Building Classifiers from Pattern Teams

Arno Knobbe[1] and Joris Valkonet[2]

[1] LIACS, Leiden University, the Netherlands
a.knobbe@kiminkii.com
[2] Avanade Nederland, the Netherlands
jorisv@avanade.com

Abstract. Over the last decade, a wealth of new pattern discovery methods has been developed. Most methods implicitly assume that the result of the method is a collection of half-products: further processing is needed to turn the set of patterns into an actionable global model, for example a classifier or regression model. Global modeling in this setting entails combining patterns effectively and dealing with possible redundancy or conflicts between the patterns reported. This paper is concerned with the question of turning sets of interesting patterns into classifiers. Specifically, we approach the problem of building classifiers from pattern teams [12], a previously presented framework for selecting small but highly effective subsets of patterns. We not only describe how a single classifier can be constructed from the final pattern team, but also how this procedure can be applied in a wrapper-approach to judge the quality of candidate pattern teams. Involving classifiers in the pattern team discovery process makes sense, because accurate classifiers require informative and non-redundant patterns (features). A number of classification procedures is considered, and compared experimentally. Furthermore, we demonstrate how specific properties of pattern teams, such as the relevance of each member pattern, can be exploited to find global models more efficiently.

1 Introduction

This paper is concerned with the construction of classifiers from sets of previously discovered patterns, more specifically, from *pattern teams* [12, 11]: small collections of predictive patterns that show little redundancy. Why are we interested in building classifiers from pattern teams? The obvious reason is simply for the sake of obtaining the classifier itself. Collections of patterns (such as rules) may be interesting because they represent relevant fragments of knowledge obtained from the data, but they are hard to use in a predictive setting because of overlapping or conflicting conclusions associated with each pattern. Therefore, it is attractive to have a procedure that resolves these conflicts and decides how the patterns collectively lead to a single prediction. This is especially relevant in domains where the initial discovery of patterns is essential, such as in relational data or graphs. A second, more important motivation for this work is related to the discovery of pattern teams. As was demonstrated in [12], one of the most fruitful methods of selecting pattern teams is by means of a wrapper [7, 14]. This means that a given classification procedure is employed to judge the predictive power of candidate pattern subsets, and the pattern team that is returned is

simply the subset that leads to the best performing classifier. Predictive power is a good measure in this setting because it nicely balances a number of desirable qualities of pattern teams. Specifically, predictive power to some degree promotes independence of patterns (and thus suppresses redundancy), while at the same time encouraging the utility of the individual patterns with respect to the classification task. It is for this second motivation that we will be considering different classes of classifiers, and comparing their performance, even though the end product of the process may often not be the actual classifier, but rather the associated pattern team.

Our work fits in the recently published LeGo (*from Local pattErns to Global mOdels*) framework [10]. This framework suggest building global models by first discovering potentially large collections of patterns. These patterns are then filtered for redundancy and relevance in a pattern selection phase and finally combined into a global (often predictive) model. The framework dictates that the three stages are relatively independent. Specifically, the initial pattern discovery phase could be instantiated by any number of local pattern discovery algorithms, without affecting the subsequent stages. When we are interested in a final predictive model however, it makes sense to choose a pattern discovery paradigm that produces supervised patterns, such as Subgroup Discovery or Correlated Pattern Mining [2, 9, 15]. An important notion within LeGo is that patterns are considered as binary features: an example is either covered by a pattern or not. The actual syntactic structure of the patterns is ignored, which makes the framework independent of the selected pattern language (itemsets, fragments of molecules, relational structures, ...). In this interpretation, pattern discovery becomes a form of feature construction, and pattern selection can be thought of as feature selection. More importantly, the binary feature interpretation means that in the global modeling phase, any supervised machine learning algorithm may be employed to combine the patterns (features) into a single model.

Although we can state the problem of building classifiers from pattern teams as a general supervised machine learning problem, the problem has a number of specific properties that justify the attention we give it. First of all, we are dealing with binary features, and a candidate team has only a small number k of these features. Hence, building a classifier for a given team equates to assigning a prediction to each of the 2^k possible combinations of binary values. Furthermore, we know by definition that all patterns in a team are required to be relevant, which in theory limits possible classifiers to those that meaningfully involve all k features. Because of these properties, we will be considering two fairly simple classes of classifiers, which represent two ends of the spectrum of expressive power. On the expressive end, we have *Decision Table Majority* (DTM) classifiers [13, 17], which basically list all contingencies, and assign the associated majority class that is observed in the data. Clearly, this class of classifiers is prone to overfitting, and only works because of the low dimensionality of the problem at hand. On the other end, we have *linear classifiers* [6, 3], which are arguably the

least expressive models imaginable that still involve all features. These classifiers can be expected to trade off expressiveness for generalisation power.

The main contributions of this paper are as follows:

- Defining the task of building classifiers from pattern teams (or in fact pattern sets in general) as a generic induction task over the binary features made up by the patterns. As we pointed out in [10], passing patterns to arbitrary classification procedures is not entirely novel. However, in the context of pattern teams there are specific considerations that lead to new research questions and solutions.
- Experimentally providing answers to two important questions related to the successful application of the presented approach:
  - What choice of classifier achieves the highest predictive accuracy, and consequently leads to the most informative pattern team? Specifically, we compare a number of classes of classifiers that range in representational power.
  - What choice of pattern selection measure leads to optimal results? We consider both wrapper and filter-based quality measures.
- Showing how three unique properties of pattern teams can be exploited to arrive at relatively efficient solutions for finding pattern teams. These properties relate to patterns being binary in nature, pattern teams being concise, and finally, to pattern teams being non-redundant, requiring each pattern to be relevant.
- Showing experimentally that classifiers built from pattern teams can compete with more traditional classification techniques, and furthermore, that they outperform classifiers induced on the whole pattern set.

