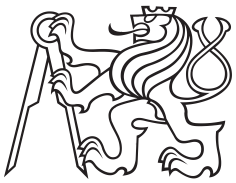
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of cybernetics

Analysis Annotations of Epileptic Seizures

Václav Příhoda

Supervisor: Ing. Lenka Vysloužilová, Ph.D.
Field of study: Biomedical Engineering and Informatics
Subfield: Biomedical Engineering
May 2016

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Bc. Václav P ř í h o d a

Study programme: Biomedical Engineering and Informatics

Specialisation: Biomedical Engineering

Title of Diploma Thesis: Analysis Annotations of Epileptic Seizures

Guidelines:

1. Study the structure of epileptic seizure annotation database, which will be continuely appended.
2. Study the classical and sequential association rules and their creation in language R.
3. Design and implement data transformation from database to proper format to create rules in R.
4. Find association rules between symptoms and diagnosis of epileptic seizures with respect to minimal senzitivity and minimal specificity.
5. Design sequence pattern search in annotation database.
6. Test the whole process of transformation and analysis.

Bibliography/Sources:

[1] AGRAWAL, Rakesh; IMIELIŃSKI, Tomasz; SWAMI, Arun. Mining association rules between sets of items in large databases. In: ACM SIGMOD Record. ACM, 1993. p. 207-216.
[2] MOONEY, Carl H.; RODDICK, John F. Sequential pattern mining--approaches and algorithms. ACM Computing Surveys (CSUR), 2013, 45.2: 19.
[3] BERKA, Petr. Dobývání znalostí z databází. Academia, 2003.

Diploma Thesis Supervisor: Ing. Lenka Vysloužilová, Ph.D.

Valid until: the end of the summer semester of academic year 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, December 17, 2015

Acknowledgements

I would like to express my sincere appreciation to Ing. Lenka Vysloužilová, Ph.D. for her patient support, guidance and advice. My thanks also belong to my family and everyone who have supported me.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date ....
Signature

Abstract

The aim of this thesis is to create the epileptic seizure analysis. This analysis might serve as a technique for finding the location of the epileptic zone. This add-on is part of already-made tool ASTEP, that was developed for purposes of Na Homolce hospital. Data analysis uses the technique called association rules mining.

This paper contains theoretical description of association rules mining and sequence pattern mining method. Theoretical description contains analysis that enables using values sensitivity and specificity during process of rules generating.

The implementation uses database containing annotation of epileptic seizures. This database is continuously growing. ASTEP is developed in *Java* programming language and so is the module enabling the analysis. Analysis is created in *R* programming language using *arules* and *arulesSequences* packages. Connection between Java is created through *JRI* library. Within the analysis, a proposition of sequence pattern mining was created in *R* programming language.

The last part of this paper documents the proof of implementation both module and data analysis. Results contain numbers of extracted rules with various input constraints and evaluate the possibility of sensitivity and specificity usage.

Keywords: annotated epileptic seizures, ictal signs, association rules, sequence patterns, SPADE, cSPADE

Supervisor: Ing. Lenka Vysloužilová, Ph.D.

Abstrakt

V rámci práce byl vytvořen modul, který umožňuje analýzu anotací epileptických záchvatů. Analýza by mohla sloužit jako technika na určení umístnění epileptogenní zóny. Modul analýzy je součástí již vyvíjeného nástroje ASTEP, který slouží pro účely Nemocnice Na Homolce. Pro analýzu dat se využívá technika dolování dat zvaná asociační pravidla.

Práce obsahuje teoretický popis způsobu dobývání asociačních pravidel a sekvenčních vzorů. Součástí teoretického popisu je i rozbor, který umožní pracovat při analýze s hodnotami senzitivita a specificita.

V rámci implementace se pracuje s databází, která obsahuje anotace epileptických záchvatů. Databáze se průběžně rozšiřuje. Program ASTEP je vytvořen v Javě a stejně tak i modul umožňující analýzu. Analýza je naprogramovaná v jazyce *R* s využitím balíčků *arules* a *arulesSequences*. Propojení jazyku Java a *R* je dosaženo prostřednictvím knihovny JRI (Java/R Interface). Součástí práce je návrh dobývání sekvenčních vzorů v jazyce *R*.

Poslední část práce obsahuje ověření implementace modulu i samotné analýzy dat. Výsledky obsahují počty získaných pravidel při různých vstupních podmínkách a hodnotí možnost využití senzitivity a specificity.

Klíčová slova: anotované epileptické záchvaty, iktální příznaky, asociační pravidla, sekvenční vzory, SPADE, cSPADE

Překlad názvu: Analýza anotací epileptických záchvatů

Contents

Figures

Tables

Chapter 1

Introduction

Development in the field of information technology allowed storing data from nearly any kind of human activity. As a result, large amount of data is stored in databases, data warehouses or other data storages. Using suitable data analysis techniques these data can reveal new information.

The aim of this thesis is to develop a tool for data analysis of annotations of Epileptic Seizures. This tool will become the part of the software named ASTEP (Annotation and Statistical Tool for EPilepsy) which has been designed and developed for the purposes of Na Homolce Hospital. This software tool which is developed in Java programming language provides support to a specialist who annotates the video-EEG recordings.

The integral part of ASTEP is its database. The database contains information about patients, their diagnosis and detailed description of their seizures. For the purpose of this thesis, the most important record are annotations of seizures. Annotations fully describe seizures with terms such as position of a lesion, duration time, sequence number or types and manners of seizures, which are called ictal signs.

Epilepsy is a common neurological disorder. Epilepsy is defined by following conditions: having at least two unprovoked (or reflex) seizures occurring greater than 24 hours apart or having one unprovoked (or reflex) seizure and probability of further seizures similar to the general recurrence risk (at least 60%) after two unprovoked seizures, occurring over the next 10 years or having diagnosis of an epilepsy syndrome.

Epilepsy is considered to be resolved for individuals who had an age-dependent epilepsy syndrome but are now past the applicable age or those who have remained seizure-free for the last 10 years, with no seizure medicines for the last 5 years [1].

Some percentage of patients can be treated by an antiepileptic drugs. Those patients who cannot be treated by antiepileptic drugs are potential candidates for an epilepsy surgery. It is crucial to define the location and size of the

epileptic zone. The epileptic zone is a cortical region where seizures arise. Removing this region is aim of the epileptic surgery.

The data analysis presented in this work should be able to find the links between ictal signs and location of the epileptic zone, thus expand existing diagnostic tests. Some relationships between the ictal signs and location were already described [2]. The goal of the project of hospital Na Homolce is to find and describe less conspicuous ictal signs.

So far the search for ictal signs has been related to the opinion of a specialist, solely based on experience. The data analysis tool allows to process not only some promising ictal sings but the entire database. A specialist will obtain the list of rules with its sensitivity and specificity and then he can focus on those which are considered to be significant. The plan of the project is to annotate at least 600 seizures. Association rules mining is considered to be the appropriate technique to analyse exactly these data primarily for expecting the relationship between the ictal sign and location.

Some ictal signs are generated like a consequence of seizure spread. There is an assumption that the fast spreading of a seizure can lead to coexisting several ictal signs during the same seizure. Hence, the aim of this thesis is also to design sequence pattern search in annotation database. Sequences in the seizures have been already observed [3] but until now there were no large amount of data describing seizures and no tool to process them. Therefore, the result of this analysis could bring a new perspective.

The thesis is divided into 6 chapters including introduction and conclusion. In chapter 2, the theory of the association rules containing basic algorithms, such as Apriori algorithm. Chapter 3 presents the theory of the sequential pattern mining and the algorithm cSPADE. Chapter 4 describes the implementation of the data extraction, data analysis in R and implementing tool into ASTEP. Chapter 5 presents experiments that should prove the correctness of the implementation. The last chapter summarizes results and suggests tasks for the future work.

Chapter 2

Association Rule Mining

This chapter presents the theory of association rules mining, basic metrics, e.g. support or confidence, and procedure of mining association rules.

Association rule mining is one of data mining technique. It is a crucial method for market basket analysis, which aims at finding behaviour of customers in the variety of domains. Besides of marketing, association rule mining has been applied in distinct domains such as medical diagnosis, census data or bioinformatics. The technique was introduced in 1993 in the paper by Rakesh Agrawal. The paper presented an algorithm that generated significant association rules between items in the database. It also presented the results of applying the algorithm to data from a large retailing company.

The aim of this technique is to search associations among items (i.e. associations among items of a market basket). Association rule uses an If-Then form to express the relation. More notations are used to express the relation, for example *antecedent* $\Rightarrow$ *consequent*. In some publications the word *consequent* is replaced by the word *succedent* [4]. Rule mining method is beneficial since the interpretation of a rule should be easily understandable, though association analysis results should be interpreted with caution.

2.1 Basic Concept

Association rule mining was described in the Agrawal's paper as follows: $I = I_1 , I_2 , \ldots , I_m$ is a set of items (binary attributes) and T is a database of transactions. Each transaction t is represented as a binary vector, then $t[k] = 1$ if transaction t contains item I_k or $t[k] = 0$ if transaction t does not contain item I_k. Let X be a set of items in I. It is agreed that a transaction t satisfies X if for all items I_k in X, $t[k] = 1$ [5].

Association rule is an implication of the form $X \Rightarrow Y$, where the itemsets X, Y are sets of some items in I, and X and Y do not intersect.

2.1.1 Metrics for Association Rules

Association rule mining uses several metrics to determine the significance of discovered rules. The most important are support and confidence, together with lift as a crucial metrics in this work. Support count and confidence count have an impact on the process of association rule mining.

Support of an itemset X is the proportion of transactions in the database T which contain the itemset. It can be defined as

$$supp(X) = \frac{\sigma(X)}{N}, \tag{2.1}$$

where N is number of all transactions, $\sigma(X) = |\{t_i | X \subseteq t_i, t_i \in T\}|$ and symbol $|\cdot|$ denotes the number of elements in a set.

Support of a rule $X \Rightarrow Y$ is the support of $X \cup Y$.

$$supp(X \Rightarrow Y) = supp(X \cup Y) = \frac{\sigma(X \cup Y)}{N} \tag{2.2}$$

Confidence of a rule $X \Rightarrow Y$ is the ratio that determines the strength of the implication. In other words it determines how frequently items in Y appear in transactions that contain X. This is taken to be conditional probability $P(X|Y)$ [6].

$$conf(X \Rightarrow Y) = P(X|Y) = \frac{supp(X \cup Y)}{supp(X)} \tag{2.3}$$

Last metric that will be mentioned is labelled lift. Lift can be computed by dividing the confidence of the rule by support of the consequent.

$$lift(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)supp(Y)} = \frac{conf(X \Rightarrow Y)}{supp(Y)} \tag{2.4}$$

Lift measures how many times more often X and Y occur together than expected if they where statistically independent. A lift value of 1 indicates independence between X and Y. Moreover, lift is susceptible to noise in small databases. Rare itemsets with low counts (low probability) which per chance occur a few times (or only once) together can produce enormous lift values [7]. Lift has range between 0 and infinity. According to the value of the lift, three results can occur.

- lift$(X \Rightarrow Y) > 1$, the X and the Y are positively correlated
- lift$(X \Rightarrow Y) < 1$, the X and the Y are negatively correlated
- lift$(X \Rightarrow Y) = 1$, the X and the Y are independent

2.2 Association Rule Discovery

Association rule discovery is a process of mining association rules. Let I be a set of items and X, Y be itemsets that are subset of I, denoted as $X, Y \subseteq I$. The implication $X \Rightarrow Y$ is a rule if $X \cap Y = \emptyset$.

Common strategy used in association rule mining algorithms splits the problem into two subtasks:

1. This subtask is called frequent itemset generation. The task is to generate all itemsets that satisfy the minimal support threshold $supp(X \cup Y) \geq minsupp$. Those itemsets that satisfy the minimal threshold are called frequent itemsets.

2. The second subtask is called rule generation. The goal is to extract all the high-confidence rules from the frequent itemsets found in the previous step. The rule is valid if if satisfies the minimal confidence threshold $conf(X \Rightarrow Y) \geq minconf$.

Minimal support and minimal confidence are given by a user.

