Using Learning of Behavior Rules to Mine Medical Data for Sequence Rules *

Jie Gao and Jörg Denzinger
Department of Computer Science
University of Calgary
{gaoj,denzinge}@cpsc.ucalgary.ca

Abstract

In fields like medical care, the temporal relations in the records (transactions) are of great help for identifying a particular group of cases. Thus there is some need for sequence rule learning in the classification problems in these fields. In this paper, a genetic algorithm for sequence rule learning is presented based on concepts from learning behavior of agents. The algorithm employs a Michigan-like approach to evolve a group of sequence rules, and extracts good ones into the result sequence rule set from time to time. It contains a novel quality-based intelligent genetic operator, and many adaptive enhancements to make implicit use of data-set-specific knowledge. The algorithm is evaluated on a real-world medical data set from the PKDD'99 Challenge. The results indicate that the algorithm can get satisfactory sequence rule sets from the sparse and noisy data set.

1 Introduction

1.1 Background

Rule learning is a core component in rule-based classification problems. These rules contain the criteria how we classify some given case. In traditional rule learning applications, the rules have preconditions that are conjunctions of predicates, so that the predicates are independent and their orders does not affect the rule applicability at all. But in many real-life situations, the temporal relations between conditions are of importance. In fields like medical

*Technical Report, Department of Computer Science, University of Calgary

care, where the cases being studied are developing in some time frame, considering temporal relations can help the research a lot. So in these fields, the classification work can perform better if it is based on what we call temporal sequence rules, where the sequence of predicates in the precondition of a rule represents also a temporal sequence.

This paper uses the medical care field as background, and the targeted data sets for learning are medical records, in which we have time stamps for each treatment or medical exam. The goal of our learning approach is to get sequence rules that can identify disease cases among all the cases that are being provided.

1.2 Related Work

There has been only few research on learning sequence rules. Instead, many existing research projects are on discovering sequential patterns (which naturally is very much related to learning sequence rules). They either employ a mathematical approach – analyze the data with statistical or stochastic methods–, or an approach derived from association mining methods. The former is often applied to quantitative streams while the latter is often used on transaction data sets. In this work the target data set is a transaction-like data set. So the discussion below is focused on this type of approaches.

Some variations of the *Apriori* algorithm are presented in [AS95]. The algorithms presented, namely *AprioriAll* and *AprioriSome*, are based on the frequency of events. They start from 1-event sequences and get the frequent sequences among them. Then they generate 2-event sequences based on frequent sequences and get frequent 2-event sequences. This process goes on until no frequent $(n+1)$-event sequences can be generated from frequent n-event sequences. This is rather similar to the *Apriori* algorithm except that the order of the elements matters here.

Algorithms like *AprioriAll* and *AprioriSome* can be modified to discover sequence rules. After finding the frequent sequences, an evaluation process can be done to get the sequences that can discriminate different classes well. However, this may not work well if the data is sparse, because these algorithms are selecting sequences based on the frequency, which will be low in sparse data sets. And if we lower down the frequency threshold, we may have too many candidates at the beginning.

2 The Problem Definition

For the problem of learning sequence rules, many different terminologies from conjunctive rule learning can have some influence, but some terms will have to be defined differently. So it is necessary to provide clear definitions of the problem and related terminologies, which we will do in this section.

2.1 The Data Set

In a medical care data set, there is information about a number of patients, and time-stamped medical records for each patient. The data set referred here is actually the training data set, which means we also have accurate class information in it, i.e. each patient is known to have the disease or not. A patient is also called a case, and a disease case is also called a *positive case*. Each individual medical record is a set of one or more treatments or medical exam results that are recorded at the same time stamp.

2.2 Temporal Sequence Rules

A *temporal sequence* is a list of *events* in their temporal order. The events are similar to the predicates in conjunctive rules but each of them is associated with a time stamp so that we can take their temporal order into account. If we use e_i to stand for an event, a temporal sequence can be represented as $e_1 \rightarrow e_2 \rightarrow \cdots \rightarrow e_n$.