In the next section, we introduce notation and terminology, and formally define the problem addressed, as well as the different classes of classifiers considered. In Section 3, we consider one of the classifiers in particular, and show how specific properties of pattern teams may be exploited to arrive at an efficient implementation of this classifier. In Section 4, we experimentally compare different combinations of quality measures for pattern teams and classifiers. Furthermore, we compare the resulting classifiers with the performance of existing classes of classifiers. Finally, a discussion and conclusions are given in Section 5.

2 Preliminaries

We start by providing a number of basic definitions related to patterns and pattern team discovery. We assume that our database d is a bag of labelled objects $i \in D$, referred to as *individuals*, taken from a domain D. Furthermore, there is a function $l : d \rightarrow \{0, 1\}$ that specifies the label of an individual. We refer to the size of the database as $N = |d|$.

We assume nothing about the syntax of the pattern language, and treat a pattern simply as a function $p : D \rightarrow \{0, 1\}$. We will say that a pattern p *covers* an individual i *iff* $p(i) = 1$. A *subgroup* $S(d, p)$ implied by a pattern is now simply

the set of individuals $i \in d$ that are covered by p: $S(d,p) = \{i \in d | p(i) = 1\}$. For brevity we will omit the d from now on. $s(p) = |S(p)|$ refers to the size of the subgroup implied by p. Furthermore, we will use expressions like $l(i) = 1$ to denote patterns related to the label of individuals, such that $S(l(i) = 1)$ for example denotes the set of positive cases.

When talking about sets of patterns $P = \{p_1, \ldots, p_k\}$ of size k, an individual may be covered by some patterns in P and not by others. In order to represent such *contingencies*, we introduce *codes* $c \in \{0,1\}^k$. Clearly, for a set of k patterns, there are 2^k possible different codes. The subgroup implied by a given pattern set P and a code c is defined by

$$S(P,c) = \{i \in d | p_1(i) = c_1, \ldots, p_k(i) = c_k\}. \quad (1)$$

$s(P,c) = |S(P,c)|$ is the size of the subgroup implied by P and c.

As mentioned, we assume that an initial mining process has produced a set of interesting patterns $\mathcal{P}$. The elements of $\mathcal{P}$ have been selected on the basis of their individual merits (using some quality measure such as frequency, χ^2, or novelty). In the subsequent pattern team discovery phase, we try to find a subset of $\mathcal{P}$ that is informative and non-redundant at the same time. In order to judge candidate pattern sets and select the optimal pattern team, we define a quality measure for pattern sets that promotes important qualities of a pattern team. [12] presents a number of useful quality measures.

Definition 1 (Quality Measure). *A quality measure for pattern sets is a function $\Phi_d : 2^{\mathcal{P}} \rightarrow \mathbb{R}$, that computes a unique numeric value for a pattern set P, given a database d.*

Definition 2 (Pattern Team Discovery). *Given a set of interesting patterns $\mathcal{P}$, and a quality measure for pattern sets $\Phi_d : 2^{\mathcal{P}} \rightarrow \mathbb{R}$, find a pattern set $P \subseteq \mathcal{P}$ of size k such that $\Phi_d(P) \geq \Phi_d(Q)$ for all $Q \subseteq \mathcal{P}$ of size k.*

Finding a pattern team of size k for any given quality measure Φ_d potentially involves the consideration of $\binom{n}{k}$ subsets of the set of interesting patterns $\mathcal{P}$, where $n = |\mathcal{P}|$. In fact, Mielikäinen et al. [16] show that the general Pattern Team Discovery problem is NP-hard by relating it to the set-covering problem, making it infeasible for all but small values of k. Fortunately, for specific quality measures, it is possible to find optimal pattern sets efficiently, or to find approximations that can be shown to perform reasonably well. For example, in [11], we provide some algorithms for *joint entropy*.

Example 1. Let us consider a database of 7 molecules labelled a, ..., g, and assume we have discovered the following four promising patterns.

id	p_1	p_2	p_3	p_4	class
a	0	0	0	1	0
b	1	0	0	1	0
c	0	1	0	0	0
d	0	0	1	0	1
e	1	0	0	0	1
f	1	1	1	1	1
g	1	1	0	1	1

Clearly, $S(p_1) = \{b, e, f, g\}$ and $s(p_1) = 4$. Assume we are interested in pattern teams of size $k = 2$. This leads to the following possible codes: (0,0), (0,1), (1,0), and (1,1). The pattern set $P = \{p_1, p_2\}$ separates the database into the following four subgroups: $S(P, (0, 0)) = \{a, d\}$, $S(P, (0, 1)) = \{c\}$, $S(P, (1, 0)) = \{b, e\}$, and $S(P, (1, 1)) = \{f, g\}$. There are six candidate pattern sets that need to be considered: $\{p_1, p_2\}$, $\{p_1, p_3\}$, $\{p_1, p_4\}$, $\{p_2, p_3\}$, $\{p_2, p_4\}$, and $\{p_3, p_4\}$.

