

From Local Patterns to Classification Models

Björn Bringmann, Siegfried Nijssen, and Albrecht Zimmermann

Abstract Using pattern mining techniques for building a predictive model is currently a popular topic of research. The aim of these techniques is to obtain classifiers of better predictive performance as compared to greedily constructed models, as well as to allow the construction of predictive models for data not represented in attribute-value vectors. In this chapter we provide an overview of recent techniques we developed for integrating pattern mining and classification tasks. The range of techniques spans the entire range from approaches that select relevant patterns from a previously mined set for propositionalization of the data, over inducing pattern-based rule sets, to algorithms that integrate pattern mining and model construction. We provide an overview of the algorithms which are most closely related to our approaches in order to put our techniques in a context.

1 Introduction

In many applications rule-based classification models are beneficial, as they are not only accurate but also interpretable. Depending on the application, these rules are *propositional*, i.e. of the kind

if income of a customer **is** high *and* loans **is** low
then predict the customer is good,

or *relational* or *graph based*, i.e. of the kind

if carbon **is connected to** a nitrogen in a molecule
then predict the molecule is active.

Björn Bringmann · Siegfried Nijssen · Albrecht Zimmermann
e-mail: `firstname.lastname@cs.kuleuven.be`. Department of Computer Science,
Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Leuven, Belgium

Finding such rules is a challenge for which many algorithms have been proposed. Initially, most of these approaches were greedy [Coh95, Mit97]; due to the use of heuristics, however, these algorithms may end up in a local optimum instead of a global one. In particular on structured domains, such as the example in chemoinformatics listed above, certain rules are hard to find by growing rules in small steps. For instance, in molecules, a benzene ring (a ring of 6 carbons) is an important structure with special properties; however, only after adding 6 bonds in a graph does this structure emerge. A greedy algorithm is not likely to find this structure automatically. Algorithms that are not greedy, but instead investigate a larger search space or even provide optimal solutions according to well-chosen criteria, may hence be preferable in these applications.

However, given the large search space of rules in most applications, making an exhaustive search feasible is a major challenge. To address this challenge, *pattern mining algorithms* may be useful. Pattern mining is one of the most studied topics in data mining and focuses mostly on the exhaustive enumeration of structures in databases. A pattern can be thought of as the antecedent of a rule. Even though patterns were originally studied for descriptive tasks [AMS⁺96], an obvious question is how to exploit them also in predictive tasks, where the aim is to exhaustively search through a space of rule antecedents.

Even once we have found a set of patterns (or rules), as mined by pattern mining techniques, these do not immediately correspond to a good classifier. The next question is how we can select and combine patterns for accurate classification models. This is the problem that we study in this chapter.

The probably most simple approach for using patterns in classification models is as follows:

- a pattern mining technique is used to find a large set of patterns;
- a new feature table is created, in which every column corresponds to a pattern, and each row corresponds to an element of the data set;
- a model is learned on this new table, where any learning algorithm can be used that can deal with binary data.

The process is illustrated in Figure 1. It is conceptually simple to use as the learning algorithm is treated as a black box. A deeper understanding of classification algorithms is not needed, thus making this approach very useful for non-computer scientists, such as biologists or chemists. Particularly in applications with structured data, such as (bio-)chemistry, this strategy turned out to be attractive and was among the first pattern-based classification approaches [LZO99]. It was used successfully to classify sequences [LZO99, KR01], graph structures [KR01, DKWK05, KNK⁺06, BZRN06] and also attribute-value data [DK02]. Its popularity today is attested by the continued use in recent publications [CYHY08, BZ09] as well as a workshop devoted to this topic [KCFS08].

Many extensions to the basic procedure are possible. The main complication that was already faced in the early days of pattern-based classification

was the large number of patterns produced by pattern mining algorithms [LHM98]. At the time few learning algorithms were commonly known that could deal with large feature spaces (for instance, SVMs for regression were only introduced by Vapnik in 1996, in the same year that the term ‘frequent itemset’ was introduced). It was necessary to reduce the number of patterns as much as possible. Two approaches can be taken to achieve this.

The direct approach, in which the classifier construction and the feature selection are combined: the selection procedure is made such that the selected patterns are directly inserted in a rule-based classifier [LHM98].

The indirect approach, in which the feature selection is separated from the classifier construction [LZO99]. This step is also indicated in Figure 1.

An advantage of the direct approach is that it no longer treats the classifier as a black box. One of the potential advantages of using patterns – interpretable classifiers – is hence not negated in this approach.

This chapter provides an overview of several recent approaches towards classification based on patterns, with a focus on methods that we proposed recently. The chapter is subdivided in sections that roughly correspond to the steps illustrated in Figure 1.

The first step in the process is the pattern mining step. The constraint which has traditionally been used to determine if a pattern is relevant and should be passed on to the next phase, is the minimum support constraint. If the patterns are used for classification purposes, frequent features may however not always be relevant; for instance, consider a pattern which covers all elements of a data set, and hence will never distinguish classes from each other. An alternative is to search for only those patterns which show a correlation with the target attribute. Section 3 discusses correlation constraints that can be applied to individual patterns to determine their relevance, and how to enforce them during mining, based on the solution proposed in [MS00].