2.3 Frequent Itemset Generation

Let I be a set of items, where $I = \{A, B, C, D\}$. Figure (2.1) shows an itemset lattice for I. An itemset which has k items can generate $2^k - 1$ frequent

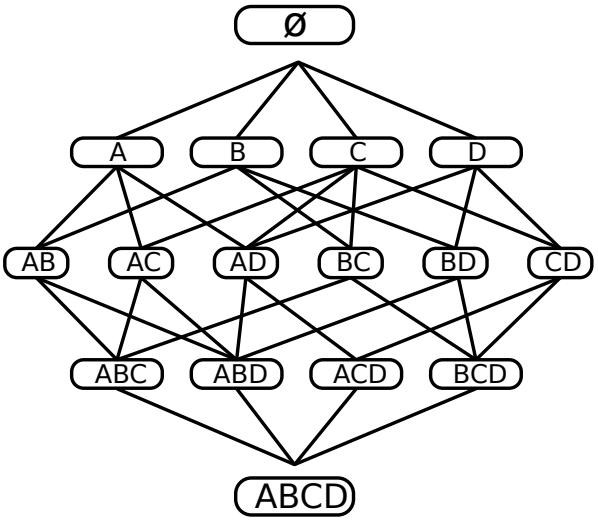


Figure 2.1: An itemset lattice shows all combinations of itemsets [8].

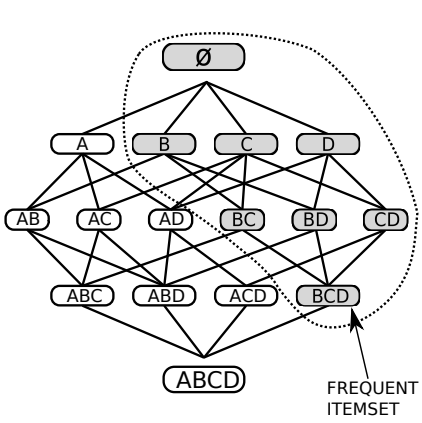
itemsets. The number of items can be large. That cause the search space of itemsets that need to be explored to grow exponentially. If the number of items becomes large, it will result in exponential growth of the search space. A brute-force algorithm for finding frequent itemsets determines the

support for each candidate itemset in the lattice structure. Each candidate is compared against each transaction. In case that the transaction contains the candidate, support count of the candidate is incremented. Complexity of a brute-force algorithm can be $\mathcal{O}(NMw)$, where N is the number of transactions, $M = 2^k - 1$ represents the number of candidate itemsets, and w is the maximal number of items present in a transaction (i.e. maximum transaction width).

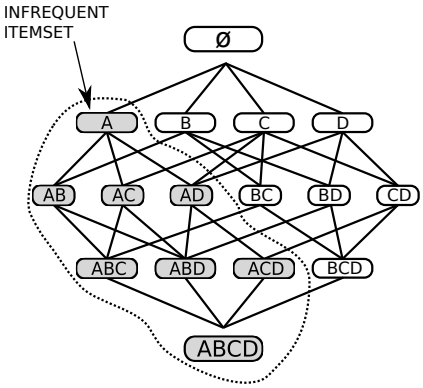
One way of how to reduce computational complexity is to reduce the number of candidate itemsets. An effective way to eliminate some of the candidate itemsets uses the Apriori principle.

2.3.1 Apriori principle

The Apriori principle states: If an itemset is frequent, then all of its subsets must also be frequent [8]. Conversely, if an itemset is infrequent then all its supersets must also be infrequent. Figure (2.2) illustrates the Apriori principle. Let the itemset {B,C,D} be a frequent itemset. This itemset contains its subsets, for example {B,C},{B,D},{B} etc. If the itemset {B,C,D} is frequent, then all its subsets must be also frequent. Conversely, in figure (2.2b) is shown that all supersets of infrequent itemset {A} are also infrequent.



(a) : Frequent itemset - If an itemset is frequent, then all its subsets must be frequent.



(b) : Infrequent itemset - If an itemset is infrequent then all its supersets must also be infrequent.

Figure 2.2: A graphic interpretation of Apriori principle [8].

Once the infrequent itemset is found, the following supersets of this itemset can be pruned. This strategy is also known as support-based pruning [8]. The strategy is made possible by a property of the support measure. This property states that the support count for an itemset never exceeds the support of its subsets. This property of support count is called the anti-monotone.

Definition of monotonicity property: Let I be a set of items, and $J = 2^I$ be

the power set of I. A measure f is monotone (or upward closed) if

$$\forall X, Y \in J : (X \subseteq Y) \rightarrow f(X) \leq f(Y), \tag{2.5}$$

that means if X is subset of Y, then *f(X)* must not exceed *f(Y)*. Conversely, f is anti-monotone (or downward closed) if

$$\forall X, Y \in J : (X \subseteq Y) \rightarrow f(Y) \leq f(X), \tag{2.6}$$

that means if X is subset of Y, then *f(Y)* must not exceed *f(X)* [8].

2.3.2 Apriori algorithm

The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemset properties [6]. The Apriori algorithm is a level-wise algorithm. It means that the algorithm completely searches current level for itemsets before moving to the next level. The pseudocode for the frequent itemset mining using Apriori algorithm is shown in algorithm (1). Set of candidate k-itemsets is C_k and set of frequent k-itemsets is F_k.

Algorithm 1 Apriori algorithm - frequent itemset mining [8]

1: k=1
2: $F_k = \{i | i \in I \wedge \sigma(\{i\}) \geq N \cdot minsup\}$ {Find all frequent 1-itemsets}
3: **repeat**
4: k=k+1.
5: C_k = apriori-gen(F_{k-1}) {Generate candidate itemsets}
6: **for** each transaction $t \in T$ **do**
7: $C_t = subset(C_k, t)$
8: **for** each candidate itemset $c \in C_t$ **do**
9: $\sigma(c) = \sigma(c) + 1$ {Increment support count}
10: **end for**
11: **end for**
12: $F_k = \{c | c \in C_k \wedge \sigma(c) \geq N \cdot minsup\}$. {Extract the frequent k-itemsets}
13: **until** $F_k = \emptyset$
14: Result=$\cup F_k$

- In step 1 and 2 the algorithm passes over the data and determines the support of each item. Subsequently, the set of all frequent 1-itemsets (F_1) is known.
- Then the algorithm repeatedly generates candidate k-itemsets C_k using the frequent itemsets F_{k-1} that were found in previous step (iteration). The function $apriori - gen$ generates candidate itemsets (step 5). The function performs two operations:
  - Candidate Generation - generates new candidate k-itemsets.

- Candidate Pruning - eliminates some of the candidate k-itemsets using the support-based pruning strategy.

There are more methods that can generate candidate itemsets (for example Brute-Force method, $F_{k-1} \times F_1$ method or $F_{k-1} \times F_{k-1}$ method [8]), yet all of them should satisfy following requirements. The method should not generate infrequent candidate itemsets and the candidate itemset should not be generated more than once. The method must ensure that all frequent itemsets are included and used in candidate generatation procedure.

- Next (steps 6 - 11) the algorithm counts the support of the candidates. It is needed to pass over the data once again. If the candidate itemset C_k is located in the subset C_t of transaction t, then the support count is updated.

- All candidate itemsets whose support is less than minimal support are eliminated (step 12).

- The algorithm terminates when there are no new frequent itemsets generated ($F_k = \emptyset$).

The algorithm needs k_{max+1} iterations, where k_{max} is the maximum size of the frequent itemsets.

2.3.3 Time Complexity

There are several factors that affect the time complexity of the Apriori algorithm.

- Support Threshold - There are two results of lowering support threshold. In the first case it increases the number of frequent itemsets. The second result of lowering threshold can cause the increase of the maximum size of frequent itemsets. Both cases have an adverse effect on the time complexity of the algorithm since more candidate itemsets must be generated and counted and bigger size causes more passes over the data set.

- Number of Items - Increasing number of items results in more space demand to store the support count of items.

- Number of Transactions - Time complexity increases with larger number of transactions.

- Average Transaction Width - The maximum size of frequent itemsets tends to increase as the average transaction width increases. As a result, more candidate itemsets must be examined during candidate generation and support counting.

2.4 Rule Generation

Each frequent k-itemset, X, can generate $2^k - 2$ association rules, excluding empty sets such as $X \rightarrow \emptyset$ or $\emptyset \rightarrow X$. If the itemset is partitioned into two non-empty subsets, then an association rule can be generated. Consider an itemset Y that can be extracted into subsets X and $Y - X$, such that $X \Rightarrow Y - X$ satisfies the confidence threshold.

For example consider $X = \{A, B, C\}$ to be a frequent itemset. There are 6 possible association rules that can be generated from X, for instance $\{A, B\} \Rightarrow \{C\}$, $\{A, C\} \Rightarrow \{B\}$ etc. Support count of such rules is same for the support of X. All rules satisfy the support threshold because they were generated from a frequent itemset which has to meet the support threshold.

The minimal confidence threshold is used to prune rules. Confidence, apart from support, does not have a monotone property like support. However, rules that are generated from same frequent itemset Y hold the following theorem for confidence measure. If a rule $X \Rightarrow Y - X$ does not satisfy the confidence threshold, then any rule $X' \Rightarrow Y - X'$, where X' is a subset of X, must not satisfy the confidence threshold as well [8]. Let $X = \{A, B, C\}$

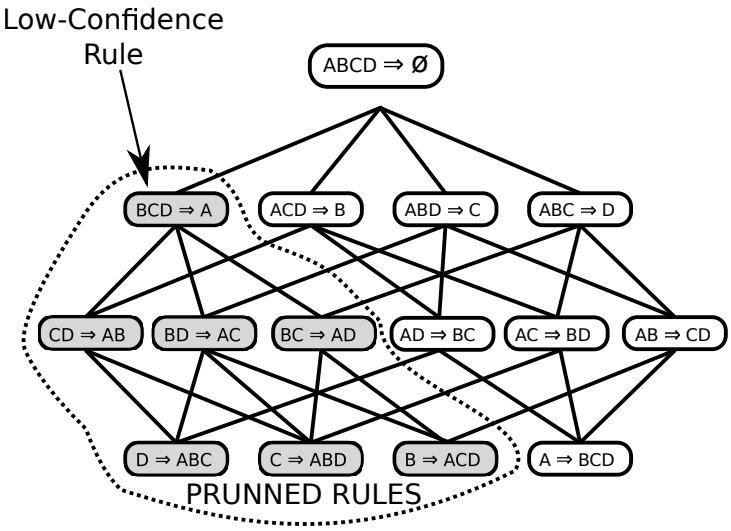


Figure 2.3: Rule generation using confidence-base pruning principle [8].

be a frequent itemset that generates two candidate rules. The first rule is $\{A, B\} \Rightarrow \{C\}$ with confidence

$$conf(A, B \Rightarrow C) = \frac{supp(A, B \cup C)}{supp(A, B)},$$

and the second rule is $\{A\} \Rightarrow \{B, C\}$ with confidence

$$conf(A \Rightarrow B, C) = \frac{supp(A \cup B, C)}{supp(A)},$$

where $A \subset \{A, B\}$. The support count $sup(A)$ must be equal or higher than $sup(A, B)$ due to the anti-monotone property of support count. Therefore, the confidence count of the rule $\{A\} \Rightarrow \{B, C\}$ is at least equal or smaller than confidence count of the rule $\{A, B\} \Rightarrow \{C\}$.

Figure (2.3) shows the confidence-based pruning principle. The itemset {BCD} → {A} has low confidence. Then all the rules that contain item A in its consequent e.g. {BC} → {AD} or {B} → {ACD} can be pruned.

2.4.1 Apriori Algorithm

The pseudocode of the apriori algorithm for the rule generation is shown in (2) and (3). The procedure $ap - genrules$ is similar to the frequent itemset mining algorithm mentioned in the section (2.3.2). The difference is that, the procedure does not need to pass over the data again to compute the confidence of the rule. The confidence is computed using the support counts.

Algorithm 2 Apriori algorithm - rule generation [8]

```
1: for each frequent k-itemset f_k, k ≥ 2 do
2:     H_1 = {i|i ∈ f_k}    {1-item consequents of the rule.}
3:     call ap-genrules(f_k, H_1)
4: end for
```

Algorithm 3 Procedure ap-genrules(f_k, H_m) [8]

```
1: k = |f_k|    {size of frequent itemset.}
2: m = |H_m|    {size of rule consequent}
3: if k > m + 1 then
4:     H_{m+1} = apriori − gen(H_m)
5:     for each h_{m+1} ∈ H_{m+1} do
6:         conf = supp(f_k)/supp(f_k − h_{m+1})
7:         if conf ≥ minconf then
8:             output the rule (f_k − h_{m+1}) → h_{m+1}
9:         else
10:            delete h_{m+1} from H_{m+1}
11:        end if
12:    end for
13:    call ap-genrules(f_k, H_{m+1})
14: end if
```

2.4.2 Item Constraints

The item constraints are used when a user is interested only in a subset of associations. These constraints involve restrictions on items that can appear in a rule. For example, it allows to search such rules that contain a specific

item I_x from an itemset I appearing in the antecedent or that have a specific item I_y appearing in the consequent. Combinations of the above constraints are also possible. For example, to request all rules that have items from some predefined itemset X appearing in the antecedent, and items from some other itemset Y appearing in the consequent [5]. While such constraints can be applied as a post processing step, integrating them into the mining algorithm can dramatically reduce the execution time [9].

