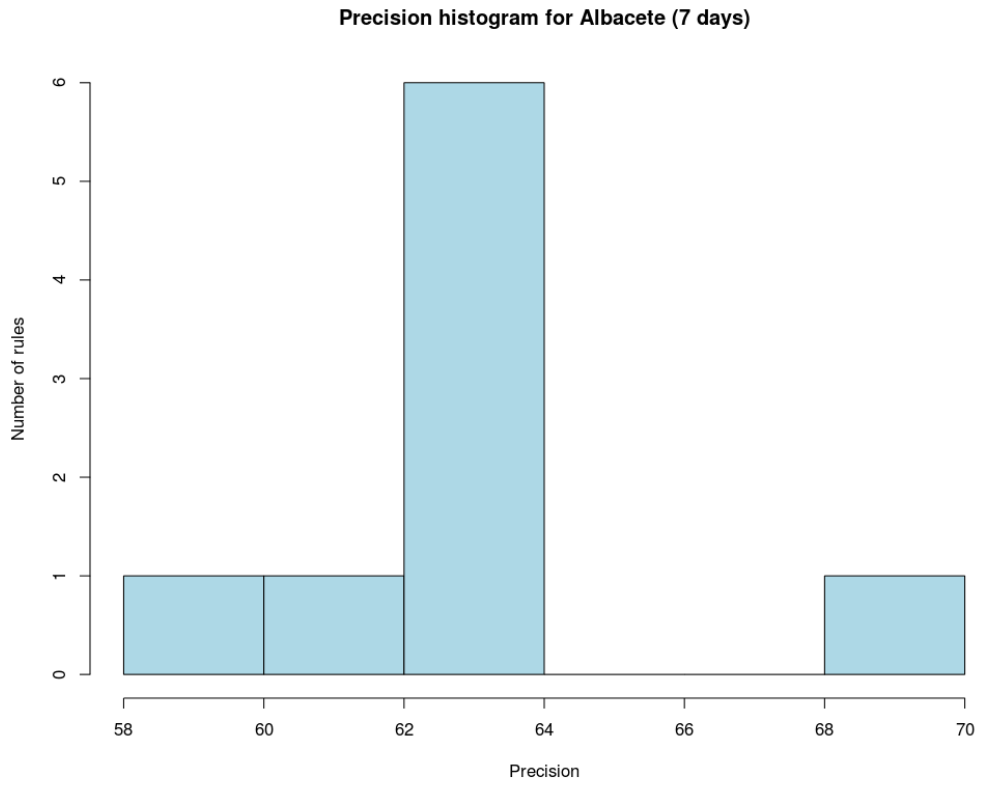

Figure 7: Rule distribution by precision in Albacete (7 days)