Once a pattern mining technique has generated patterns, it is often the case that these patterns are correlated among each other. Section 4 discusses *indirect* methods for selecting a subset of useful patterns from a given set of patterns, and places two techniques that we recently proposed [RZ07, BZ09] in a wider context [KH06a, KH06b, BYT⁺08]. These methods are independent of the classifier that is applied subsequently.

Given that patterns can be seen as rules, several algorithms have been proposed that take a set of patterns as input and use these patterns as rules to build a rule-based classification model

Subsequently, Section 5 discusses *direct* methods for selecting patterns from a pre-computed set of patterns. Next to traditional methods [LHM98, LHP01] we summarize a method we proposed [ZB05].

All approaches mentioned so far assume that the patterns, and the corresponding feature table, are created in a separate, first step. However, in more recent papers a tight integration has been studied, in which the learning algorithm calls the pattern mining algorithm *iteratively* as required to create new

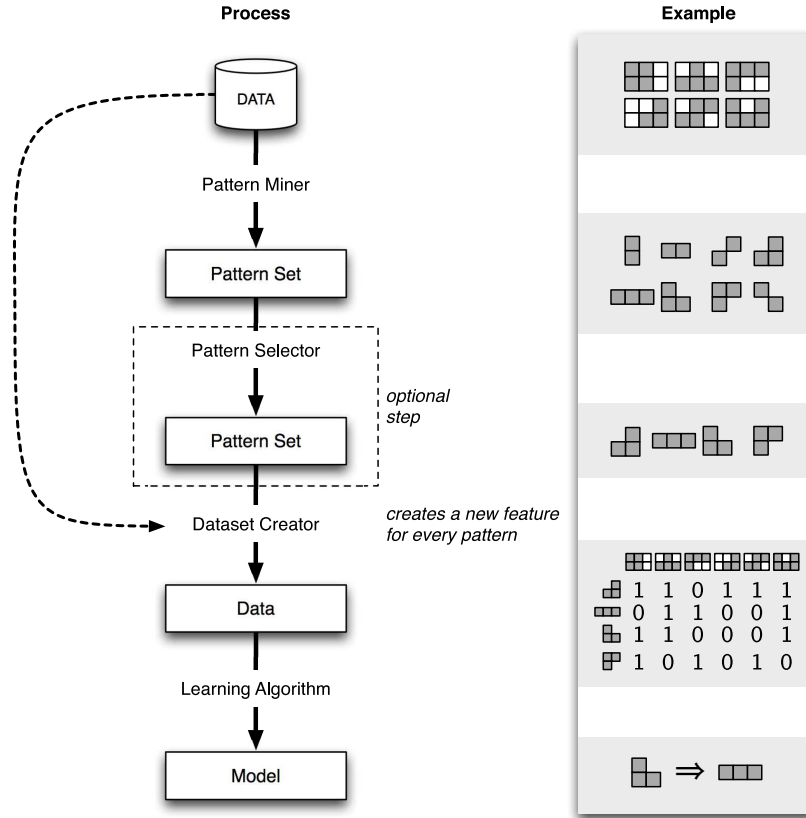


Fig. 1 The stepwise approach to use patterns in classification models.

features, and the construction of a feature table is not a separate phase. In addition to discussing approaches we proposed [CGSB08, NF07, BZ05], Section 6 also presents an argument for upgrading the third technique described in Section 4 [KH06a] to the integrated setting.

An overview of the methods is provided in Table 1.

First, however, Section 2 briefly reviews the basic principles of pattern mining, and introduces the notation that we will be using.

2 Preliminaries

We assume given a data set \mathcal{D} consisting of tuples (\mathbf{x}, y) , where $y \in \mathcal{C}$ is a class label and $\mathbf{x} \in \mathcal{A}$ is a description of an element of the data set. In the simplest case, this description is a tuple of binary attributes, that is, $\mathcal{A} = \{0, 1\}^n$; however, \mathbf{x} may also be more complex, for instance, a labeled

	Classifier Construction	
	Indirect (<i>Separate from pat. selection</i>)	Direct (<i>Integrated with pat. selection</i>)
Separate Pattern Mining (<i>Post-processing patterns</i>)	Section 4 <ul style="list-style-type: none"> • Exhaustive search • The chosen few • Maximally informative sets 	Section 5 <ul style="list-style-type: none"> • CBA • CMAR • CtC
Integrated Pattern Mining (<i>Iterative pattern mining</i>)	Section 6 <ul style="list-style-type: none"> • Maximally informative sets 	Section 6 <ul style="list-style-type: none"> • FitCare • DL8 • Tree²

Table 1 Comparison of pattern set mining techniques discussed in this chapter.

graph (V, E, Σ, λ) , where V is a set of nodes, $E \subseteq V \times V$ is a set of edges, and λ is a function from V to labels in Σ .

We can partition a dataset according to the class labels; that is, for a given $c \in \mathcal{C}$, we define that $\mathcal{D}^c = \{(\mathbf{x}, y) \mid (\mathbf{x}, y) \in \mathcal{D}, y = c\}$.