2.1 Classifiers

Once a pattern team P has been selected, it can be turned into a global model using a *classifier*. In the context of this paper, we define a classifier as a function $f : \{0, 1\}^k \rightarrow \{0, 1\}$. Note that f assigns a 0 or 1 to each of the 2^k possible codes. We can completely characterise a classifier by listing the 2^k assignments. Such a vector $v \in \{0, 1\}^{2^k}$ will be referred to as a *configuration*. There are 2^{2^k} distinct configurations for a given k, and hence 2^{2^k} possible classifiers. Because f will typically depend on the set of patterns P, we will often write f_P. If we apply a classifier to an individual i in the database, we will simply write $f_P(i)$ to denote the result of f applied to the binary vector $(p_1(i), \ldots, p_k(i))$.

In this paper, we do not only employ classifiers in order to turn the final pattern team into a predictive model. Rather, we involve the classifier in the Pattern Team Discovery phase, by embedding it in a *wrapper approach* [7, 14]. This means that we will be using the predictive accuracy of a classifier induced from a candidate pattern set P to judge the quality of P. There are various ways of estimating the predictive accuracy of a classifier (e.g. cross-validation), but for reasons of efficiency, we will simply be using its *purity*: the quality of the classifier on the training set. Assuming we have selected a process for inducing a classifier f_P from a given set P, we can simply define our quality measure as follows:

$$\Phi(P) = |\{i \in d | f_P(i) = l(i)\}|/N \tag{2}$$

Linear classifiers The simplest classifier we will be considering is the linear classifier. A linear classifier determines its prediction by computing the weighted sum of the features involved. If this sum plus a bias value b is positive, then the prediction is 1, and 0 otherwise. This binary decision can be interpreted as a

hyperplane in the feature space.

$$f_P(c) = \begin{cases} 1 & \text{if } \sum_j (w_j \cdot c_j) + b \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

The weights w_j can be positive or negative, and are typically determined from the data by means of some optimisation procedure. A well-known procedure for computing such weights is the *Support Vector Machine* (SVM) [3, 18]. This procedure attempts to find a hyperplane (i.e. weight vector) that separates the positive (1) from the negative (0) cases, such that the *margin* between these classes is optimised. For datasets that cannot be separated perfectly, an optimisation criterion that balances margin width and misclassification has been defined. Because of this maximal margin (and the relatively simple model), the SVM has shown good generalisation performance, and therefore good predictive accuracy on unseen data. In order to induce a linear SVM, we employ the relatively efficient Sequential Minimal Optimisation (SMO) procedure [18]. Although SVMs, and SMO in particular are able to work with non-linear decision boundaries by using non-linear kernels, we only use linear models in order to obtain a classifier of low expressive power and to prevent overfitting.

Assuming the wrapper-approach, the quality of a pattern set P simply becomes the purity of the induced SVM $f_{P,SVM}$, which we will refer to as SVMp:

$$\text{SVMp}(P) = |\{i \in d | f_{P,SVM}(i) = l(i)\}|/N \quad (4)$$

Apart from the wrapper approach, we will also consider an alternative quality measure that uses properties of the hyperplane and its margin. If SVMs attempt to optimise a margin-based function (for a given pattern set), it makes sense to adopt this optimisation function for comparisons between pattern sets also. In this setting, we will hence select the pattern team that leads to a set of features that allows the widest margin. The following quality measure is hence directly inspired by the usual SVM optimisation function [3]. The minus changes the minimisation into maximisation for pattern sets. The first term is derived from the margin-width, which is maximised. The second term is a penalty for individuals that are misclassified.

$$\text{SVMq}(P) = -(\sum_j (w_j \cdot w_j) + C \sum_i \epsilon_i) \quad (5)$$

The SVM classifier combines the advantages of a relatively simple (and hence general) model and margin maximisation. Unfortunately, this second goal comes at a computational cost. It is fair to wonder whether the notion of margin maximisation actually provides any additional benefit in our specific setting of building classifiers from pattern teams. In order to test this, we consider an additional linear classifier that simply optimises the purity (i.e. minimises the misclassifications on the training set), while ignoring any margin considerations. We simply refer to this classifier as LC, with the associated purity-based quality measure LCp. For now, we assume that we can compute such an optimal linear classifier

relatively efficiently. In Section 3, we will consider LC specifically, and provide a method that effectively exploits a number of the peculiarities of pattern teams to deal with this issue.

Decision Table Majority classifier The second class of classifiers considered is the *Decision Table Majority* classifier [12, 13, 17], also known as a *simple decision table*. The idea behind this classifier is to build from the pattern set a contingency table for each possible code, and compute the relative frequency of positive cases for each contingency. For contingencies that do not appear in the database, the relative frequency of positive cases is based on that of the whole database (i.e. the prior). An individual is now classified by computing its code, and returning the majority class within the associated subgroup. This simple approach works surprisingly well, under two conditions: the features (i.e. patterns) have a low cardinality, and the decision table should be based on a relatively small number of features selected from a larger set by means of a wrapper [7]. These conditions clearly hold for our application. The following definition captures the workings of a DTM classifier. The function t computes a conditional probability estimate for $l(i) = 1$ for a code c, given a set of patterns P:

$$t_P(c) = \begin{cases} \frac{|S(P,c) \cap S(l(i)=1)|}{s(P,c)} & \text{if } s(P,c) > 0 \\ \frac{s(l(i)=1)}{N} & \text{if } s(P,c) = 0 \end{cases} \tag{6}$$

