How Rules Determine the Operator

Analysis of Water Height Prediction

Alexander Nezhinsky

October 31, 2007

Abstract

The project concerns water height prediction by the Rijkswaterstaat employees. The employees predict the water height based on some rules. We want to find out by which rules an employee can be identified. We use a validation set corresponding to some unknown employee. We find out how we can use the association rules to retrieve which operator the validation set represents. In our main approach we use association rule properties such as confidence and lift to calculate the chance that a given validation set corresponds with a given operator. Over 70% of the tested sets could be successfully retrieved with this approach.

Chapter 1

Introduction

This research was done for Rijkswaterstaat (RWS), the Dutch government organization for water management. The project that was proposed concerned water height prediction by the RWS employees. Different kinds of data are available, from which the operators predict the waterheight. Can we recognize the operator (employee) by looking at his/her prediction of the waterheight? We try to solve the problem by the use of data mining approaches. As a general reference for datamining we refer to [7].
First we look at the creation of association rules from these data for each operator. This is done by the use of an Apriori [1] like algoritm. The database used is provided by Rijkswaterstaat. We will try to retrieve only those rules, that actually matter for operators decision.
Then, after a list of rules is created for each operator-place combination, we want to test the usability of these rules. To do this we try to retrieve the operator (and perhaps the location) that belongs to some dataset by looking at its association rules.
The research was done as a master thesis for Computer Science at Leiden University. This thesis is supervised by dr. Jeannette de Graaf and dr. Walter Kosters.

Chapter 2

Definitions

In this section we will give all necessary definitions. For a given database of events (records), we will define association rules. Then we will explain the meaning of the terms support, confidence, expected confidence and lift.

2.1 Items and itemset

Items are units. In our case items are variable values such as ”waterheight is low”. Multiple units build up an itemset.

2.2 Association rule

An association rule is an implication consisting of two parts, the rule body and the rule head. The rule body consists of items which together implicate the rule head. The rule head is also called antecedent, the rule body is called consequent. For example, consider the following association rule: $\{ab\}$ implies $\{c\}$, for brevity $\{ab\} \rightarrow \{c\}$ (with elements a, b and c) or even $ab \rightarrow c$. In this case the items a and b together form the rule body. The item c is the rule head.

2.3 Support

The support s of an association rule indicates how often a certain rule is encountered, considering all the records from a dataset. The support of a rule is in fact the support of the underlying itemset, containing the rule body and the rule head. This value is usually expressed as a percentage. The support value is calculated in the following way: the number of records containing the itemset divided by the total number of records. For example consider the following data set:

abc
dc
de
dc

The support of the rule $ab \rightarrow c$ equals the support of the itemset $\{a, b, c\}$, and is then

$$s(\{ab\} \rightarrow \{c\}) = 1/4$$

2.4 Confidence

The confidence of an association rule is a value that indicates how likely it is that the rule body implies the rule head if we consider all the records which contain the same rule bodies. For the calculation of the confidence *conf* for a rule r we must divide the support of r by the support of its rule body. For example, consider the following dataset:

abc
dc
de
dc

The confidence of $d \rightarrow c$ is

$$conf(\{d\} \rightarrow \{c\}) = s(\{d\} \rightarrow \{c\})/s(\{d\}) = (2/4)/(3/4) = 2/3$$

This means there is a chance of 2/3 that if a record contains d it also contains c.

2.5 Expected Confidence

The expected confidence *exp* of a rule is the confidence value that a rule would have if the rule head and the rule body were in fact independent. It can be calculated by multiplying the support of the rule head with the support of the rule body and after that dividing by the support of the rule body: $exp(rule) = s(head) * s(body)/s(body)$. Since $s(body)$ can be removed from the formula, the expected confidence can be calculated by simply looking at the support head $s(head)$. Again, consider the example:

abc
dc
de
dc

The expected confidence of the rule $d \to c$ is

$$exp(\{d\} \to \{c\}) = s(\{c\}) = 3/4$$

2.6 Lift

Another important measurement is the lift. Lift indicates the amount of statistic independency of the rule body and rule head. If the rule is completely random then the lift will be 1, since then the combination of rule body and rule head is found as often as expected. If the lift is lower then 1, this means that rule body and rule head appear less often together than expected. If the lift is larger than 1, this means that rule body and rule head appear more often together than expected. The lift can vary between 0 and infinity. The lift is calculated as follows:

$$lift(rule) = conf(rule)/exp(rule) =$$

$$= conf(rule)/s(head)$$

In our case the rule head is always one of the five values, while the rule body can be anything, so often the lift is just slightly higher than 1 for the rule heads, that are often encountered. The lift is often higher for the rule heads that are not often encountered.

Chapter 3

Creating association rules

3.1 Place and operator

The places for which data files are available, are the places in the Netherlands which have a collection of measurement utilities and where the water height and other factors are measured and recorded.

The people who predict the water height values for each of the places are called *operators*. Operators have several kinds of different data they can access and use these and their knowledge of different situations for their prediction. The data include, i.e., current measurements of the water height, wind direction and speed.

The operators can also consult several water height prediction algorithms, that are running at different places. These algorithms are *Sobek*, *CSM8*, *Neural Network*. Not all of these algorithms are running in all places considered.

Besides looking at the measurements and the prediction algorithms, the main part of the operators job is to make his or her own judgement of the situation and decide which data to use for prediction. The quality of each operator prediction is recorded in the datafile. The better the operator's prediction matches the actual water height, the lower the prediction error. The two possible errors in the prediction are *overguess* and *underguess*. We speak of *overguess* when the value predicted by an operator is higher than actual water height. We speak of *underguess* when the value predicted by an operator is lower than actual water height.

The number of operators considered in this project equals 7. The operators are numbered 2 till 8. The records for operator number 1 did exist, but this number was used as an alias by different operators, who were not logged in as themselves, and thus unusable for this project.

For all the places at a certain timestamp only one operator is working. When he or she is done with his or her shift, the shift of another operator starts.

The rules we are going to search for are always matching a certain operator at work at a certain place. An operator is unlikely to use the same rules for all the places he has worked at, since different influences are present at different places. We introduce the therm *PlaceOperator*, which represents an *operator-place* pair, and can later be used to identify the connection between a rule and such a pair. For abbreviation we will use the notation *po*.

3.2 Explanation of the variables

The creation of association rules will be described in this section.

The database that was initially provided by RWS consisted of raw data corresponding to different variables, measured on certain time intervals. The database files that were provided by RWS are the records of the measurements for 7 places. The different places and their abbrevations used within the database are shown in the following table:

DENH	Den Helder
DLFZ	Delfzijl
EPL	Europlatform
HA	Harlingen
HARL	Haarlem
HVH	Hoek van Holland
YMD	IJmuiden

Data files are available for all 7 locations on the time interval between October 2005 and November 2006. For 4 of these locations (HVH, HA, YMD,EPL) also the data files are given on the interval between January 2005 and October 2006. For operators 2 and 4 the prediction values are not reliable for the locations DENH, HARL and DELFZ.

For each place a datafile that consists of multiple lines is available. Each line represents a certain timestamp and consists of values for different variables.
Each dataline consists of 20 different variable values written in sequence and delimited with single spaces. Each variable has a different meaning. Here is a table with a short description of the variable meanings.

Number	*Variable*	*Example value*
0	low water or high water	1
1	operator	4
2	time	732586.1667
3	time astro	732586.625
4	time measurement	732586.6111
5	time HMR	732586.625
6	time CSM8	732586.625
7	time Neural	732586.6181
8	water 1-height astro	93
9	water height measured	115
10	water height HMR	119
11	water height CMS8	118
12	water height Neural	123
13	time difference	0.44444
14	time difference HMR	0.013889
15	HMR	4
16	time difference CSM8	0.013889
17	CSM8	3
18	time difference NN	0.0069444
19	NN	8

The above variables are used for the construction of several properties, that will later be used as rule elements. A new database is built up from these properties. Properties are in this context also variables, but with (very) limited number of variable values. Properties can sometimes also be gained from multiple variables. We will discuss these transitions of variables into properties in more detail.

Variable 0 stands for low or high water and can have a value of 0 or 1. It is used for the property *Water height 1* which can take the values *Low water* if the water level is considered to be low and *High water* otherways.

Variable 1 stands for operators ID. Each operator is identified by his number. In total there are 7 operators. At each time some operator is present to predict the water height. Sometimes the ID of the operator is unknown. The operator ID value is not used for a property. This value is to define the filename to which the properties are written. If, for example the operator ID equals 6, all the properties will be written to a file *Operator6.txt*.

Variable 2 stands for the shift starting time. The time is formatted as *MATLAB* time. This representation means, that the time value is seen as a float value. Date numbers are serial days where 1 corresponds to 1 January of the year 0000. Each whole day is represented by a 1. We can easily decode each value into our current time. For example 732678 is decoded into 1 January 2006. The next day is then 732679. The decimals are used to denote the time of day. For example 732678.5 is decoded into 1 January 2006 12:00. The advantage of this representation is that time is presented as a linear value, which is more intuitive and makes calculations easier.

Variable 3 stands for *Time astro*. It is the time at which the *astro* is measured. *Astro* is an automated water height prediction based on the locations of the sun, the moon and the planets. The measured water height differs from this *astro* prediction, because local influences like wind speed, wind direction, waves, etc. are not considered by *astro*. We do use the prediction by astro in variable 8 to check the difficulty of the weather, but variable 3 is not used.

Variable 4 stands for the current time — time at which the actual water height was measured. All the predicting algorithms (CSM8, Sobek, Neural) are started at this time. We use the current time value to set the *Time of day* property to either *morning*, *afternoon*, *evening* or *night*. We also use this variable to retrieve the shift in which the current operator was working. There are three shifts — from 22:45 till 6:45 is the night shift, from 6:45 till 15:00 the day shift and from 15:00 till 22:45 the evening shift. We set the property *Shift* to either *Night shift*, *Afternoon shift* or *Evening shift*.

Variable 5 stands for the time the water height was predicted by a HMR operator (current operator at work). HMR stands for *Hydro Meteo Centrum Rijnmond*. In the future this name will be changed into HMCN, which stands for *Hydro Meteo Centrum Noordzee*. This seems to be useful, but we have no need to use this variable, since variable 14 gives the difference between this variable value and the actual time of measurement (variable 4).

Variable 6 stands for the time at which CSM8 made the water height prediction. Variable 7 stands for the time at which Neural made the water height prediction. The information that both variables provide seems useful, but is redundant. To see how big the delay of both prediction algorithms is we make use of variables 16 (difference between variable 4 and variable 6) and 18 (difference between variable 4 and variable 7). The delay is how long the algorithms were calculating the value.

Variable 8 stands for *water height astro*. This variable stands for the water height value that is predicted by the *astro*.

Variable 9 represents the actual water height at the time defined in variable 4. This variable is used for the property *Water height 2* which can take 7 different values, each representing a certain water height interval.
This variable is also used together with variable 8 (water height astro) to define a boolean property called *Heavy weather*. When the difference between these two variables is high it means that the weather was difficult to predict, and thus the operator will have more difficulties with his or her predictions. This will be the case when there is for example a hard wind. *Heavy weather* is then set to *true*. When the weather is quiet, the predictiong is considered easier and *Heavy weather* is set to *false*.

Variable 10 stands for the water height HMR. Water height HMR is the water height that is predicted by the operator. This seems to be useful, but we have no need to use this variable, since variable 15 gives the difference between this variable value and the actual water height.

Variable 11 stands for the water height predicted by the *CSM8* algorithm.
Variable 12 stands for the water height predicted by the *Neural* algorithm.

The information that both variables provide is useful but redundant. To see how big the error of both prediction algorithms is we make use of variables 17 and 19.

Variable 13 is the time difference. It represents the time passed since the beginning of the shift (variable 2), until the time the measurement was made (variable 4). Variable 13 is not used for any properties.

Variable 14 stands for time difference HMR, it is the difference in time between the time an operator made a prediction and the time (variable 4) the water height (variable 5) was measured. It represents the delay with which an operator made his prediction. We can see how long an operator was *thinking* about the prediction. We set the property *Operator time* to either *Operator early*, *Operator on time*, *Operator late* and *Operator very late*.

Variable 15 is the difference between water height predicted by the operator and the actual water height, and leads to the property *Operator error*. This is the most important value, since it represents the error which an operator makes in his or her prediction. The property *Operator error* can be set to 5 different values. If the operator predicts a water height lower than the actual water height, when the error is large (for more precise definition of large and small see [5]) the property is set to *Operator large under* or when the error is small it is set to *Operator small under*. If the operator predicts a water height higher than the actual water height, if the error is large the property is set to *Operator large over* or if the error is small to *Operator small over*. If the prediction an operator makes is very close to the real water height value, then the property is set to *Operator OK*. This property has no value, if for any reason, an operator has made no measurement at all.

Varable 16 — time difference CSM8, is the difference in time between the time the CSM8 algorithm made a prediction (variable 6) and the time the water height was measured (variable 4). It represents the delay with which CSM8 made his prediction. We set the property *CSM8 time* to either *CSM8 early*, *CSM8 on time*, *CSM8 late* and *CSM8 very late*.