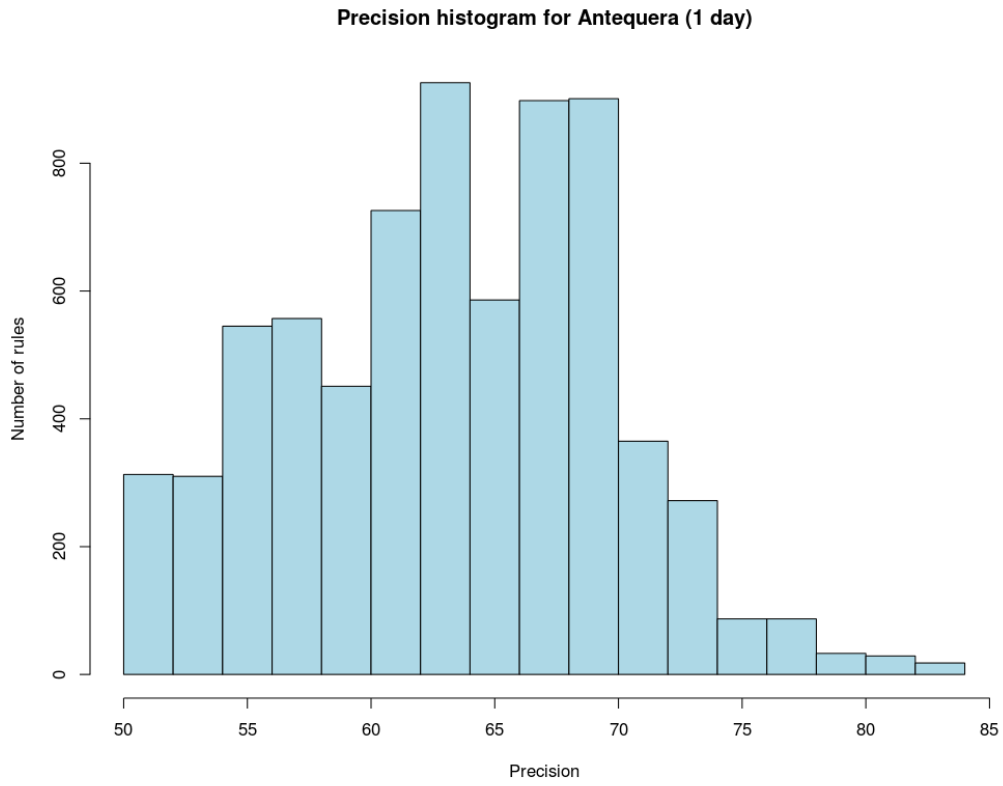

Figure 8: Rule distribution by precision in Antequera (1 day)

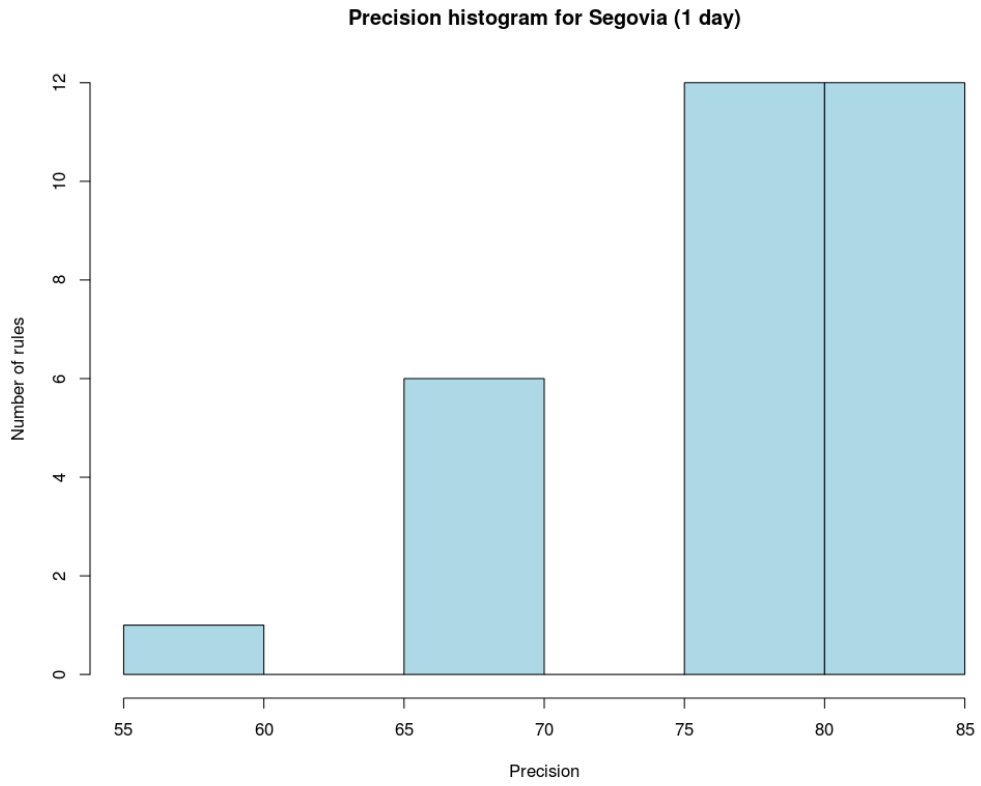

Figure 9: Rule distribution by precision in Segovia (1 day)