$$f_{P,DTM}(c) = \begin{cases} 1 & \text{if } t_P(c) \geq 1/2 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Analogous to SVMp, we define the DTM-based quality measure as the purity of the induced DTM classifier $f_{P,DTM}$, which we will refer to as DTMp:

$$\text{DTMp}(P) = |\{i \in d | f_{P,DTM}(i) = l(i)\}|/N \tag{8}$$

Example 2. Consider again the database from Example 1. For pattern sets of size $k = 2$, the $2^2 = 4$ contingencies lead to $2^{2^k} = 16$ configurations: (0,0,0,0), (0,0,0,1), ..., (1,1,1,1), each of which forms a potential classifier. Consider the pattern set $P = \{p_1, p_2\}$ and the classifier (0,0,0,1), which classifies a molecule as positive iff both p_1 and p_2 hold. Clearly, $f_P(a) = 0$ and $f_P(g) = 1$. The purity of this classifier is

$$\Phi(P) = |\{a, b, c, f, g\}|/7 = 5/7 \approx 71.4\%$$

The quality measure DTMp leads to the pattern team $\{p_2, p_4\}$, with the associated classifier (1,0,0,1), which classifies positively iff p_2 and p_4 have the same value. This classifier is 100% pure. The quality measure SVMp (and LCp) leads to the pattern team $\{p_1, p_3\}$. The associated linear classifier is as follows

$$3 \cdot p_1 + 4 \cdot p_2 - 2 \geq 0$$

which is equivalent to (0,1,1,1). This leads to a purity of 85.7%.

dimensions	configurations	linear decision functions	hyperplanes	relevant hyperplanes
1	4	4	1	1
2	16	14	6	4
3	256	104	51	36
4	65,536	1,882	940	768
5	$4.29 \cdot 10^9$	94,572	47,285	43,040
6	$1.84 \cdot 10^{19}$	$1.50 \cdot 10^7$	7,514,066	
7	$3.40 \cdot 10^{38}$	$8.38 \cdot 10^9$	$4.19 \cdot 10^9$	

Table 1. Numbers of hyperplanes.

3 Hyperplane Enumeration

In the previous section, the LC classifier was introduced as one of the possible classifiers. There remains the issue of finding the optimal linear classifier efficiently. In this section, we will present a method for doing so that exploits a number of important characteristics of the pattern team setting. The first characteristic is related to the small number of patterns found in a typical pattern team. The second characteristic has to do with the binary nature of patterns. Whereas in general, a linear classifier is defined as a linear hyperplane in Euclidean space, in our application we only deal with examples from the space $\{0,1\}^k$, which effectively reduces the number of candidate classifiers that need to be considered.

Infinitely many different hyperplanes will map to a single linear decision function. Two hyperplanes map to the same linear decision function if none of the corners of the hypercube defined by the patterns lies between the two hyperplanes. Therefore, it makes sense to only consider the finite number of linear decision functions possible for a hypercube of given size k, rather than using some time-consuming convergence process that attempts to find an optimal hyperplane. If k is small, as is the case in our setting, this number of linear decision functions may be reasonably small.

So how many unique linear decision functions actually exist for a given hypercube? This question has been addressed by Oswin Aichholzer [1]. In Table 1 (third column), we list the number of linear decision functions for $k = 1$ to $k = 7$. For example, for $k = 2$, we see that there exist $2^{2^k} = 16$ configurations, 14 out of which are linear. The two remaining configurations correspond to the two instances of the XOR function, which are clearly not linearly separable. The fourth column is concerned with the number of hyperplanes that need to be considered as candidate linear classifiers. This number does not include the two linear decision functions that lie outside the hypercube, and hence do not actually decide between 0 and 1. Furthermore, for all pairs of decision functions that swap 0's and 1's, we only need to consider one hyperplane. Hence, for h hyperplanes, we have $2h + 2$ linear decision functions.

For a further reduction in the number of hyperplanes to consider, we use a third property of pattern teams. By definition, we require each pattern in the

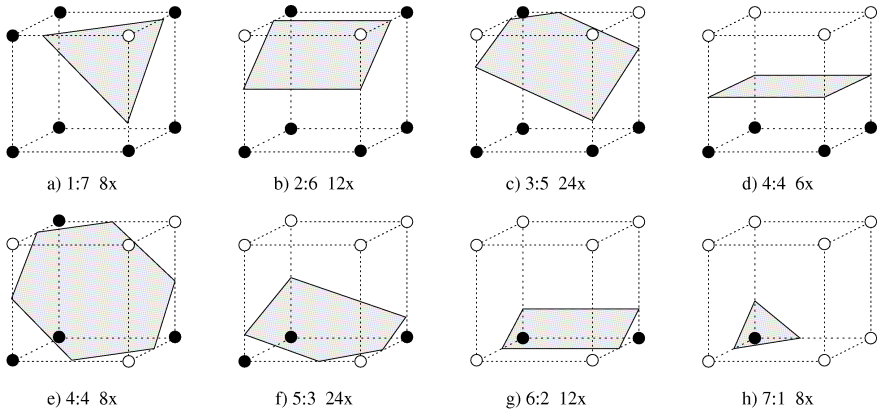


Fig. 1. Hyperplanes for k=3 (courtesy of O. Aichholzer).

