EFFICIENT FEATURE DETECTION FOR SEQUENCE CLASSIFICATION IN A RECEPTOR DATABASE*

Edgar H. de Graaf   Walter A. Kosters

Leiden Institute of Advanced Computer Science
Leiden University, The Netherlands
{edegraaf,kosters}@liacs.nl

Abstract

The analysis of sequences is one of the major research areas of bio-informatics. Inspired by this research, we investigate the discovery of sequential patterns for use in classification. We will define variations of a fit function that enables us to tell if one pattern better fits to a class than another. Furthermore we will show how domain knowledge can be used for faster discovery of better sequential patterns in specific types of databases, in our case a receptor database.

1 Introduction

Sequence analysis has many application areas, e.g., protein sequence analysis and customer behavior analysis. We investigate extraction of features for protein sequence classification where features are *sequential patterns*: ordered lists of items (for proteins the items are amino acids). As a motivating example, we would like to know if a protein sequence, an ordered list of amino acids, belongs to the olfactory family or not, where the olfactory family is a group of proteins that deals with smell. We focus on a special group of proteins called GPCRs. These G-protein-coupled receptors (GPCRs) play fundamental roles in regulating the activity of virtually every body cell [12]. Usually classification is done unsupervised using alignment, however in the case of GPCRs this turned out to be difficult. Fortunately, we know for some protein sequences whether they are of the olfactory family or not. These sequences can thus be divided into two disjoint classes: olfactory and no-olfactory, and from these classes we can extract sequential patterns to be used as attributes in a classification algorithm (as is being proposed in [9]). The question we try to answer in this paper is: which sequential patterns are the *best* features? And how can domain knowledge be used to improve the search for such patterns?

Classification based on sequential patterns is also applicable in many other areas. For example, in the case of customer behavior analysis, we might want to characterize groups of clients based on sequential patterns in their behavior.

The "best" sequential patterns are discovered through a function that judges patterns. In Section 2 we will discuss different instances of this function and select one for our purposes. Section 3 adapts the PrefixSpan algorithm of [10] to deal with this function. In addition, a pruning strategy is introduced in Section 4 increasing efficiency by first searching in a certain area of the sequence, the *probable time window*. Section 4 also describes how preferring small patterns can further increase classification performance. The effectiveness of these improvements will be shown in Section 5.

2 Related Work

Our algorithms will be based on the pattern growth approach called PrefixSpan proposed in [10]. Classification by means of patterns has been done before but not so much in the sequence domain. We now mention related work in the non-sequence domain. Apriori-C [6] constructs classification

*This research is carried out within the Netherlands Organization for Scientific Research (NWO) MISTA Project (grant no. 612.066.304).

rules by extending the Apriori algorithm [2, 3]. Apriori-C discovers a large number of rules from which a fixed number of rules with the highest support are selected. Apriori-SD [7] solves the problem of selecting the right rules with *subgroup discovery.* This algorithm selects a subgroup of rules by calculating their *weighted relative accuracy.* This means that the probability of a pattern occurring in a class is compared with the probability of its occurrence outside the class. This is weighted with the probability of a class. Most class association rule mining algorithms work with unordered sets of items frequently occurring together in *item sets.* CorClass [13] describes an algorithm that also works with item sets. It introduces a new method of pruning. Specialized rules are only added if the upper bound of its correlation is higher than the minimal correlation of k rules. In our work we use a similar method of pruning. Much work has been done in the field of molecular feature mining, e.g., the MolFea algorithm described in [8]. MolFea employs a level-wise version space algorithm to discover those molecule fragments often occurring in one data set and less often in another.

3 The Maximal Discriminating Patterns

One would like to select the best patterns for use as attributes in a classification algorithm. But how can we tell if one pattern is better than the other? In this section we will first explain the notion of support and why it is less useful for selecting the best pattern. Next we introduce the notion of confidence which will give more useful patterns, but it also has disadvantages. Finally we will discuss and motivate so-called maximal discriminating patterns, enabling us to have patterns specific to one class, but without the disadvantages of confidence.