Variable 17 is the difference between the water height predicted by the CSM8 algorithm and the actual water height. This value represents the error which the CSM8 algorithm makes. The property *CSM8 error* is deduced from this variable. Analogous to the *Operator error* the property can be set to 5 different values *CSM8 small under*, *CSM8 large under*, *CSM8 large over*, *CSM8 small over* or *CSM8 OK*. This property has no value if for any reason the CSM8 algorithm has made no measurement at all.

Varable 18 stands for *time difference Neural*, it is the difference in time between the time the Neural algorithm made a prediction (variable 7) and the the time the water height was measured (variable 4). It represents the delay with which Neural made his prediction. We set the property *Neural time* to either *Neural early*, *Neural on time*, *Neural late* and *Neural very late*.

Variable 19 is the difference between the water height predicted by Neural algorithm and the actual water height. This value represents the error which the Neural algorithm makes. The property *Neural error* is set by this variable. Analogous to the *Operator error* the property can have 5 different values *Neural small under*, *Neural large under*, *Neural large over*, *Neural small over* or *Neural OK*. This property has no value if for any reason an Neural algorithm has made no measurement at all.

The properties that are created from the variables that are described in this section are presented in the following table:

Property	*Number of values*
Water height 1	2
Time of day	4
Shift	3
Water height 2	7
Heavy weather	2
Operator time	4
Operator error	5
CSM8 time	4
CSM8 error	5
Neural time	4
Neural error	5

3.3 Encoding values

A Perl script *decode5.pl* is written, to parse these raw data files into a different format. This is done in a few sequential steps. First the user is asked to enter the file name and the minimal support. The file that is used for parsing can be an original raw data file that was provided by the RWS. Later, in Section 4 after the terms *learning set* and *validation set* are introduced, it will be shown why it is better to use only a part of this file.

Some minimal support should be selected to prevent the program from creating too many itemsets and to save running time. Since the database is quite large the itemsets should only be considered if the items occur together more often than a certain threshold.

Each of the variables in a line is looked at and parsed. The new format for each variable consists solely of 0 and 1 in the new encoding.
For example, look at variable 15 which stands for *Operator error*. This variable stands for the difference of the operator predicted water height and the real water height that was measured at this time. This means we can see the error an operator introduces in making his or her prediction of the water height. We then look at the interval of the possible error values. We then divide this interval into some small number of intervals (in this case 5 is enough), to make the encoding of the values easier and to limit the number of possible property values. Sometimes a variable has no value at all. In the datafile this is denoted as "not a number" — *NaN*. This means that for some reason no measurement was done. If the value of the variable is unknown (or in this case — no measurements), then all the encoding bits of a property are set to 0. For example, a property with 5 values is coded into a string with length 5. The new encoding for the different values of the property *measurement correctness* can become the following:

```
0 0 0 0 0 - operator no measurement
0 0 0 0 1 - operator large overguess
0 0 0 1 0 - operator small overguess
0 0 1 0 0 - operator good guess
0 1 0 0 0 - operator small underguess
1 0 0 0 0 - operator large underguess
```

Note that we cannot use standard binary encoding and we can make use of a maximum of one occurrence of 1 for each variable value. This is due for the next step of decoding these values. We do this step for every variable, starting at variable 0 and ending at variable 20. For different variables the number of bits encoding its value can be different. It depends on the number of intervals in which the range of a variable is divided. The property (new variable) will have that number of possible values. We use one bit for each possible variable value. For the example (*Operator error*) above 5 bits are enough. But for the *low or high water* variable we only need 2 bits, since water can be either high or low.

The new representations are then appended after each other in one line. The new representation for the example timestamp (a line) will look like this:

```
1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0
```

Each line in the representation, that now consists of 0's and 1's, must be represented as a collection of numbers. Each number stands for a place in the line where a 1 was encountered. Counting of 1's starts at place 0.
The new format for the example line shown earlier looks like this:

```
0 3 11 15 18 23 27 32 36 41
```

This line means that a 1 was encountered at place 0, 3, 11, etc.

3.4 Decoding values

The same script calls for a program called *fim_all* [3] that finds frequent itemsets. The *fim_all* program has as input the file encoded as shown above and writes a new file in which all found frequent itemsets are presented with the number of their occurrences in the database. An output file created by this program looks like this:

```
(311)
5 (4)
5 12 (2)
5 12 14 (2)
5 12 14 21 (1)
```

```
5 12 14 22 (1)
5 12 14 34 (2)
...
```

The above example can be read as follows. The sequence of numbers in each row that are not placed between brackets is the itemset. The number between brackets following this sequence is the number of its occurences in the database. The very first row always consists of only one number between brackets and represents the total number of lines (timestamps) in the data file. In this case there are 311 lines. Item 5 occurs 4 times in the database. Numbers 5 and 12 together occur 2 times, etc.

The support for an itemset which occurs x times in the example database can thus be computed as $s = x/311$.

A certain support threshold can be set. The itemsets which have a frequency lower than this threshold will be omitted in the output file.

3.5 Creating association rules

The creation of association rules that are presented to the user will be discussed in this section. The files that consist of frequent itemsets are used as input for the creation of these rules. These files exist for each *po* (place, operator combination). At program start the files for the selected *po*'s are read. The itemsets that will be transformed to rules are those, that have a high support and confidence value. One of the common algorithms for retrieving freqent itemsets that can be transformed into rules is the Apriori algorithm.

3.5.1 The Apriori algorithm

Apriori [1] generates frequent itemsets, which are then used for the generation of the association rules.

First the 1–itemsets are generated. To do this item occurrences are counted. We will call this the first iteration. For each next iteration i the following applies: for each iteration i the itemsets that were created in the $(i-1)$th iteration L_{i-1} are used to generate candidate frequent i—itemsets L_i. This generation is done by joining these itemsets. A candidate frequent i–itemset can be discarded if it contains a subset which is not present in L_{i-1}. If an itemset from the frequent itemset has a support lower than some threshold value it is also discarded.

For example consider the following collection of frequent 2–itemsets L_2 (for $i = 2$):

$$L_2 = \{ab, bc, dc, de, ce\}$$

We assume that the support of all itemsets in this collection is higher than the preset threshold. We now want to construct L_3. The join step applied to L_2 gives the following candidate itemsets:

$$\{abc, bcd, bce, cde\}$$

The itemset *abc* should not be present in L_3, because the subset *ac* is not present in L_2. For the same reason *bcd* and *bce* should be removed. The subsets of *cde* are present in L_2 and if its support is higher than the minimal support this itemset should remain in L_3. The candidate frequent itemset L_3 (for $i = 3$) is:

$$L_3 = \{cde\}$$

3.6 Rule creation

In this section we will discuss how we mine the rules. We first describe the basic algorithm, and then discuss several options to distinguishthe interesting rules. First the available frequent itemsets (that are created by APRIORI) are read into memory. Here also a check is performed whether the itemset actually contain a so–called *answer item* as one of the items. If an answer item is available this itemset is marked as a possible rule. Answer values are special values that should be present in every rule as the rule head. In our case we are interested in the quality of operators prediction of the water height. The five possible answer items are: *Operator large under* (operator predicts with a large underguess), *Operator small under* (operator predicts with a small underguess), *Operator OK* (operator predicts almost the correct height), *Operator large over* (operator predicts with a large overguess) and *Operator small over* (operator predicts with a small overguess). Thus the answer items are 21, 22, 23, 24 and 25 (corresponding to the 5 values of property *Operator error*). If an itemset contains one of these values, the itemset is marked as having an answer. Note that an itemset can contain at most 1 answer item in our case.

So, for example, consider the following encoding fragment:

```
(311)
5 (4)
5 12 (2)
5 12 14 (2)
5 12 14 21 (1)
5 12 14 22 (1)
5 12 14 34 (2)
...
```

The two itemsets $\{5, 12, 14, 21\}$ and $\{5, 12, 14, 22\}$ contain an answer item (21 resp. 22). The possible rules, corresponding to the itemsets are $\{5, 12, 14\} \rightarrow \{21\}$ and $\{5, 12, 14\} \rightarrow \{22\}$.

All itemsets are read into a list. If the same itemset occurs more than once we do not save each occurrence, but remember the number of times an itemset occurs. After that the list is sorted. The itemsets are ordered in such a way, that retrieving them from the list can be done with a simple binary search.

Then the itemsets from the list are read one by one. First we retrieve the *GetBodySupport(I)* and the *GetMinSupport(I)* for each itemset I.

The function *GetBodySupport(I)* retrieves the support of the itemset J which has the same items as the itemset I, but without the answer item. So $J = I \backslash \{answeritem\}$. For example, if we are looking at an itemset $\{1, 2, 23\}$, which occurs 4 times ($s(1, 2, 23) = 4$), the *GetBodySupport(I)* retrieves the support of the itemset $\{1, 2\}$. Since this itemset is a subset of $\{1, 2, 23\}$ its support in this case will be ≥ 4.

The function *GetMinSupport(I)* retrieves the minimal support value of all itemsets which have the same items as the itemset I but is missing one item (this item can not be the answer value). For example, consider the following itemsets that are present in the list:

$\{1, 23\}$ occurs 4 times
$\{2, 23\}$ occurs 6 times
$\{1, 2, 23\}$ occurs 4 times
$\{1, 2\}$ occurs 5 times

Suppose we want to retrieve $GetMinSupport\{1, 2, 23\}$. The itemsets we will consider will then be $\{1, 23\}$ and $\{2, 23\}$. We take the minimum of the support values of these two itemsets:

$$GetMinSupport\{1, 2, 23\}) = min(s(\{1, 23\}), s(\{2, 23\})) =$$
$$= min(4, 6) = 4$$

The rule that will be created from the itemset I is already known here, since I contains 1 answer item. The rule r that is created from I will then be $I \backslash \{answeritem\}$. We compute the confidence $conf$ of r.

$$conf(r) = s(I)/s(GetBodySupport(I))$$

Then we compute the lift of the rule r. Remember that

$$lift(r) = conf(r)/exp(r)$$

We need to retrieve the expected confidence exp first. The function $GetExpectedConfidence(r)$ returns $exp(r)$. This is $s(answeritem)$. For the itemset $\{1, 2, 23\}$ the expected confidence is the support of the answer item 23.

3.7 Reducing the number of rules

The list of rules created in Section 3.6 can become very large. There are two options to reduce the number of rules.

3.7.1 Check Direction

The option *Check Direction* is one of the approaches to lower the number of possible rules. When this option is enabled it allows us to check if the *conf* of a candidate rule I such as $\{A, B\} \to \{C\}$ (in which C is an answer item) is higher than the *conf* of any other rule containing the same items, but with another *body-head* distribution of these items. If this value (let us call it $cd(I)$) is *true*, we can say that the answer item is indeed treated as an answer item, so the rule reads in one direction. Thus we check whether the rule is really $\{A, B\} \to \{C\}$ and not $\{A, C\} \to \{B\}$ or $\{B, C\} \to \{A\}$. If the *Check Direction* option is enabled but does not return a *true* value, the itemset will be marked as not interesting. To check the direction we use the values of *GetMinSupport* and *GetBodySupport*:

$$cd(I) = \begin{cases} true, & if\, GetMinSupport(I) > GetBodySupport(I) \\ false, & otherwise \end{cases}$$

3.7.2 Highest only

The option *Highest only* is another approach to lower the number of possible rules. The set of rules contains rules with a different body length. The bodies are itemsets, and for some rules these itemsets can be supersets of other rule bodies. For example the rule $\{0, 34\} \to \{25\}$ has a body which is a superset of the body of rule $\{34\} \to \{25\}$.
If the confidence of a rule r_1 with a rule body which is a superset of the body of rule r_2, is lower than the confidence of r_2 but has the same rule head, the rule r_1 will be removed from the rule list. This is done, because only if the confidence is higher, a longer rule will actually be interesting. For example, consider the following list of rules and their confidences, that were found for some *po*:

$\{34\} \to \{25\}$ $conf = 0.571$
$\{0, 34\} \to \{25\}$ $conf = 0.566$
$\{1, 34\} \to \{25\}$ $conf = 0.576$

The rules $\{0, 34\} \to \{25\}$ and $\{1, 34\} \to \{25\}$ both contain the item 34 in their rule body. The rule body of rule $\{34\} \to \{25\}$ consists only of item 34, and thus the other two rule bodies are a superset of this one.The confidence of $\{1, 34\} \to \{25\}$ is higher than the confidence of $\{34\} \to \{25\}$ ($0.576 \geq 0.571$) but the rule $\{0, 34\} \to \{25\}$ has a lower confidence ($0.566 < 0.571$). The rule $\{0, 34\} \to \{25\}$ therefore will be removed from the list if the option *Highest only* is enabled.