When matching a temporal sequence with a case, the events in the temporal sequence are not required to take place adjacently in the case. The only thing that matters is the order of the events. For example in Figure 1, the cases A, B and C all match the temporal sequence S.

$$
\begin{aligned}
S: &\quad e_1 \rightarrow e_5 \rightarrow e_7 \\
A: &\quad e_2 \rightarrow e_1 \rightarrow e_5 \rightarrow e_7 \\
B: &\quad e_1 \rightarrow e_3 \rightarrow e_5 \rightarrow e_7 \\
C: &\quad e_4 \rightarrow e_1 \rightarrow e_2 \rightarrow e_6 \rightarrow e_5 \rightarrow e_8 \rightarrow e_7
\end{aligned}
$$

Figure 1: A temporal sequence that matches 3 cases

A *temporal sequence rule* has a right-hand term in addition to a temporal sequence. This right-hand term states whether a matching case is a positive case or not. In the targeted field –medical care–, the majority of the cases are

not positive cases. So we can use non-disease as a default class and let our sequence rules always classify positive cases. In this way, the right-hand term is always "positive" or "disease". If we denote a temporal sequence as S, a temporal sequence rule can be denoted as $S \Rightarrow disease$. Since we always have the same right-hand term, we can omit it and use only a temporal sequence to stand for the temporal sequence rule that classifies positive cases.

2.3 Accuracy and Coverage

The *accuracy* of a sequence rule is the ratio of the number of correctly identified disease cases to the number of all cases that is claimed by this sequence rule to be a disease case. If we use *true positive* (TP) to represent the number of correctly identified disease cases, and *false positive* (FP) to represent the number of disease cases claimed in error, we can use the equation below to calculate the accuracy:

$$accuracy = \frac{TP}{TP + FP}$$

Coverage used in this paper is the ratio of the number of correctly identified disease cases to the number of all actual disease cases in the data set. If we use *true positive* (TP) to represent the number of correctly identified disease cases, and *positive* (P) to represent the number of all actual disease cases , we can use the equation below to calculate the coverage (of a particular rule or a rule set):

$$coverage = \frac{TP}{P}$$

2.4 The Learning Problem

The learning problem can be defined as follows: $\mathcal{T}$ is a two-class training data set, in which the cases are labeled with classes $\mathcal{C} = \{c_1, c_2\}$. Our learning task is to find a set of temporal sequence rules $\mathcal{R}$. This sequence rule set $\mathcal{R}$ can be used to predict the class of a given case from a data set $\mathcal{D}$ without class labels, thus working as a function $R : \mathcal{D} \rightarrow \mathcal{C}$.

3 Genetic Algorithm Learning Approach

A genetic algorithm framed according to the Michigan approach (see [Gol89]) will be employed in our approach to learn temporal sequence rules from a data set, i.e. each individual is a single temporal sequence rule. We will then need some techniques to identify the final sequence rule set. Compared with

the Pittsburgh approach (in which an individual would be a whole rule set), algorithms based on the Michigan approach have simpler representations for individuals and usually simpler genetic operators, as well as fewer parameters. For learning tasks on large data sets, it is worth doing some extra rule set selection work to gain simplicity and smaller size of individuals.

3.1 Representation of an Individual

An individual in our genetic algorithm is a temporal sequence rule. According to the assumptions in Section 2.2, we can use a temporal sequence to represent a positive-class sequence rule. So an individual here is actually a temporal sequence. In our implementation, it is a list of events in temporal order.

The events are using a fix-length binary encoding representation. In the binary encoding, the length is the number of possible events and each bit corresponds to an event. In this way, compound events (multiple events happen at the same time) can be easily expressed by setting multiple bits to 1. An individual is a variable-length list of events. As already discussed in Section 2.2, the right-hand class label is not necessary in the representation of an individual. So an actual individual is just a temporal sequence. The detailed meaning of an individual is task specific. In Section 4.2 these details will be discussed.

When matching an event e_i from an individual with an event e_t in the training data set, we do not require e_i to be exactly the same as e_t. The matching rule is $e_i \& e_t = e_i$ (& is bitwise `and` operator), i.e. e_t contains all the event bits in e_i. This allows the matching to ignore irrelevant event bits in the training data set.

3.2 Genetic Operators

There are two different types of genetic operators in our genetic algorithm. One type are the "standard" genetic operators, including *mutation* and *crossover*. Mutation is to change an individual randomly employing one of the following 3 methods: delete an event, change an event, and add a new event. Crossover is to generate a new offspring out of two parent individuals. Crossover points are randomly chosen for both individuals and then the subsequence from the beginning to the crossover point in the first individual is connected with the subsequence from the crossover point to the end in the second individual to form the offspring.