2.5 Transformation of Basic Metrics

As shown in section (2.1), the support and confidence count are basic metrics that describes association rules. However, support and confidence are not common terms among medical specialists. They are rather associated with terms such as sensitivity and specificity. These terms indicate quality of a statistical hypothesis or a statistical test.

The basic metrics of association rules can be illustrated using the contingency table, see the table (2.1), where a represents the number of transactions that satisfies the antecedence (X) and the consequence (Y), b represents the number of transactions that satisfies antecedence and does not satisfy the consequence etc. Total number of all transactions indicates n. The table is

	Y	¬ Y	∑
X	a	b	r=a+b
¬ X	c	d	s=c+d
∑	k=a+c	l=b+d	n = a+b+c+d

Table 2.1: Contingency table for a rule [10].

useful in defining support, confidence and lift, as shown in following equations.

$$support = \frac{a}{n} \tag{2.7}$$

$$confidence = \frac{a}{r} \tag{2.8}$$

$$lift = \frac{a \cdot n}{r \cdot k} \tag{2.9}$$

Sensitivity and specificity can be also derived from the table, see equation (2.10) and (2.11).

$$sensitivity = \frac{a}{a + c} \tag{2.10}$$

$$specificity = \frac{d}{d + b} \tag{2.11}$$

Given task is to enable specialists to work with association rules while using sensitivity and specificity instead of support and confidence. This task leads to two subtasks. The first subtask is to determine sensitivity and specificity of a rule based on its support, confidence and lift count. The second subtask

is to estimate support and confidence count that is used as the threshold in the algorithm.

Determining Sensitivity and Specificity of a Rule

In the first case we are given a rule with its support, confidence and lift count and the task is to compute sensitivity and specificity. Based on the previous equations (2.7), (2.8) and (2.9) it is now possible determine all values in the table (2.1).

$$\begin{aligned} a &= n \cdot support & \qquad l &= n - k \\ r &= \frac{a}{confidence} & \qquad b &= r - a \\ k &= \frac{a \cdot n}{r \cdot lift} & \qquad d &= l - b \end{aligned}$$

Then it is possible to count sensitivity and specificity using support, confidence and lift.

$$sensitivity = \frac{a}{k} = \frac{a \cdot support}{k} = \frac{n \cdot support}{\frac{confidence \cdot n}{lift}} = \frac{support \cdot lift}{confidence} \tag{2.12}$$

$$\begin{aligned} specificity &= \frac{d}{d+b} = \frac{l-b}{l} = 1 - \left(\frac{r-a}{n-k}\right) \\ &= 1 - \left(\frac{\frac{n \cdot support}{confidence} - n \cdot support}{n - \frac{confidence \cdot n}{lift}}\right) \\ &= 1 - \left(\frac{\frac{support}{confidence} - support}{1 - \frac{confidence}{lift}}\right) \end{aligned} \tag{2.13}$$

Estimating Support and Confidence Threshold

Second case is the inverse of the previous. In this case, the known variables are sensitivity, specificity and number of all transactions n and it is to determine the estimation of support and confidence value with respect to them. This task represents setting of minimal support and confidence in the association rules mining. The equation (2.14) expresses the sum of all transactions.

$$n = a + b + c + d \tag{2.14}$$

In the next step, the variables b a c are substituted. This substitution is derived from equations number (2.10) and (2.11). Subsequently, the equation

is modified into its final form.

$$\begin{aligned} n &= a + \frac{d(1 - specificity)}{specificity} + \frac{a(1 - sensitivity)}{sensitivity} + d \\ &= a + \frac{d}{specificity} - d + \frac{a}{sensitivity} - a + d \\ &= \frac{a}{sensitivity} + \frac{d}{specificity} \end{aligned} \tag{2.15}$$

In this form the equation have 2 unknown variables a and d. Assuming that $d >> a$, since there is more record in the database that do not satisfy the rule, and thus it allows us to substitute $k \cdot a$ for variable d.

$$n = \frac{a}{sensitivity} + \frac{k \cdot a}{specificity} \tag{2.16}$$

One more substitution is needed to derive equation for estimation of support and confidence. Variable n in the support equation (2.7) is replaced with equation (2.16). Similarly, the variable b in the confidence equation (2.8) is replaced. Equations (2.17) and (2.18) represents the final form for the estimation of support and confidence count.

$$\begin{aligned} support = \frac{a}{n} &= \frac{1}{\frac{1}{sensitivity} + \frac{k}{specificity}} \\ &= \frac{sensitivity \cdot specificity}{specificity + k \cdot sensitivity} \end{aligned} \tag{2.17}$$

$$\begin{aligned} confidence = \frac{a}{a+b} &= \frac{a}{a + \frac{k \cdot a(1 - specificity)}{specificity}} \\ &= \frac{specificity}{specificity + k(1 - specificity)} \end{aligned} \tag{2.18}$$

Task is to find an estimation of the support and confidence count. The estimation should be the minimum, therefore the parameter k has to be maximal. The estimation of the parameter k is derived from equation (2.16), where sensitivity and specificity are equal to 1.

$$\begin{aligned} a + k \cdot a &\geq n \\ k &\geq \frac{n}{a} \end{aligned} \tag{2.19}$$

The task is considered to be an optimization problem. The straightforward solution is to set support and confidence threshold to a very small value. With this setting, there is an assurance that all rules will be found, yet this setting can cause long duration time of the algorithm especially if a big dataset is used. Therefore, the estimation of the support and confidence threshold could be an alternative solution.

Chapter 3

Sequential Pattern Mining

Sequential pattern mining is another approach in mining association rules. It uses sequences in a transactions to discover subsequences. This approach was firstly mentioned by Rakesh Agrawal in 1995 [11].

This thesis introduces the apriori-based algorithm called SPADE. However, there are more apriori-based algorithms, such as Apriori-All, GSP, PSP, SPAM.

3.1 Basic Concept

Let $I = i_1, i_2, ..., i_m$ be a set of m distinct items comprising the alphabet. An event is a non-empty unordered collection of items. A sequence is an ordered list of events. An event is denoted as $(i_1, i_2, ..., i_k)$ where i_j is an item. A sequence α is denoted as $(\alpha_1 \rightarrow \alpha_2 \rightarrow ... \rightarrow \alpha_q)$, where α_i is an event. The length of the sequence α is q and width of the sequence is the maximum size of any α_i for $1 \leq i \leq q$ [12]. A sequence with k items $k = \sum_j |\alpha_j|$ is called a $k-sequence$. If an event α_i is before another event α_j it is denoted as $\alpha_i < \alpha_j$.

Consider a sequence α and β. The sequence α is subsequence of β if there exists a function f that maps events in α to events in β and satisfies following conditions.

1. $\alpha_i \subseteq f(\alpha_i)$.
2. if $\alpha_i < \alpha_j$ then $f(\alpha_i) < f(\alpha_j)$. In the other words the order of events must be preserved [13].

The Database D contains a collection of input-sequences. Each input-sequence has the following fields: *sequence_id* (sid), *event_id (eid)* and *items*, see table (3.1). Sequence_id is a unique identifier for each input sequence. The assumption is that there is no sequence having more than one event with the same time-stamp. Therefore, this time-stamp can be used as a unique event identifier.

Support of a sequence, denoted $\sigma(\alpha, D)$, is defined as the number input-sequences in the database D that contain α. A sequence is called frequent

if it satisfies minimal support threshold that is specified by a user. The task is to find all frequent sequences from database D. The set of frequent $k - sequences$ is denoted as $\mathcal{F}_k$ [14].

Rules are generated using frequent sequences and minimum confidence threshold which is defined by a user.

Sequence ID	Event ID	Items
1	1	C D
1	2	A B C
1	3	A B F
1	4	A C D F
2	1	A B F
2	2	E
3	1	A B F
4	1	D G H
4	2	B F
4	3	A G H

Table 3.1: An input-sequence database [13].

3.2 SPADE Algorithm

The SPADE (Sequential PAttern Discovery using Equivalence classes) algorithm is used to discover frequent sequences. The algorithm uses lattice theory [13], especially the observation that the subsequence relation $\preceq$ defines a partial order on the set of sequences. In other words, if β is a frequent sequence, then all subsequences $\alpha < \beta$ are also frequent [12]. The search space is very large, considering that the total number of all sequences n of length at most k is $\mathcal{O}(n^k)$.

A		B		D		F	
sid	eid	sid	eid	sid	eid	sid	eid
1	2	1	2	1	1	1	3
1	3	1	3	1	4	1	4
1	4	2	2	4	1	2	2
2	2	3	1			3	1
3	1	4	3			4	3
4	4						

Table 3.2: The vertical database format for items A, B and C of input-sequence format of table (3.1).

This algorithm uses vertical database layout that is shown in table (3.2). It is an alternate manner compared with horizontal layout. In the vertical layout each item X in the sequence lattice is associated with its list called $id - list$

and denoted $\mathcal{L}(X)$. Each record in the list contains a pair of input-sequence identifier (sid) and event identifier (eid). The number of rows in the list corresponds with the number of occurrence of the item in the database. The list contains pairs of input-sequence identifier and event identifier that contain the item.

Pseudo code of the SPADE algorithm is shown in algorithm (4). The algorithm computes the frequent 1-sequences and 2-sequences. Then it decomposes the frequent 2-sequences into sub-lattices and then enumerates all other frequent sequences using Breadth-First Search (BFS) or Depth-First Search (DFS). All

Algorithm 4 SPADE (min_sup, D): [13]

1: $F_1 = \{\text{frequent items or 1-sequences}\}$;
2: $F_2 = \{\text{frequent 2-sequences}\}$;
3: $\varepsilon = \{\text{equivalence classes } [X]_{\theta_1}\ \}$;
4: **for** all [X] $\in \varepsilon$ **do** Enumerate-Frequent-Seq([X]);

frequent 1-sequences can be discovered by one scan of the vertical database. There are two ways how to discover 2-sequences.

1. Preprocessing and collecting all 2-sequences above a user specified lower bound.

2. Performing vertical-to-horizontal transformation. The transformation creates a new list. A pair that consist of item and its event identifier in the sequence is inserted into each input-sequence (sid), see table (3.3). Afterwards, it is needed to create a list of all 2-sequences for each input-sequence, and update counts in a 2-dimensional array indexed by the frequent items [13].

SID	(Item, EID) pair
1	(A,2) (A,3) (A,4) (B,2) (B,3) (C,1) (C,2) (C,4) (D,1) (D,4) (F,3) (F,4)
2	(A,2) (B,2) (E,3) (F,2)
3	(A,1) (B,1) (F,1)
4	(A,4) (B,3) (D,1) (F,3) (G,1) (G,4) (H,1) (H,4)

Table 3.3: The vertical-to-horizontal database format of input-sequence format of table (3.1).

3.2.1 Equivalence classes

The process continues by decomposing the 2-sequences into prefix-based parent equivalence classes. The algorithm recursively decomposes the sequences at each new level into even smaller independent classes due to limited amount of memory. At level one the suffix classes have length equal to one and they are called parent classes. To decompose the 2-sequences it is possible to

use Breadth-First Search or Depth-First Search. In Breadth-First Search, the space is explored in a bottom-up manner. All child classes at each level are processed before moving to next level. In Depth-first search all child equivalence classes for a path are processed before moving to the next path [14].