Bringing it all together, the computation of the confidence is shown in the following pseudo code.

```
procedure ComputeConfidence

    inputs: L, a list filled with itemsets

    for each itemset I from list L do

    if I contains an answer item then do

        MS := GetMinSupport(I);

        BS := GetBodySupport(I);

        if BS > 0 then do

            I.conf = I.s / BS;

            I.lift = I.conf / GetExpectedConfidence(I);

            if Check Direction option is enabled then do

                if  MS < BS then do

                    mark I as not interesting

            if Highest only option is on and HighestConfBody(I) > I.conf then do

                mark I as not interesting
```

The procedure that computes the confidence *ComputeConfidence* is also the one that marks the rules in the list as *not interesting.*

Chapter 4

Retrieving the operator

In the previous section we discussed the creation of interesting association rules. After these rules are created we would like to use them for operator retrieving. We will explain this in more detail.

The association rules were created from a set (or multiple sets) of data for different *po*'s. We will call the sets used for rule creation *learning sets.* One learning set is made for exactly one *po*. This means, that every place–operator combination (*po*) has its own set. There are, thus, $7 * 7 = 49$ learning sets (there are 7 different places and 7 operators).

Now suppose that there is another database which contains the same type of information as the learning sets and was created in a similar way. It contains the data for some *po*, but it was created in a different time interval. Like a learning set, it only contains the data for some single *po*. We assume that we do not know for which *po* this dataset was made and thus which learning set is the matching one. We will call this dataset a *validation set.* Note that the term "validation set" is often used in a somewhat different context.

One reason for retrieving the operator number is to check the correctness of rules we created from the *learning set.* The rules which are wrong, will not be usable for correct *po* retrieval. If the operator can be predicted by looking at the rules, this means, that the operator is really using them and is distinguished from other operators.

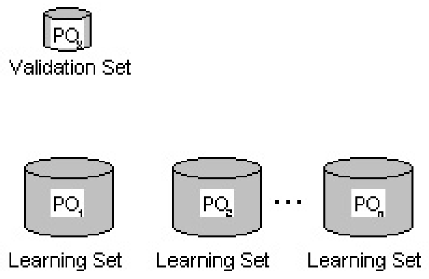


Figure 4.1: The *validation set* and the *learning sets*

The number of different learning sets is quite large (49), so it might be a good idea to limit the scope of our investigation to only a small number of *po*'s (learning sets). To do this we look at a certain set of *po*'s, which is a small subset of all *po*'s. The new set of *po*'s we are considering should include the *po* we are looking for.

We can use a set of all operators present at only one place. The place in this case will be fixed.

We look at a *validation set* for some po_{ia} (operator o_a at a fixed place p_i). Though we know what the value of a is, we try to retrieve it automatically. To do this we calculate the chances for operator o_a being operator o_2, operator o_3, etc (at place p_i). The considered learning sets are created for all the operators at place P (since there are 7 operators, there will be 7 learning sets considered). In this way we have 7 different *po*'s with one of them equal to o_a. For example, consider the *validation set* that was created for po_{42}. The identifier 42 in this context means, that it is operator number 2 working at place 4. We use all the *learning sets* corresponding to the operators from place 4 which are po_{42}, po_{43}, po_{44}, po_{45}, po_{46}, po_{47} and po_{48}.

For our tests we use multiple learning sets and one validation set. The learning sets that are used are corresponding to the *po*'s which all belong to a single place. We take a *validation set* which is made for a known *po* (for the same place as the learning sets).

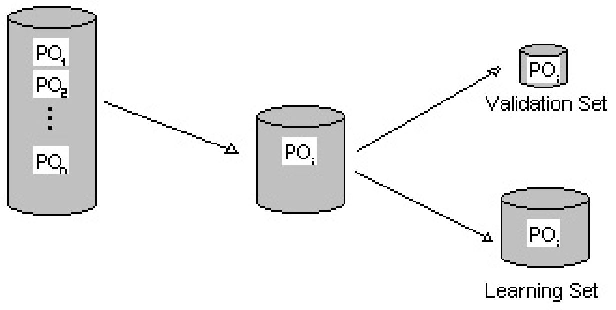


Figure 4.2: Creation of the *validation set* and the *learning set* for *po* at a fixed place *i*

Creation of the validation set and the learning set corresponding to one single place for a *po*–combination is shown in Figure 4.2. In this case we are considering only one place and the place is known. The datafile for a chosen place is parsed and split up into data files for each po_{ij} (place p_i operator o_j). The datafile for po_{ij} is then split into the validation set and the learning set.

To make a known *validation set*, which is not a subset of a *learning set* we have to split the initial database file for the *po* we are looking at (both place and operator are fixed) into two pieces. Simply splitting such a file in two pieces would not suffice, since certain events can be clustered and thus become part of only one of the two files. We traverse the initial file line by line and write each line with a chance p to the file *learningset* and with a chance $1-p$ to the file *validation set*. In this way all the lines are randomly distributed and $learningset \cap validation\ set = \emptyset$.

For splitting the file we use the Perl script *splitfile.pl*. A file with the name *filename* is split up into two files: *a_filename*, which consists of $\frac{1}{3}$ of the lines of the original file and *b_filename* which consists of $\frac{2}{3}$ of the lines. The larger file *b_filename* is used as the *learning set*. The smaller file *a_filename* is used as the *validation set*. We have chosen for this distribution for the following reason. The *learning set* must contain as many lines as possible for better rule construction, while it must not intersect with the *validation set*. The *validation set* can be smaller than the *learning set*, but it must contain enough examples so that as many rules as possible can be checked.

Until now we have considered different operators in only one place. Another interesting comparison would be to choose a set of the *po*'s which represent one operator, but at different places. The operator in this case will be fixed. By looking at such a set we can compare the behaviour of the same operator for different places. Some operator might make use of a certain rule while working at some place, but ignore this rule at some other place. For example, consider again the *validation set* that was created for po_{42}. We can use the *learning sets* corresponding to the *po*'s for operator 2 at all places which are po_{02}, po_{12}, po_{22}, po_{32}, po_{42}, po_{52} and po_{62}.

4.1 Definitions

4.1.1 Rules

To make the following calculations more clear to the user we will introduce two terms: rules and events. A rule consists of a rule body and a rule head. The rules are calculated by the program and presented to the user. A rule is of the form:

$$\{b_1, b_2, \ldots, b_n\} \rightarrow \{h_1\}$$

The length of the rule body can differ. Its length can vary between 1 and the maximal number of variables known. The rule head contains exactly one answer.

4.1.2 Events

An event is a line containing data that is read from some input file. It represents a state at some time t for some operator and place, *po* (in total $7*7 = 49$ different input files). Such an input line looks like this:

```
0 3 11 15 18 23 27 32 36 41
```

A single input file only contains events for the same *po*. Within an event for all the available variables (called properties earlier) at time t the value is given. An event is of the form:

$$\{b_1, b_2, \ldots, b_m\}$$

The number of variables in the event is always fixed. It is always set to 10 (since that is the number of different properties). The event contains exactly one answer item. For example, an event might look like this:

$$\{0, 3, 11, 15, 18, 23, 27, 32, 36, 41\}$$

4.1.3 Rule Collections

The number of different rule bodies is very large, since a lot of combinations of different items are possible. But a single rule body can have only a limited number of rule heads. In our case there are only 5 possible answer items, so there are 5 different rule heads possible, for each rule body.

To improve the effectiveness of looking at the rules we would like to look at a rule body with all possible rule heads.

We introduce a new term: *Rule Collection*. A Rule Collection consists of the collection of all found rules with the same body (the answer item should be encountered at least one time). A set consisting of all the rules considered can thus be divided into sets we will call Rule Collections. These Rule Collections consist of a number of rules with the same rule body. The number of rules in a single Rule Collection is thus equal to the total number of possible answer items. We assume that the number of answer items for a rule is always 5. Due to the support threshold and since we only consider the interesting rules it can occur that certain answer items are not present in the list of the available association rules. this rules still will be in a Rule Collection, but their confidence will be unknown.

A Rule Collection looks like this for the rule body b and all possible answer items a_i. In our case Rule Collections always consist of 5 rules:

$$C = \begin{cases} b \rightarrow a_1 \\ b \rightarrow a_2 \\ b \rightarrow a_3 \\ b \rightarrow a_4 \\ b \rightarrow a_5 \end{cases}$$

A Rule Collection is identified by the rule body b. For example, consider the following set of rules,. We will only consider the Rule Collections with a maximal number of answer items:

$\{2, 10\} \rightarrow \{21\}$
$\{10\} \rightarrow \{21\}$
$\{10\} \rightarrow \{22\}$
$\{2, 10\} \rightarrow \{22\}$
$\{2, 10\} \rightarrow \{23\}$
$\{10\} \rightarrow \{24\}$
$\{10\} \rightarrow \{25\}$
$\{10\} \rightarrow \{23\}$
$\{2, 10\} \rightarrow \{24\}$
$\{2, 10\} \rightarrow \{25\}$

There are two Rule Collections within this set. The first Rule Collection can be identified by the rule body [10] and contains $\{10 \rightarrow 21\}$, $\{10 \rightarrow 22\}$, $\{10 \rightarrow 23\}$, $\{10 \rightarrow 24\}$, $\{10 \rightarrow 25\}$. The other can be identified by the rule body $\{2, 10\}$. The advantage of looking at Rule Collections instead of simple rules is, that we can treat these rules as a single unit. This actually is true, since the sum of all the confidences of the rules in a collection C is always 1. In further calculations we can make use of this property.

$$conf(b \rightarrow a_1)+conf(b \rightarrow a_2)+conf(b \rightarrow a_3)+conf(b \rightarrow a_4)+conf(b \rightarrow a_5) = 1$$

4.2 Choosing Rules

The program presents the user with a list of rules (these are created from the learning set). For a fairly large database the number of these rules can be very large. The rules have a certain *confidence* and *lift*. We assume that at this point only the rules with a certain minimal *support* and *lift* are presented. Not all the rules that are presented are actually interesting rules. There are a few ways to try and locate the interesting rules and in this way decrease the total number of rules considered. In this section we will focus on the choice of these.

4.2.1 Combining Rules

The first thing we can do to decrease the total number of rules is combining multiple rules into one rule.

For a Rule Collection sometimes not all the confidences are known for every o (operator). But if we want to compare different rules in a Rule Collection to each other we need to know all of these confidences. For example, consider the following table, which presents the values of a Rule Collection for the rules with body 10 for a single place. o_2 till o_8 are in this example the operators. All of them are working in the same place. The values in the table correspond to the *conf* of a *rule* for each o combination. We consider only one place (in the table below the place id is 0).

	o_2	o_3	o_4	o_5	o_6	o_7	o_8
$\{10\} \rightarrow \{21\}$	—	—	0,147	—	—	0,144	—
$\{10\} \rightarrow \{22\}$	0,128	0,144	0,168	—	—	0,156	0,191
$\{10\} \rightarrow \{23\}$	0,353	0,394	0,462	0,381	0,400	0,311	0,397
$\{10\} \rightarrow \{24\}$	0,250	0,256	0,147	0,170	0,191	0,150	0,214
$\{10\} \rightarrow \{25\}$	0,179	0,119	—	0,245	0,155	0,240	0,130

One way would be to guess the unknown confidences, but this would be imprecise and has a big chance to lead to erroneous answers.

Let us look at how these unknown confidences can occur and how they can be retrieved. Unknown confidences are present for a rule if the rule has no occurrences for this *po* or if the support (or lift) of the rule is lower than the threshold. If the rule indeed never occurs the confidence should be 0. The support threshold is set outside the application. It is set in the step when the raw data is parsed into the data used by the application. Therefore, if a confidence is not present due to support being lower than this threshold, this can not directly be seen in the data.

The way to find out what the confidence of an empty cell is, is to look at the other rules in the Rule Collection and their confidences for the same *po*. We know that the sum of these should always be 1. This was already shown in the definition of a Rule Collection. Therefore, for instance the following holds:

$$\begin{gathered} conf(\{10\} \rightarrow \{21\}) + conf(\{10\} \rightarrow \{22\}) + \\ conf(\{10\} \rightarrow \{23\}) + conf(\{10\} \rightarrow \{24\}) + conf(\{10\} \rightarrow \{25\}) = 1 \end{gathered}$$

If only one of these values is unknown for a o, it can easily be calculated. In the example the confidence of $\{10\} \rightarrow \{25\}$ for po_{04} is:

$$1 - (0,147 + 0,168 + 0,462 + 0,147) = 0,076$$

The answer is not 0, which means that the support threshold of the rule was just too low. If more than one of the values is unknown for a o, it is more difficult to calculate the unknown confidences. We can, however, calculate the confidence of the occurrence of these rules together, which is the sum of these confidences. In the example there are 2 unknown confidences for o_5: $\{10\} \rightarrow \{25\}$ and $\{10\} \rightarrow \{21\}$. We can calculate the confidence of $\{10\} \rightarrow \{21 \vee 25\}$ (a rule with a body 10 and a head which contains 21 or 25). This confidence is :

$$1 - (0,381 + 0,17 + 0,245) = 0,204$$

Comparing rules to each other is difficult if for some po the confidences are not known for all of these rules. Therefore, if at least one o shares its confidence with another rule, these two rules should be seen as one rule (but with multiple answer item possibilities) for all po's. In our example this means that the rules $\{10\} \rightarrow \{21\}$ and $\{10\} \rightarrow \{22\}$ must be combined into one rule $\{10\} \rightarrow \{21 \vee 22\}$. For the sake of simplicity all rules which have at least one unknown confidence are combined (so in our example, also the rule with answer 25 is combined, although in this case the missing value could have been computed). Thus, the new Rule Collection table will look like this:

	o_2	o_3	o_4	o_5	o_6	o_7	o_8
$\{10\} \rightarrow (\{21\} \vee \{22\} \vee \{25\})$	0,397	0,350	0,392	0,449	0,409	0,539	0,389
$\{10\} \rightarrow \{23\}$	0,353	0,394	0,462	0,381	0,400	0,311	0,397
$\{10\} \rightarrow \{24\}$	0,250	0,256	0,147	0,170	0,191	0,150	0,214

In short the algorithm for computing rule confidence for rule r works as follows. We find all the rules which are in the same Rule Collection as r. If these are found, we check each of them for completeness (a complete rule has no unknown confidences for the po interface we are considering). If the rule is incomplete, the incomplete fields are calculated as described before by the use of other, complete rules. A new table is created for the Rule Collection with the combined rules. This table contains no fields with unknown confidences.