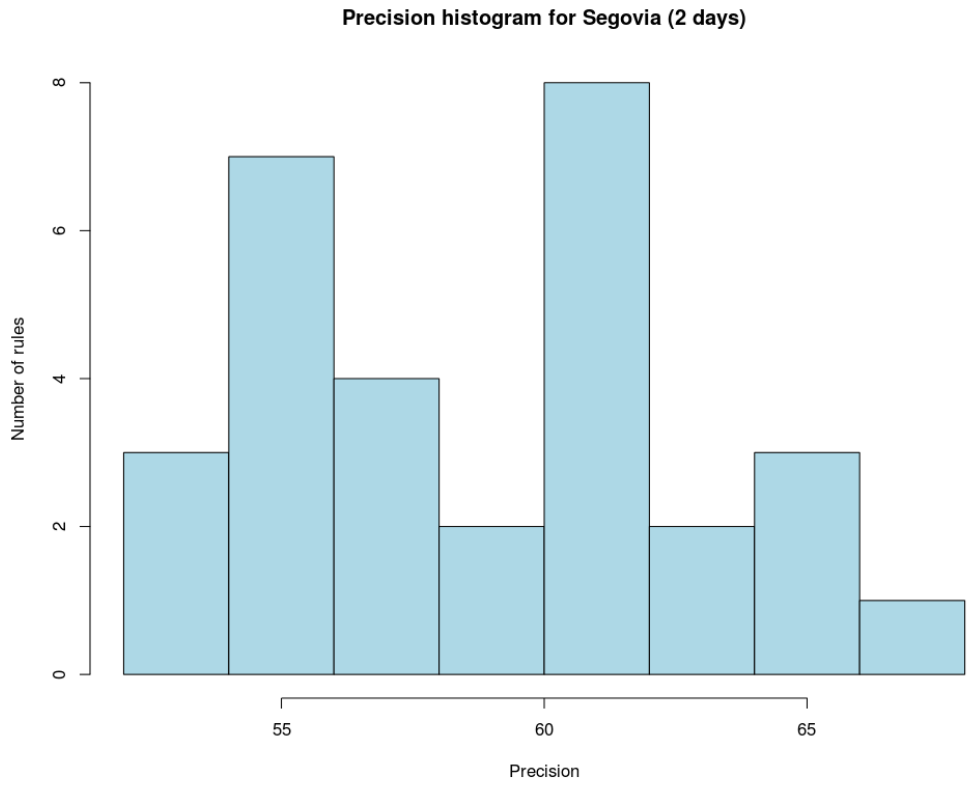

Figure 10: Rule distribution by precision in Segovia (2 days)