team to contribute something to the decision. That is, we require a pattern to be *strongly relevant* within the team: for at least one example in the dataset, this is the only pattern in the team that decides between a positive and negative prediction [14]. In terms of hyperplanes, a pattern is irrelevant if the hyperplane is parallel to at least one of the sides of the hypercube[3]. Consider Figure 1, which shows all hyperplanes (modulo rotation and mirroring) organised by the number of positive and negative contingencies (p:n). For example, the first diagram a) shows the class of hyperplanes that has only a single positive contingency. Clearly, the hyperplane is not parallel to any of the sides of the hypercube, and hence all patterns are relevant. On the other hand, diagrams b), d) and g) have one, two and one irrelevant patterns, respectively. It should be noted that in theory the (optimal) pattern team could contain only a single strongly relevant pattern. One could argue that in this case, only a single pattern should be returned, and hence all candidate hyperplanes should be considered. If on the other hand, one is strictly interested in teams of size k, then only the relevant hyperplanes need to be considered, with a moderate advantage in computation time.

In the fifth column of Table 1 (our contribution), we list the number of hyperplanes that do not contain any irrelevant patterns.[4] As can be seen from Table 1, both the number of hyperplanes and the number of relevant hyperplanes grow rapidly with increasing k. These numbers are however considerably smaller than the total number of possible classifiers (configurations). Assuming that pattern teams are typically very small collections of patterns, just enumerating and testing all (relevant) hyperplanes may be feasible, and preferable over computing

[3] To be more precise, a pattern is irrelevant if at least one of the associated hyperplanes is parallel to at least one of the sides of the hypercube.

[4] Lists of all hyperplanes and relevant hyperplanes for $k \leq 5$ can be obtained from the authors, in flat file or XML.

dimensions	hyperplanes	relevant hyperplanes	SMO, WDBC	SMO, Ionosphere
2	6	4	4,218	15,149
3	51	36	29,141	6,610
4	940	768	10,704	56,026
5	47,285	43,040	24,109	44,245
6	7,514,066		20,114	39,522

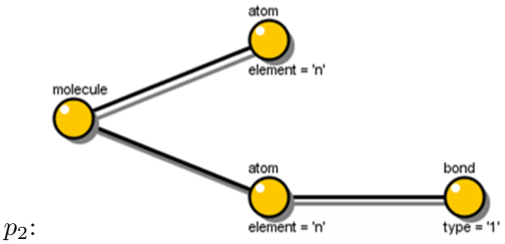
Table 2. Relevant hyperplanes compared to typical number of iterations for two UCI datasets.

hyperplanes using some convergence process such as SMO. We have performed a number of experiments to test typical numbers of iterations (and hence scans of the data) involved with the SVM implementation we have been using. Table 2 compares the number of table scans needed with the proposed enumeration approach compared to SVMs computed on two UCI datasets (see next section on experiments). As can be seen, hyperplane enumeration is advantageous for $k \leq 4$, which is a small but very reasonable number for our pattern team setting.

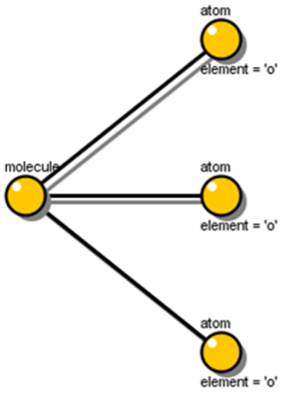
4 Experiments

In this section, we present a number of experiments that examine the effectiveness of the different classifiers and quality measures for pattern teams. We start off by an informal demonstration of our method on a database of molecules. We then proceed with a systematic comparison on a number of well-known datasets. The presented methods have been implemented in the Safarii Data Mining environment [9], which already features a Subgroup Discovery method for producing the initial set of patterns, as well as the necessary pattern team machinery. The system is able to find interesting patterns in complex data including graph and multi-relational data. As a demonstration of our methods on such data, we now consider the well-known Mutagenesis database [19]. This database describes 188 molecules falling in two classes, mutagenic (66.5%), and non-mutagenic. The structural description consists of the atoms and the bonds that make up the compound. In particular, the database consists of 3 tables that describe directly the graphical structure of the molecule (molecule, atom, and bond). In the pattern discovery phase, we look for fragments of the molecules that are indicative of mutagenicity. Fragments were selected using the Novelty evaluation measure (also known as *weighted relative accuracy*). From the initial set of 50 fragments discovered, we select pattern teams of size 3. Using the DTMp quality measure, the following pattern team is produced:

p_1:



p_2:



p_3:

Each so-called selection graph represents a fragment, where the graphical structure of the graph represents constraints on the molecules in terms of atoms and bonds. We are using the *object identity* interpretation, which means that different nodes in the selection graph are required to map to different parts (atoms, bonds) of the molecule. Therefore, for example, p_3 should be interpreted as molecules containing at least three oxygen-atoms. Note that because of the structural nature of the data, standard attribute-value systems cannot be applied directly.