The program has a *Combine conf* option, which enables combination of the rules in the way described above.
With the *Combine conf* option switched on, the number of rules is reduced. This is due to the combination of multiple rules into one.

It also becomes more clear which rules are of no use to us. If, for example, after combination all the rules in a collection are combined into one rule the rule will always be *true* (confidence of 1) and thus not interesting (for example $\{10\} \rightarrow \{21 \vee 22 \vee 23 \vee 24 \vee 25\}$). It is also possible, that after combination rules get combined confidences, that do not differ a lot from each other. For example if the new table consists of the following 2 rules for some *po*: $\{10\} \rightarrow \{21 \vee 22 \vee 23 \vee 24\}$ with confidence 0.5 and $\{10\} \rightarrow \{25\}$ with confidence 0.5. Then the rule is perhaps also not interesting.

4.2.2 Choice based on lift value

Interestingness of a rule is mainly determined by its lift value. It is important to look at only those rules that are actually interesting and therefore we set a minimal lift threshold. Lift threshold can be applied for the rules which are not combined.

4.2.3 Lift modification

The number of possible rule heads is only 5 for our database. But each rule head is not encountered with the same frequency for each *po*. Some are encountered more often than others. For example in most operator data answer 25 is encountered with a frequency of about 0.3, while the answer value 23 has a frequency of around 0.05. The lift values can then fluctuate. Lift is calculated as:

$$lift = conf/exp$$

Because confidence can have a maximum value of 1, the maximum value of lift can be:

$$lift_{max} = 1/exp$$

With fluctuating expected confidence the value can differ a lot for different answer values. It makes comparing rules on lift with different expected confidence values very difficult, because the lift value is not equally distributed.

We would like to give all the lift values the same interval, therefore we introduce a new measurement function which we call λ. λ is a normalization function of lift and will be used as a replacement for lift.
We want to give λ a maximum value of 1, and a minimum of 0; 1 means interesting, 0 means not interesting.

We will use *conf* for rule confidence and *exp* for the expected rule confidence. For $conf/exp \geq 1$ we will use the following formula:

$$\lambda = \frac{conf/exp - 1}{1/exp - 1}$$

We also need to define λ for the case where $conf/exp < 1$. In this case *lift* is always between 0 and 1, so we only need to flip the value (we want 0 to stand for not interesting and 1 stand for interesting):

$$\lambda = 1 - mathitconf/mathitexp$$

Putting all the above rules together:

$$\lambda = \begin{cases} \frac{conf/exp-1}{1/exp-1}, & if\, conf/exp \geq 1 \\ 1 - conf/exp, & if 0 < conf/exp < 1 \end{cases}$$

We can now set a minimum λ for our algorithm. We can select to show only those rules with a λ above a certain threshold.

4.2.4 Lift threshold for a rule

Until now we looked at the lift value of a single rule at a certain *po*. For our calculation we want to filter out interesting rules by looking at their lift (we are looking at the rules created by the learning set). The lift value can of course differ a lot for different *po*'s, but in general the really interesting rules will have a relatively high λ value for most *po*'s. This is important for us, since we compare the validation set events with all the available *po*'s (in our case all *po*'s at a certain place). Thus, the best way to look at lift is to look at the average value of λ. Before the parsing process starts the user can select a minimal λ value. If the average λ value of a rule is below this value, this rule will be considered as non interesting. The formula for computing λ for some rule with m *po*'s at place 0 is:

$$\lambda(rule) = (\lambda_{po_{01}}(rule) + \lambda_{po_0 2}(rule) + \ldots + \lambda_{po_m}(rule))/m$$

4.3 Approach 1: Computing weights for each *po*

In this section we will in short describe the process of operator retrieval for a given learning set and a validation set. We will assume that the place P for which the test is done is fixed. As a validation set we will use some event sequence E. We want to retrieve by which operator E was created.

In the previous section we have described multiple approaches to find interesting rules from a learning set. Let us assume that we now have a list of association rules that are all considered to be interesting. We will use these rules for further calculations. We will look at each Rule Collection in the association rules list.

Suppose we are looking at the Rule Collection R. R has the confidence values that are given for each of the rules inside R and each of the 7 operators that are working at P. We use these values to assign a certain weight for each R–E–operator combination (for each of the 7 operators). The higher the weight value for a certain operator x is (for the R–E combination), the higher the chance that E defines an operator x under the consideration of R.

The calculation of the weight is explained in more detail in the following sections.

After we have calculated the weights considering R, we also calculate the weights for other available Rule Collections. The combination of these weights will then be used for operator prediction.

A *Rule Collection* can contain combined rules. The confidence of combined rules is calculated as described in Section 4.2.1. A *Rule Collection* is considered interesting, if at least one rule of this *Rule Collection* is considered interesting. For example, consider the following Rule Collection that is found to be interesting for some *po*:

$\{10\} \rightarrow (\{21\} \vee \{22\} \vee \{25\})$ (a combined rule)
$\{10\} \rightarrow \{23\}$
$\{10\} \rightarrow \{24\}$

The weight will in this case be calculated for the *Rule Collection* which is identified by the body $\{10\}$. If for example rule $\{10\} \rightarrow \{23\}$ would not be an interesting rule but the others are, the *Rule Collection* $\{10\}$ remains interesting (because at least one other rule with a body $\{10\}$ is interesting).

When we have identified an interesting *Rule Collection C* for some *po* we first must check if the *Rule Collection* is also interesting for all other *po*'s that will be considered for weight computation. This is important, because we will want to make sure all the *po*'s are equally depending on the rules. For example, if we are considering only the place *HVH* then the *Rule Collection* must be interesting for all operators (2 till 8) that are working at *HVH*. If that is *true* we can use it to compute the weight for a sequence of events E for each *po*. The weight for $E - C - po_x$ is computed by considering only those events from E that contain the items which are also inside the body of C, and one item which is an answer value (21,22,23,24,25).

For example, consider a seqence of events E (for an unknown *po*–combination with a known place):

$$\{0, 3, 10, 16, 19, 22, 28, 31, 36, 41\}$$

$$\{0, 3, 12, 15, 18, 25, 26, 32, 36, 41\}$$

$$\{0, 2, 10, 15, 17, 23, 27, 32, 36, 42\}$$

$$\{0, 3, 11, 15, 18, 21, 27, 31, 35, 41\}$$

$$\{1, 3, 11, 15, 18, 25, 27, 32, 36, 41\}$$

$$\{1, 3, 11, 15, 18, 24, 27, 32, 36, 42\}$$

$$\{1, 2, 11, 15, 18, 24, 26, 31, 35, 42\}$$
$$\{1, 3, 10, 15, 18, 25, 27, 32, 35, 42\}$$

If we want to compute the weight for E for different *po*'s for the Rule Collection which is identified by the body 10 we get the following updated event sequence E_{new} (we only consider the events which contain item 10):

$$\{0, 3, 10, 16, 19, 22, 28, 31, 36, 41\}$$
$$\{0, 2, 10, 15, 17, 23, 27, 32, 36, 42\}$$
$$\{1, 3, 10, 15, 18, 25, 27, 32, 35, 42\}$$

We then assign a weight w_{po_i} to E_{new} for each considered po_i. We calculate the weight by comparison of E_{new} with the Rule Collection for each *po*.

4.3.1 Weight computation for one event

Let us first look at retrieving the chance of an event sequence being some *po* for Rule Collection C, when the event sequence consists of exactly one event e. We can treat this chance as a weight. From this event we will only consider the items that are also present in C and the answer item.

For example such event e sequence might look like this:

$$E_{new} = \{0, 3, 10, 16, 19, 22, 28, 31, 36, 41\}$$

Suppose that the Rule Collection C has the body $\{0, 16\}$. Then only the items $\{0, 16, 22\}$ from e will be considered. We need to know the confidences for all rules in the Rule Collection C.

Suppose A and X are some stochastic events. For a given rule r (*body* $\rightarrow$ *head*) the event A means that the rule holds. $P(A|X)$ stands for the chance that A happens if X happens. Consider a rule A. The confidence of the rule A for some operator X at work will be treated as the chance $P(A|X)$. In this section we will only consider 7 operators working at a fixed placed place. We assume that all operators are working equally often at place p. The chance for an operator at work at a certain time is thus the same for m *po*'s.

$$P(po_{p1}) = P(po_{p2}) = \ldots = P(po_{pm})$$

$P(A)$ can be computed as the sum of all chances related to A, divided by the total number of *po*'s m.

$$P(A) = \sum_{i=1}^{m} P(A|o_i)/m$$

$$P(B) = \sum_{i=1}^{m} P(B|o_i)/m$$

$P(X|A)$ stands for the chance that operator X is at work when the rule r holds. $P(X|A)$ can now be computed with the use of the Bayes' theorem. The Bayes' probability theorem looks like this:

$$P(X|A) = \frac{P(A|X)P(X)}{P(A)}$$

Let us consider an example. The following table with confidences is given:

	po X	po Y
rule A	0.6	0.1
rule B	0.4	0.9

The different o's are denoted here by X and Y. In this example we assume that there are no other o's. Two rules that are excluding each other are denoted by A and B. The rules A and B together form a Rule Collection that contains only two rules.

We assume that both operators (X or Y) are working equally often (place is fixed).

$$P(X) = P(Y)$$

$$P(X) + P(Y) = 1$$

$$P(X) = P(Y) = 1/2$$

Consider the confidences for the rules A and B for operators X and Y, that are taken from the table:

$$P(A|X) = 0.6$$

$$P(A|Y) = 0.1$$

$$P(B|X) = 0.4$$

$$P(B|Y) = 0.9$$

The formula for computing $P(A)$ and $P(B)$ is:

$$P(A) = P(A|X)P(X) + P(A|Y)P(Y)$$

$$P(B) = P(B|X)P(X) + P(B|Y)P(Y)$$

For the example this gives:

$$P(A) = (0.6 + 0.1)/2 = 0.35$$

$$P(B) = (0.4 + 0.9)/2 = 0.65$$

Suppose we are considering a single event which contains the rule body and the rule head of A. The chance that operator X is at work if such an event happens is:

$$P(X|A) = \frac{P(A|X)P(X)}{P(A)} = \frac{0.6 * 0.5}{0.35} = \frac{6}{7}$$

and analogous for event which contains the rule head and the rule body of Y:

$$P(Y|A) = \frac{P(A|Y)P(Y)}{P(A)} = \frac{0.1 * 0.5}{0.35} = \frac{1}{7}$$

Since always either Y or X is valid, also the following is true:

$$P(X|A) + P(Y|A) = 1$$

We can check it for our case:

$$P(X|A) + P(Y|A) = \frac{6}{7} + \frac{1}{7} = 1$$

In the above example we considered the operators working at a fixed place. If we consider a fixed operator at work at different places the calculations are done analogously.

4.3.2 Chance computation for event sequence

In our case test files contain not one event, but a sequence of events. We only use itemsets from events which match the Rule Collection body and have an answer item. Now let us look at how the chances are calculated if more than one event happens in sequence. For example we are matching an event set against the Rule Collection with body $\{8\}$. The event set contains the following itemsets which are matching the rule body and contain an answer item.

$\{8, 21\}$
$\{8, 21\}$
$\{8, 24\}$

We will denote this itemset sequence (which can also be seen as a sequence of rules) as AAB. We assume that there are only two operators, which are X and Y. What is the chance, it will be X? We assume that A and B are independent, so the order in which they happen does not matter (AAB is treated in the same way as ABA and BAA). The main step is the same as described previously (use Bayes' theorem):

$$P(X|AAB) = \frac{P(AAB|X)P(X)}{P(AAB)} =$$

$$= \frac{P(A|X)P(A|X)P(B|X)P(X)}{P(AAB)}$$

Here we assume that A and B are independant from each other, given X:

$$P(AAB|X) = P(A|X)P(A|X)P(B|X)$$

Calculating $P(AAB)$ is more difficult. Analogous to the calculation of $P(A)$ (as described before) we get:

$$P(AAB) = P(AAB|X) * P(X) + P(AAB|Y) * P(Y) =$$

$$P(A|X) * P(A|X) * P(B|X) * P(X) + P(A|Y) * P(A|Y) * P(B|Y) * P(Y)$$

The above formula is created for 2 operators and 3 rules (or itemsets). For a large number of operators or rules this formula will be very long. However, if we are comparing different operators for the same event sequence, and want to see which one has higher chance it is enough to calculate only the numerators of the fraction, and compare them to each other. The largest numerator is equivalent to the highest chance. This can be done, because the denominator is a constant for the same event sequence. Thus, in the above example, we only need to determine the value of $P(A|X)P(A|X)P(B|X)P(X)$. For different operators we can simply normalize the retrieved nominators to retrieve the actual chance.