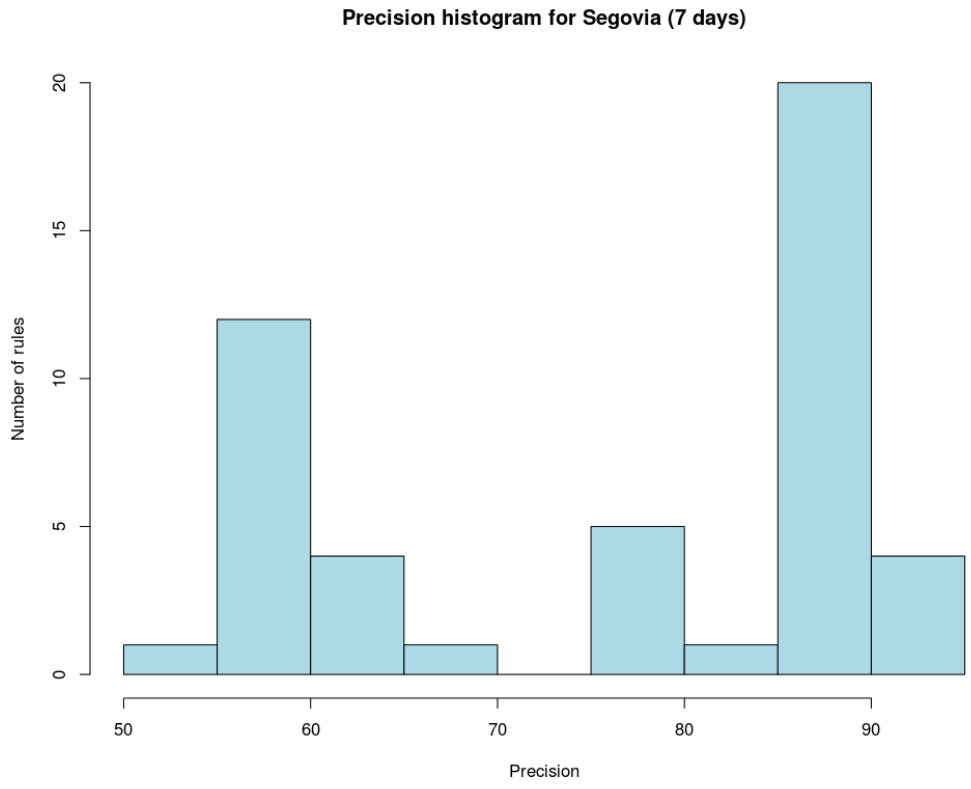

Figure 11: Rule distribution by precision in Segovia (7 days)

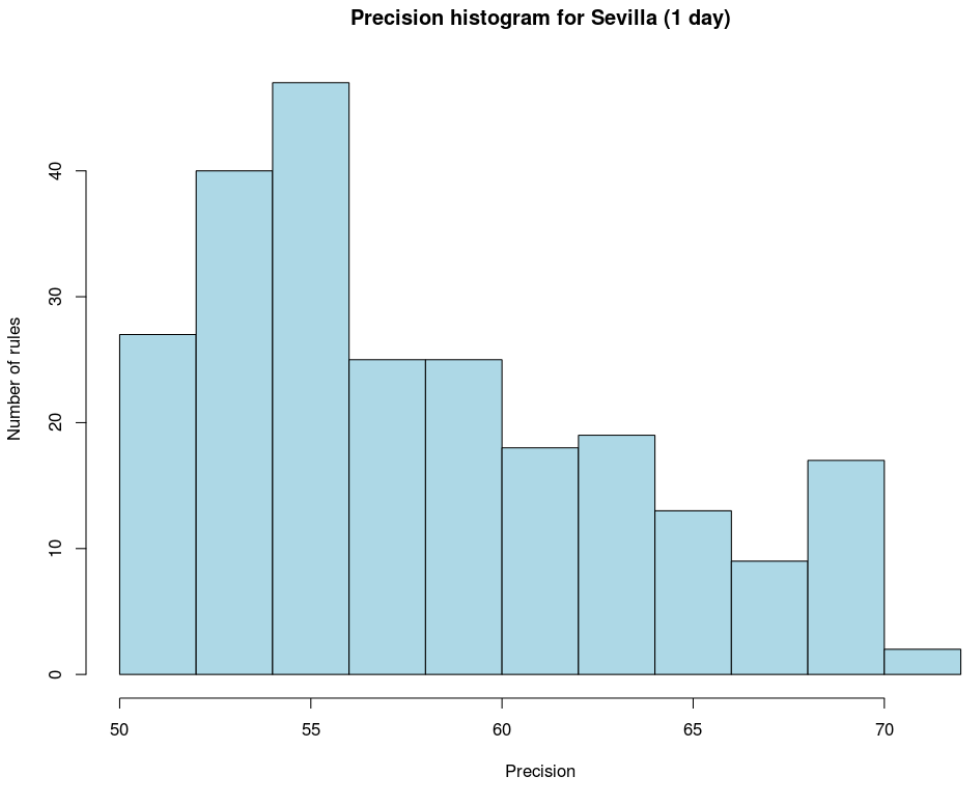

Figure 12: Rule distribution by precision in Sevilla (1 day)

operator could select a higher threshold for predictions or disregard those raised predictions with a low confidence.