A *pattern* π is a function from \mathcal{A} to $\{0, 1\}$. There are many types of patterns. The simplest type of pattern is an *itemset*. An itemset is represented by a set $I \subseteq \{1, \dots, n\}$. When an itemset is applied to a binary vector \mathbf{x} , it predicts 1 iff for all $i \in I$: $x_i = 1$. We usually say that pattern I is included in element \mathbf{x} . Also patterns can be more complex, for instance, graphs G . Usually, a graph G predicts 1 for a graph that is an element of the data set iff G is subgraph isomorphic with this graph.

Given a dataset \mathcal{D} and a pattern π , by $\pi(\mathcal{D})$ we denote the set of transactions containing the pattern, i.e. $\pi(\mathcal{D}) = \{(\mathbf{x}, y) \mid \pi(\mathbf{x}) = 1, (\mathbf{x}, y) \in \mathcal{D}\}$.

A pattern mining algorithm is an algorithm that enumerates *all* patterns satisfying certain *constraints* within a certain *pattern language* \mathcal{L} . The most popular constraint is the minimum support constraint. The support of a pattern is the number of elements in a data set that includes the pattern, i.e., if π is a pattern, its support is $|\pi(\mathcal{D})|$. A pattern is *frequent* for minimum support threshold θ iff $|\pi(\mathcal{D})| \geq \theta$. Examples of pattern languages are itemsets ($\mathcal{L} = 2^{\mathcal{I}}$) and graphs ($\mathcal{L} = \{G \mid G \text{ is a graph}\}$).

If we associate to a pattern a class label $c \in \mathcal{C}$ we obtain a class association rule $\pi \Rightarrow c$. The support of a class association rule is defined as

$$|\{(\mathbf{x}, y) \in \mathcal{D}^c \mid \pi(\mathbf{x}) = 1\}|.$$

To find all patterns satisfying a constraint φ in a space of patterns \mathcal{P} , algorithms have been developed which traverse the search space in an efficient way. Despite the large diversity in methods that exist for many pattern domains, they have certain properties in common.

First, they all assume that there is a generality order \sqsupseteq between the patterns. This order satisfies the property that

$$\forall \mathbf{x} : \pi_1(\mathbf{x}) = 1 \wedge \pi_2 \sqsubseteq \pi_1 \rightarrow \pi_2(\mathbf{x}) = 1.$$

In other words, if a pattern is included in an element of the data set, all its generalizations are also included in it.

Consequently, for the minimum support constraint we have that

$$|\pi_1(\mathcal{D})| \geq \theta \wedge \pi_2 \sqsubseteq \pi_1 \Rightarrow |\pi_2(\mathcal{D})| \geq \theta$$

for any possible dataset. Any constraint which has a similar property is called *anti-monotonic*. While anti-monotonic constraints were the first ones used and exploited for efficient mining, current pattern mining algorithms exist also for non-monotonic constraints such as *average* value of items in a set or *minimum* χ^2 -score.

All descendants of a pattern in the pattern generality order are called its *refinements* or *specializations*. Pattern mining algorithms search for patterns by repeatedly and exhaustively refining them, starting from the most general one(s). Refinements are not generated for patterns that do not satisfy the anti-monotonic constraint. By doing so some patterns are never generated; these patterns are *pruned*.

In the case of itemsets, the subset relation is a suitable generality order. Assuming itemsets are sets $I \subseteq \{1, \dots, n\}$, the children of an itemset I can be generated by adding an element $1 \leq i \leq n$ to I , creating sets $I \cup \{i\}$. For the subset partial order it holds that

$$I_1 \subseteq I_2 \Rightarrow |I_1(\mathcal{D})| \geq |I_2(\mathcal{D})|.$$

Frequent itemset mining algorithms traverse the generality order either breadth-first (like APRIORI [AMS⁺96]) or depth-first (like ECLAT [ZPOL97] and FP-GROWTH [HPY00]). When it is found for an itemset I that $|I(\mathcal{D})| < \theta$, the children of I are not generated as all supersets can only be infrequent too.

Similar observations can be exploited for graphs, where usually the subset relation is replaced with the subgraph isomorphism relation.

Frequent pattern mining algorithms for many types of datastructures exist, among which sequences, trees, and relational queries; essential in all these algorithms is that they traverse the pattern space in such a way that the generality order is respected.

We will see in the next section how we can search for patterns under other types of constraints.

In this paper, we will often be using a set of patterns $P = \{\pi_1, \dots, \pi_n\}$ to create a new binary dataset from a dataset $\mathcal{D} = \{(\mathbf{x}, y)\}$:

$$\{(\mathbf{z}_{\mathbf{x}}^P, y) = (\pi_1(\mathbf{x}), \dots, \pi_n(\mathbf{x}), y) \mid (\mathbf{x}, y) \in \mathcal{D}\}.$$

In this new dataset, every attribute corresponds to a pattern; an attribute has value 1 if the element of the data set includes the pattern.

3 Correlated Patterns

In traditional frequent pattern mining the class labels, if present, are ignored. We will refer to a pattern mining technique which takes the class labels into account as a *correlated pattern mining* technique, but many alternative names have also been proposed, among others *emerging pattern mining* [DL99], *contrast pattern mining* [BP99] and *discriminative pattern mining* [CYHH07, CYHY08] and as subgroup discovery in the first paper in [Wro97]. We will use the notation we proposed in [NK05].