Assume given a database D with $D = D_1 \cup D_2 \cup \ldots \cup D_c$, with c classes. The D_i's $(1 \leq i \leq c)$ are mutually disjoint and not empty.

Each record in the database is a non-empty finite *sequence* (i.e., an ordered list) of items from the set $\Sigma = \{\mathtt{A},\mathtt{B},\mathtt{C},\ldots\}$, e.g., $(\mathtt{C},\mathtt{B},\mathtt{G},\mathtt{A},\mathtt{A},\mathtt{A},\mathtt{C},\mathtt{B})$. Now fit_0 is defined as support (as used in association rule mining algorithms like Apriori [3]), because support can be seen as a measure of how well a pattern fits the data. Commonly a sequence d is said to support a *pattern* s if the pattern is contained (in the set sense) in the sequence:

$$\mathit{supp}_0(s,d) = \begin{cases} 1 & \text{if for all } i \ (1 \leq i \leq k) \text{ there is a } j \ (1 \leq j \leq \ell) \text{ with } s_i = d_j; \\ 0 & \text{otherwise,} \end{cases}$$

for $s = (s_1, s_2, \ldots, s_k)$ and $d = (d_1, d_2, \ldots, d_\ell)$, $k \geq 1$ and $\ell \geq 1$. This means that s is a *subset* of d. We then can define fit_0:

$$\mathit{fit}_0(s, D_i) = \frac{1}{|D_i|} \sum_{d \in D_i} \mathit{supp}_0(s,d)$$

$(1 \leq i \leq c)$, where s is a pattern.

We now specialize support to sequences. A sequence $d = (d_1, d_2, \ldots, d_m)$ is called a *super-sequence* of a sequence $s = (s_1, s_2, \ldots, s_k)$ if $k \leq m$ and for each s_i $(1 \leq i \leq k)$ there is a d_{j_i} $(1 \leq j_i \leq m)$ with $s_i = d_{j_i}$ and $j_{i-1} < j_i$ $(i > 1)$. We denote this with $s \prec d$. The sequence s is called a *sub-sequence* of d. This defines *sequential patterns* on sequences items. (Another definition of sequential patterns was given by Agrawal et al. in [3], in which they define sequential patterns on sequences of item sets). We now let

$$\mathit{supp}_1(s,d) = \begin{cases} 1 & \text{if } s \prec d; \\ 0 & \text{otherwise,} \end{cases}$$

and define fit_1 in the same way as fit_0 was defined using supp_0.

Now fit_1 or fit_0 by itself is not useful for selection of features for classification. One of the patterns of size 1 will always have the highest fit and these small patterns are probably often present in more than one D_i. Thus the presence of such a pattern will not give a good distinction between classes.

The next most logical step is to use confidence to select the best patterns. The patterns x_r $(1 \leq r \leq c)$, one for each class, are then chosen to maximize *confidence* $(\mathit{fit}_1(x_r, D_r)|D_r|)/(\mathit{fit}_1(x_r, D)|D|)$.

The class t of sequence s is the t ($1 \leq t \leq c$) where $x_t \prec s$. If more than one t is possible we select based on the highest confidence. One is selected at random if more than one class t has a pattern with the highest confidence. If there is no t where $x_t \prec s$ then the sequence could be said to be "undecided".

A problem is that we only pick one pattern per class. This is plausible if a family of a sequence is only decided by one sequence of features. However, it is often the case that the class of a sequence is decided by multiple patterns. Moreover there can be constraints on the pattern. This means that the class deciding pattern x_t *with* the constraint is not necessarily equal to the x_t *without* the constraint, e.g., patterns are in different areas of the sequence. As a consequence it is usually possible to find a combination of patterns with a better classification performance. Finally it is possible that a single sequential pattern x_t is equal for two or more classes, and as a consequence a classification will be done at random. This problem will occur with a lower probability if we use multiple patterns for each class.