4.3.3 Correcting the answer

When we have computed the weights (product of the chances) for a certain Rule Collection over all events from the validation set we expect that the highest weight value gives us the correct o. This will not always be the case.

One reason for this is that the rule chosen for comparison is actually still not correct or totally valid. The incorrectness might be because there was not enough data in the learning set or the rule was incidental. There is very little that can be done to remove incidental rules. Rules that are created with not enough data can be ignored by increasing the support threshold.

Another reason is that for some po's the same rules can behave similar. This means the following. If two or more rules from the same Rule Collection have the same confidences for po_i and po_j, then they can not be distinguished for this Rule Collection.

For example, look at Figure 4.3. It represents the graphs of confidences for different po's for a Rule Collection R (for a certain place p).

It can clearly be seen, that the shape of the graphs for po_2 and po_4 are rather similar, while the shape is different from po_3, po_5, po_6, po_7, po_8. po_3, po_5, po_6, po_7, po_8 also all have approximately the same shape, but for instance po_3 and po_5 differ more from each other, than others. Because po_2 in the graph looks a lot like po_4 we can not decide which of the two it is if po_2 is guessed to be the correct po. But we can almost certainly say that if the program guesses po_2 to be the correct po, the chance for po_3, po_5, po_6, po_7,

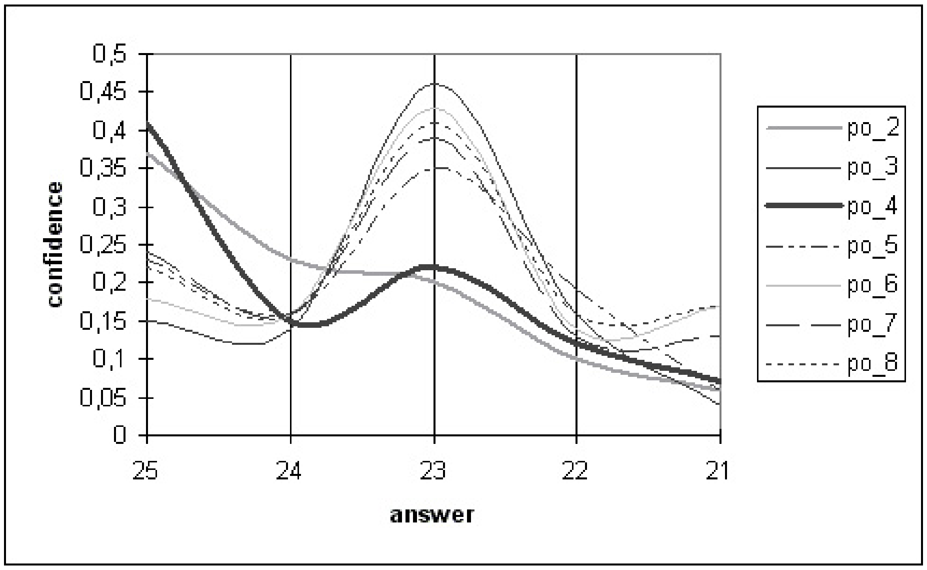


Figure 4.3: Comparison of confidences for different *po*'s for a Rule Collection (for a certain place)

po_8 to be the correct *po* is very low. The term "looks like" we use in this context is for now rather vague and seems to be intuitive, but we will make the definition more precise by determining the distance between operators (*po*'s) in the next subsection.

4.3.4 Distance between operators

One approach to see if the *po* answer curve for a certain Rule Collection looks a lot like the curves for other *po*'s is calculating a distance between the different *po*'s for that Rule Collection.

The distance between two operators for some Rule Collection at a fixed place (or in general different *po*'s) is computed in the following way: we compute the square of the difference between the confidences of the rule r for both operators. Then we sum up the computed distances from rules inside the Rule Collection (5 rules: r_1,r_2,r_3,r_4 and r_5). Since we are doing the test for a certain place, we consider 7 *po*'s which are available for this place. We compute the distance between *po* x and y for a certain Rule Collection with the following formula:

$$distance_p(x,y) = \sum_{i=1}^{5} \left(conf_{xp}(r_i) - conf_{yp}(r_i)\right)^2$$

We use the square for two reasons: to make the distance always positive and to improve the distance measurement (small differences will remain small, bigger ones will be larger).

The computed distances between the *po*'s are compared to some threshold value. A 2–dimensional array is created with all *po*–distances for a single Rule Collection. If the distance between po_a and po_b is smaller than a user preset threshold, then the corresponding array element is set to *true*. We calculate each distance. Later on this table is used as a reference to find similar *po*'s.

For only three example *po*'s such table might look like this:

	po_1	po_2	po_3
po_1	—	0.1	0.5
po_2	0.1	—	0.4
po_3	0.5	0.4	—

If the user has set the minimal threshold to 0.15, then the new reference table will look like this.

	po_1	po_2	po_3
po_1	false	true	false
po_2	true	false	false
po_3	false	false	false

From this table it can be seen that only po_1 looks like po_2 (and the other way around). This information can be used in the following way. If the program predicts for some events that po_1 is the correct *po*, automatically also po_2 becomes a guess. If po_3 is a prediction — no other events are selected.

What we have seen in this subsection is that not all the answers (*po* predicted by our algorithm) can be equally trusted. The answer can give an indication which set of *po*'s contains the best predictions.

4.3.5 Finding best match

In the previous sections we have described how to compute the weight for one Rule Collection C and how to choose the candidate *po* (or multiple candidate *po*'s). In this section we will focus on how to predict candidate *po*'s for multiple Rule Collections.

Predicting the *po* by looking at only one Rule Collection is the base of our prediction technique. But we want to use multiple Rule Collections that are available for the *po*'s, so we do not lose the available information we have.

First, we want to ensure to use only those Rule Collections (created from the learning set) that actually are interesting. The interestingness of a Rule Collection is already described in a previous section. If the step of removing the non interesting Rule Collections from the total rule list is done correctly, we should now be able to use all the remaining rules (and Rule Collections) from this list. In general the following is true: the longer the rule, the lower its support. We already applied a support threshold on the rules, but we can limit it even more. In case we want to limit our search to only rules with a certain length there is an option *end at rule* which can be used for this. This option defines until which rule in the list we should go, while computing the total weight. The rules in the list are ordered by length. We can set the *end at rule* at the ID of the last rule in the rule list, to traverse the entire list. We can set this to a lower value, in order to look at shorter rules.

Suppose we now have chosen a list of Rule Collections (from the learning set) we would like to include into the prediction process. First we make a list of *po*'s that are considered for the prediction. For example we can use all *po*'s with $place = 1$. We assign a label to each of the list elements.

For each Rule Collection we compute the candidate *po*'s. We assign a counter to each *po*. For these *po*'s we add 1 to the *po* counter.

$\{8\} \rightarrow \{23\}$
$\{8\} \rightarrow \{24\}$
$\{8\} \rightarrow \{25\}$

The corresponding Rule Collection is identified by the rule body $\{8\}$. The rules $\{8\} \rightarrow \{21\}$ and $\{8\} \rightarrow \{22\}$ are not considered interesting, since they are not present in the rule list. Now suppose we have a validation set. The first valid itemset from this validation set contains the item $\{8\}$ and some answer value. The program predicts that the correct *po*'s are po_2 and po_4. This means, that 1 is added to the counter of po_2 and po_4. The counters for the other *po*'s will not change their value.

In this way, after all the Rule Collections are checked, we can compare the value of the *po* counters to each other. For example consider the following set of rules corresponding to some *po* combination:

$\{3\} \rightarrow \{21\}$
$\{3\} \rightarrow \{22\}$
$\{3\} \rightarrow \{23\}$
$\{7\} \rightarrow \{23\}$
$\{8\} \rightarrow \{23\}$
$\{8\} \rightarrow \{24\}$
$\{8\} \rightarrow \{25\}$
$\{10\} \rightarrow \{23\}$
$\{10\} \rightarrow \{24\}$

There are 4 Rule Collections in this set of rules. The Rule Collections of these rules are identified by the rule bodies $\{3\}$, $\{7\}$, $\{8\}$ and $\{10\}$.
Suppose the *validation set* contains the following itemsets:

$\{8, 23\}$
$\{7, 23\}$
$\{8, 24\}$
$\{9, 22\}$
$\{10, 24\}$

The itemset $\{9, 22\}$ will not be considered, because there is no interesting Rule Collection which is identified by the body $\{9\}$. The rest of the items in the *validation set* do reference an interesting Rule Collection and therefore will be considered.
Let us assume that we are looking at 7 *po*'s numbered from 2 till 8. Now suppose the algorithm predicts the following *po*'s for the itemsets. For $\{8, 23\}$ po_2 or po_4 (has the highest weight) are predicted. For $\{7, 23\}$ po_2 or po_3 (has

the highest weight) or po_6 are predicted. For $\{8, 24\}$ po_3 (has the highest weight) is predicted. For $\{10, 24\}$ po_2 or po_3 or po_4 (has the highest weight) or po_6 or po_7 are predicted.

We do not need to take into account yet what the *po* with the highest weight was for every rule. We only look at the predicted *po*'s. The counters for every considered *po* will be updated as follows (all counters start at value 0):

$po_2 = 0 + 1 + 1 + 1 = 3,$
$po_3 = 0 + 1 + 1 + 1 = 3$
$po_4 = 0 + 1 + 1 = 2$
$po_5 = 0$
$po_6 = 0 + 1 + 1 = 2$
$po_7 = 0 + 1 = 1$
$po_8 = 0$

The highest counter value is that of po_2 and po_3, which equals 3. These two *po*'s will be assumed to be the best prediction for this validation set; po_5 and po_8 will be considered as worst prediction.

The *po* corresponding to the counter with the highest value is thus considered as the best prediction. If multiple counters have approximately the same highest value, they are all considered as best prediction.

4.3.6 Best bonus

The above method works well if we are careful with the predictions of the algorithm and do not trust the best prediction for each line. For example for $\{8, 23\}$ po_2 or po_4 are predicted. As already described in the previous section, two or more *po*'s are predicted, when the *distance* between these *po*'s is very small, and one of them is actually predicted as the best *po* match. The best *po* match has the highest weight value.

The option *Best bonus* switched on allows us to give the *po* with the highest weight an additional bonus value if that *po* is predicted, that will be added to this *po* counter. The value of the bonus can be set by the user and can be equal to any integer value. For the example described earlier, if the bonus value is set to 2 the counters for every considered *po* will be updated as follows (all counters start at value 0):

$po_2 = 0 + 1 + 1 + 1 = 3,$
$po_3 = 0 + (1 + 2) + (1 + 2) + 1 = 7$
$po_4 = 0 + (1 + 2) + (1 + 2) = 6$
$po_5 = 0$
$po_6 = 0 + 1 + 1 = 2$
$po_7 = 0 + 1 = 1$
$po_8 = 0$

The highest counter value is that of counter of po_3, which equals 7. Now po_3 will be assumed as the best prediction for this validation set. Note that po_4 is now considered a better prediction than po_2.

4.3.7 Only high lift

The option *only high lift* can be enabled or disabled. If the option *only high lift* is enabled, we filter out the rules which are chosen as candidates, but have at least one λ (see Section 4.2.3) value for some *po* which is lower than the selected threshold (only those rules remain which have $\lambda \geq$ *threshold* for all considered *po*'s). This option looks a lot like the minimal threshold that can be set in the beginning, but is different. The minimal threshold considers an average λ which must be higher than some threshold, while when the option *only high lift* is used one λ is enough to ignore the rule. This is done because for the correct *po* retrieving process we must be sure about every rule for every *po*.

4.4 Approach 2: Comparing of confidences

There is a second, easier approach for retrieving the correct *po*. However, the prediction proved to be less accurate than the first approach. With this approach we compare the interesting rules to events. This is a very simple approach which boils down to checking the confidences.