In statistics, correlation describes a linear dependency between two variables; in our case the class value and the occurrence of a pattern. While in theory more than two possible class values could be handled, we will restrict ourselves to the binary setting. The aim of correlated pattern mining is to extract patterns π from a data set whose occurrences are *significantly* correlated with the class value y . The main motivation for this approach is that patterns which do not have significant correlation with the target attribute, are not likely to be useful as features for classifiers.

Correlated pattern mining techniques use correlation scores that are computed from contingency tables. Every transaction is either covered by a pattern or not; furthermore, the transaction is either positive or negative. This gives us four disjunct possibilities for each transaction, which can be denoted in a 2×2 contingency table such as Table 2. Statistical measures are employed to calculate a correlation score between the class value and the pattern at hand from the number of transactions for each of the four possibilities. Correlated pattern mining aims for the extraction of patterns that

Table 2 Example of a contingency table.

	Covered by pattern	Not covered by pattern	
Positive example	$p = \pi(\mathcal{D}^+) $	$P - p = \mathcal{D} \setminus \pi(\mathcal{D}^+) $	P
Negative example	$n = \pi(\mathcal{D}^-) $	$N - n = \mathcal{D} \setminus \pi(\mathcal{D}^-) $	N
	$p + n$	$P + N - p - n$	$N + P$

have a high correlation with the target attribute. There are two settings that have been considered:

- find all patterns that reach at least a user-defined score;
- find the k patterns scoring best on the given data set.

We will first provide on how to compute patterns in these settings, before discussing the advantages and disadvantages.

3.1 Upper Bound

Obviously it is not efficient to enumerate all possible patterns to find the desired subset. As noted before, the property allowing for an efficient enumeration of all frequent patterns is the anti-monotonicity of the frequency constraint. In correlated pattern mining, we would like to exploit a similar anti-monotonic constraint. Unfortunately, interesting correlation scores are usually not anti-monotonic. A correlation score should at least have the following properties:

- (A) a pattern which covers negative and positive elements of the data set in equal proportion as occurs in the full data should not be considered a correlated pattern;
- (B) a pattern which covers many elements of one class, and none of the other, should be considered correlated.

As a pattern of type (B) can be a refinement of a pattern of type (A) useful correlation measures cannot be anti-monotonic. Hence,

$$\pi \sqsubseteq \pi' \not\Rightarrow score(\pi) \geq score(\pi')$$

The main approaches proposed to solve this problem involve introducing an *upper bound*. An upper bound is an anti-monotonic measure which bounds the correlation values that refinements of a pattern can reach. Hence, an upper bound allows for an efficient search;

$$\pi \sqsubseteq \pi' \Rightarrow bound(\pi) \geq bound(\pi') \text{ with } bound(\pi) \geq score(\pi')$$

The bound allows us to find all correlated patterns exceeding a minimum correlation score in a similar way as we can find frequent patterns; i.e. branches of the search tree are pruned using the anti-monotonic bound constraint, instead of the minimum support constraint. In contrast to frequent pattern mining, not all patterns that exceed the score are output. Only patterns that exceed the threshold on the original score function are part of the result.

3.2 Top-k Correlated Pattern Mining

In order to search for the top- k patterns, a naïve approach would be to simply select these patterns in a post-processing step for a given (low) threshold. However, an important question is if we can find the top- k patterns more efficiently if we search for top- k patterns during the mining step.

It turns out that the bounds discussed in the previous section can be used relatively easily to find the top- k patterns. The main idea is to update the correlation threshold θ during the search starting from the lowest possible

threshold. A temporary list of the top- k patterns is maintained during the search. Every time a enumerated pattern π exceeds the score of the worst scoring pattern in the top- k list, this newly found pattern π is inserted into the list, which is cropped to the current top- k patterns found. As long as the length of the list is still below the desired k , patterns are simply added. Once the list reaches the desired size of k patterns the threshold used for pruning is always updated to the score of the worst pattern in the list. This threshold is used as a constraint on the bound value of patterns. Thus, while the mining process goes on, the pruning-threshold continues to rise and consequently improves the pruning-power. The final list contains the top- k patterns.¹

Note that due to the changing threshold the search strategy can influence the enumerated candidates in the top- k setting. This has been investigated and exploited in e.g. [BZRN06].

Both constraints – the top- k and the minimum score – can be combined by setting the initial threshold to a user defined threshold. As a result, the patterns extracted will be the k best scoring patterns exceeding the initial user-defined threshold.

3.3 Correlation Measures

As said before, in correlated pattern mining we need a non-trivial upper bound to allow for pruning². This upper bound should be as tight as possible to achieve the maximum amount of pruning.

In this chapter, we will use the PN-space based methodology of [NK05] to introduce the bounds. The use of bounds in a correlated pattern mining setting was first proposed in [BP99]; however, we present the bounds introduced in [MS00] as they are more tight.

As stated earlier, instances in the data set are associated with class labels. We consider a binary class setting with positive labeled instances \mathcal{D}^+ and negative labeled instances \mathcal{D}^- such that $\mathcal{D}^+ \cup \mathcal{D}^- = \mathcal{D}$ while \mathcal{D}^+ and \mathcal{D}^- are disjunct.