Another major drawback of the confidence method is that the size of the D_i's seriously influences the classification. E.g., assume we have databases D_1 and D_2. Furthermore assume D_1 contains 500 sequences and D_2 only 100. The pattern p_1 occurs 100 times in D_2 and 60 times in D_1, thus a confidence with respect to D_2 of 0.625. Another pattern p_2 occurs 70 times in D_2 and 10 times in D_1, giving a confidence of 0.875. The pattern p_2 will be used for classification if no other pattern has a higher confidence. However p_1 occurs in every sequence of D_2 and only in a small percentage of the sequences in D_1. One could argue that p_1 should be preferred over p_2.

Therefore we define fit_2, which we use in the remainder of this paper:

for a pattern s and $1 \leq q, r \leq c$ we define $\delta(s, D_q, D_r) = \mathit{fit}_1(s, D_q) - \mathit{fit}_1(s, D_r)$,
and we let $\mathit{fit}_2(s, D_r) = \min\{\delta(s, D_r, D_q) \mid 1 \leq q \leq c \ \wedge \ q \neq r\}$.

We then choose patterns x_r ($1 \leq r \leq c$) with maximal $\mathit{fit}_2(x_r, D_r)$. We can then use them to classify sequences as before, without the drawbacks mentioned above. Unlike confidence, it calculates difference independent of the size of each database. We will usually find those patterns that are characteristic for one class. With *characteristic* we mean that fit_1 will have a high value in D_t and a lower value in the other $D_i's$, $i \neq t$.

Our new fit has some similarities with the concept of *emerging patterns* [4]. In order to discover emerging patterns, patterns are preferred where the ratio $\mathit{fit}_1(s, D_1)/\mathit{fit}_1(s, D_2)$ is the highest, where D_1 and D_2 are two databases each containing one class of sequences. Bailey et al. [4] further investigate jumping emerging patterns. These are patterns that have a support of zero in D_2 and a non-zero support in D_1. Emerging patterns can also be defined in a way similar to fit_2, but now using $\mathit{fit}_1(s, D_q)/\mathit{fit}_1(s, D_r)$ instead of $\delta(s, D_q, D_r)$.

Classification algorithms usually need a limited number of attributes. In order to classify a sequence s we use a finite number of n sequential patterns $p_1^t, p_2^t, \ldots, p_n^t$ per class t, where $\mathit{fit}_2(p_1^t, D_t) \geq \mathit{fit}_2(p_2^t, D_t) \geq \ldots \geq \mathit{fit}_2(p_n^t, D_t)$ and p_n^t has the n-th highest support for all possible patterns. These patterns, the so-called *maximal discriminating patterns*, could be used by any classification algorithm when we first convert each sequence into a vector indicating for each pattern if it is contained in the sequence, see [9]. However it is possible that, e.g., p_1^t is supported by all or most of the sequences supporting p_2^t. Thus p_2^t might not improve classification. This problem could be solved by removing all sequences containing p_1^t from D_t. The algorithm for searching the sequence with maximal fit is then again applied to this subset of D_t in order to find p_2^t. In this paper we do not further focus on the precise classification performance, but rather on the discovery of the discriminating patterns. Our algorithm aims at finding the set $P = P^t$ of maximal discriminating patterns.

4 Algorithm without Domain Knowledge

Our pattern search algorithm, coined PrefixTWEAC (Time Window Exploration And Cutting), is based on PrefixSpan. The algorithm does not generate candidates, but it grows patterns from smaller patterns. This principle makes it faster than most Apriori like algorithms because it doesn't generate candidates that do not exist in the database [10]. PrefixSpan is a depth first algorithm, which will be explained in more detail in Section 5 when we adapt this algorithm to our current needs (see Table 1 and Table 2). PrefixSpan as described in [10] searches for those patterns with *support* larger than or equal to a given support threshold *minsupp*, where support is defined as fit_1. The algorithm starts with all frequent sub-sequences of size one. For each sub-sequence a projected database is created. These frequent sub-sequences are extended to all frequent sub-sequences of size two by only looking in the projected database. This *projected database* is a database of pointers to the first item occurring after the current pattern, also called the *prefix*. A sequence is only in the projected database if it contains the prefix, making PrefixSpan faster than other algorithms that count in the entire database. Again for each frequent sub-sequence of size two a corresponding projected database is created. This process continues recursively until no extension is frequent anymore.