Traverse all the rules. For each rule, look at all the events in the input. Count the number of times the rule body is a subset of the event body and the number of times the rule head is the same as the event head. From these numbers we can calculate the confidence of a rule in the validation set and match it to the confidence of the rule in the set of rules corresponding to a given *po*. The less the difference between the two is, the better match is found. We will call the sum of these differences the *weight*.

$$weight = \sum_{all-rules-r} |confFound - confRule(r)|$$

We consider the algorithm below used for comparing rules to events and computing the weight. The algorithm gives a percentage as result, which represents the matching factor of the validation to the learning set. The closer the percentage is to 100, the better the match is.

```
Match = 0;
MatchNumber = 0;
for each rule R do
{
  TotalLeft = 0;
  TotalRight = 0;
  for each event E do
  {
    if body(R) is subset of E
    {
      TotalLeft = TotalLeft + 1;
      if also answer matches
      {
        TotalRight= TotalRight + 1;
      }
    }
  }
  if TotalLeft > 0
  {
    CalcConfidence = TotalRight / TotalLeft;
    MatchNumber = MatchNumber+1;
    Match = Match + (CalcConfidence/R.FConfidenceArray[OperatorAndPlace]);
  }
}
```

```
if  (Match / MatchNumber) > 0
{
  result = 200 - (Match / MatchNumber)*100;
}
else
{
  result = (Match / MatchNumber)*100;
}
```

Chapter 5

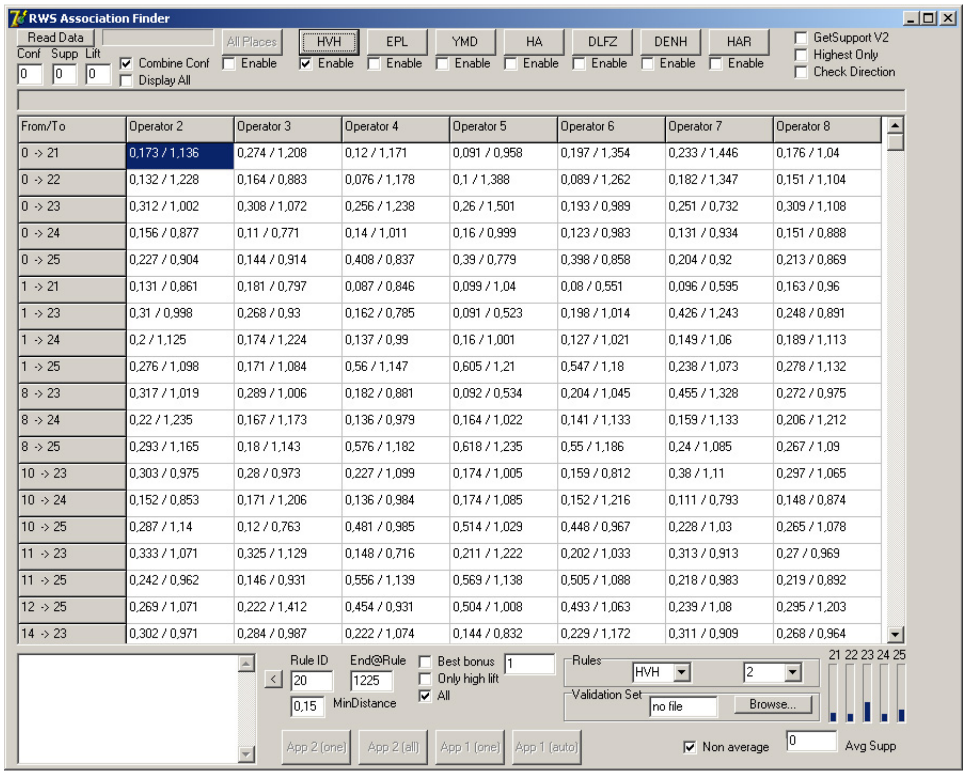
About the program

The program *RWS association finder* was built to be able to create association rules and predict the *po* with the use of these rules. As input data the files are used that are created by the *fim_all* program. The process of this creation and the input data format are described in Section 3.4. The input data files are made available as one file per *po*. The files are ordered in folders by place. Folder names equal the matching place name. The program is written in Delphi and can be used as a stand alone application.

5.1 User interface

The user can select the places he wishes to analyse. With this selection the *po*'s which are available for this place will automatically be selected. It is not possible to select any combination of *po*'s in some other way, since each place contains exactly 7 operators, with the selection of one place we select 7 *po*'s. In Figure 5.1 the place *HVH* is selected. As a result the rules are created for po_2 till po_8 in *HVH*.

The parsing process can be initiated with the use of different options. The options *Highest Only* and *Check Direction* can be switched on or off by checking the boxes next to them. these options enable or disable the variables *Highest Only* and *Check Direction* that were described in Sections 3.7.2 and 3.7.1. The other two options are *Combine conf* and *Display All*. The *Combine conf* option is described in Sections 4.2.1. Option *Display All* forces the display of all rules, even if the rules are found not to be interesting.



From/To	Operator 2	Operator 3	Operator 4	Operator 5	Operator 6	Operator 7	Operator 8
0 -> 21	0,173 / 1,136	0,274 / 1,208	0,12 / 1,171	0,091 / 0,958	0,197 / 1,354	0,233 / 1,446	0,176 / 1,04
0 -> 22	0,132 / 1,228	0,164 / 0,883	0,076 / 1,178	0,1 / 1,388	0,089 / 1,262	0,182 / 1,347	0,151 / 1,104
0 -> 23	0,312 / 1,002	0,308 / 1,072	0,256 / 1,238	0,26 / 1,501	0,193 / 0,989	0,251 / 0,732	0,309 / 1,108
0 -> 24	0,156 / 0,877	0,11 / 0,771	0,14 / 1,011	0,16 / 0,999	0,123 / 0,983	0,131 / 0,934	0,151 / 0,888
0 -> 25	0,227 / 0,904	0,144 / 0,914	0,408 / 0,837	0,39 / 0,779	0,398 / 0,858	0,204 / 0,92	0,213 / 0,869
1 -> 21	0,131 / 0,861	0,181 / 0,797	0,087 / 0,846	0,099 / 1,04	0,08 / 0,551	0,096 / 0,595	0,163 / 0,96
1 -> 23	0,31 / 0,998	0,268 / 0,93	0,162 / 0,785	0,091 / 0,523	0,198 / 1,014	0,426 / 1,243	0,248 / 0,891
1 -> 24	0,2 / 1,125	0,174 / 1,224	0,137 / 0,99	0,16 / 1,001	0,127 / 1,021	0,149 / 1,06	0,189 / 1,113
1 -> 25	0,276 / 1,098	0,171 / 1,084	0,56 / 1,147	0,605 / 1,21	0,547 / 1,18	0,238 / 1,073	0,278 / 1,132
8 -> 23	0,317 / 1,019	0,289 / 1,006	0,182 / 0,881	0,092 / 0,534	0,204 / 1,045	0,455 / 1,328	0,272 / 0,975
8 -> 24	0,22 / 1,235	0,167 / 1,173	0,136 / 0,979	0,164 / 1,022	0,141 / 1,133	0,159 / 1,133	0,206 / 1,212
8 -> 25	0,293 / 1,165	0,18 / 1,143	0,576 / 1,182	0,618 / 1,235	0,55 / 1,186	0,24 / 1,085	0,267 / 1,09
10 -> 23	0,303 / 0,975	0,28 / 0,973	0,227 / 1,099	0,174 / 1,005	0,159 / 0,812	0,38 / 1,11	0,297 / 1,065
10 -> 24	0,152 / 0,853	0,171 / 1,206	0,136 / 0,984	0,174 / 1,085	0,152 / 1,216	0,111 / 0,793	0,148 / 0,874
10 -> 25	0,287 / 1,14	0,12 / 0,763	0,481 / 0,985	0,514 / 1,029	0,448 / 0,967	0,228 / 1,03	0,265 / 1,078
11 -> 23	0,333 / 1,071	0,325 / 1,129	0,148 / 0,716	0,211 / 1,222	0,202 / 1,033	0,313 / 0,913	0,27 / 0,969
11 -> 25	0,242 / 0,962	0,146 / 0,931	0,556 / 1,139	0,569 / 1,138	0,505 / 1,088	0,218 / 0,983	0,219 / 0,892
12 -> 25	0,269 / 1,071	0,222 / 1,412	0,454 / 0,931	0,504 / 1,008	0,493 / 1,063	0,239 / 1,08	0,295 / 1,203
14 -> 23	0,302 / 0,971	0,284 / 0,987	0,222 / 1,074	0,144 / 0,832	0,229 / 1,172	0,311 / 0,909	0,268 / 0,964

Rule ID: 20
End@Rule: 1225
MinDistance: 0,15
Best bonus: 1
Only high lift
All [✓]
Rules: HVH 2
Validation Set: no file
Non average [✓]
Avg Supp: 0

Figure 5.1: RWS association finder interface

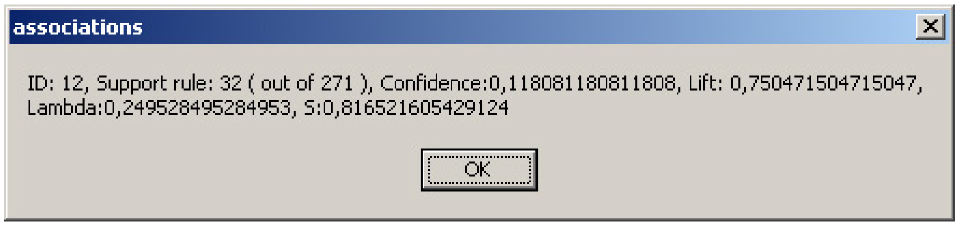
After pressing the *Read data* button, the learning set files will be read and the created association rules will be shown on the grid in the middle of the interface.

The grid where the association rules are shown, has a vertical and a horizontal bar. The *po*'s which are selected for the rules are shown on the horizontal bar. On the vertical bar the descriptions of the rules are placed. The rule items in the description are shown as numbers. For better readability these numbers can be translated into words. This happens when the mouse pointer is moved over any cell in a row on the grid. The rule is then decoded into words. The itemset names can be custom set by the user. For each item number a name can be used. An external file is used to store these data. This file is stored in the same directory as the executable and is called *config.txt*.

An example of this files contents is shown below:

```
10 NIGHT
11 MORNING
12 DAY
13 EVENING
14 EVENING_SHIFT
15 DAY_SHIFT
16 NIGHT_SHIFT
17 OPERATOR_EARLY
18 OPERATOR_ON_TIME
19 OPERATOR_LATE
20 OPERATOR_VERY_LATE
```

Each rule (represented as a row) contains 7 cells. Each cell provides information on the rule in the row for a different *po*. A cell contains a confidence and a lift value of the rule for a *po*. The user can get more information on a rule for a certain po, by left–clicking on a cell. A pop–up will appear with additional information, such as rule *ID*, *support*, total number of occurrences, *confidence*, *lift*, λ, see Figure 5.2.



```
ID: 12, Support rule: 32 ( out of 271 ), Confidence:0,118081180811808, Lift: 0,750471504715047,
Lambda:0,249528495284953, S:0,816521605429124
```

Figure 5.2: More information on a selected rule

It is also possible to quickly analyse each *po* for answer item distribution. Holding a mouse over a *po* column header in the bottom right of the interface a distribution is shown of the answer items 21 till 25, see Figure 5.3.

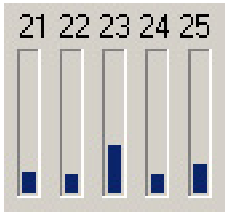


Figure 5.3: Answer item distribution

After a list of rules is created for the selected place we can test the two approaches for *po* retrieval. A validation set can be selected with the *Browse* button. After a validation set file has been selected we can select an approach for *po* retrieval (*App 1 (auto)* for Approach 1 and *App 2 (Auto)* for Approach 2). The options *Best Bonus* (its value can be set, see Section 4.3.6) and *Only High Lift* can be switched on or off (see Section 4.3.7). We can set the minimal distance in the *Min dist* box and set the number of rules that are considered for retrieval in the *End At* box.

The results of the test are shown in the text field at the bottom left of the interface. A list of the tested *po*'s is presented. Behind the *po* number its weight is given.

Chapter 6

Testing

6.1 Predicting *po* with default values

In this subsection we will try to retrieve the correct *po* of a validation set. We will use the program with all the default values and see how well the prediction will be. The default options are:

option	value	notation
Place	HVH	p
Check Direction	disabled	cd
Confidence	0	c
Best bonus	disabled	bb
Support	0	s
Only high lift	disabled	ohl
Lift	0	l
minDistance	0.15	md
Combine Conf	enabled	cc
end at rule	5225	end
Display all	disabled	da
Highest Only	disabled	ho

To give the reader some ompression of the algorithm, we provide some detailed results of experiments in Section 6.1.1 to 6.2.

6.1.1 Testing the approaches

First we will test both available approaches (Approach 1 and Approach 2) for their rate of correct predictions. To do this we will use the learning set also as the validation set. The program should always return the correct prediction. We use the learning set for the place HVH. We use the validation sets for HVH, and thus do the testing for a fixed place. The testing will first be done with Approach 1. We will use default settings. The output that will be presented throghout this chapter shows the weight assigned to each of the tested *po*'s. The higher the weight is, the higher the chance that the *po* is the correct prediction according to the algorithm we use.The *po* with the highest weight is thus the best prediction.

We use the validation set for po_2 (*b_hvh.dat.invoer2.txt*).

```
GLOBAL po 2 ->209
GLOBAL po 3 ->97
GLOBAL po 4 ->67
GLOBAL po 5 ->41
GLOBAL po 6 ->87
GLOBAL po 7 ->45
GLOBAL po 8 ->71
```

The algorithm predicts po_2. This is a correct prediction.