The patterns represents subgroups of the dataset of size 126, 58 and 88, with novelties 0.054, -0.046 and 0.035, respectively. Note that p_1 and p_3 are positive indicators of mutagenicity, and p_2 is negative. Although these fragments appear to be relatively basic, further analysis shows that they represent important classes of molecules. For example, the majority of molecules in p_1 contain a *phenyl*-group, which is an aromatic ring of six carbon atoms attached to the remainder by a single bond. The carbon atom and aromatic bond (*type* = '7') mentioned in p_1 apparently are enough to recognize such a functional group. Further mining on p_2 shows that most of these molecules are *amines* (contain a nitrogen atom connected with three single bonds), and additionally contain a *nitroso*-group ($-$N=O). Within most of the molecules in p_3, the first oxygen atom turns out to be part of a nitroso-group connected to an aromatic ring. The location of the second oxygen is not specified, but it is connected via a double bond (O=), as is the case in for example esters and ketones, but not in ethers and alcohols ($-$O$-$). The third oxygen atom appears in many different roles in p_3. It is important to note that there is overlap between the three patterns, for example because they either share an aromatic ring (p_1 and p_3) or a nitroso-group

(p_2 and p_3). Whereas the presence of a nitroso-group is a negative indicator for mutagenicity (p_2), in combination with an aromatic ring and more oxygen atoms (p_3), the molecules are more likely mutagenic. This interaction between patterns demonstrates the usefulness of having classifiers built from pattern teams, as the class of a molecule depends on its code based on the three patterns.

The three patterns lead to the following decision table (with supports for individual contingencies indicated). The associated classifier has a purity of 74.5%. For a cross-validated score, the process would have to be repeated of course, resulting in slight variations in patterns and pattern teams produced.

p_1	p_2	p_3	support	class
0	0	0	22	1
0	0	1	21	1
0	1	0	15	0
0	1	1	4	0
1	0	0	47	1
1	0	1	40	1
1	1	0	16	0
1	1	1	23	1

The SVMq measure leads to the following linear classifier, with a purity of 72.3%. Note the negative weight of p_2:

$$p_1 - 0.96 \cdot p_2 + 0.85 \cdot p_3 + 0.83 \geq 0$$

quality measure		joint entropy	BDeu	DTMp		SVMp		SVMq	LCp
classifier	default	DTM	DTM	SVM	DTM	SVM	DTM	SVM	LC
Chess (KRvsKP)	52.0	81.2	92.8	80.3	**92.9**	89.7	90.4	77.4	92.7
Ionosphere	64.1	85.5	89.7	88.0	89.7	85.2	**90.3**	73.5	88.3
Pima	65.0	71.3	**75.0**	70.0	**75.0**	73.2	71.9	69.4	74.2
TicTacToe	65.3	66.3	72.4	62.6	72.2	70.7	**72.8**	68.0	71.1
WBCD	65.5	93.8	93.8	90.0	93.8	92.7	93.8	86.7	**94.4**
WDBC	62.0	85.4	94.4	93.5	**94.7**	94.5	94.5	88.8	92.4
Credit Screening	55.5	82.8	82.3	82.0	84.1	82.6	82.6	82.3	**84.2**

Table 3. Cross-validated results using Novelty.

In the following experiments, we compare the performance of the different classifier classes in terms of their predictive accuracy. For our experiments, we have selected a number of datasets from the well-known UCI repository. All selected datasets have 2 classes. The initial set of patterns was produced by running Subgroup Discovery using beam-search with moderate search parameters, such that pattern sets of approximately 50 patterns were produced. Again, patterns were selected using the Novelty evaluation measure. In order to obtain reliable

dataset	default	best pattern team	SD + SVM	J48	Neural Network	PART
Chess (KRvsKP)	52.0	92.9	85.4	**99.3**	99.1	98.9
Ionosphere	64.1	**90.3**	84.6	89.7	**90.3**	90.0
Pima	65.0	**75.0**	68.9	71.2	74.3	74.2
TicTacToe	65.3	72.8	63.8	**98.3**	97.1	93.7
WBCD	65.5	94.4	89.3	94.3	94.3	**94.7**
WDBC	62.0	**94.7**	89.8	93.3	93.3	93.1
Credit Screening	55.5	84.2	81.9	**85.8**	83.3	84.8

Table 4. Pattern team-based results compared to other methods.

estimates of the predictive accuracies, the whole process (including Subgroup Discovery) was embedded in a 5-fold cross-validation procedure. In Table 3, we show predictive accuracies for a classifier built from the pattern team obtained. All numbers represent predictive accuracies using cross-validation in percentages. For comparison, the second column ("default") provides the frequency of the majority class as a baseline.

Separate experimentation (to be published elsewhere) has shown that, depending on the extent of the Subgroup Discovery phase, k should preferably be 3 or 4. Although at first glance, this appears to too small a number, it should be noted that 3 or 4 patterns can generate fairly complex models. For example, assuming $k = 3$, a model can be constructed that is equivalent to a decision tree of 15 nodes (7 internal and 8 leafs). Furthermore, each of these three patterns may be rather complex, and involve multiple attributes of the original data. For a typical size of 3 conditions per pattern, the global model conceivably could involve up to $3 * 3 = 9$ attributes. The experimentation has shown that with larger pattern teams, the risk of overfitting due to this model complexity hinders optimal scores for the classifier.

For these reasons, we continue our experiments with pattern teams of size 3. The first row of Table 3 indicates the quality measure used to discover pattern teams. The second row indicates the classifier that was used to obtain the (cross-validated) predictions using the final pattern team. Note that this classifier is not necessarily the same classifier as used in the wrapper approach (specifically for DTMp and SVMp). Furthermore, for the quality measures that do not employ a wrapper approach (joint entropy and BDeu), the DTM classifier was used. Joint entropy (defined in [12]) is an unsupervised measure based on information theory. It favours independent patterns, and hence balanced contingencies. BDeu (Bayesian Dirichlet equivalent uniform) is a measure from Bayesian theory [8] that, in this context, can be interpreted as an estimate for the performance of a DTM classifier (without actually computing the DTM) with a penalty for small contingencies that may lead to overfitting. Winners (and ties) per dataset are indicated by bold typeface.