PrefixTWEAC (Table 1) is different from PrefixSpan in that it searches for the maximal fit_2 instead of the maximal support fit_1. The function fit_2 is by definition not anti-monotone (so $fit_2(s_1, D_t) > fit_2(s_2, D_t)$ might happen, where s_1 is a super-sequence of s_2). However the anti-monotone property for fit_1 can still be used in two ways, when looking for the one pattern with maximal fit_2. First of all in PrefixTWEAC we only examine an extended pattern p if $fit_1(p, D_t) \geq minsupp$ where *minsupp* is the support threshold. Secondly p is not further examined if $fit_1(p, D_t) <$ *current maximal fit*, where *current maximal fit* is the current best fit of all patterns found while searching. The value of $fit_2(p, D_t)$ will never become larger than the current maximal fit, because it can at most become $fit_1(p, D_t)$. Note that CorClass uses similar methods to prune [13].

```
PrefixTWEACCore(prefix, projected_database)
1. For all items $i$ that can extend the prefix
2.     new_prefix = prefix extended with item $i$
3.     Count $w_1 = fit_1$ in the projected_database$_t$ for new_prefix
4.     Calculate $f_2 = fit_2$ for new_prefix
5.     Create a projected database new_projected_database with new_prefix
6.     Get $\delta_{min}$, the lowest $fit_2$ in $P$
7.     Get $s_{min}$, $fit_1$ corresponding with the lowest $fit_2$ in $P$
8.     if $w_1 \geq minsupp$ and $|P| < n$ then
9.         Add new_prefix to $P$
10.        Call PrefixTWEACCore(new_prefix, new_projected_database)
11.    else if $w_1 \geq minsupp$ and $w_1 \geq \delta_{min}$ then
12.        if $f_2 > \delta_{min}$ or
13.            ($f_2 = \delta_{min}$ and $w_1 > s_{min}$) or
14.            ($f_2 = \delta_{min}$ and $w_1 = s_{min}$ and new_prefix $\prec p_n$) then
15.            Remove $p_n$ from $P$ and add new_prefix to $P$
16.        Call PrefixTWEACCore(new_prefix, new_projected_database)
```

Table 1: The PrefixTWEAC algorithm

5 Domain Specific Improvements

In the previous section we stated that fit_2 can be used to "prune": certain pattern extensions are not further examined because they can never lead to the maximal fit_2. The faster we get to a large fit_2 for the n-th pattern in $P = P^t$ the better, because all extensions with a lower $fit_1(p, D_t)$ can be pruned. The improved version of PrefixTWEAC will be explained in the sequel.

If we consider protein sequences then pattern discovery might be done faster when using certain knowledge about the sequences:

- Protein sequences are sequences of amino acids. Certain parts of such a sequence are shaped like a helix in 3D space. These helices will probably contain most of the maximal fitting sequences since parts outside the helix have more variation in size and content. Patterns (partially) outside the helix are less likely to occur in most members of the protein family.

- Small patterns are preferred. Smaller patterns are less specific and biologists prefer smaller patterns in their analysis.

```
PrefixTWEACExt(prefix, projected_database)
1. For all items $i$ that can extend the prefix
2.    new_prefix = prefix extended with item $i$
3.    Count $fit_1$ for new_prefix:
4.       $w_1 = fit_1$ in the projected_database$_t$ without the inclusion vector, using $supp_1$
5.       $w_2 = fit_1$ in the projected_database$_t$ with the inclusion vector, using $supp_1^{\mathrm{PTW}}$
6.    Calculate $f_2 = fit_2$ for new_prefix (without the inclusion vector)
7.    Create new_projected_database (without using the inclusion vector)
8.    Get $\delta_{min}$, the lowest $fit_2$ in $P$
9.    Get $s_{min}$, $fit_1$ of the lowest $fit_2$ in $P$
10.   if $w_1 \geq minsupp$ and $|P| < n$ then
11.      Add new_prefix to $P$
12.      Call PrefixTWEACExt(new_prefix, new_projected_database)
13.   else if ($w_1 \geq minsupp$ and $w_2 < minsupp$) or
14.            ($w_2 \geq minsupp$ and $w_1 \geq \delta_{min}$ and $w_2 < \delta_{min}$) then
15.      storeState($S$,new_prefix, new_projected_database)
16.   else if $w_2 \geq minsupp$ and $w_2 \geq \delta_{min}$ then
17.      if $f_2 > \delta_{min}$ or
18.         ($f_2 = \delta_{min}$ and $w_1 > s_{min}$) or
19.         ($f_2 = \delta_{min}$ and $w_1 = s_{min}$ and new_prefix $\prec p_n$) then
20.         Replace $p_n$ with new_prefix
21.      Call PrefixTWEACExt(new_prefix, new_projected_database)
```