We use the validation set for po_3 (*b_hvh.dat.invoer3.txt*).

```
GLOBAL po 2 ->105
GLOBAL po 3 ->205
GLOBAL po 4 ->75
GLOBAL po 5 ->56
GLOBAL po 6 ->56
GLOBAL po 7 ->73
GLOBAL po 8 ->57
```

The algorithm predicts po_3. This is a correct prediction.

We use the validation set for po_4 (*b_hvh.dat.invoer4.txt*).

```
GLOBAL po 2 ->110
GLOBAL po 3 ->94
GLOBAL po 4 ->119
GLOBAL po 5 ->45
GLOBAL po 6 ->69
GLOBAL po 7 ->42
GLOBAL po 8 ->103
```

The algorithm predicts po_4. This is a correct prediction.

We use the validation set for po_5 (*b_hvh.dat.invoer5.txt*).

```
GLOBAL po 2 ->37
GLOBAL po 3 ->44
GLOBAL po 4 ->18
GLOBAL po 5 ->218
GLOBAL po 6 ->41
GLOBAL po 7 ->120
GLOBAL po 8 ->25
```

The algorithm predicts po_5. This is a correct prediction.

We use the validation set for po_6 (*b_hvh.dat.invoer6.txt*).

```
GLOBAL po 2 ->81
GLOBAL po 3 ->36
GLOBAL po 4 ->43
GLOBAL po 5 ->33
GLOBAL po 6 ->218
GLOBAL po 7 ->35
GLOBAL po 8 ->77
```

The algorithm predicts po_6. This is a correct prediction.

We use the validation set for po_7 (*b_hvh.dat.invoer7.txt*).

```
GLOBAL po 2 ->51
GLOBAL po 3 ->67
GLOBAL po 4 ->37
GLOBAL po 5 ->124
GLOBAL po 6 ->47
GLOBAL po 7 ->208
GLOBAL po 8 ->45
```

The algorithm predicts po_7. This is a correct prediction.

We use the validation set for po_8 (*b_hvh.dat.invoer8.txt*).

```
GLOBAL po 2 ->74
GLOBAL po 3 ->59
GLOBAL po 4 ->101
GLOBAL po 5 ->34
GLOBAL po 6 ->92
GLOBAL po 7 ->49
GLOBAL po 8 ->204
```

The algorithm predicts po_8. This is a correct prediction.

All the predicted aswers were correct. Approach 1 seems to retrieve correct *po*'s.

Now the testing will be done with Approach 2. To enable the usage of Approach 2 the Combine answer option must be disabled, the rest of the settings remains default.

We use the validation set for po_2 (*b_hvh.dat.invoer2.txt*).

```
Best match found for operator 2 at place HVH (match 100%)
```

The algorithm predicts po_2. This is a correct prediction.

We use the validation set for po_3 (*b_hvh.dat.invoer3.txt*).

```
Best match found for operator 3 at place HVH (match 99,94%)
```

The algorithm predicts po_3. This is a correct prediction.

We use the validation set for po_4 (*b_hvh.dat.invoer4.txt*).

```
Best match found for operator 4 at place HVH (match 99,85%)
```

The algorithm predicts po_4. This is a correct prediction.

We use the validation set for po_5 (*b_hvh.dat.invoer5.txt*).

```
Best match found for operator 5 at place HVH (match 100%)
```

The algorithm predicts po_5. This is a correct prediction.

We use the validation set for po_6 (*b_hvh.dat.invoer6.txt*).

```
Best match found for operator 6 at place HVH (match 99,85%)
```

The algorithm predicts po_6. This is a correct prediction.

We use the validation set for po_7 (*b_hvh.dat.invoer7.txt*).

```
Best match found for operator 7 at place HVH (match 100%)
```

The algorithm predicts po_7. This is a correct prediction.

We use the validation set for po_8 (*b_hvh.dat.invoer8.txt*).

```
Best match found for operator 8 at place HVH (match 99,9%)
```

All the predicted aswers were correct. Approach 2 seems to retrieve correct *po*'s.

6.1.2 Predicting *po* for *HVH* with Approach 1

Now we use the learning set for the place *HVH*. Thus, all the *po*'s in this range are po_2 till po_8 corresponding to the 7 operators at *HVH*. The testing will be done with Approach 1.

We start with the validation set for po_2 (*a_hvh.dat.invoer2.txt*).

```
GLOBAL po 2 ->123
GLOBAL po 3 ->93
GLOBAL po 4 ->67
GLOBAL po 5 ->63
GLOBAL po 6 ->82
GLOBAL po 7 ->75
GLOBAL po 8 ->91
```

The algorithm predicts po_2. Correct prediction.

We use the validation set for po_3 (*a_hvh.dat.invoer3.txt*).

```
GLOBAL po 2 ->112
GLOBAL po 3 ->90
GLOBAL po 4 ->76
GLOBAL po 5 ->64
GLOBAL po 6 ->108
GLOBAL po 7 ->67
GLOBAL po 8 ->100
```

The algorithm predicts po_6. This is an incorrect prediction.

We use the validation set for po_4 (*a_hvh.dat.invoer4.txt*).

```
GLOBAL po 2 ->110
GLOBAL po 3 ->94
GLOBAL po 4 ->119
GLOBAL po 5 ->45
GLOBAL po 6 ->69
GLOBAL po 7 ->42
GLOBAL po 8 ->103
```

The algorithm predicts po_4. This is a correct prediction.

We use the validation set for po_5 (*a_hvh.dat.invoer5.txt*).

```
GLOBAL po 2 ->59
GLOBAL po 3 ->59
GLOBAL po 4 ->20
GLOBAL po 5 ->142
GLOBAL po 6 ->49
GLOBAL po 7 ->124
GLOBAL po 8 ->29
```

The algorithm predicts po_5. This is a correct prediction.

We use the validation set for po_6 (*a_hvh.dat.invoer6.txt*).

```
GLOBAL po 2 ->91
GLOBAL po 3 ->42
GLOBAL po 4 ->72
GLOBAL po 5 ->49
GLOBAL po 6 ->135
GLOBAL po 7 ->66
GLOBAL po 8 ->107
```

The algorithm predicts po_6. This is a correct prediction.

We use the validation set for po_7 (*a_hvh.dat.invoer7.txt*).

```
GLOBAL po 2 ->47
GLOBAL po 3 ->69
GLOBAL po 4 ->45
GLOBAL po 5 ->123
GLOBAL po 6 ->67
GLOBAL po 7 ->137
GLOBAL po 8 ->62
```

The algorithm predicts po_7. Correct prediction.

We use the validation set for po_8 (*a_hvh.dat.invoer8.txt*).

```
GLOBAL po 2 ->98
GLOBAL po 3 ->123
GLOBAL po 4 ->101
GLOBAL po 5 ->42
GLOBAL po 6 ->59
GLOBAL po 7 ->71
GLOBAL po 8 ->98
```

The algorithm predicts po_3. Wrong prediction. The correct po_8 has a weight of 98 and is not close to the highest prediction weight.

The algorithm provided 5 times a correct prediction and 2 times an incorrect prediction.

6.1.3 Predicting *po* for *HVH* with Approach 2

We now will do the same test but we will use Approach 2 instead of Approach 1. To enable the usage of Approach 2 the Combine answer option must be disabled. The rest of the settings remain default.

We use the validation set for po_2 (*a_hvh.dat.invoer2.txt*).

```
Best match found for operator 5 at place HVH (match 99,88%)
```

We use the validation set for po_3 (*a_hvh.dat.invoer3.txt*).

```
Best match found for operator 8 at place HVH (match 98,1%)
```

We use the validation set for po_4 (*a_hvh.dat.invoer4.txt*).

```
Best match found for operator 8 at place HVH (match 94,72%)
```

We use the validation set for po_5 (*a_hvh.dat.invoer5.txt*).

```
Best match found for operator 6 at place HVH (match 99,95%)
```

We use thevalidation set for po_6 (*a_hvh.dat.invoer6.txt*).

```
Best match found for operator 7 at place HVH (match 99,67%)
```

We use the validation set for po_7 (*a_hvh.dat.invoer7.txt*).

```
Best match found for operator 5 at place HVH (match 99,3%)
```

We use the validation set for po_8 (*a_hvh.dat.invoer8.txt*).

```
Best match found for operator 8 at place HVH (match 93,73%)
```

Only one of the *po*'s was guessed correctly. But since po_8 was also predicted for two other tests it is possible that even this one correct prediction was just a coincidence.

6.1.4 Predicting *po* for ***EPL***

We now will do the same test but we use the learning set for the place *EPL*. Thus, all the *po*'s in this range are po_{12} till po_{18}, corresponding to the 7 operators at *HVH*. The testing will be done again with Approach 1.

We use the validation set for po_{12} (*a_epl.dat.invoer2.txt*).

```
GLOBAL po 12 ->146
GLOBAL po 13 ->144
GLOBAL po 14 ->175
GLOBAL po 15 ->205
GLOBAL po 16 ->192
GLOBAL po 17 ->82
GLOBAL po 18 ->119
```

The algorithm predicts po_{15}. This is an incorrect prediction.

We use the validation set for po_{13} (*a_epl.dat.invoer3.txt*).

```
GLOBAL po 12 ->172
GLOBAL po 13 ->158
GLOBAL po 14 ->105
GLOBAL po 15 ->225
GLOBAL po 16 ->183
GLOBAL po 17 ->141
GLOBAL po 18 ->142
```

The algorithm predicts po_{15}. Wrong prediction.

We use the validation set for po_{14} (*a_epl.dat.invoer4.txt*).

```
GLOBAL po 12 ->127
GLOBAL po 13 ->182
GLOBAL po 14 ->213
GLOBAL po 15 ->135
GLOBAL po 16 ->132
GLOBAL po 17 ->89
GLOBAL po 18 ->113
```

The algorithm predicts po_{14}. Correct prediction.

We use the validation set for po_{15} (*a_epl.dat.invoer5.txt*).

```
GLOBAL po 12 ->102
GLOBAL po 13 ->100
GLOBAL po 14 ->47
GLOBAL po 15 ->290
GLOBAL po 16 ->130
GLOBAL po 17 ->198
GLOBAL po 18 ->100
```

The algorithm predicts po_{15}. This is a correct prediction.

We use the validation set for po_{16} (*a_epl.dat.invoer6.txt*).

```
GLOBAL po 12 ->188
GLOBAL po 13 ->112
GLOBAL po 14 ->127
GLOBAL po 15 ->239
GLOBAL po 16 ->286
GLOBAL po 17 ->65
GLOBAL po 18 ->78
```

The algorithm predicts po_{16}. This is a correct prediction.

We use the validation set for po_{17} (*a_epl.dat.invoer7.txt*).

```
GLOBAL po 12 ->91
GLOBAL po 13 ->181
GLOBAL po 14 ->112
GLOBAL po 15 ->169
GLOBAL po 16 ->102
GLOBAL po 17 ->204
GLOBAL po 18 ->143
```

The algorithm predicts po_{17}. This is a correct prediction.

We use the validation set for po_{18} (*a_epl.dat.invoer8.txt*).

```
GLOBAL po 12 ->233
GLOBAL po 13 ->303
GLOBAL po 14 ->239
GLOBAL po 15 ->251
GLOBAL po 16 ->246
GLOBAL po 17 ->176
GLOBAL po 18 ->379
```

The algorithm predicts po_{18}. This is a correct prediction.

The results are the same as for *HVH*: the algorithm provided 5 times a correct prediction and 2 times an incorrect prediction.

6.1.5 Predicting *po* with minimal distance disabled

Let us now try to repeat the previous tests for *EPL*, but look at the influence of the change of the minimal distance. First we lower the minimal distance to 0.

We use the validation set for po_{12} (*a_epl.dat.invoer2.txt*).

```
GLOBAL po 12 ->64
GLOBAL po 13 ->39
GLOBAL po 14 ->97
GLOBAL po 15 ->90
GLOBAL po 16 ->47
GLOBAL po 17 ->20
GLOBAL po 18 ->37
```

The algorithm predicts po_{14}. This is an incorrect prediction

We use the validation set for po_{13} (*a_epl.dat.invoer3.txt*).

```
GLOBAL po 12 ->55
GLOBAL po 13 ->64
GLOBAL po 14 ->36
GLOBAL po 15 ->116
GLOBAL po 16 ->57
GLOBAL po 17 ->38
GLOBAL po 18 ->28
```

The algorithm predicts po_{15}. This is an incorrect prediction

We use the validation set for po_{14} (*a_epl.dat.invoer4.txt*).

```
GLOBAL po 12 ->45
GLOBAL po 13 ->73
GLOBAL po 14 ->136
GLOBAL po 15 ->57
GLOBAL po 16 ->43
GLOBAL po 17 ->19
GLOBAL po 18 ->21
```

The algorithm predicts po_{14}. This is a correct prediction.