Any pattern π covers $p = |\pi(\mathcal{D}^+)|$ of the positive instances and $n = |\pi(\mathcal{D}^-)|$ of the negative instances and can be represented as point in a pn -space, as illustrated in Figure 2. Accordingly the total frequency of the pattern π is $|\pi(\mathcal{D})| = |\pi(\mathcal{D}^+)| + |\pi(\mathcal{D}^-)|$. The upper bound has to be defined such that for any specialisation $\pi' \sqsupseteq \pi$ the inequality $bound_{\mathcal{D}}(\pi) \geq score_{\mathcal{D}}(\pi')$ holds, and the bound is anti-monotonic. Due to the anti-monotonicity any specialisation $\pi' \sqsupseteq \pi$ can only cover a subset of π such that $\pi'(\mathcal{D}^+) \subseteq \pi(\mathcal{D}^+)$ and $\pi'(\mathcal{D}^-) \subseteq \pi(\mathcal{D}^-)$. Therefore any π' can only reach values located in the grey area defined by π (Figure 2). As a result we need to define the upper

¹ The list can be shorter if there were less than k patterns exceeding the lowest threshold.

² There is always a trivial upper bound that does not allow for any pruning and thus is worthless

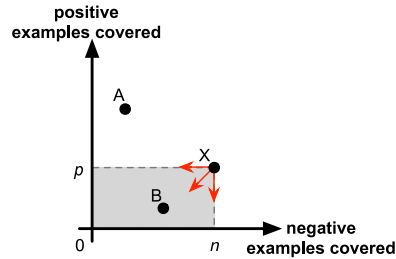


Fig. 2 The pn -space with patterns a , b , and x covering different parts of the data set.

bound to be greater or equal to the maximum score over all $p' \leq p = |\pi(\mathcal{D}^+)|$ and $n' \leq n = |\pi(\mathcal{D}^-)|$,

$$\text{bound}_{\mathcal{D}}(p, n) \geq \max_{p' \leq p, n' \leq n} \text{score}_{\mathcal{D}}(p', n')$$

If the correlation measure is *convex*, the calculation of the upper bound is straightforward. Convex functions are known to reach their extreme values at the borders. Thus, only the scores of these borders have to be computed - the maximum of which specifies the upper bound.

Morishita and Sese [MS00] introduce the general idea and discuss two popular and frequently used convex functions, χ^2 and *information gain (IG)*. While in these cases convexity is exploited, in general, any function that allows for an efficient calculation of an upper bound in a restricted domain can be used to guide the search [Bri09]. One example is the non-convex Hellinger distance. As it still reaches its extrema at the borders, the upper bound can be calculated in the very same way as for χ^2 or Information Gain.

Next to the technique introduced above, other methods to extract correlated patterns based on ‘pure’ frequent-pattern mining have been proposed. [NK05] shows that separate mining on each class with a threshold derived from the desired minimum correlation and post-processing can be done efficiently to obtain the desired result. Alternatively, given a threshold on correlation, the minimum of the class-specific support thresholds can be used as a support threshold on the entire dataset. This approach was essentially proposed in [CYHH07, CYHY08]; note however that this class-ignorant pruning strategy is suboptimal compared to the approach which takes class labels into account.

3.4 Type I Errors

All methods discussed evaluate a correlation score for every pattern. For instance, we used χ^2 to evaluate how much each pattern correlates with a class attribute. In statistics a common recommendation is that if the χ^2 value

exceeds 3.84, the correlation is significant. Does this mean that it is always correct to use 3.84 as threshold?

Unfortunately, this is not the case. In statistics it is well-known that if we repeatedly evaluate a significance test, chances are very high that we will incorrectly reject the null hypotheses in some of these tests. This kind of error is called a *type I error*. Given the large number of patterns considered, type I errors are very likely in pattern mining. How to deal with them is an issue, which has not received much attention.

A common way to deal with type I errors is the *direct adjustment approach*, which modifies the minimum correlation score depending on the number of hypotheses evaluated: the more hypotheses considered, the higher the correlation threshold should be.

In bound-based pattern mining, it is however not clear how this method should be applied correctly. Intuitively, one would say that the choice of threshold should be independent from the algorithm used, and hence, independent from the number of patterns enumerated by the search process of the algorithm. However, normalizing by the total size of the search space is not always an option, as some pattern domains (such as graphs) are infinitely large.

Given these problems, until now most direct adjustment approaches take a practical approach. In [BP99] the score of a pattern is normalized by the number of frequent patterns of the same size; [Web08] presents an attempt to make this approach more correct by including the total size of the search space; this approach is only applicable on itemset domains however.

An alternative approach is the *holdout* approach. In this case, the dataset is split in two parts. A large set of patterns is first mined in the first part; this set of patterns is evaluated in the second part to determine which are relevant. The advantage is that it is justified in this case to normalize the correlation threshold by the number of patterns returned in the first phase. The approach, however, is not applicable in direct top- k pattern mining [Web08].

3.5 Closed and Free Pattern Mining

Pattern sets can often be redundant (see Section 4). One such redundancy is caused by considering patterns that are neither closed nor free. The principles of closedness and freeness were originally proposed for unlabeled data sets, in which case a pattern π was called *closed* iff there was no pattern $\pi' \sqsupset \pi$ such that $\pi(\mathcal{D}) = \pi'(\mathcal{D})$; a pattern was called *free* iff there was no pattern $\pi' \sqsubset \pi$ such that $\pi(\mathcal{D}) = \pi'(\mathcal{D})$. Restricting the set of patterns to closed or free ones can often already reduce the number of patterns under consideration significantly and is hence one of the most common approaches to restricting the initial set of patterns.