Another type of genetic operator is based on quality criteria associated with parts of a parent individual, which is why we call these kinds of operators

"intelligent genetic operators". If we have an individual $e_1 \to e_2 \to \cdots \to e_n$, our intelligent genetic operator is defined by the following steps:

1. Calculate the fitness of each subsequence that starts at e_1 and ends at e_i ($i = 1$ to n). These fitnesses are $f_1, f_2, \cdots, f_n$.
2. Find the maximum in $f_1, f_2, \cdots, f_n$, say f_m. Compare f_m with f_n.
   (a) If f_m is much better than f_n, cut the sequence after e_m and erase $e_{m+1}, e_{m+2}, \cdots, e_n$. Then randomly decide either to append a new event at the end or append a "tail" from another good individual.
   (b) If f_m is very close to f_n, find the minimum in $f_{m+1}, f_{m+2}, \cdots, f_n$, say f_k, and erase e_k.

This intelligent genetic operator is based on the following consideration: When a sequence becomes longer, intuitively we can tell that the coverage (number of matching cases) is decreasing. To keep the sequence as generic as possible, we should eliminate some bad points in the sequence to maximize the coverage. And also the quality may drop dramatically after a bad part has been added into the sequence. So the intelligent genetic operator tries to find such bad parts and its application is intended to keep the sequence generic. In some sense, it is a retraction of the bad mutations and crossovers.

3.3 Evolution Parameter Selection

One of the most important and difficult tasks in genetic algorithms is to select the many evolution parameters. They include the application probabilities of each genetic operator, the population size, the elite group size, the fitness measure and some parameters particular to the Michigan approach used here.

The application probabilities of each genetic operators, the population size and the elite group size cannot be easily decided a priori. Their values are mostly decided by experiences and experiments. A basic principle of selecting these values is to allow enough diversity among the individuals, and during the evolution process, the good parts will be kept over generations.

The fitness measure for our task should consider both accuracy and coverage of the sequences so that we can evolve good generic sequences. Without the coverage part in the fitness measure, we can easily run into the over-fitting problem and in the worst case, each sequence will only predict one case in the data set. The accuracy measure is the percentage of true positive cases in all the matching cases. And the coverage measure is the percentage of true positive cases related to the actual positive cases. To balance between accuracy

and coverage, the fitness is then the product of the two percentages. This makes sure that a high accuracy will not mask a low coverage, and vice versa.

The other parameters that are related with the Michigan approach –the ones that decide how a final sequence rule set is built from the individuals, will be discussed in Section 3.4.

3.4 Building the Final Sequence Rule Set

It is a necessary step that we build a final sequence rule set in addition to evolving the individual sequence rules. A commonly used method is to run the genetic algorithm multiple times. In each run the winning (fittest) individual is extracted into the final rule set. This is also the method we used in our approach. A fitness threshold and an accuracy threshold are given. During the evolution process once the thresholds are met by the winning individual, it is extracted into the final sequence rule set. Afterwards, the positive cases covered by this rule are removed from the training data set and the individuals are re-generated to start a new evolution run. So, later evolutionary processes only run with the positive cases not covered by earlier found sequence rules.

3.5 Adaptive Behaviours

Adaptive behaviours are used in various places in our genetic algorithm to make the evolutionary process more efficient:

- An event set is generated before-hand. It contains all the events in positive cases in the training set. The first generation of individuals are 1-event sequences whose events are randomly picked up from this event set. This ensures that the randomly generated individuals can match the positive cases.

- In mutation, if the method is adding new events to the individual, the events are also randomly picked from the above-mentioned event set so that the mutated individual does have more potential to fit the positive cases.

- A generation number limit is set to avoid improper thresholds in the building of the sequence rule set. When the maximum fitness gets stuck around a value that is below the threshold, the generation number counter is checked against the limit. If the limit is exceeded, the algorithm will lower the thresholds to fit the situation. After extracting an individual into the sequence rule set, the thresholds are reset to the initial values to avoid having them ever-decreasing.