We use the validation set for po_{15} (*a_epl.dat.invoer5.txt*).

```
GLOBAL po 12 ->36
GLOBAL po 13 ->9
GLOBAL po 14 ->11
GLOBAL po 15 ->139
GLOBAL po 16 ->41
GLOBAL po 17 ->145
GLOBAL po 18 ->13
```

The algorithm predicts po_{17}. This is an incorrect prediction

We use the validation set for po_{16} (*a_epl.dat.invoer6.txt*).

```
GLOBAL po 12 ->57
GLOBAL po 13 ->33
GLOBAL po 14 ->32
GLOBAL po 15 ->83
GLOBAL po 16 ->148
GLOBAL po 17 ->17
GLOBAL po 18 ->24
```

The algorithm predicts po_{16}. This is a correct prediction

We use the validation set for po_{17} (*a_epl.dat.invoer7.txt*).

```
GLOBAL po 12 ->34
GLOBAL po 13 ->79
GLOBAL po 14 ->30
GLOBAL po 15 ->49
GLOBAL po 16 ->41
GLOBAL po 17 ->140
GLOBAL po 18 ->21
```

The algorithm predicts po_{17}. This is a correct prediction

We use the validation set for po_{18} (*a_epl.dat.invoer8.txt*).

```
GLOBAL po 12 ->35
GLOBAL po 13 ->145
GLOBAL po 14 ->96
GLOBAL po 15 ->23
GLOBAL po 16 ->30
GLOBAL po 17 ->22
GLOBAL po 18 ->43
```

The algorithm predicts po_{13}. This is an incorrect prediction The difference with the correct po_8 is very large.

The algorithm provided 3 times a correct prediction and 4 times an incorrect prediction with sometimes a very large prediction error.

6.2 Predicting *po* with the value Highest Only enabled

In this subsection we will again try to retrieve the correct *po* of a validation set. We will now use the program with not all the default values, but with the option Highest Only enabled and see how well the prediction will now be. Enabling Highest Only option should lower the number of rules in the rule list. We will again look at the place *EPL*, so we can compare the results of this test with the test from the previous subsection.

We use the validation set for po_{12} (*a_epl.dat.invoer2.txt*).

```
GLOBAL po 12 ->95
GLOBAL po 13 ->98
GLOBAL po 14 ->123
GLOBAL po 15 ->121
GLOBAL po 16 ->110
GLOBAL po 17 ->43
GLOBAL po 18 ->86
```

The algorithm predicts po_{12}. This is an incorrect prediction

We use the validation set for po_{13} (*a_epl.dat.invoer3.txt*).

```
GLOBAL po 12 ->120
GLOBAL po 13 ->106
GLOBAL po 14 ->71
GLOBAL po 15 ->135
GLOBAL po 16 ->119
GLOBAL po 17 ->80
GLOBAL po 18 ->93
```

The algorithm predicts po_{15}. This is an incorrect prediction

We use the validation set for po_{14} (*a_epl.dat.invoer4.txt*).

```
GLOBAL po 12 ->82
GLOBAL po 13 ->118
GLOBAL po 14 ->152
GLOBAL po 15 ->72
GLOBAL po 16 ->71
GLOBAL po 17 ->46
GLOBAL po 18 ->78
```

The algorithm predicts po_{14}. This is a correct prediction

We use the validation set for po_{15} (*a_epl.dat.invoer5.txt*).

```
GLOBAL po 12 ->73
GLOBAL po 13 ->62
GLOBAL po 14 ->26
GLOBAL po 15 ->179
GLOBAL po 16 ->70
GLOBAL po 17 ->124
GLOBAL po 18 ->62
```

The algorithm predicts po_{15}. This is a correct prediction

We use the validation set for po_{16} (*a_epl.dat.invoer6.txt*).

```
GLOBAL po 12 ->124
GLOBAL po 13 ->80
GLOBAL po 14 ->86
GLOBAL po 15 ->132
GLOBAL po 16 ->176
GLOBAL po 17 ->35
GLOBAL po 18 ->62
```

The algorithm predicts po_{16}. This is a correct prediction

We use the validation set for po_{17} (*a_epl.dat.invoer7.txt*).

```
GLOBAL po 12 ->65
GLOBAL po 13 ->118
GLOBAL po 14 ->80
GLOBAL po 15 ->108
GLOBAL po 16 ->66
GLOBAL po 17 ->127
GLOBAL po 18 ->81
```

The algorithm predicts po_{17}. This is a correct prediction

We use the validation set for po_{18} (*a_epl.dat.invoer8.txt*).

```
GLOBAL po 12 ->71
GLOBAL po 13 ->151
GLOBAL po 14 ->146
GLOBAL po 15 ->50
GLOBAL po 16 ->61
GLOBAL po 17 ->62
GLOBAL po 18 ->130
```

The algorithm predicts po_{13}. This is an incorrect prediction

The algorithm provided 4 times a correct prediction and 3 times an incorrect prediction.

6.3 Different configurations put together

In this chapter we will consider a table filled with test results for different configurations. The tests that were described in detail in the previous sections are also included in the table. Of other tests only the results are shown. Only values other that default values are shown in the table for better readability. The places we have investigated are *HVH* and *EPL*. This was done, because these are the places which gave most correct results (this will be shown in Section 6.4).
The abbrevation $ap2$ stands for approach 2 (by default approach 1 is enabled). In the last column the percentage of correct predictions is given. The 1 in the binary string between brackets stands for the operators (from 2 till 8) which were correctly predicted (0 stands for incorrect prediction). For example string 0011111 means that operator 2 and 3 were not predicted correctly, but the rest of the operators was.

place	s	l	cc	da	ho	cd	md	end	bb	ohl	ap2	%
EPL												71%(0011111)
EPL									1			57%(0010111)
EPL						on						28%(0001010)
EPL							0					43%(0010110)
EPL							0.3					57%(0011101)
EPL					on							57%(0011110)
EPL											on	43%(1010100)
HVH												71%(1011110)
HVH									1			71%(1011110)
HVH						on						71%(1001111)
HVH					on							57%(1001110)
HVH											on	14%(0000001)
HVH							0					71%(1011110)
HVH							0.3					57%(1011100)
HVH	30											71%(1011110)
HVH	30								1			71%(1011110)
HVH	30				on					0.3		43%(1000110)
HVH	30						0.3					57%(1011100)
HVH	30				on							57%(1011100)
HVH	30		off	on							on	29%(0001001)
HVH	30										on	14%(0000001)

6.4 Different *po*'s behaviour

In this chapter we will consider a table filled with test results for different places in the default configuration. For each test we consider a certain place and all (7) operators that are working at this place. A 1 in the table denotes a correct prediction, while a 0 denotes an incorrect prediction. Approach 1 is used. The same validation sets (for each *po*) are used as in the previous tests.

Place	op_2	op_3	op_4	op_5	op_6	op_7	op_8	%
HVH	1	0	1	1	1	1	0	71%
EPL	0	0	1	1	1	1	1	71%
YMD	1	0	0	1	0	0	0	28%
HA	0	0	0	0	0	1	1	28%
DLFZ	0	0	1	1	1	1	0	57%
DENH	0	0	1	0	1	1	1	57%
HAR	0	1	0	1	1	1	0	57%

The *po* predictions for different operators give in places *HVH* and *EPL* high percentage of correct predictions for all operators (71%). The predictions for the places *YMD* and *HA* were mostly incorrect (28% of correct predictions).

6.5 One operator in different places

In the previous sections we considered a certain place and the operators (*po*'s) working at that place in which he works. In this section we will consider a fixed operator and different places.

For example, if we try to predict the place for operator 2, the *po*'s that are considered will be po_{02}, po_{12}, po_{22}, po_{32}, po_{42}, po_{52}, po_{62}, po_{72}.

Below is a table consisting of the predicted results for different operators.

Place	HVH	EPL	YMD	HA	DLFZ	DENH	HAR	%
op_2	1	0	0	0	1	1	1	57%
op_3	0	1	1	0	1	0	1	57%
op_4	0	1	1	1	0	1	0	57%
op_5	1	1	1	1	1	1	0	85%
op_6	0	1	0	1	1	1	1	71%
op_7	0	1	1	1	1	1	1	85%
op_8	0	1	1	1	1	1	1	85%

The *po* predictions for different places (with a fixed operator) give a high percentage of correct predictions for all operators (all 57%).

Chapter 7

Discussion and Conclusion

We have introduced two approaches for recognising the operators by means of their rules.
Approach 2 only gives good results when the validation set is a subset of the learningset. In the other cases most of its predictions were incorrect (only 14% correct).

Using the option *Check Direction* for Approach 1 for *HVH* and *EPL* can be useful, since it reduces the number of rules dramatically. For *HVH* the number of correctly predicted operators remains the same. For *EPL* the number of rules seems to be reduced so much, that there are not enough rules to give a correct prediction.

Making a higher support threshold value makes no difference for the place *HVH*. We only tested with a support threshold set to 30.

Setting the *minimal distance* to 0 for Approach 1 gave worse results for *EPL* (see the table in Section 6.3). Less predictions were correct. However, when the algorithm did guess correctly, the weight for the correct *po* stood out. Setting the *minimal distance* to a higher value (0.3) than the default value (0.15) also gave worse results. Apparently the *minimal distance* must have such a value, that lookalike *po*'s are included in the prediction, but not too many.

Enabling the option *best bonus* gave worse results for *EPL*. For the place *HVH* the results were the same as when default options were used. This means that in the weight calculation for *HVH* very often the highest weight (when comparing an event sequence with a rule) is actually the correct prediction. Considering lookalike *po*'s for this place is thus not important and so is the minimal distance. We can also see that if we look at the case where minimal distance was set to 0 for *HVH* — the results were the same as with the default values (the default minimal distance is set to 0.3).

Using the option *only high lift* for Approach 1 also gave worse results (see the table in Section 6.3). Apparently looking only at Rule Collections with an average high lift filters out too many other interesting Rule Collections.

Using Approach 1 with the option *highest only* gave 57% of correct predictions (for both tested places). The number of correct predictions was lower than with this option disabled, but the number of rules that needed to be considered was drastically lowered by the use of this option.

Approach 1 gives good results when used with the default settings. For the tested cases 75% of the predictions were correct (see the table in Section 6.3). The incorrect predictions often were close to the correct predictions (the correct operator had a high weight).

As we can see in the table in Section 6.4 the *po* prediction for different places has a very different error rate. For the places *HVH* and *EPL* 71% of the *po*'s were correctly predicted. For the places *DLFZ*, *DENH* and *HAR* the error rate was higher; 57% of the *po*'s was correctly retrieved. For the places *YMD* and *HA* only 28% of the *po*'s could be retrieved correctly. Still this percentage is higher than the percentage of a completely random prediction. There are 7 operators (*po*'s) considered per place. The chance that a correct *po* is predicted if the prediction is random is 1/7 per prediction. The average percentage of the randomly predicted *po*'s for all seven predictions would thus be $1/7 * 100 \approx 14\%$. Thus Approach 1 gives a better prediction and in some cases even a good prediction.

In Section 6.5 we considered a prediction table that was created for a certain fixed operator and different places. The error rate is low. This means operators actually follow some rules for prediction at different places, but these rules differ per place. This also means that different operators are using more often the same rules for the predictions at a certain place, while a certain operator uses various rules for different places.

Some operators follow rules more often than others. For example the behaviour of operator 2, operator 3 and operator 4 is more difficult to predict (high error rate for these operators) than that of other operators.

We conclude that it is in principle possible to predict the operators by their rules

7.1 Future Research

The research that was done in this thesis focused only on the database that was provided by Rijkswaterstaat. The application accepts only this database as an input. The algorithms proposed in this paper could be used for solving other similar problems. Therefore the application should be adapted for other databases as an input.

We tried to solve the problem by the use of different data mining approaches. In this paper two of these approaches are described. The main approach was based on the Bayes's theorem. In a previous document that was focusing on the same problem and involvong the same dataset [5] we tried to solve the problem by the use of the Decision Tree algorithm [6]. Other data mining approaches might provide other results.

The validation sets that were used for testing were made for a fixed place (and considering of 7 operators) or a fixed operator (and considering of 7 places). The testing should be done for more combinations. For example we should consider all places and all operators at once. Also the testing should be done more often for all the different options.

Bibliography

[1] R. Agrawal, T. Imielinski, A. Swami; *Mining Association Rules Between Sets of Items in Large Databases*; 1993; 207–216.

[2] R. Agrawal, R. Srikant; *Fast Algorithms for Mining Association Rules*; VLDB; 1994; 487-499.

[3] W.A. Kosters, W. Pijls; *Apriori: A Depth First Implementation*; FIMI'03, The First Workshop on Frequent Itemset Mining Implementations; 2003; Melbourne, Florida, USA (CEUR Workshop Proceedings, http://CEUR-WS.org/Vol-90/; Bart Goethals and Mohammed J. Zaki (eds.)).

[4] P.D. McNicholas, T.B. Murphy, M. O'Regan; *Standardising the Lift of an Association Rule*; Department of Statistics, Trinity College; Dublin, Ireland; 2007.

[5] A. Nezhinsky; *Decision Tree Algorithm for Rijkswaterstaat*; LIACS, Leiden University; 2006.

[6] S. Russell, P. Norvig; *Artificial Intelligence, A Modern Approach*; Second edition; Prentice Hall; 2003.

[7] P.N. Tan, M. Steinbach, V. Kumar; *Introduction to Data Mining*; Addison-Wesley; 2006.