The extension of this principle to labeled data is possible in multiple ways. The most straightforward approach is to ignore the class labels. However, assume we have two patterns with these properties:

$$|\pi(\mathcal{D}^+)| = |\pi'(\mathcal{D}^+)|, \quad |\pi(\mathcal{D}^-)| < |\pi'(\mathcal{D}^-)| \quad \text{and} \quad |\pi(\mathcal{D}^-)|/|\mathcal{D}^-| < |\pi(\mathcal{D}^+)|/|\mathcal{D}^+|,$$

where the last condition states that π is more correlated with the positive class than with the negative, then any sensible correlation score would determine that π' is less correlated. This shows that one can also consider a pattern uninteresting if it is closed or free on only one of the two classes. An approach for finding closed patterns, given support thresholds for two classes, was studied in [GKL06]. An alternative approach is to perform top- k free pattern mining [BZRN06], essentially by not inserting patterns in the queue of the search procedure which are not free compared to the patterns already in the queue.

4 Finding Pattern Sets

The preceding section did not only cover top k -correlated pattern mining, but also algorithms to derive all correlated patterns. As we have argued in the introduction, in classifier building an optional second step is often useful in which a set of patterns is filtered. The main goals of this filtering are to remove redundancy, and to bring the potentially large amount of patterns down to a reasonable number. Achieving these goals will benefit both machine learning techniques that use the patterns either directly or indirectly (through data propositionalization) and human users that want to inspect the patterns or the models created from said patterns. For machine learning techniques, a large amount of redundant features means that selecting the relevant features in the process of building a model becomes harder, not to mention the increased computational complexity that goes with it. For humans there is simply the question of human perception – no one can be expected to make sense of hundreds or even thousands of patterns, especially if some of them are largely redundant. In this section we focus on stand-alone techniques that have been proposed for selecting patterns from a set. Techniques which directly use patterns as rules in classifiers are discussed in Section 5.

4.1 Constrained Pattern Set Mining

There is actually a straightforward way of selecting subsets of (potentially correlating) patterns from an existing large set of mined patterns: search for

subsets under constraints as done in pattern mining algorithms. In the same way as *itemsets* can be enumerated in a principled way, *pattern sets* can be assembled. Of these sets, only those are kept that satisfy certain properties, e.g. regarding the redundancy among patterns in the set, the size of the pattern set, or in- or exclusion of particular patterns.

This idea was formalized and the framework and preliminary results were published in [RZ07]. The main insight lies in the fact that the problem of pattern set mining is dual to the problem of pattern mining: while *all* items of an itemset have to occur in an element of the data set to *match*, for pattern sets the most intuitive interpretation is to match an element of the data set if *any* member pattern of the pattern set occurs in the element. This duality leads to an exact reversal of the direction of the (anti-)monotonicity property, which means that the pruning strategies behind the pattern mining algorithms can also be applied in pattern set mining. For instance, the maximum frequency constraint (which is monotonic in itemset mining), is anti-monotonic in pattern set mining, and can be used to prune the search space when growing pattern sets. Useful constraints for pattern set mining were defined, such as *pairwise redundancy* constraints, which allow the user to effectively control the level of redundancy that holds in the data set and together with size constraints allows the selection of compact, informative pattern subsets. These constraints also fulfill the anti-monotonicity property. Finally, the dual nature of the problem allows for the *lower* bounding of interestingness measures such as χ^2 and the effective upper bounding of accuracy, something that is not possible with individual patterns.

The experimental results reported in [RZ07] show the effectiveness of the approach, demonstrated in pattern selection for classifier building, but also the limitations in that pattern set mining faces quite a few of the challenges that local pattern mining encounters. Specifically, depending on the selectiveness of the constraints, many pattern sets satisfy them and enumerating them can quickly exhaust computational resources, given the large amount of patterns forming the basic elements of the language. These observations point towards two promising avenues: 1) it is often not necessary to return *all* pattern sets, a view corresponding to top-*k* mining for local patterns. Also, 2) the set returned does not have to be the optimal one if optimality can be traded off against efficiency in a reasonable manner. The following two approaches follow these directions.

4.2 The Chosen Few

The method we introduced in [BZ09] focusses on optimizing a single pattern set heuristically. The main point of this work is that if two pattern sets partition the data in the same manner, the smaller one of those is considered preferable, carrying the same information as the larger one while being easier

to peruse by humans and easier to process by machine learning techniques. Given a set of patterns, instances from which they have been mined can be described in terms of the patterns in the set that are present in the instance and in terms of those are absent. Instances agreeing on all patterns' presence form an equivalence class or block; the set of all blocks makes up the partition on the data. In terms of machine learning techniques, for instance, *all* instances from a particular block appear *equal* to the algorithm since they are encoded in an indistinguishable manner.