Table 2: PrefixTWEAC Extended: extension using the probable time window

For certain problems we know the approximate area of important features, e.g., protein sequences should have most of the discriminating patterns in the helix. Also in other problems this might be the case, for example — in the case of customer relations — customers tend to behave differently during the night. These *probable time windows* can easily be defined with an *inclusion vector*. An inclusion vector is a vector $v = (v_1, v_2, \ldots, v_n)$, $v_i \in \{0, 1\}$ $(1 \leq i \leq n)$. This vector will indicate where to search in the first phase of the algorithm, see Table 2. We then let

$$supp_1^{\mathrm{PTW}}(s, d) = \begin{cases} 1 & \text{if } s \prec d/v; \\ 0 & \text{otherwise,} \end{cases}$$

where $(d/v)_i = d_i$ if $v_i = 1$ and `$` otherwise (`$` $\notin \Sigma$).

First PrefixTWEACExt (Table 2) is applied to the databases D_t, one at a time, each time starting with an empty $P = P^t$. After using PrefixTWEACExt with the inclusion vector we apply PrefixTWEAC (Table 1) without the vector to the remaining states stored in the state database S.

Figure 1 shows an example of the extensions made to a sequence `A`. The dotted lines are extensions that do not have a high enough *fit₁* and *fit₂* inside and outside the probable time window. These extensions and their extensions are pruned. The dashed lines indicate extensions that are currently good enough with regards to the entire sequence only. Finally the solid lines are already good enough when we only count patterns inside the probable time window.

If we prefer small patterns, then we can add new rules, using so-called *smallest maximal discriminating patterns*:

- $fit_1(s, D_r) = 0$ for all r $(1 \leq r \leq n, r \neq t)$. Then fit_2 of the extended patterns will never increase.

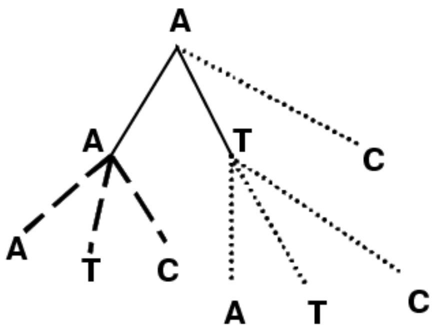


Figure 1: Extending the single item sequence A

- $\mathit{fit}_1(s, D_t) \leq \mathit{fit}_2(p, D_t)$ where both p and s are sequences and s is created by extending p. Then fit_2 of the extended patterns will never be better than the fit_2 of p.

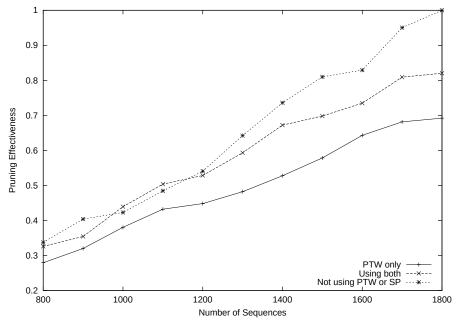
These rules sacrifice some completeness for classification performance; if extensions do not improve a smaller pattern then they are not always explored further. Patterns that normally would have been added are now skipped because the extended pattern doesn't distinguish more sequences. These pruning rules will not lower classification performance because they leave out only non-improving extensions. Rather the classification is expected to improve because the set of patterns will contain less small variations of the same pattern.