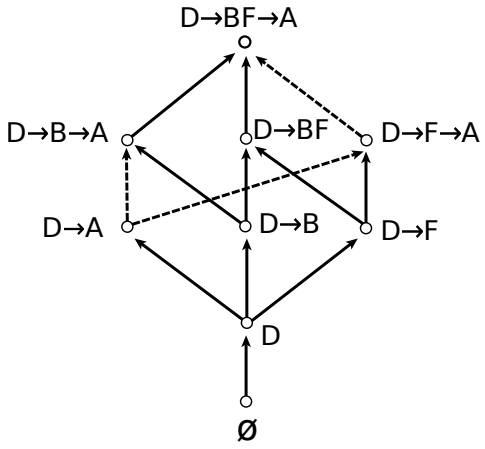


Figure 3.1: Recursive decomposition of class D into smaller sub-classes results in lattice of equivalence classes [13]

3.2.2 Enumerate Frequent Sequences

Algorithm (5) shows the pseudo-code for breadth-fist and depth-first search. Input of the procedure *Enumerate-Frequent-Seq* is suffix class with the id-list

Algorithm 5 Enumerate-Frequent-Seq(S): [13]

```
1:  for all sequences A_i ∈ S do
2:      T_i = ∅;
3:      for all sequences A_j ∈ S, with j ≥ i do
4:          R = A_i ∨ A_j;
5:          if (Prune(R) == FALSE) then
6:              L(R) = Temporal − Join(L(A_i), L(A_j));
7:              if (σ(R) ≥ min_sup) then
8:                  T_i = T_i ∪ R; F_|R| = F_|R| ∪ {R};
9:      end
10:     if (Depth-First-Search) then Enumerate-Frequent-Seq(T_i);
11: end
12: if (Breadth-First-Search) then
13:     for all T_i ≠ ∅ do Enumerate-Frequent-Seq(T_i);
14: delete(S);
```

for each of its elements. Frequent sequences are generated using joins of id-lists and verifying the support of a new id-list against the minimal support

threshold.

The algorithm can contain a pruning step. This step tests whether all subsequences of the discovered sequence are frequent. Discovered frequent sequences at the current level are used to discover frequent sequences at the next level. This process repeats recursively until all frequent sequences are discovered [12]. The last step of the algorithm is to delete the current level sequences S if the next level sequences T_i are found.

3.2.3 Temporal id-list Join

Let's have a sequence S and items A and B. The enumeration of the frequent sequences is performed by joining the id-lists in one of three ways.

1. Itemset and Itemset: joining AS and BS results in a new itemset ABS.
2. Itemset and Sequence: joining AS with $B \rightarrow S$ results in a new sequence $B \rightarrow AS$.
3. Sequence and Sequence: joining $A \rightarrow S$ with $B \rightarrow S$ gives three possible results: a new itemset $AB \rightarrow S$, and two new sequences $A \rightarrow B \rightarrow S$ and $B \rightarrow A \rightarrow S$. One special case occurs when $A \rightarrow S$ is joined with itself resulting in $A \rightarrow A \rightarrow S$

The enumeration process is the union or join of a set of sequences or items whose counts are then calculated by performing an intersection of the id-lists of the elements that comprise the newly formed sequence [14].

3.2.4 Pruning Sequences

Let α_1 denote the first item of sequence α. Before generating the id-list for a new k-sequence β, we check whether all its k subsequences of length k-1 are frequent. If they all are frequent then we perform the id-list join. Otherwise, β is dropped from consideration [13].

3.2.5 Rule Generating

The pseudo code in algorithm (6) presents rule generating procedure. Input of this procedure are frequent sequences and the minimal confidence. The Function fr(α) determine frequency in which the subsequence α occurs in the input-sequences. The confidence of a rule is computed as a ratio of frequency β and frequency of α. If the rule satisfies the minimal confidence threshold, then the procedure returns the rule and its confidence.

3.3 cSPADE Algorithm

The algorithm cSPADE extends the algorithm SPADE. In the algorithm (7) and (8) the pseudo code is presented. The constraints restriction is implemented in the Enumerate-Frequent procedure.

Algorithm 6 RuleGen(F, min_conf): [13]

1: **for** all frequent sequences $\beta \in F$ **do**
2: **for** all subsequences $\alpha \prec \beta$ **do**
3: conf = fr(β)/ fr(α);
4: **if** $conf \geq minconf$ **then**
5: **output** the rule $\alpha \rightarrow \beta$, and conf

Algorithm 7 cSPADE (min_sup): [12]

1: $P = \{$parent classes $P_i\}$;
2: **for** each parent class $P_i \in \mathcal{P}$ **do** Enumerate-Frequent(P_i);

A definition states that a constraint can be class-preserving or not class-preserving. A constraint is class-preserving if in the presence of the constraint a suffix-class retains it's self-containment property, i.e., support of any k-sequence can be found by joining the $id-lists$ of its two generating sub-sequences of length (k - 1) within the same class. [12].

The algorithm considers several constraints such as:

- Length and width limitations constraints are useful in tasks which has to restrict maximum length or width of a pattern to avoid exponential growth in the number of discovered frequent sequences. The constraint is implemented using IF clauses in the step 8 in the algorithm (8).
- Minimum gap and maximum gap between sequence elements. This constraint works with the time stamp of the input-sequences. The case of minimum gap allows to define a gap (a certain amount of time) that at least has to pass between two sequences. The case of the maximum gap is analogical, yet the maximum gap constraint is not class-preserving, therefore it requires a different approach of enumerating the frequent sequences. If the maximum gap constraint is used, then must be performed a join with set of frequent 2-sequences, instead of self-join [12]. See the step 2 - 6.
- The window constraint specifies the area of the interest. Using the minimum and maximum gap constraint allows only to define a length between two sequence elements. A window defines a time length of a whole sequence. This constraint is class-preserving. If a sequence α is within the time-window, then any subsequence β must also be within the same window. Implementation of this constraint is throw adding an extra column to the $id-list$ called *diff*. The column *diff* is computed as a differential between the first and last *eid* (time stamp) of a sequence.
- Item constraint allows either excluding items, including items or creating a super-item. A user defines which items will be excluded, included or considered as a super-item. It is possible due to using the vertical

format database and equivalence classes. In case of excluding constraint the specified item is removed from the parent class, so that the item will never appear in any discovered sequences. In case of inclusion, a new class is generated if a sequence contains the item. The super-item constraint allows considering a group of items or sequences to be a single item.

- Class constraint is able to use in case of classificated database, in which each input-sequence has a class label.

Algorithm 8 Enumerate-Frequent (S): [12]

```
1: for all sequences A_i ∈ S do
2:     if (maxgap) then // join with F_2
3:         p= Prefix-Item(A_i)
4:         N= { all 2-sequences A_j in class [p]}
5:     else// self-join
6:         N = { all sequences A_j in S, with j ≥ i }
7:     for all sequences α ∈ N do
8:         if (length(R) <= max_l and width(R) <= max_w and
               accuracy(R) ≠ 100%) then
9:             L(R)=Constrained-Temporal-Join(L(A_i),L(α),min_gap,
               max_gap,window);
10:            if (σ(R,c_i) ≥ min_sup(c_i)) then
11:                T = T∪ R; print R;
12:    Enumerate-Frequent(T)
13: delete(S);
```

Chapter 4

Design and Implementation of Data Analysis

This chapter describes developing the data analysing tool and implementing it into ASTEP. Following text that describes data analysis is split into four parts:

- Data extraction and data format,
- Data analysis in *R*,
- Implementation of previous into ASTEP and
- Design of Sequence Pattern Search.

The first part of the text describes the database structure, extracting data from the database and creating the dataset which is used to data analysis. The second part is introduction into data analysis using association rules and its implementation into *R*. The third part of this chapter shows ASTEP interface for data analysis and implementation of both previous parts into ASTEP. The last part presents the design of sequence pattern search in *R*.

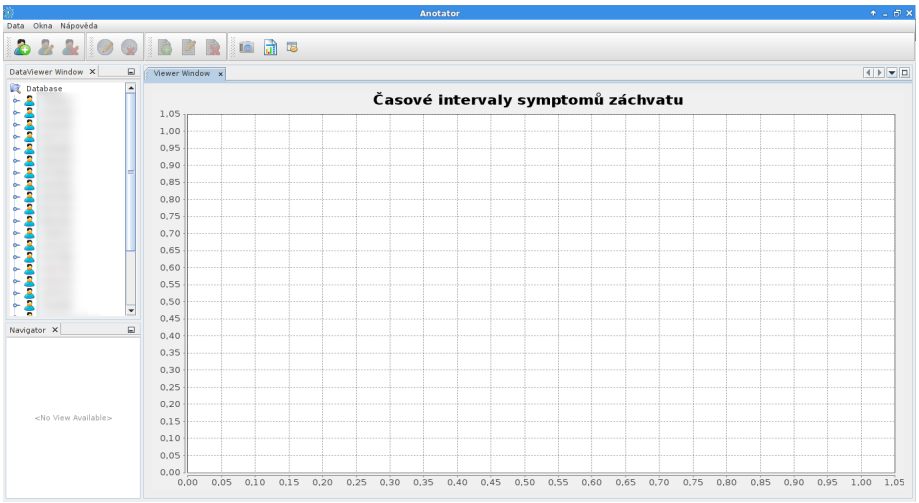


Figure 4.1: The interface of Annotation and Statistical Tool for EPilepsy.

4.1 Database and Data Extraction

The software tool ASTEP contains database of the seizures. The database has 6 tables:

- GENERATOR,
- PATIENT,
- TERMSPATIENT,
- MEASURE,
- SYMPTOM and
- TERMSEPILEPTICFIT.

For the structure of the database, see figure (4.2). The description of all

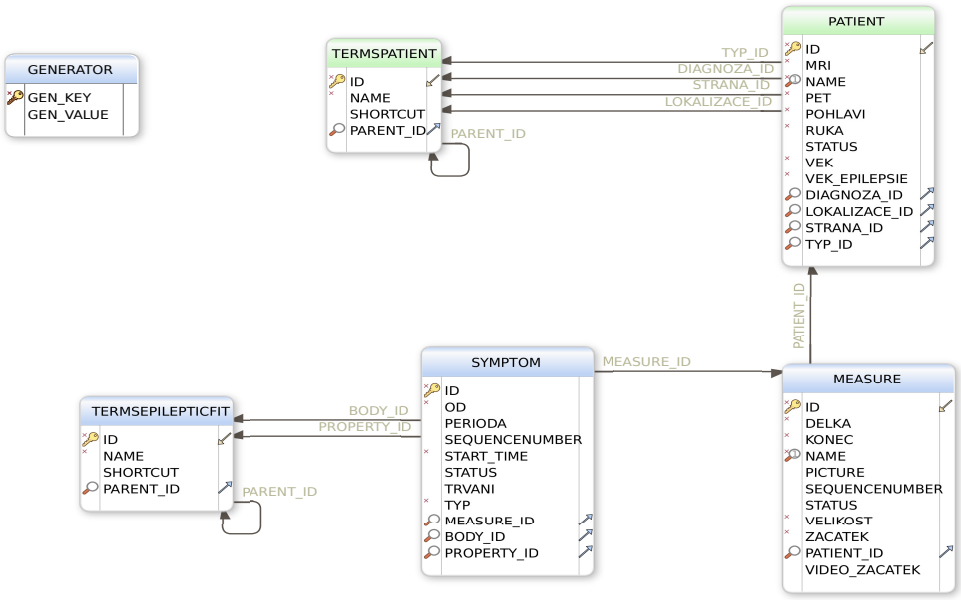


Figure 4.2: The Scheme of the database.

tables is in the appendix (A). Worthy of noticing are tables TERMSPATIENT and TERMSEPILEPTICFIT which use recursive association. The recursive association allows using a hierarchical structure. Types and manners of the seizures, also called ictal signs, are stored in the table TERMSEPILEPTICFIT. For example, hierarchy of body ictal sign *right eye* or *right hand* shows the figure (4.3). The maximal level (i.e. depth) of this hierarchy is 4.

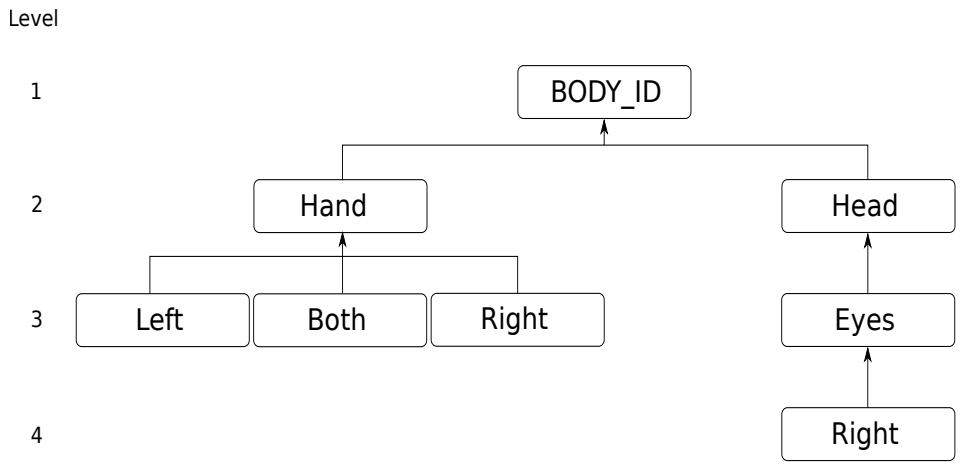


Figure 4.3: An example of the hierarchy in the table TERMSEPILEPTICFIT

4.1.1 Data Extraction

In order to extract data from the database into a dataset, all tables except GENERATOR are needed. The dataset has 15 variables (features). The main variables that are used for the association rules analysis are *localisation*, *side*, *body_i* and *property_i*, where *i* refers to hierarchical structure with numbers 1, 2, 3, 4. For the full list of variables see appendix (B).