On the whole, the DTMp/DTM combination seems to produce the most informative pattern teams, with optimal scores being achieved in almost half

of the datasets. Its average rank over all datasets is 2.07. Not surprisingly, the DBeu/DTM combination shows very similar results. Interestingly however, the additional feature of avoiding overfitting in BDeu does not appear to lead to a better generalisation capability. The SVMp/DTM combination performs slightly worse than DTMp/DTM, although it is still responsible for two optimal pattern teams out of seven. On average, SVMp/DTM and LCp/LC perform almost equally well, although the difference in performance ranges from -2.3% to 2.1%, depending on the dataset. It is interesting to note that SVMp/SVM almost consistently performs worse than SVMp/DTM. Apparently, the SVMp quality measure is able to select the right patterns, but in the end, a more expressive classifier is needed to fully exploit these patterns. To our surprise, the SVMq/SVM consistently is among the worst results of the wrapper-based combinations (average rank 7.36). Apparently, optimising the margin over all pattern subsets does not select a predictive combination of patterns. As a general trend, the average rank of the different combinations appears to coincide with the level of purity (in contrast to margin optimisation) of the quality measure/classifier combination: DTMp/DTM (average rank 2.07), SVMp/DTM (3.07), LCp/LC (3.14), SVMp/SVM (4.86), SVMq/SVM (7.36), from pure to optimal margin.

On the whole, the joint entropy/DTM combination performs relatively badly. This should come as no surprise, as joint entropy is an unsupervised measure.

For a more rigorous comparison of the performance of the different combinations, we have performed a procedure outlined by [5]. A Friedman test performed on the average ranks per combination (columns in Table 3) shows that the differences in performance are significant ($p = 3.2 \cdot 10^{-7}$). A post-hoc Nemenyi test shows that the critical distance in ranks is 3.97 at a 5% significance level. As a result, one can say that the differences between the top 6 ranking combinations are not significant. However, DTMp/DTM performs significantly better than both DTMp/SVM and SVMq/SVM. Furthermore, DBeu/DTM, SVM/DTM and LCp/LC significantly outperform SVMq/SVM.

We would like to repeat that building a classifier with a high classification accuracy is not our goal per se, although a good classification score is a good indicator for having selected informative patterns. Still, one may wonder how well our pattern team-based approach compares to existing classification procedures, as well as to building classifiers from *all* patterns rather than a small subset. In Table 4, we provide a comparison between the pattern team results and those obtained using a number of reference classification procedures (as well as the majority class baseline). The fourth column presents results obtained by applying an SVM to all patterns produced by Subgroup Discovery (rather than to subsets of size 3). Here, the same SMO implementation was used as in the pattern team approaches. The last three columns present results produced using three well-known procedures available through the Weka package. Note that these algorithms were run on the data directly (i.e. not on the pattern set). It is important to note here that the last three columns can only be computed thanks to the propositional nature of the selected UCI datasets. With structured datasets,

such as multi-relational or graphical ones, the reference algorithms could not have been applied.

Comparing the third and fourth column of Table 4, we observe that the best pattern team ($k = 3$) always outperforms an SVM induced from all patterns discovered (approx. 50). This even holds when comparing with just the SVMp/SVM results (not shown in this table). The fact that better results can be obtained using only three, easily inspectable, patterns than using all patterns discovered, is clear support for our pattern team approach. Comparing the five approaches listed in Table 4, we note that the classifier based on the best pattern team can compete with the other four methods (ranking first on three datasets). In fact, our approach (average rank 2.36) slightly outperforms the three reference methods (2.43, 2.43, 2.71, respectively), but not significantly so. SD + SVM consistently performs the worst (5.0). The two datasets where pattern teams do not perform well are Chess and TicTacToe, which both relate to board games. These datasets tend to require the combination of many low frequency or low Novelty patterns. It might well be that the required patterns do not appear in the first 50-odd patterns produced during Subgroup Discovery. Again, the comparison with existing methods is only provided for illustration, as our main purpose is selecting actionable patterns rather than building optimal classifiers.

5 Discussion and Conclusion

In this paper, we have presented a number of methods for building classifiers from pattern teams. The different methods have been compared experimentally, using a number of UCI dataset. In terms of the predictive quality of these methods, the experiments show a tendency towards purity, meaning that methods with high expressive power and a disregard for the risk of overfitting perform well. In our experiments, the model with the highest expressiveness, the decision table, tends to outperform models of lesser expressive power (the different linear classifiers). Furthermore, within the linear models, the simple LC classifier, which optimises purity, almost consistently outperforms the Support Vector Machine. Although these methods use the same linear model, the margin-based generalisation facilities of the SVM seem to be an obstacle for optimal performance in this case. Although the LC classifier was initially chosen as a relatively efficient alternative to SVMs for small k, it also seems to be preferable from an accuracy standpoint. Note that this makes two of the best performing classifiers, DTM and LC, also the most efficient ones.