These adaptive behaviours are actually the implicit use of topic specific knowledge in the training data set. They make the genetic algorithm more knowledge-based.

4 Testing

4.1 Data Set

The data set used for testing is from the medical data set in the PKDD'99 Discovery Challenge (see [ECM99]). This data set contains the data on collagen disease patients collected at Chiba University hospital. The patients in the data set have medical test results with time stamps. And each patient also has a record whether he/she had thrombosis. The learning task on this data set is to discover sequence rules that predict thrombosis.

This data set contains real-world data. So it is different from artificially generated data sets in the following aspects:

- The data set contains large amounts of data. In the data set, there are nearly 1300 patients, over 57000 test records and each test record has at most 44 different test results.
- The data set content is not strictly formatted. In each field, there could be quantitative numbers, range expressions (e.g. > 3000), "+", "−" or "+−" signs and other symbolic marks to indicate different results for the same medical test.
- Not all the data in the data set is useful. There are test records without patient personal information, and there are also patients without test records. We have to decide what part of data is useful.
- The data set is sparse. Among the over 1000 patients there are only 66 patients who really have thrombosis. And in the test records there are many empty fields.

These characteristics make this data set a real challenge for our learning approach.

4.2 Data Transformation

As the records are not consistently valued, cleaning and transformation have to be done to make the data usable for our learning algorithm. The following steps were taken to make the transformation:

1. Assign a arbitrary value for each range expression. This is suggested by the data set guide (see [ECM99]). For cases like "$> N$", $1.5N$ is assigned; and for "$< N$", $\frac{2}{3}N$ is used.

2. Convert each quantitative value to a symbolic expression like "+" and "−". In medical records, the signs "+" and "−" are often used to indicate an abnormal result and normal result respectively, and "+−" indicates a result at the boundary of normal and abnormal. According to the given normal ranges in the data set description (see [ECM99]), the quantitative values with a single normal boundary are converted to "+" and "−" expressions. And a 10% overlap is made manually here to get the "+−" cases. For the tests whose normal range is like "$a < N < b$", the symbols "l", "n" and "h" are used to express low, normal and high respectively. A 10% overlap is made at each boundary so that we can also have "ln" and "nh", like the "+−" in single-boundary fields. Other types of values are preserved in their original forms.

3. For each possible value in each field, assign a bit in the binary encoded event expression. And convert each record in the data set into a binary encoded event.

4. Link each patient's events into a temporal sequence chronologically.

Number of cases	1235
Number of disease cases	66
Number of events	57541
Number of unique events	39264
Number of unique events in disease cases	4268
Number of bits in event binary encoding	234

Figure 2: Statistics from the training data set

In the data transformation work, we do not remove any fields. And we keep all non-quantitative values as they are. Because in this data set, the provided medical background knowledge is only enough to discriminate normal values from abnormal results. So all other values are kept untouched and encoded into the binary representation directly so that we will not lose any possibly useful information.

Finally the set of temporal sequences, together with the flag whether the patient has thrombosis or not, forms the training data set. Some statistics from the data set are listed in Figure 2 to show the data characteristics.

4.3 Testing Results

The testing is made with different parameter values to make the genetic algorithm work efficiently. From the tuning with these parameters, we can find the population size should be at least around 300, and the mutation operator should have a relatively high probability to be selected. This is because the number of possible events is very large, and the number of different sequences is even larger (virtually infinity because there is no sequence length limit). Only a large population size and a high mutation probability can make the individuals explore more diversity and thus produce more useful results. After a few experimental runs, the genetic operator parameters are decided and a new generation is created from the previous one as follows: Elite group is top 30% of the population according to the fitness and the remaining 70% are generated by crossover. After that 30% of the new population are mutated. And finally 40% of the population is modified using our intelligent genetic operator.

Another parameter that has much impact on the result are the thresholds for extracting individuals into the sequence rule set (refer to Section 3.4). We should avoid unnecessarily high values for them. Although the adaptable behaviours in the algorithm can lower the thresholds when we give too high values, it will take extra time for the algorithm to run until it exceeds the generation limit. The domain experts' opinions also suggest we start from not-too-high thresholds because the doctors can accept some error rate if all the disease cases can be successfully covered. So in the testing, the correctness threshold is set between 0.5 and 0.7.