The program ASTEP has implemented API to interact with the database and objects equivalent to the tables in the database. This API is used to extract data from the database. For the purpose of creating the dataset, two methods are used.

Method getDBData

The first method is called *getDBData*. This method returns type *StringBuilder* which contains extracted data.

For each single feature, there is created an *ArrayList*. The first value in the list is always the name of the feature. The reason for this is to create the header of the dataset making the dataset understandable.

The API that allows interacting with the database provides class *IDatabaseManager*. Its instance is called *manager* providing the list of the patients. The for loop iterates over the patients list to get the record of the *measures* and in like manner the record of each *symptom* in the table *measure* is obtained.

```
public static StringBuilder getDBData() {
 patientsList = manager.getPatients();
 for (int i = 0; i < patientsList.size(); i++) {
            for (int j = 0; j < patientsList.get(i).getMeasures()
                .size(); j++) {
                IMeasure measure = patientsList.get(i).
                    getMeasures().get(j);
```

```
                for (int k = 0; k < measure.getSymptoms().size();
                     k++) {
                    ISymptom symptom = measure.getSymptoms().get(
                        k);
                    symptomsList.add(symptom);

                    ...
                }
            }
    }
}
```

It is important to note that the object *symptom* contains variables of the ictal signs *Body* and *Property*.

As previously mentioned the ictal signs use the hierarchical structure in the database hence it is necessary to reflect it and split the ictal sign into single levels. The maximal level of the hierarchy is currently considered to be 4. If an ictal sign doesn't utilize all levels of the hierarchy, the empty levels are filled with the tabulate space (horizontal tab). In the code below is shown how to obtain the hierarchy of the ictal sign for the variable *body*. The procedure is same in the case of variable *property*.

```
        StringBuilder body = new StringBuilder();
                for (int l = 0; l < 4; l++) {
                    if (l < symptom.getBody().getHierarchy().
                        size()){
                        body.append(symptom.getBody().
                            getHierarchy().get(l).getTermName
                            ());
                    } else{
                          body.append("");
                    }
                    if (l<3){
                        body.append("\t");
                    }
                }
    ...
```

In each iteration over the list of the symptoms all extracted values are appended into corresponding *ArrayLists* and as result the dataset has the same number of records (rows) as the table *symptom*. In the end when all for loops terminate, the values from *Arraylists* are appended into the final *StringBuilder* called dataExport.

Method writeDataFile

The Second method is called *writeDataFile* and it writes data into a text file. The method has a parameter *StringBuilder* and its data are written into the file. The Java class called *FileWriter* (java.io.FileWriter) provides writing characters to a file. The parameter of the *FileWriter* sets the path and name of the file.

```
public static void writeDataFile(StringBuilder data) throws
    IOException {
```

```
        String filePath = System.getProperty("user.dir") + File.
            separator + "Analyzator" + File.separator + "RFiles"
            + File.separator + "sequencesData";

        FileWriter writer = new FileWriter(filePath);
        writer.write(data.toString());
        writer.close();
    }
```

4.2 Associative Rules in R

Package *arules* provides infrastructure for association rules learning. Function *apriori* is part of this package and it implements the Apriori algorithm [15].

4.2.1 Function Apriori

The *apriori* function is necessary for this work especially in the R script which is used to rules mining. The arguments of the function are data, parameter, appearance and control. The output of the function is an object of class rules or item sets.

The data given to the function must be *transactions*. The transaction is a *data.frame* with vectors of the same length as the number of transactions. Transactions can be created by coercion from lists containing transactions, but also from matrix and *data.frames*. However, it is necessary to have the data prepared. Association rule mining can only use items and does not work with continuous variables [16].

The argument parameter holds the information about the behaviour of the mining algorithm. Using this argument allows setting the value for the minimal support and the minimal confidence of an item set using the parameters *support* and *confidence*. There are more parameters to use, e.g. *minlen*, *maxlen* which define the minimal or the maximal number of items per item set or the parameter *target* that defines the type of association mined (e.g. frequent item sets, maximally frequent item sets, rules etc.).

The argument appearance specifies the restrictions for the associations mined. The list that specifies the restriction can contain following elements *lhs*, *rhs*, *both* or *none*. These elements contain character vectors that give the labels of the items which can appear in the specified place. The element *lhs* stands for left-hand-side and *rhs* stands for right-hand-side. The place *none* is special and it specifies items that cannot appear in the rule. The Parameter *default* defines the default appearance for all items which are not mentioned in the other elements of the list.

The argument control holds the parameters for the used algorithms and controls the algorithmic performance. This argument enables specifying the

type of sorting with parameter *sort* and it allows reporting progress using parameter *verbose*. There are more parameters in this argument to use, for example *filter*, *tree*, *heap*, *memopt*, *load* and *sparse*.

4.2.2 Description of the R Script

The main R script is called *epilepticRules*. In brief, this script loads data in R (function *readData*), then creates vectors which are used for appearance restriction (function *getARvector*). At last, the script uses data and these list to search association rules (function *findRules*).

```
data <- readData("sequencesData")

rhsFeatures <- c("localisation","side")
rhsList <- getList(data,rhsFeatures)

if (switch_val==1){
lhsFeatures <- c("body_1","body_2","body_3","body_4","property_1"
    ,"property_2","property_3","property_4")
lhsList <- getList(data,lhsFeatures)
}

export <- findRules(data,lhsList,rhsList,sens_value,spec_value)
```

Function readData

Purpose of the function *readData* is to load and store data in the *data frame* form. The function also verifies if the data file exists based on the name of the file. If the file exists, then data are loaded. The function returns the *data frame* called *data*.

```
readData <- function(file_name){
  fileBool <- file.exists(file_name)

  if(fileBool==TRUE){

    data <- read.csv(file_name,sep="\t", header= TRUE);
    data[names(data)] <- lapply(data[names(data)] , factor)
  }

  return(data)
  }
```

Function getARvector

The function returns vector of labels that is necessary in case of using appearance restriction (i.e. item constraints) in the Apriori function. Function *getList* can be used to create vectors for both sides right and left. The input of the function is the dataset and vector containing names of the features.

Features in the right-hand-side are *localisation* and *side*. Features in left-hand-side are *body_1* - *body_4* and *property_1* - *property_4* or the vector of

labels can be defined directly by a user from the program interface.

Based on the name of the features (columns), corresponding columns are extracted from the dataset. If there is no record in a column, then such column is not included. Remaining columns are saved in the variable *newData*.

```
columnNames <- features
  columnNumbers <- which(names(data)%in%columnNames)
  newData <- list()
  vect <- c()
  k <- 1;
  for (i in columnNumbers){
    if (length(levels(factor(data[[i]])))!=0){
      vect <-c(vect,i);
      k <- k+1
    }
  }
 newData <- data[vect]
```

Following step after extracting features is to obtain the unique values of the variables which is attained by using functions *levels* and *factor*.

```
nameList <- list()
  for (i in 1:length(newData)) {
    nameList[i]  <- list(levels(factor(newData[[i]])))
  }
```

Due to the fact that the dataset has empty fields in the columns, one of the unique value can be an empty character. Therefore, this value has to be removed.

```
  for (i in 1:length(nameList)) {
    vect <- do.call("cbind",nameList[i])
    for (j in 1:length(vect)) {
      if (("" %in% vect)) {
        vect <- vect[-j]
      }
    }
    nameList[i] <- list(vect);
  }
```

Each entry of the final vector consists of the column name, the unique value of the column and the symbol of equality "=".

```
 for(i in 1:length(nameList)){
    ARvector <- c(lhsList,paste(colnames(newData[i]),nameList[[i
        ]],sep = "="))
    }
 return(ARvector)
```

Function findRules

Function *findRules* processes given data and searches for the association rules. Input of this function is dataset, both appearance restriction vectors, sensitivity and specificity value. Initially, it is crucial to compute support and confidence count stated by equations (2.17) and (2.18).

Then the parameters are matched with the arguments of function Apriori. Parameters such as the minimal length of the rule, the minimal value of support and confidence, the type of the association mining and appearance vectors.

```
rules <- apriori(mydata, parameter = list(minlen=minlen_value,
    supp = supp_value, conf = conf_value, target = "rules"),
    appearance = list(lhs=lhsVect, rhs=rhsVect,default="none"))
```

Besides the association rules the output of the function *apriori* contains also value of the support, confidence and lift for each rule. Based on these three values it is possible to determine the value of sensitivity and specificity using equations (2.12 and (2.13).

```
rules <- as(rules, "data.frame");
rules[5] <- list(sensitivity = sensitivity)
rules[6] <- list(specificity = specificity)
```

Function Apriori searches rules using values support and confidence threshold. Therefore, detected rules can have lower sensitivity and specificity than it is stated by a user. It is necessary to select rules that specifically satisfy minimal sensitivity and specificity value. All the values, including the rules, are stored in the variable called *export*.

```
export <- rules[which(rules[6] >= spec & rules[5] >= sens),]
```

4.3 Implementation into ASTEP

Data extraction and the R script, that were described in (4.1.1) and (4.2.2), are both implemented in ASTEP. The module for data analysis has its own window. The figure (4.4) shows all graphical objects that are used. There are two combo boxes called *Symptom* and *Místo* (place on the body) to define ictal signs and two text fields to assign the value for the minimal sensitivity, the minimal specificity. These values are used to estimate the minimal value for support and the minimal value for confidence. Results are printed in the text area in the bottom part of the window. The button *hledej* starts the analysis.

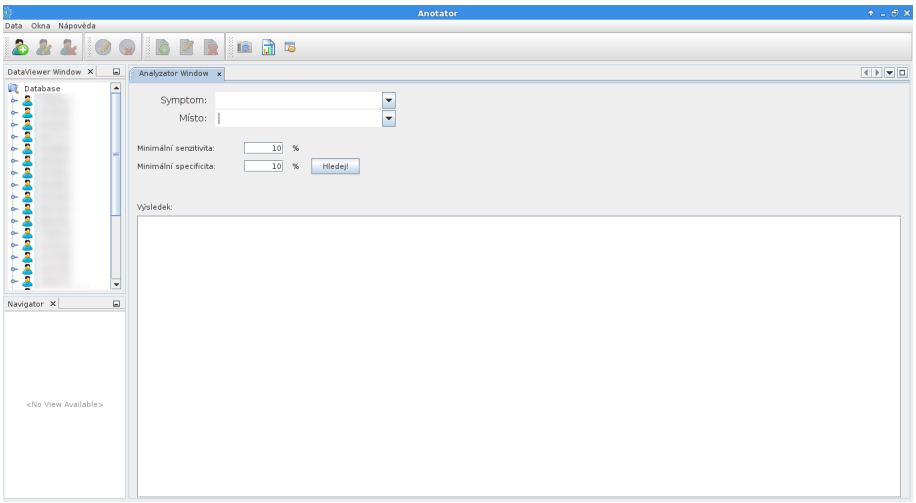


Figure 4.4: ASTEP - the module of data analysis.

4.3.1 The Dataset

The text file which contains data is created every time during the launch of the analysis module. It is provided by method *componentOpened*. The method ensures that the dataset is up to date. Methods *writeDataFile* and *getDBData* are described in the section (4.1.1).

```
public void componentOpened() {
 try {
     writeDataFile(getDBData());
 } catch (IOException ex) {
     Exceptions.printStackTrace(ex);
 }
}
```

4.3.2 Analysis in Java with the use of R

The button *hledej* has an *ActionListener* which executes the code inside the method when the button is pressed. The action that is performed clears the

text area for the results, starts the analysis and prints the results in the text area.

```
private void processBTActionPerformed(java.awt.event.ActionEvent
    evt) {
        outputTA.setText("");
        RList results = runRSript();
        outputTA.append(printResults(results).toString());
    }
```

Method runRScript

The method *runRScript* executes the R code as well as the analysis. Java class called Java/R Interface (org.rosuda.JRI) allows running R code in Java. At first, it is needed to initialize the R engine.

```
Rengine engine = Rengine.getMainEngine();
```

If a user decides to choose an ictal sign, then an instance of the object *StringBuilder* is created. This instance contains single levels of the selected ictal signs.

