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

Contents

List of Figures

List of Tables

1 Data mining procedure

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 Data Mining techniques from which the actual knowledge will be obtained.

As we already mentioned, there are different groups in which we can categorize most of Data Mining procedures:

1. **Classification:** Learning a function that maps an item into predefined classes.

2. **Regression:** Learning a function that maps an item to a predicted variable.

3. **Segmentation:** Identifying a set of clusters to categorise the data.

4. **Summarization:** Finding a compact description for the data.

5. **Association:** Finding significant dependencies between different variables.[6].

6. **Sequence analysis:** Finding frequent sequences or episodes in data [7, 2].

Three of these categories can be useful for our learning objectives. *Sequence analysis* seems the most immediately appropriate category in which our objectives might fall – we have a large amount of sequential data from which we want to obtain patterns which might be useful to make predictions in the future. From these obtained patterns, we may be able to obtain *association* rules – obtain antecedent-consequent pairs from frequent sequences which would associate occurrence of alarms in a given period with the occurrence of other alarms in any other future period.

Alternatively, we can convert our problem into a *classification* task by modelling the alarms of the current period as variables, and the possible alarms for the prediction period as categories to classify our situation into.

The most immediate and convenient (in terms of later implementation and integration) for us is the first approach: sequential analysis from which we will obtain association rules. The whole procedure comprises three main steps:

First of all we must 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 2.

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.

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 4.

2 Mining 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

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 [4, 7, **?**]. 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.

2.1 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[5]. 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 2, 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[5, 3] 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.

3 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 2, *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 2.
- 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:

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

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

4 Validation and evaluation

5 Evaluation criteria

The predictive information obtained in the data mining process, will lead to the implementation of systems which will give us a prediction using current events as its input. This prediction will be given in the form of an alarm or set of alarms, which are likely to be raised within a given prediction period. In this section we will approach the problem of *evaluation* of this predictive information.

As a first thought, it might seem appropriate to evaluate 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 completely significant of the overall 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[1]. 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.

6 Validation

Once the data mining process has been successfully performed, it is of essential importance to *validate* the obtained results. This is, once we have learnt to make predictions based on observation of events, we must actually test how good our predictions are. As we count on a vast amount of data logs, we can easily check our predictions against this historic data. 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 system, 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.

References

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

[2] 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.

[3] 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.

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

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

[6] 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.

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