The intelligent genetic operator and adaptive behaviours in the genetic algorithm help the work a lot. While monitoring the evolutionary process, from time to time we can observe the phenomenon that the maximum fitness of a generation drops and then goes up back to the previous high value in later generations (Figure 3). This is an evidence of the intelligent genetic operator's work. And if we turn off the adaptive adjustment of thresholds in extracting the individuals into the sequence rule set, the evolutionary process

```
Max fitness: 1.16377
Max fitness: 1.16377
Max fitness: 1.16377
Max fitness: 1.18906
Max fitness: 1.16377
Max fitness: 1.18906
Max fitness: 1.18906
Max fitness: 1.18906
```

Figure 3: Fitness temporarily decreases and goes back again

```
Exceed stuck_limit. Adjust the thresholds. The new thresholds are:
fitness_threshold=1.2803        correctness_threshold=0.675
Exceed stuck_limit. Adjust the thresholds. The new thresholds are:
fitness_threshold=1.2803        correctness_threshold=0.6075
Exceed stuck_limit. Adjust the thresholds. The new thresholds are:
fitness_threshold=0.476336      correctness_threshold=0.517628
Exceed stuck_limit. Adjust the thresholds. The new thresholds are:
fitness_threshold=0.556522      correctness_threshold=0.357252
                              ... ...
Exceed stuck_limit. Adjust the thresholds. The new thresholds are:
fitness_threshold=0.498339      correctness_threshold=0.529448
Exceed stuck_limit. Adjust the thresholds. The new thresholds are:
fitness_threshold=0.498339      correctness_threshold=0.373754
Exceed stuck_limit. Adjust the thresholds. The new thresholds are:
fitness_threshold=0.5   correctness_threshold=0.53033
Exceed stuck_limit. Adjust the thresholds. The new thresholds are:
fitness_threshold=0.886427      correctness_threshold=0.706127
Exceed stuck_limit. Adjust the thresholds. The new thresholds are:
fitness_threshold=0.886427      correctness_threshold=0.66482
                              ... ...
```

Figure 4: Adaptive adjustment of thresholds

gets stuck with some peak fitness values under the threshold. Figure 4 shows the adjustments in a test run. These logs show that the adjustments not only adapt to the data set, but also avoid ever decreasing thresholds by resetting them back to the initial value after each sequence rule extraction.

A typical sequence rule set from the testing results is given in Figure 5. They are listed in the order of extraction from the evolved individuals. The "True Positive" column is the number of disease cases correctly identified by each sequence rule; "Total Covered" is the total number of cases claimed by the sequence rule to be positive cases; and the "Disease Cases" is the number of remaining disease cases while evolving and extracting this sequence rule. In this rule set, we can clearly tell that the rules are mainly of two types. The first several sequence rules are general rules that have relatively large coverage, while the rest are mostly specialized sequence rules that only cover 1 or 2 disease cases but with high accuracy. This is because the data set is very sparse with only about 5% of disease cases in it. After extracting several large-coverage sequence rules, the rest of the disease cases are too few. They cannot prove whether the sequence rules covering them are general or not.

In Figure 6, a sequence rule discovered by the test runs is given. The presented sequence rule is already translated into a better understandable form from its original binary encoding. This sequence rule shows that if a patient has *fibrinogen* normal at the beginning, then with *uric acid* below normal value but *lactate dehydrogenase*, *total protein*, *albumin*, *urea nitrogen*, *creatinine* and *C-reactive protein* are all normal, then with repeatedly *prothrombin*

No.	Fitness	True Positive	Total Covered	Disease Cases
1	1.2803	13	20	66
2	1.15566	7	8	53
3	0.652174	3	3	46
4	0.523256	3	4	43
5	0.5	2	2	40
6	1.18421	6	8	38
7	0.625	2	2	32
8	1	3	3	30
9	0.740741	2	2	27
10	0.8	2	2	25
11	0.869565	2	2	23
12	0.952381	2	2	21
13	1.05263	2	2	19
14	1.17647	2	2	17
15	0.666667	1	1	15
16	0.714286	1	1	14
17	1.53846	2	2	13
18	0.909091	1	1	11
19	1	1	1	10
20	1.11111	1	1	9
21	1.25	1	1	8
22	1.42857	1	1	7
23	1.66667	1	1	6
24	2	1	1	5
25	2.5	1	1	4
Too few disease cases left. Evolution process stops.				