The tendency towards purity is surprising, as generally avoiding overfitting is an important issue in classification tasks. It should be noted that we have only considered pattern teams of relatively small size. Clearly, the risk of overfitting when combining only 3 patterns is relatively minor, and secondary to the benefit of optimal purity. With larger numbers of patterns involved, one can expect this trade-off between purity and generality to shift more towards the latter. Especially the decision tables, which grow exponentially in size with the number

of dimensions, will likely break down in this case, although the risk of overfitting can be somewhat avoided by picking redundant patterns in the team.

In our experiments, we have primarily considered the classification score of the different methods, which might give the impression that predictive accuracy is the primary, or in fact only virtue when building classifiers from pattern teams. Although having accurate models is important, we would like to stress again the value of having a small set of informative patterns, and having the assurance that these few patterns contribute optimally to the prediction task at hand. In many cases, the predictive setting will only be used by a domain expert to choose those patterns that matter.

One issue that was not considered in great detail is the computational cost of our approach. For all of our classifiers, the running times will significantly increase with larger values of k. However the biggest influence on the overall efficiency is the exponential nature of the pattern team discovery process. In the context of this paper, we have assumed this process to be exhaustive, which amounts to testing all $\binom{n}{k}$ possible pattern teams of size k, and building a classifier for each. Although the pattern team framework is clearly intended for small teams, a more scalable consideration of candidate teams is desirable. In [11], a number of methods were presented for pruning the candidate space, by exploiting properties of the quality measure in question (joint entropy). This paper also described an approximate search algorithm that greedily adds patterns, starting from the empty set. For each addition, the pattern that improves the quality of the team the most is selected, resulting in a $O(kn)$ time complexity. For joint entropy, it turns out that this 'quadratic' forward selection results in a near-optimal pattern team. Cheng *et al.* [4] demonstrate that such a greedy approach can produce positive results. As an alternative, [20] presents a 'linear' method that picks the relevant patterns in $O(n)$ steps. This method assumes that the patterns are ordered, for example by support or complexity of the pattern, before starting the actual selection. This sorting can of course be done in $O(n \lg n)$ time.

The success of forward selection processes (either linear or quadratic) highly depends on the quality measure chosen. Unfortunately, in the case of supervised quality measures (as opposed to joint entropy), forward selection methods are less likely to find near-optimal pattern teams. One can easily construct examples where the greedy method would completely ignore patterns that lead to an optimal team. Think for example of two patterns that constitute an XOR-problem. Of course, one could argue that this will not be a problem for some of the classifiers we use, which cannot deal with the XOR-problem any way. Note that the obvious alternative, backward selection (repeatedly removing patterns from the complete set), would not be an option as most of the chosen classification procedures only work for small sets of patterns. In further work, we intend to investigate different heuristic pattern selection procedures that could provide efficient approximations of pattern teams.

References

1. O. Aichholzer and F. Aurenhammer. Classifying hyperplanes in hypercubes. *SIAM Journal of Discrete Mathematics*, 9(2):225–232, 1996.
2. B. Bringmann, A. Zimmermann, L. De Raedt, and S. Nijssen. Don't be afraid of simpler patterns. In *Proceedings PKDD'06*, pages 55–66, Berlin, Germany, 2006. Springer-Verlag.
3. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
4. H. Cheng, X. Yan, J. Han, and C.-W. Hsu. Discriminative frequent pattern analysis for effective classification. In *ICDE*, pages 716–725, 2007.
5. J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
6. R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley, 2001.
7. I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.
8. D. Heckerman, D. Geiger, and D. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:179–243, 1995.
9. A. Knobbe. Safarii multi-relational data mining environment. http://www.kiminkii.com/safarii.html, 2006.
10. A. Knobbe, B. Crèmillieux, J. Fürnkranz, and M. Scholtz. From local patterns to global models: The LeGo approach to data mining. In *Proceedings LeGo'08 workshop at ECML PKDD'08*, pages 1–16, http://www.ecmlpkdd2008.org/files/pdf/workshops/ lego/1.pdf, 2008.
11. A. Knobbe and E. Ho. Maximally informative k-itemsets and their efficient discovery. In *Proceedings KDD'06*, pages 237–244, Philadelphia, PA, 2006.
12. A. Knobbe and E. Ho. Pattern teams. In *Proceedings PKDD'06*, pages 577–584, Berlin, Germany, 2006. Springer-Verlag.
13. R. Kohavi. The power of decision tables. In *Proceedings ECML '95*, pages 174–189, London, UK, 1995. Springer-Verlag.
14. R. Kohavi and G. John. The wrapper approach. In *Feature Extraction, Construction and Selection: a data mining perspective*, pages 33–50. Kluwer Academic Publishers, 1998.
15. N. Lavrac. Subgroup discovery techniques and applications. In *Proceedings PAKDD'05*, pages 2–14, 2005.
16. T. Mielikäinen and H. Mannila. The pattern ordering problem. In *Proceedings PKDD'03*, pages 327–338, Cavtat-Dubrovnik, Croatia, 2003. Springer-Verlag.
17. B. Pfahringer. Compression-based feature subset selection. In *Proceedings IJCAI'95*, Vienna, 1995.
18. J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods: support vector learning*, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.
19. A. Srinivasan, S. Muggleton, and R. King M. Sternberg. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1,2), 1996.
20. A. Zimmermann, B. Bringmann, and L. De Raedt. The chosen few: On identifying valuable patterns. In *Proceedings ICDM'07*, 2007.