7.1 Predictions raised by event type

First of all we will observe the number of predictions raised each day of our sample. This will give us an idea of the number of predictions our system would give for an average day. This can be seen in figure 13.

As we can see, for this station and rule set the number of predictions obtained in an average day is around 20. This does not include repeated predictions (more than one rule which can be fired simultaneously to predict the same event with different confidences).

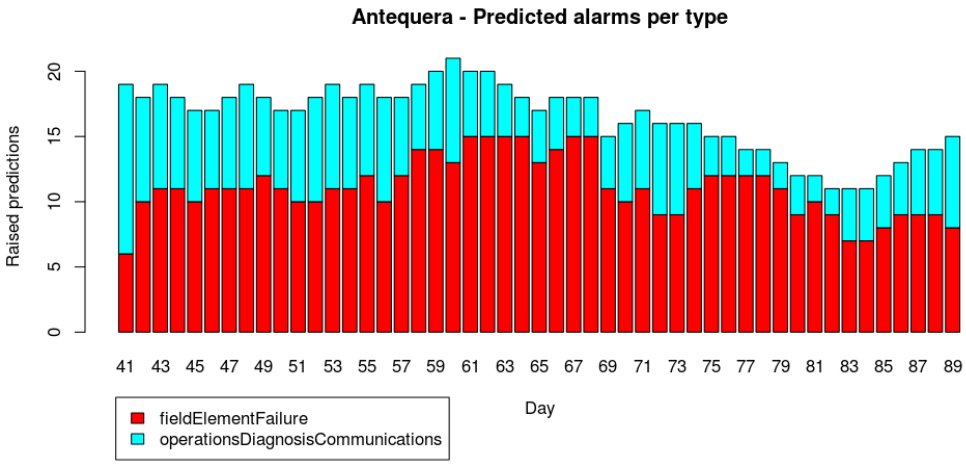

Figure 13: Timeline of predictions during a sample period

7.2 Percentage of events predicted

The next thing we can analyse is the number of events which happen during an average day which are actually predicted by our system. The result is illustrated in figure 14. This value corresponds with what we defined as *recall* in section 3.3.