Figure 5: A typical sequence rule set

time above normal value, he/she is probably having thrombosis. This temporal sequence matches 12 patients in the data set, and 10 out of them are positive cases. This result is very similar to one of the discoveries stated in [LMC+99], which also indicates that abnormal *prothrombin time* value indicates thrombosis. Although our sequence rule and the discovery in [LMC+99] have the same coverage, our sequence rule has a higher accuracy.

Further examination of the sequence rule in Figure 6 shows the advantage of sequence rules over conjunctive rules. If we convert this sequence rule into a conjunctive rule by dropping the temporal order and combining redundant events, and match this conjunctive rule against the training data set, we find it is covering 32 cases and 15 out of them are positive cases, which gives an accuracy of 0.46875. Comparing with the original sequence rule's accuracy,

Event No.	Symptom
1	*fibrinogen* normal
2	*uric acid* **below normal** *lactate dehydrogenase* normal *total protein* normal *albumin* normal *urea nitrogen* normal *creatinine* normal *C-reactive protein* normal
3	*prothrombin time* **above normal**
4	*prothrombin time* **above normal**

Covering 12 cases, 10 are positive cases.
coverage=0.18182, accuracy=0.83333

Figure 6: A sequence rule of high quality

we can tell the temporal sequence rule is really in higher quality.

In the tests, we used the entire data set as the training data and no classification tests on new cases is done. This is due to the sparseness of the data. If we split the already limited disease cases into training set and test set, we will end up with data sets lacking enough disease cases for learning. The classification can only be done when there is enough data available.

In the results presented above, we do not have a single sequence rule that can cover most of the disease cases but still has high accuracy. This is expected because the real-world rules of the diseases are very complex and cannot be easily discovered by a unified method. And many sequence rules in the rule set are only covering one or two disease cases but with a high accuracy –rather specialized sequence rules. However, we cannot tell if these sequence rules are actually too specialized in the real world because in the data set we have, the disease cases are sparse. We can expect better sequence rules if we have more disease cases in the data set.

5 Conclusion

The test results in the last section show that the genetic algorithm learner can get satisfactory (from the point of view of doctors) sequence rules from a sparse, noisy and large-scale real world data set. On such a data set these discoveries are already very good. Some innovative enhancements are introduced into the genetic algorithm, including an intelligent quality-based genetic operator and adaptive adjustment of thresholds. The former helps stabilize the evolutionary process, and the latter helps decrease the effect of improper

parameters. Both of them are actually using the data-set-specific knowledge implicitly, thus making the genetic algorithm more "intelligent".

At this point, this algorithm is still in an unmatured stage. There is much work to do to improve the performance. To help the algorithm scale up, one potential work is to distribute the evolutionary process on several machines and on each machine a group of individuals are evolved and the best results are exchanged. This will increase the total population and makes the individuals explore the search space more diversely. Another possible improvement is to couple this algorithm with other methods, say *AprioriAll*. This may help each algorithm to overcome its own weaknesses with the help of the other algorithm. Other potential extensions of this algorithm may be considering each event's effective time limit (the maximum interval between two events take place), and adding more tricks to the binary encoding to allow "`or`" relations in a compound event.

References

[AS95] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee S. P. Chen, editors, *Proceedings of the 11th International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[DK00] Jörg Denzinger and Michael Kordt. Evolutionary on-line learning of cooperative behavior with situation-action-pairs. In *Proceedings of the 4th International Conference on MultiAgent Systems (ICMAS-2000)*, pages 103–110, Boston, July 2000. IEEE Press.

[ECM99] ECML/PKDD Discovery Challenges, `http://lisp.vse.cz/challenge/`. *Guide to the Medical Data Set, PKDD99 Discovery Challenge*, 1999.

[Fre02] Alex A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Natural Computing Series. Springer, 2002.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[LMC+99] B. Levin, A. Meidan, A. Cheskis, O. Gefen, and I. Vorobyov. PKDD99 discovery challenge – medical domain. In P. Berka, editor, *Workshop Notes on Discovery Challenge, PKDD-1999, Prague, Sep. 15-18*, pages 55–57, Univ.of Economics, Prague, 1999.