The function *eval* evaluates an expression. Therefore, it can be used to transfer the values into the R environment.

```
engine.eval("lhsList<-" + signs);
engine.eval("sens_value<-" + sens_value);
engine.eval("spec_value<-" + spec_value);
```

The function *eval* launches the whole R script which is described in (4.2.2).

```
engine.eval("source('epilepticRules.R')");
```

After the evaluation of the code, the results are transferred from the R environment to Java.

```
results = engine.eval("export").asList();
```

The method *printResults* prints the rules with sensitivity and specificity in the text area and describes the style and the format of the printed string.

There is one more method to notice and it is called *getRulesArray*. The string with the rules contains some symbols that can make reading the rules unpleasant. This method is used to remove unwanted characters from the string. As the result, the rules are more clear.

4.4 The Design of Sequence Pattern Search

One of the tasks is to design sequence pattern search. The implementation described in this section uses *R* package *arulesSequences* [17] and its functions.

4.4.1 Function cSpade

Function *cSpade* generates frequent sequential patterns. The function has four arguments *data*, *parameter*, *control* and *tmpdir*.

The input of argumet *data* must be an object of class *transactions* with temporal information, such as event identifier.

The argument parameter provides the constraint parameters for the algorithm. The argument allows adjusting the function behaviour. Available options are *support*, *maxsize*, *maxlen*, *mingap*, *maxgap* or *maxwin*. More information about constraints in cSPADE, see section (3.3).

The argument control holds controlling parameters for the cSPADE algorithm. For example, parameter *memsize* defines the maximum amount of memory to use. The other options are *numpart*, bfstype, *verbose*, *summary* and *tidLists*.

Arguments *tmpdir* stands for temporal directory and defines a name of the directory where temporary files are written.

4.4.2 R Implementation

Following text explains the implementation of the sequence pattern mining in *R*.

Firstly, it is necessary to prepare the data for function *read-baskets*. This function requires the data in a proper form. The columns that obtain sequence or event identifier must occur before columns that contain information about items. Therefore, several steps, such as replacing empty columns with a string *NA* or feature selection, must be fulfilled.

```
data <- read.csv("sequencesData",sep="\t",na.strings=c("","NA"),
    header= TRUE);
features <- c("measure_id","sequence","body_1","body_2","body_3",
    "body_4","property_1","property_2","property_3","property_4",
    "localisation","side")

columnNumbers <- which(names(data)%in%features)
data <- data[,columnNumbers]
```

Since the implementation of cSPADE algorithm does not support direct item constraints, values of location and location side has to be renamed. It will be useful in defining the *lhs* and *rhs* vectors of items.

```
colNumLocalisation <- which(names(data)%in%"localisation")
colNumSide <- which(names(data)%in% "side")

levels(data[,colNumLocalisation ]) <- c(paste(colnames(data[
    colNumLocalisation ]),levels(data[,colNumLocalisation ]),sep=
    "="))
levels(data[,colNumSide ]) <- c(paste(colnames(data[colNumSide ])
    ,levels(data[,colNumSide ]),sep="="))
```

```
LevLocalisation <-  levels((data[,colNumLocalisation ]))
LevSide <- levels(factor(data[,colNumSide]))
```

modified data are written in to a file then the function *read-baskets* can read the modified dataset. The function reads the data and creates a new object of *transactions* class. The dataset contains values *NA* and the function *cSPADE* considers these values to be an item. Removing *NA* values from *transactions* class is a straightforward process.

The object of class *transactions* is an extension of class *itemMatrix* that allows to store a binary incidence matrix [16]. The procedure is to extract this matrix from the *transactions* class, then to find the row that corresponds to the value *NA* and to set all values of this row to *FALSE* and to return this modified matrix to the original object of *transactions* class.

```
sequencesData <- read_baskets("basketData",sep="\t",info=c("
    sequenceID","eventID"))
BImatrix <- sequencesData@data
BImatrix[match('NA',sequencesData@itemInfo$labels),] <- FALSE
sequencesData@data <-BImatrix
```

Parameters of function *cSPADE* were mentioned in section 4.4.1. Once the function returns frequent sequences it is possible to extract a subset. In this case, the extracted subset must have the size of the sequence equal or higher than 2 and sequences must contain an item from right hand side vector (*rhsVect*). It decreases frequent sequences, so it will eventually decrease the number of rules.

```
seqs <- cspade(sequencesData, parameter = list(support=supp)

lhsVect <- subset(seqs@elements@items@itemInfo$labels, !(
    seqs@elements@items@itemInfo$labels %in%  c(LevSide,
    LevLocalisation) ))
rhsVect <- subset(seqs@elements@items@itemInfo$labels, (
    seqs@elements@items@itemInfo$labels %in% c(LevSide,
    LevLocalisation) ))

subSeqs <-subset(seqs, size(seqs) >= 2 & seqs %in% rhsVect)
```

The function *ruleInduction* is used to generate sequences rules. A sequence rule has to satisfy the minimum confidence threshold and contain the last element of the sequence as the consequent of a rule [17]. The last modification is to select subset of rules again. The condition is that the items from vector *rhsVect* cannot occur in left hand side (antecedent) of a rule and must occur in right hand side (consequent). The condition is similar for the vector *lhsVect*.

```
rules <- ruleInduction(subSeqs,sequencesData, confidence = 0.1,
    control = list(verbose = TRUE))

rules <- (subset(rules, (rhs(rules) %in% rhsVect) & !(rhs(rules)
    %in% lhsVect ) & !(lhs(rules) %in% rhsVect) & (lhs(rules) %in
    % lhsVect)))
```

Chapter 5

Experiments

This chapter presents results of some experiments that were made to prove the correctness of the implementation. Main experiments that will be demonstrated are:

- verification of the function *apriori* and association rule mining,
- visualization of the association rules,
- verification of the basic metrics transform,
- proof of functionality of the analysis module in ASTEP and
- analysis of the sequence pattern mining.

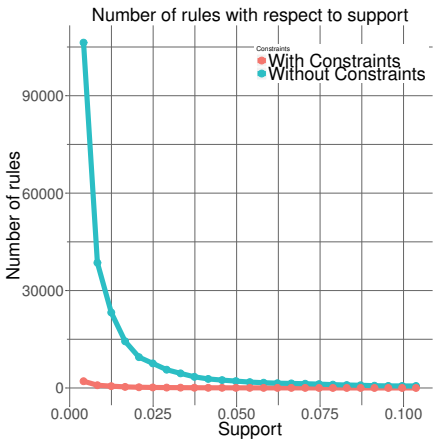
The database contains record of 25 patients, 57 annotated seizures and 241 annotated symptoms. The size of the dataset is 241 record, denoted as n.

5.1 Verification of the Association Rule Mining

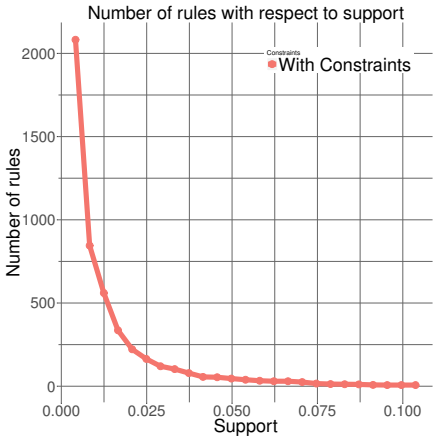
The aim of this section is to demonstrate results of association rules mining and to determine the number of discovered rules.

Two approaches have been used to show influence of the support count on the number of discovered rules, see Figure (5.1). The first approach is to generate all available association rules without any constraints. The second approach is to generate association rules with item constraints usage. The implementation of item constraints was mentioned in section (4.2.1). In both cases the value of confidence was set to $1/n$.

Figure (5.1a) and (5.1b) shows the manner of decreasing number of discovered rules with respect to increasing support count. The maximal count of discovered rules is 106 343 in case of no constraints and 2 082 rules in the case of constraints. The value of support is equal to $1/n = 1/241$ which corresponds to the occurrence of a rule at least in one record of the dataset. Applying constraints in *aprirori* function leads to dramatic decrease of discovered rules, especially in cases with very low support count.



(a) : The number of rules with respect to support.



(b) : A detailed view of the number of rules using item constraints.

Figure 5.1: Experiment of the dependency between support count and number of rules.

In Figure (5.2) the execution time is presented. Shown values are the average of 1000 iterations. The shape of execution time curves approximately correlates with the shape of curves in Figure (5.1a).

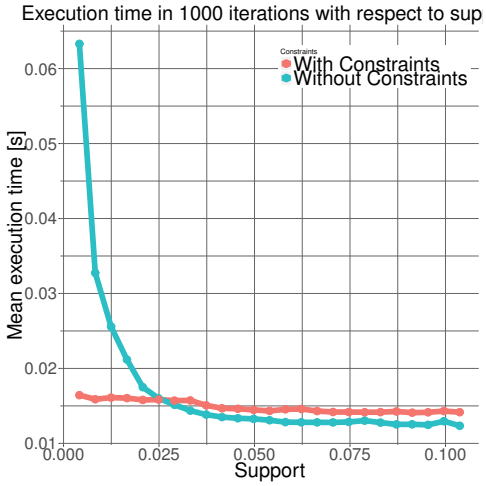


Figure 5.2: The execution time of the function *apriori* with respect to support in 1000 iterations.

It is necessary to be aware of the fact that support count is used in the first phase of association rule mining. The first phase is searching for frequent item sets. An adjustment of the support threshold affects the number of frequent itemsets which affects number of rules in the end. Therefore, support count has more influence on time consumption of *apriori* algorithm than

confidence count. Figure (5.3) shows how the number of rules is related with confidence count. The value of support count in this test was set to $1/n$.

(a) : The number of rules with respect to confidence.

(b) : A detailed view of the number of rules with constraints.

Figure 5.3: En experiment of the dependency between confidence count and number of rules.

As mentioned above, the dataset consists of 47 seizures (measures). The estimation is that 600 seizures will be annotated. It is about 12.5 times more then current dataset has. We can assume that the number of symptoms will also increase by 12.5 times, thus the number of record in dataset could be 3 076.

Nevertheless, it is difficult to estimate the number of frequent itemsets that could be generated. As it was mentioned in section (2.3), itemset with k items can generate $2^k - 1$ frequent itemsets. There are two extreme situations that can occur.

1. The itemset will grow same range. It means there will be new items appended into the itemset. It will result in many new frequent itemset with very low support, because their appearance will be rare. If we simply multiply current 2082 rules by 12.5 than we get estimation of 26 000 rules.

2. In the second situation the number of items in the itemset will stabilize. Instead of growing the number of items, the support of existing rules will increase, because new record will contain same symptoms.

5.1.1 Visualisation of the Association rules

To check all 2 082 rules one by one can be difficult. Therefore, it can be useful to display rules in graphs. *R* package *arulesViz* is used to visualize association rules.
Figure (5.4) shows all discovered rules with respect to support count, confidence count and lift. The most of the rules have support count lower than

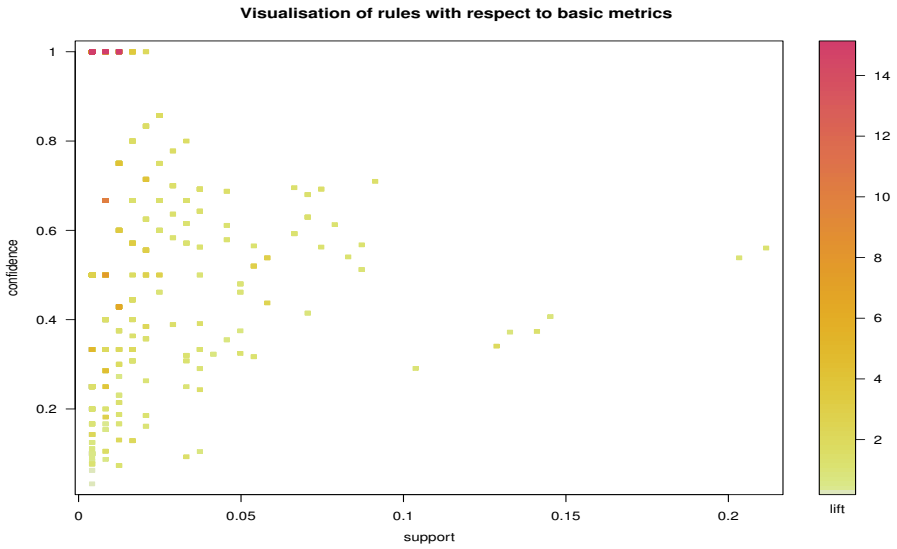


Figure 5.4: An illustration of all rules using scatter plot and basic metrics.