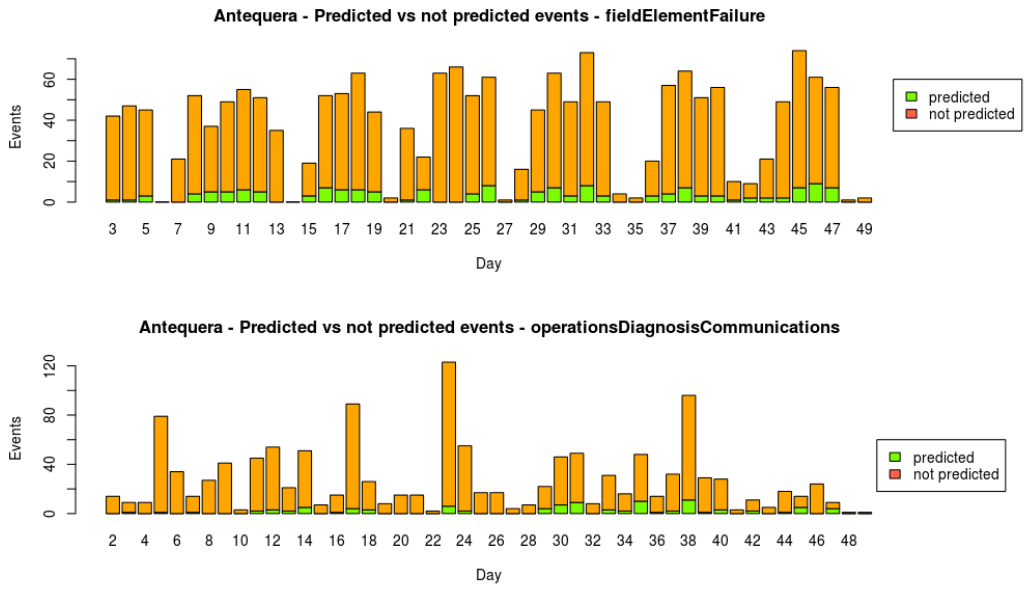

Figure 14: Timeline of predictions during a sample period

7.3 Percentage of right predictions

To end with, we will analyse how many of the raised predictions are actually true. In order to illustrate better this aspect and take into account also the confidence parameter of the predictions, we will perform three different analysis: one disregarding predictions with $c < 0.5$, a second one with $c < 0.7$ and a third with $c < 0.8$. The results can be seen in figures 15, 16 and 17 respectively.

As expected, when increasing the precision threshold we have less and less predictions, but these tend to be more accurate.