Protein sequences usually are very long, about 300 amino acids. However these sequences are constructed out of only 20 types of amino acids. We need to use constraints to make the problem tractable. It was chosen to use the time window constraint, because the discovered patterns will be concentrated in one area. The *time window constraint* means that the distance between the first and last item of the sequence is bounded by some constant. For example if all sequences in a database are equal to (`A`,`C`,`G`,`Q`), and the time window is 2, then (`A`,`Q`) is not a frequent pattern because the distance between the `A` and `Q` in the occurrences is more than 2. This constraint is easily implemented in the algorithm used. It was also considered to use the *gap constraint* [1], that allows some gaps in the matches. However this constraint would have required more memory, e.g., if we count fit_1 of (`A`,`C`,`G`) and we want to know whether the sequence (`A`,`C`,`C`,`C`,`G`) contains it. Furthermore assume the maximal gap is 1, thus in the sequence one letter is allowed between two letters of the pattern. If the algorithm only looks at the first `C` then the gap constraint will be broken because the gap between the `C` and the `G` is 2. An algorithm has to check two `C`'s to match (`A`,`C`,`G`). PrefixSpan will have to add both projections to the projected database for at least two `C`'s. One other reason for not using the gap constraint is that it would allow patterns to be spread all over the sequence as long as it doesn't break the gap constraint.

6 Experimental Results

The experiments are aimed at showing the effectiveness of the pruning rules we described. The protein sequences used during our experiments where extracted from the GPCRDB website [5]. The effectiveness was also tested on a synthetic data set: the two classes consist of 1000 sequences of length 130, having 20 item types. First each item is chosen with a uniform probability and then we insert one of ten patterns at each starting position within the time window (position 20 to 60) of class one with 80% probability.

The results are shown in Figure 2 and Figure 3. All experiments were done on a Pentium 4 2.8 GHz with 512MB RAM. On the horizontal axis in the graphs we have the number of used elements in the data set. As both synthetic and protein data set have two classes, we take half of these elements from the first class and the rest from the second class. On the vertical axis we have the pruning effectiveness indicated by a real number between 0 and 1. This effectiveness is calculated by dividing the search time by the worst search time in the experimental results. During the experiments we searched for the 100 maximal discriminating patterns in the GPCRDB and 10 in the synthetic data set, each with a time window of 8 and a *minsupp* of 0. Note that time window and probable time window are different concepts.



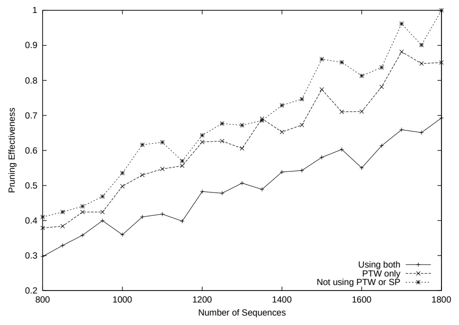


Figure 2: Effectiveness on the GPCRDB data set Figure 3: Effectiveness on the synthetic data set

Figure 2 shows the effectiveness of using probable time windows (PTW) and pruning when using "small patterns" (SP) on the GPCRDB data. Note that SP lowers pruning effectiveness with regards to the GPCRDB data, because less variations of the same pattern fill up the set of patterns. Some of the patterns discovered with this data set were used for classification: two protein families could be correctly distinguished in more than 90% of the cases, depending on the chosen time window size and the classification algorithm at hand.