Given a set of patterns, and the partition it induces, adding a new pattern to the set can either increase the number of blocks or leave it unchanged. In the latter case, this pattern is clearly redundant, since it can be expressed either by another pattern, by its complement, or by a combination of patterns. In processing the full result set of a local pattern mining operation, it can hence be useful to reject patterns that do not change the partition, whittling the pattern set down to a smaller number of patterns carrying the same amount of information. While it could be argued that there is information which is not recovered by the partition alone, this is not true for all machine learning techniques; in principle, if this is useful, many machine learning techniques could deduce one feature from the others. While the *minimal* number of patterns needed is logarithmic in the number of blocks in the partition, typically more patterns are *actually* needed. Deciding on the minimal set of patterns needed to induce the same partition as the original result set is computationally rather expensive, however, setting the stage for the application of heuristic techniques, for which we developed two alternative approaches.

The first of the two algorithms, BOUNCER, uses a user-defined order and considers each pattern exactly once for potential inclusion. This order is augmented by a measure which evaluates the contribution of a pattern in terms of the granularity of the partition. Using a threshold on the minimal contribution a pattern has to make, the first pattern encountered that exceeds this threshold is added to the set, before patterns that appear later in the order are evaluated further. The combination of the order, the selection of the first pattern for inclusion and the, necessarily, local quality measures allows the efficient mining of locally optimal pattern sets. Notwithstanding their local optimality, the resulting pattern sets were shown to improve on the set of all patterns in terms of utility as features for classification. The size and quality of the pattern sets is strongly influenced by the order and the measure used, however.

Heuristically optimizing a pattern set will of course have drawbacks that balance the faster execution. Consider the example in Table 3: $\square\square$ has a joint entropy of 1, the highest entropy for a single pattern, and would therefore be chosen first. The joint entropy of the set $\{\square\square, \square\square, \square\square\}$ is 2.72193, however, the highest possible with three patterns, and this does not include the highest-scoring individual pattern at all.





























































Elements												
π_1												
π_2												
π_3												
π_4												

Table 3 A data set of 10 elements with the coverage of four different patterns.

While the adoption of a global optimality criterion would prove to be impossible for non-exhaustive search, the heuristic nature of BOUNCER can be improved by changing the used order and the selection of the pattern used for inclusion. This is implemented in the PICKER* algorithm, in which an upper bound on the contribution of individual patterns to the set is calculated and the patterns ordered in descending value of this upper bound. By traversing and evaluating patterns until none of the remaining upper bound values exceeds the contribution of the currently best pattern anymore, a closer approximation of global optimality can be expected. Recalculating the upper bounds and reordering the patterns potentially leads to several evaluations of each individual pattern, as opposed to BOUNCER’s approach. The resulting pattern sets do not show the fluctuation in cardinality that different orders caused and in most cases improve on the quality of pattern sets mined by using a user-defined order. Both techniques are able to handle a far larger amount of patterns than the complete method introduced in the preceding section.

4.3 Turning Pattern Sets Maximally Informative by Post-Processing

While the approaches in the previous section greedily compute a set of patterns aimed to be diverse, no global optimization criterion is used to measure this diversity. In [KH06a, KH06b, BYT⁺08] measures were studied which can be used to measure diversity of a set of patterns. Nevertheless, the BOUNCER algorithm is related to algorithms which optimize under such global measures.

We can distinguish two types of measures. First, there are measures that do not take into account the class labels, such as joint entropy [KH06a]. By picking a set of patterns P we can construct a new representation for the original data, in which for every element (\mathbf{x}, y) we have a new binary feature vector $\mathbf{z}_{\mathbf{x}}^P$ (see Section 2). Ideally, in this new representation we can still distinguish two different elements from each other by having different feature vectors. However, in case this is not possible, one may prefer a set of patterns in which any two elements are not likely to have the same feature vector. One

way to measure this is by using joint entropy,

$$H(P) = \sum_{\mathbf{b} \in \{0,1\}^{|P|}} -p(\mathbf{b}) \log p(\mathbf{b}),$$

where

$$p(\mathbf{b}) = |\{(\mathbf{x}, y) \mid \mathbf{z}_{\mathbf{x}}^P = \mathbf{b}\}| / |\mathcal{D}|.$$

Hence $p(\mathbf{b})$ denotes the fraction of elements of the data set that have a certain feature vector \mathbf{b} once we have chosen a set of patterns P , and we consider such fractions for all possible feature vectors. Joint entropy has desirable properties for measuring diversity: the larger the number of vectors \mathbf{b} occurring in the data, and the more balanced they occur, the higher the entropy value is. When the entropy is maximized, the patterns are chosen such that elements of the data set are maximally distinguishable from each other.

Another class of measures are the supervised measures. An example of such a measure is [BYT⁺08]:

$$Q(P) = |\{(\mathbf{x}, \mathbf{x}') \mid (\mathbf{x}, y) \in \mathcal{D}^+, (\mathbf{x}', y') \in \mathcal{D}^- : \mathbf{z}_{\mathbf{x}}^P = \mathbf{z}_{\mathbf{x}'}^P\}|.$$

This measure calculates the number of pairs of elements in different classes that have the same feature vector when patterns in P are used to build feature vectors.