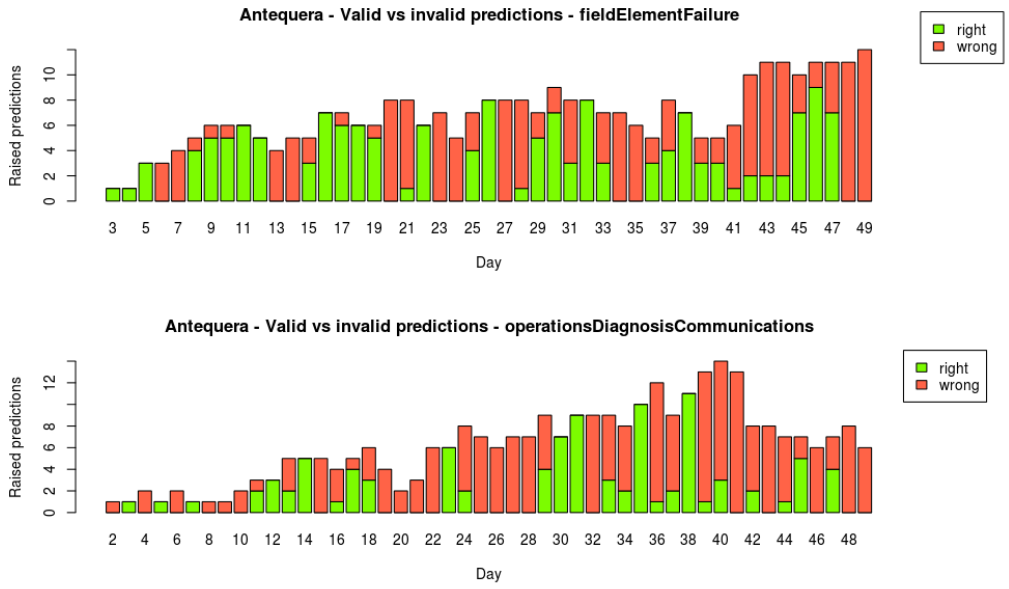

Figure 15: Timeline of predictions during a sample period

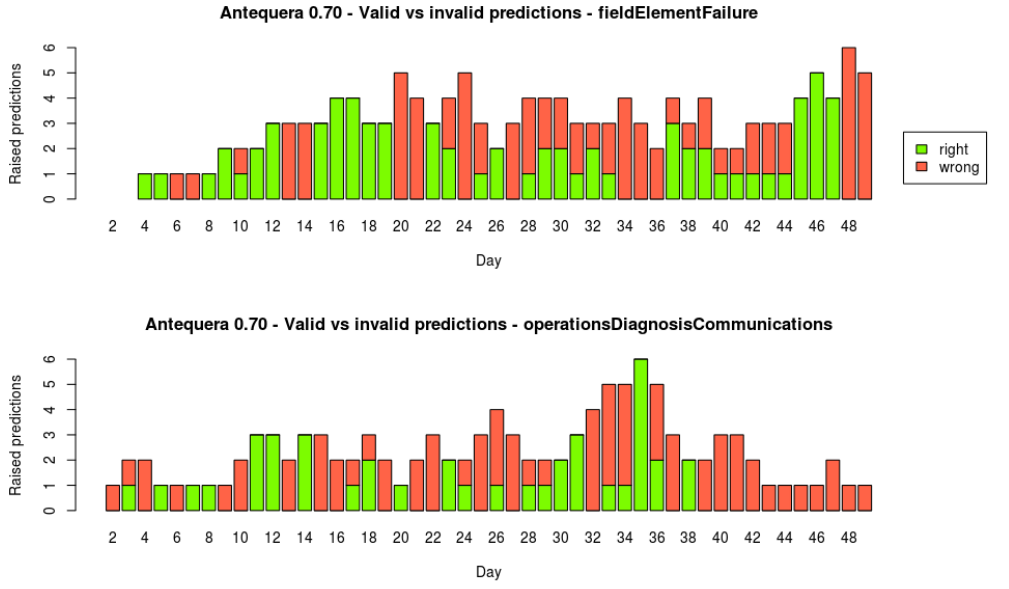

Figure 16: Timeline of predictions during a sample period

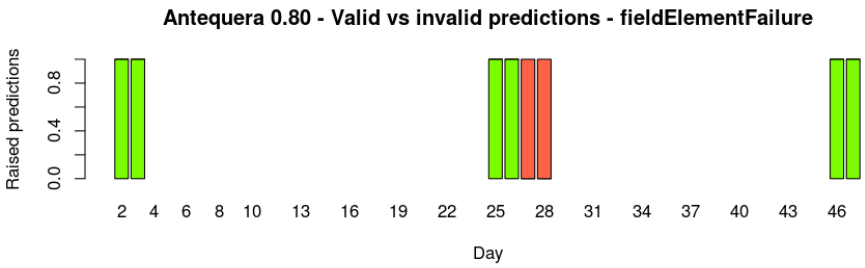

Figure 17: Timeline of predictions during a sample period

8 Conclusions

References

[1] M J Druzdzel. SMILE: Structural Modeling, Inference, and Learning Engine and GeNIe: a development environment for graphical decision-theoretic models. In *Proceedings of the national conference on artificial intelligence*, pages 902–903. JOHN WILEY & SONS LTD, 1999.

[2] R Srikant and R Agrawal. Mining sequential patterns: Generalizations and performance improvements. *Advances in Database Technology—EDBT'96*, pages 1–17, 1996.

[3] L Torgo. *Data mining with R*. CRC Press, 2003.

[4] G M Weiss and Others. Predicting telecommunication equipment failures from sequences of network alarms. *Handbook of Knowledge Discovery and Data Mining*, pages 891–896, 2002.

[5] D Wu, X Wang, T Zuo, T Sun, and F Yang. A Sequential Pattern Mining algorithm with time constraints based on vertical format. In *Information Science and Engineering (ICISE), 2010 2nd International Conference on*, pages 3479–3482. IEEE, 2010.

[6] M J Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1):31–60, 2001.

[7] Mohammed J Zaki. Sequences Mining in Categorical Domains: Incorporating Constraints. In *9th ACM International Conference on Information and Knowledge Management*, November 2000.

[8] Q Zhao and S S Bhowmick. Association rule mining: A survey. *Available on WWW: http://www. cse. unsw. edu. au/˜ cs9318/readings/ARMining-Survey2. pdf (July 2007)*, 2003.

[9] Q Zhao and S S Bhowmick. Sequential pattern mining: A survey. *ITechnical Report CAIS Nayang Technological University Singapore*, pages 1–26, 2003.