0.1. It indicates that some ictal signs occur rarely in the dataset. Rules with low support count and confidence count near or equal to 1 are probably discovered based on one record in the dataset. Similarly, Figure (5.5) shows the length of rules instead of lift. Discovered rules have length up to 6.

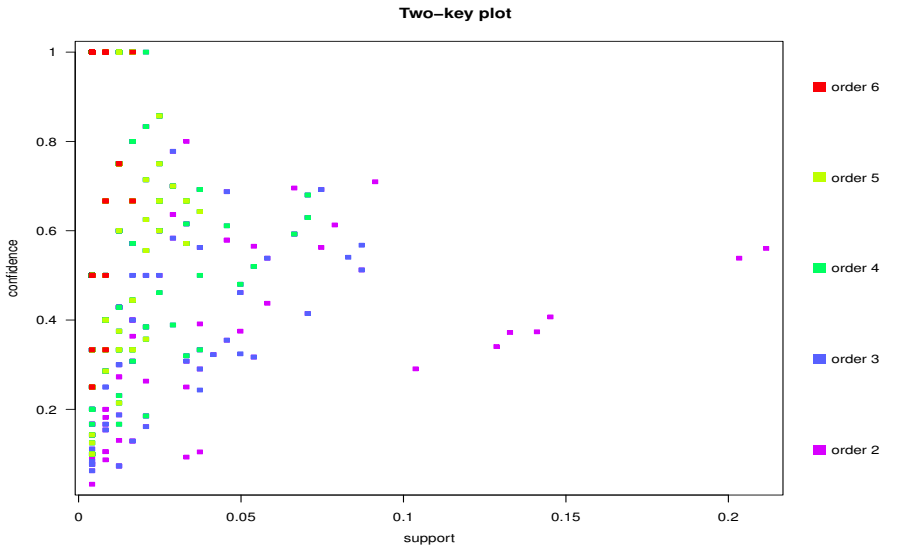


Figure 5.5: An illustration of the length of rules using scatter plot.

Group-based visualisation uses k-means clustering to handle a larger number of rules. Rules are presented as a group (cluster). The colour in Figure (5.6) represents the aggregate lift count of a group with a certain consequent. The size of a circle depends on the aggregate support count. The aggregation

function is the median value of the group. Columns and rows in the graph are reordered. The groups with the highest lift are located in the top left corner. Conversely, the groups with the least lift are located in bottom right corner.

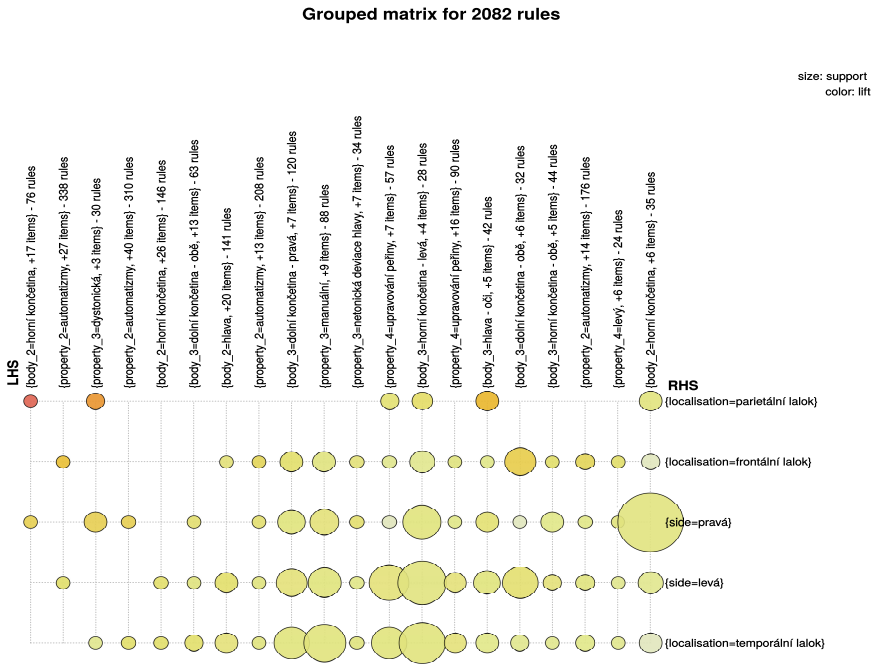


Figure 5.6: Grouped matrix for 2082 rules.

5.2 Estimation of Support and Confidence

Chapter (2.5) introduced transformation of basic metrics. Presented task was to estimate support count and confidence count. Following experiments will test the correctness of equations (2.17) and (2.18). Both equations contain parameter k and the value of this parameter affects the final estimation. There are several options of setting the value. Extreme values for the parameter k is 0 and n.

Firstly the demonstration of support count estimation will be presented. For purpose of this demonstration the confidence is $1/n$ and the value of specificity is 0.01. Results are depicted in Figure (5.7). Constant value of support $1/n$ (red line), estimated value, where $k = 0$ (green line) or $k = n$ (blue line) and maximal allowed support count (ideal, purple line) are shown in the figure. The ideal line is the upper bound of support. Values of this line are determined as the minimal support from all rules which satisfy given sensitivity.

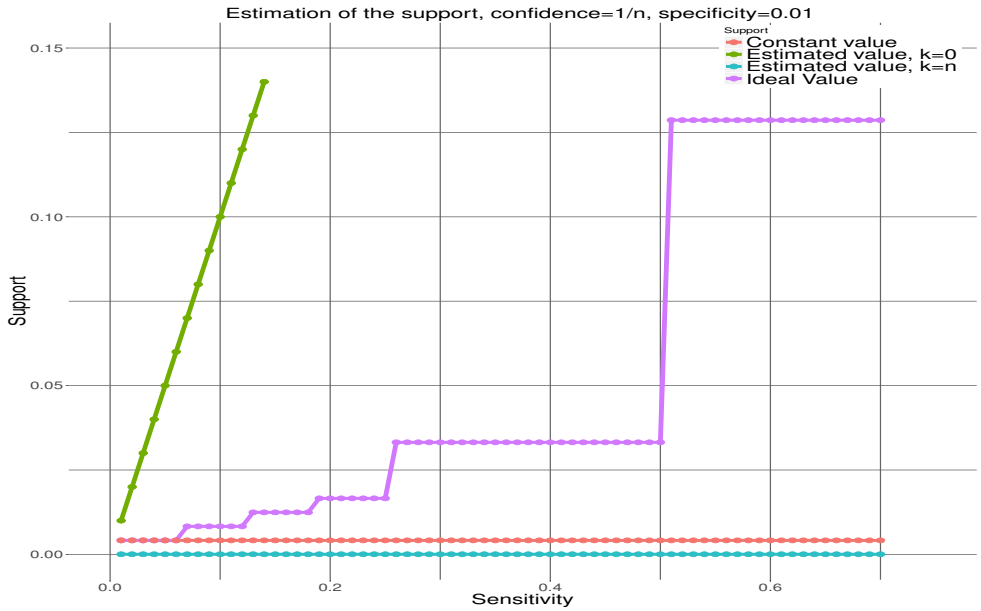


Figure 5.7: The estimation of support count: Influence of sensitivity on support count.

If the task is to find all rules with given sensitivity and specificity, then the estimated value must not cross the ideal line. Either the constant or the estimated value, where $k = n$ does not cross the maximal value of support count. The estimated value, where $k = 0$, crosses the limit line and thus it is not appropriate estimation for the support count.

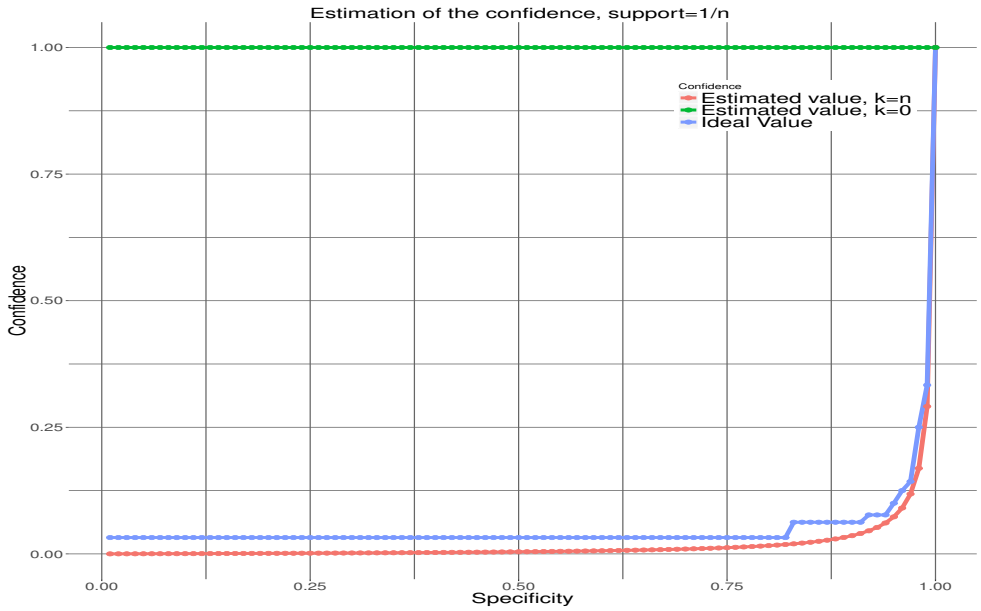


Figure 5.8: The estimation of confidence count: Influence of specificity on confidence count.

Estimation of the confidence count compared with the limit (ideal) trend is shown in figure (5.8). The ideal line is the upper bound of confidence. Its values are the minimal confidence of rules that satisfy given specificity. As it is shown, the estimated value, where $k = n$ (red line) seems to be near to the ideal value in contrast with the estimated value, where $k = 0$ (green line).

It is not possible to clearly determine the support count with the sensitivity and specificity. The trend of the ideal curve can change in future by adding new record in the data set. Since there is no direct link among support, sensitivity and specificity, any rule with low support count can satisfy the sensitivity and specificity.

5.3 Functionality of the analysis in ASTEP

Previous sections focused on the analysis and generating rules only. This section presents the functionality of the analysis in ASTEP and can be used as a user manual. Figure (5.9) shows the part of the analysis module in ASTEP.

Through combo boxes called *symptom* and *místo* are selected some ictal signs. These ictal signs define seizure symptom and body part. The purpose is to define which ictal signs will appear in the antecedent (left hand side). A

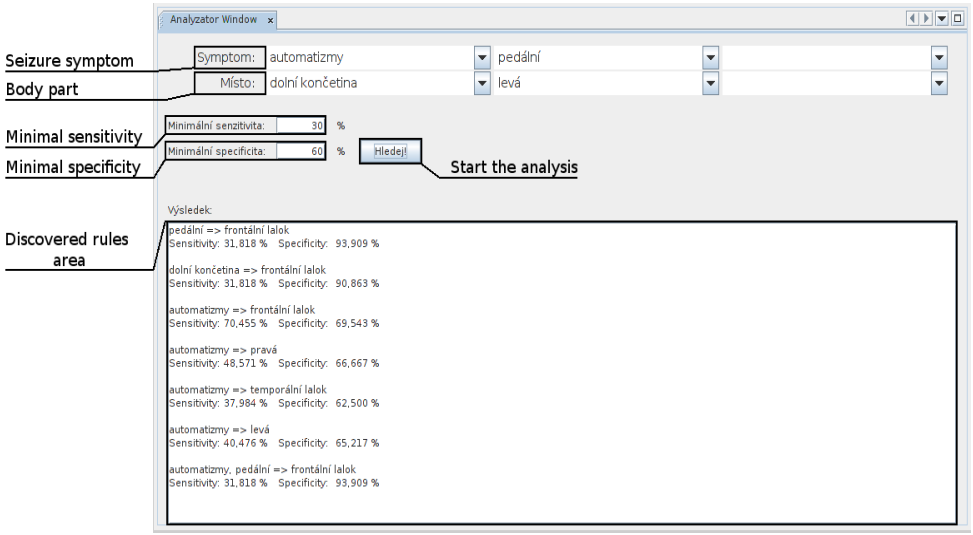


Figure 5.9: An illustration of the analysis in ASTEP.

user can define the minimal sensitivity and specificity that rule has to satisfy. The default value for both parameters is 10 %. The range of allowed values is $(0, 100\rangle$. If a user defines a value out of this range then it is not possible to start the analysis. The analysis is started through the button called *hledej!* and it is disabled in the case of values out of the range.