In the synthetic data set we have most of the best patterns in the probable time window. The n-th pattern p will get a large fit_2 earlier in the search, thus more extensions can be ignored. Figure 3 shows the effectiveness as the number of sequences in the synthetic data set increases when searching for the 10 maximal discriminating patterns. The "small pattern" rules (SP) increase the effectiveness even further, because in the synthetic data set many patterns are quickly non-improving.

	classified as no-olfactory	classified as olfactory
no-olfactory	2015	22
olfactory	16	1909

	classified as no-olfactory	classified as olfactory
no-olfactory	2024	13
olfactory	22	1903

Table 3: Confusion matrices of GPCRDB patterns without (left) and with (right) SP

The confusion matrices of Table 3 were generated using the C4.5 implementation by Weka [11] with the 10 best patterns discovered in the GPCRDB data. If we use SP we get a slightly better classification when doing 10-fold cross-validation: 99.12% instead of 99.04%. This is as expected because the set of 10 patterns used in Table 3 will contain less small variations of the same pattern. Unfortunately the difference in classification performance is not large because in this case only one pattern is different.

7 Conclusion

In this paper we introduced and compared two sequential pattern mining algorithms by using knowledge from the application area of protein sequence analysis. Given some assumptions, we can improve mining for the maximal discriminating patterns. The effectiveness depends on the quality of the assumptions, e.g., how probable a discriminating pattern is within a certain time window. It is shown that using probable time windows in protein sequences can speed up the search. Protein sequences are long but contain only a few types of items; constraints are required to make the discovery of patterns in these sequences tractable.

In future research we will further investigate methods for automatically discovering the probable time window. Furthermore we plan to use maximal discriminating patterns in other application areas like work-flow analysis.

References

[1] Antunes, C., Oliveira, A.L.: Generalization of Pattern-Growth Methods for Sequential Pattern Mining with Gap Constraints. In *Machine Learning and Data Mining in Pattern Recognition (MLDM 2003)*, Lecture Notes in Computer Science 2734, Springer, pp. 239–251.

[2] Agrawal, R., Imielinski, T., Srikant, R.: Mining Association Rules between Sets of Items in Large Databases. In *Proc. of ACM SIGMOD Conference on Management of Data (1993)*, pp. 207–216.

[3] Agrawal, R., Srikant, R.: Mining Sequential Patterns. In *Proc. International Conference Data Engineering (ICDE 1995)*, pp. 3–14.

[4] Bailey, J., Manoukian, T., Ramamohanarao, K.: Fast Algorithms for Mining Emerging Patterns. In *Proc. 6th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2002)*, Lecture Notes in Artificial Intelligence 2431, Springer, pp. 39–50.

[5] *GPCRDB: Information System for G Protein-Coupled Receptors (GPCRs)*, Website `http://www.gpcr.org/7tm/`.

[6] Jovanoski, V., Lavrač̃, N.: Classification Rule Learning with APRIORI-C. In *Proc. 10th Portuguese Conference on Artificial Intelligence (EPIA 2004)*, pp. 44–51.

[7] Kavsẽk, B., Lavrac̃, N., Jovanoski, V.: APRIORI-SD: Adapting Association Rule Learning to Subgroup Discovery. In *Proc. International Symposium on Intelligent Data Analysis (IDA 2003)*, Lecture Notes in Computer Science 2810, Springer, pp. 230–241.

[8] Kramer, S., Raedt, L. De, Helma, C.: Molecular Feature Mining in HIV Data. In *Proc. Conference on Knowledge Discovery in Data (SIGKDD 2001)*, pp. 136–143.

[9] Lesh, N., Zaki, M.J., Ogihara, M.: Mining Features for Sequence Classification. In *Proc. International Conference Knowledge Discovery and Data mining (KDD 1999)*, pp. 342–346.

[10] Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. *IEEE Trans. Knowl. Data Eng.* 16(11) (2004), pp. 1424–1440.

[11] *Weka 3: Data Mining Software in Java*, Website `http://www.cs.waikato.ac.nz/ml/weka/`.

[12] Wess, J.: G-Protein-Coupled Receptors: Molecular Mechanisms Involved in Receptor Activation and Selectivity of G-Protein Recognition, *FASEB Journal* 11 (5) (1997), pp. 346–354.

[13] Zimmermann, A., Raedt, L. De: CorClass: Correlated Association Rule Mining for Classification. In *Proc. International Conference on Discovery Science (DS-2004)*, pp. 60–72.