Discovered rules are shown in the text area bellow the text *"Výsledek"*. Only those rules that satisfy the minimal sensitivity and specificity are written to the text area. For example, in the text area is a rule *pedální* $\Rightarrow$ *frontální lalok* with sensitivity 31.8 % and specificity 93.9 %. If no rules have been found, then the text area displays such information, see Figure (5.10).

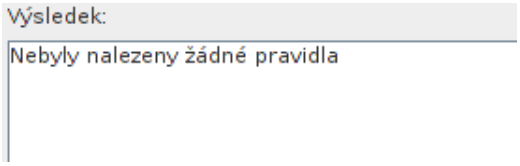
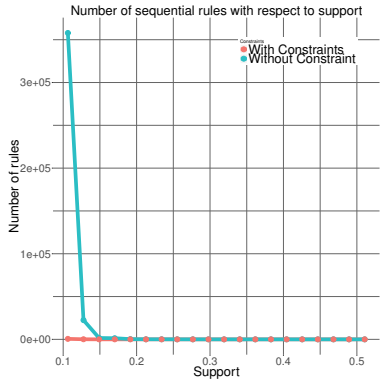


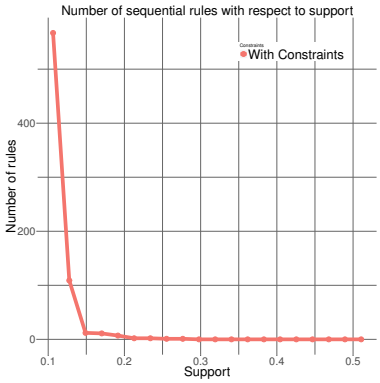
Figure 5.10: ASTEP: No rules have been found warning.

5.4 Analysis of the Sequence Pattern Mining

This section presents results of sequence pattern mining. Two approaches of the adjustment of the function *cSpade* are demonstrated.



(a) : The number of sequential rules with respect to support.



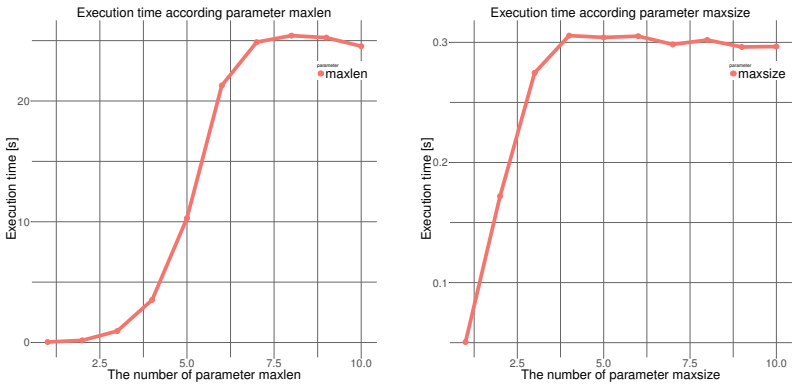
(b) : A detailed view of the sequential number of rules using item constraints.

Figure 5.11: A demonstration of the dependency between support count and number of sequential rules.

Figure (5.11a) shows the relation between the support and the number of mined rules. Similarly to previous demonstration in section (5.1), the number of rules mined without any constraints is enormous. In this case, total number of rules is 357 888 compared with 567 rules that were mined with item constraints, see Figure 5.11b. As a result, Figure (5.11) proves two facts. Support threshold has a significant effect on the number of frequent sequences. Current database consists of transactions that are rare, therefore discovered sequences have low support.

For the purpose of this demonstration the confidence value is $1/m$, where m is the number of seizures (input-sequences). The minimal possible value of

the support is $1/m$, yet the initial value of support is $5/m$.

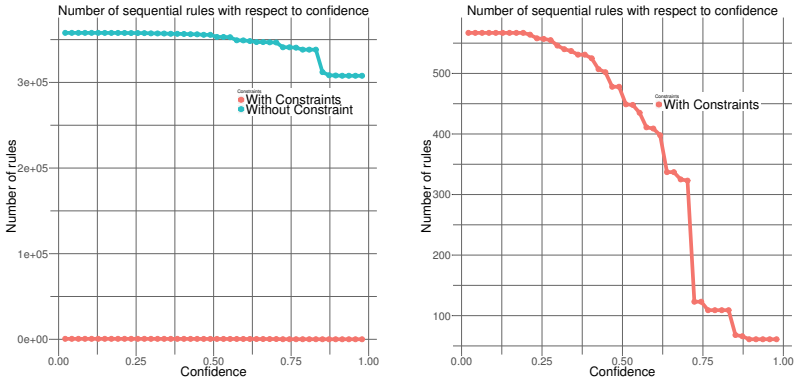


(a) : The execution time according to parameter maxlen.

(b) : The execution time according to parameter maxsize.

Figure 5.12: The execution time of function cSPADE according to the size of parameters maxlen and maxsize in 100 iterations.

Since no constraints have been used, the execution time and memory demand are high. There are two parameters that affect the execution time, *maxsize* and *maxlength*. Both are adjusted to 10 by default. The result of experiment with these parameters shows figure (5.12). The value of support is $5/m$ in this experiment. In both cases, the execution time increases, however, the figure shows the effect of parameter *maxlen*. This is the reason why it was decided to use higher support value.



(a) : The number of sequential rules with respect to confidence.

(b) : The number of sequential rules with with item constraints.

Figure 5.13: A demonstration of the dependency between confidence count and number of sequential rules

Figure (5.13a) shows number of rules with respect to confidence. In both cases the number of rules decreases. However, the number of rules mined without constraints is over 300 000 and the number of rules mined with constraints decreases under 100.

Chapter 6

Conclusion

This thesis focuses on the usage of the association rule mining with annotated record of epileptic seizures. Theoretical part provides an introduction to the field of association rule mining and sequence pattern mining and presents the algorithm *apriori* and *SPADE*. The primary objective, to create the analysis module, has been fulfilled. The implementation includes the data extraction, generating association rules in R and link to graphical user interface in ASTEP.

The analysis generates rules that satisfy the minimal sensitivity and specificity. However, the first idea of adjusting the support count with respect to sensitivity and specificity has not been successful. Results presented in section (5.2) prove that the support cannot be simply estimated with sensitivity and specificity. On the other hand, the estimation of confidence nearly fits the ideal line, thus it is considered to be correct.

In accordance to the assignment, the design of sequence pattern search has been made and tested. Results provide the information about the number of discovered sequential rules. Experiments illustrate that the execution time of the function *CSPADE* is affected by the value of parameters *maxlen* and *maxsize*. However, the parameter *maxsize* has a bigger influence on the execution time. Therefore, presented results are limited with support threshold which is not the minimal.

Since a huge amount of the association rules is mined, it is worthy to take into account any method that visualizes all the rules. Specialists could decide faster which rules are interesting and which are not, and then focus on a specific group of rules. Therefore, a visualization would be useful part of the analysis module.

It is assumed that the number of record in the database will increase, yet there is no exact prediction about the execution time. Results showed that using item constraints in function *apriori* reduces the execution time in case of low support threshold. The space for future development seems to be particularly in an optimization of sequence pattern mining, visualization of rules and interpretation of rules.

Bibliography

[1] Robert S. Fisher, Carlos Acevedo, Alexis Arzimanoglou, Alicia Bogacz, J. Helen Cross, Christian E. Elger, Jerome Engel, Lars Forsgren, Jacqueline A. French, Mike Glynn, Dale C. Hesdorffer, B.I. Lee, Gary W. Mathern, Solomon L. Moshé, Emilio Perucca, Ingrid E. Scheffer, Torbjörn Tomson, Masako Watanabe, and Samuel Wiebe. Ilae official report: A practical clinical definition of epilepsy. *Epilepsia*, 55(4):475–482, 2014.

[2] Soheyl Noachtar and Astrid S. Peters. Semiology of epileptic seizures: A critical review. *Epilepsy & Behavior*, 15(1):2 – 9, 2009. Management of Epilepsy: Hope and HurdlesCritical Reviews and Clinical Guidance.

[3] Hans O. Lüders. Simple motor seizures: localizing and lateralizing value. In *Textbook of epilepsy surgery*, pages 450 – 461. Informa Healthcare, London, 2008.

[4] Petr Berka. *Dobývání znalostí z databází*. Academia, Praha, vyd. 1. edition, 2003.

[5] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93*, pages 207–216, 1993.

[6] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[7] Michael Hahsler. Association rules, 2015.

[8] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson Addison Wesley, Boston, 1. edition, 2006.

[9] Ramakrishnan Srikant, Quoc Vu, and Rakesh Agrawal. Mining association rules with item constraints. In *Proceedings of the Third International Conference of Knowledge Discovery and Data Mining*, pages 67–73. AAAI Press, 1997.

[10] Vladimír Mařík, Olga Štěpánková, and Jiří Lažanský. *Umělá inteligence*. Academia, Praha, vyd. 1. edition, 2003.

[11] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pages 3–14, Washington, DC, USA, 1995. IEEE Computer Society.

[12] Mohammed J. Zaki. Sequences mining in categorical domains: Incorporating constraints. In *9th ACM International Conference on Information and Knowledge Management*, Nov 2000.

[13] Mohammed J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Mach. Learn.*, 42(1-2):31–60, January 2001.

[14] Carl H. Mooney and John F. Roddick. Sequential pattern mining – approaches and algorithms. *ACM Comput. Surv.*, 45(2):19:1–19:39, March 2013.

[15] Michael Hahsler, Bettina Gruen, and Kurt Hornik. arules – A computational environment for mining association rules and frequent item sets. *Journal of Statistical Software*, 14(15):1–25, October 2005.

[16] Michael Hahsler, Christian Buchta, Bettina Gruen, and Kurt Hornik. *arules: Mining Association Rules and Frequent Itemsets*, 2015. R package version 1.3-1.

[17] Christian Buchta, Michael Hahsler, and with contributions from Daniel Diaz. *arulesSequences: Mining Frequent Sequences*, 2016. R package version 0.2-15.

Appendix A

Description of the Database

- GENERATOR
  - GEN__KEY
  - GEN__VALUE
- PATIENT
  - ID
  - MRI
  - NAME
  - PET
  - POHLAVI
  - RUKA
  - STATUS
  - VEK
  - VEK__EPILEPSIE
  - DIAGNOZA__ID
  - LOKALIZACE__ID
  - STRANA__ID
  - TYP__ID
- TERMSPATIENT
  - ID
  - NAME
  - SHORTCUT
  - PARENT__ID
- MEASURE
  - ID
  - DELKA

- KONEC
- NAME
- PICTURE
- SEQUENCENUMBER
- STATUS
- VELIKOST
- POCATEK
- PATIENT__ID
- VIDEO__ZACATEK

- SYMPTOM
  - ID
  - OD
  - PERIODA
  - SEQUENCENUMBER
  - START__TIME
  - STATUS
  - TRVANI
  - TYP
  - MEASURE__ID
  - BODY__ID
  - PROPERTY__ID
- TERMSEPILEPTICFIT
  - ID
  - NAME
  - SHORTCUT
  - PARENT__ID

Appendix B

Features of the Dataset

- sequence_id
- sequence_number
- diagnosis
- localisation
- side
- diagnosis_type
- symptom_type
- body_1
- body_2
- body_3
- body_4
- property_1
- property_2
- property_3
- property_4

Appendix C

CD Content

```
|-- Java
|   `-- Analyzator.zip
|-- R
|   |-- AssociationRules
|   |   |-- epilepticRules.R
|   |   |-- findRules.R
|   |   |-- getARVector.R
|   |   |-- readData.R
|   |   `-- sequencesData
|   `-- SequencePatterns
|       |-- basketData
|       |-- SequencePatternSearch.R
|       `-- sequencesData
|-- README
`-- Text
    `-- MT-Prihoda.pdf
```