A set of k patterns that maximizes a global measure is called a maximal informative k -pattern set. To compute such a pattern set, two kinds of approaches have been studied:

- complete approaches [KH06a], which enumerate the space of subsets up to size k , possibly pruning some branches if a bound allows to decide that no solutions can be found. Such an algorithm would be capable of finding the set $\{\begin{smallmatrix} \blacksquare & \blacksquare \\ \blacksquare & \blacksquare \end{smallmatrix}, \begin{smallmatrix} \blacksquare & \blacksquare \\ \blacksquare & \blacksquare \end{smallmatrix}, \begin{smallmatrix} \blacksquare & \blacksquare \\ \blacksquare & \blacksquare \end{smallmatrix}\}$ from Example 3.
- a greedy approach, which iteratively adds the pattern that improves the criterion most, and stops when the desired pattern set size is reached [KH06a, BYT⁺08].

The first approach is very similar to the pattern set mining approach of Section 4, while the second approach is similar to the approach of Section 4.2. Indeed, one can show that the pattern that is added to a pattern set in each iteration by the PICKER* algorithm is within bounded distance from the pattern chosen by an entropy measure when using difference in entropy as distance measure.

One could wonder how well some of these greedy algorithms are performing: how close to the optimum do they get? In this regard, an interesting property of some optimization criteria is *submodularity*. A criterion $F(P)$ is submodular iff for all $P' \subseteq P$ and patterns π it holds that

$$F(P' \cup \{\pi\}) - F(P') \geq F(P \cup \{\pi\}) - F(P),$$

in other words: there are diminished returns when a pattern is inserted later in a set. For a submodular criterion, it can be shown that the greedy algorithm (operating similar to BOUNCER) approximates the true optimum: let P_{opt} be the optimal pattern set of size k , then the greedy algorithm will find a set for which $F(P) \geq (1 - \frac{1}{e})F(P_{opt}) \approx 0.63F(P_{opt})$ [NWF78]. Both $Q(P)$ and $H(P)$ are submodular, and hence the greedy algorithm achieves provably good results. Finally, as BOUNCER approximates the choices made by a greedy algorithm that uses entropy as an optimization criterion, also BOUNCER is guaranteed to approximate a global optimum under the joint entropy criterion.

5 Direct Predictions from Patterns

Most of the methods discussed in the previous section can be seen as feature selection methods. They ignore the fact that features are actually patterns, and do not construct classifiers. In this section, we study techniques that construct a classifier while taking into account that the used components are in fact patterns. These algorithms are *rule-based*, combining rules of the form *pattern* \Rightarrow *class-label*.

The technique of *associative classification* was first proposed in the work introducing the CBA algorithm [LHM98] in 1998, quickly followed by the CMAR [LHP01] approach in 2001 that extended both the pattern selection step and the actual classification model of CBA (and arguably improved on them). We will therefore discuss these approaches first.

A potential limitation of both of these approaches was however that they were centered on the classical minimum-support, minimum-confidence framework. In recent years, a consensus has developed that these patterns are not necessarily best suited to the task of classification. We developed a new method, called CTC [ZB05], which uses other measures as a starting point. We will discuss it in the third part of this section and elaborate on the differences.

5.1 CBA

As mentioned above, CBA was the first algorithm to use the minimum-support, minimum-confidence association rule framework for constructing classifiers, coining the term *associative classification*. The main difference to traditional rule-based machine learning approaches lies in that first a large set of reasonably accurate rules are mined from the data, and in a second step a subset of those is selected that forms the final classifier. The mining step itself is performed using the well-known APRIORI algorithm.

In CBA, patterns are used to build class association rules $\pi \Rightarrow c$, and patterns π need to satisfy a minimum relative support constraint $support_{rel}(\pi \Rightarrow c) = \frac{|\pi(\mathcal{D}^c)|}{|\mathcal{D}|} \geq \theta_s$ and a minimum *confidence* constraint $confidence(\pi \Rightarrow c) = \frac{|\pi(\mathcal{D}^c)|}{|\pi(\mathcal{D})|} \geq \theta_c$.



















The resulting set of all *class association rules* (CARs) that satisfy the constraints, which we will refer to as \mathbb{S} , is then ordered according to a $<_{CBA}$ relation. Given two CARs $car_1 : \pi_1 \Rightarrow c_1$, $car_2 : \pi_2 \Rightarrow c_2$, the relation $<_{CBA}$ between those two rules is

1. Let w.l.o.g. $confidence(car_1) > confidence(car_2)$ then $car_1 <_{CBA} car_2$
2. If $confidence(car_1) = confidence(car_2)$, let w.l.o.g. $support_{rel}(car_1) > support_{rel}(car_2)$ then $car_1 <_{CBA} car_2$
3. If $confidence(car_1) = confidence(car_2)$, and $support_{rel}(car_1) = support_{rel}(car_2)$, let w.l.o.g. $|\pi_1| < |\pi_2|$ then $car_1 <_{CBA} car_2$

The last check holds since both π_1 and π_2 are simply sets of items whose cardinality can be measured. If even the last check fails, the tie is broken arbitrarily. The order used by CBA can thus be summarized as “higher confidence is preferable”, “in case of equal confidence, higher support is preferable”, and “all things being equal, shorter patterns are preferable”.

Using this order, \mathbb{S} is turned into an *ordered* set. Starting from the minimal rule according to this order, rather similar to the kind of pattern selection encountered in Section 4.2, \mathbb{S} is traversed and each rule in turn considered for inclusion in the final classifier. For each rule, all elements in the data set it matches are collected, and it is evaluated whether the rule predicts at least one of those elements’ class label correctly. If it does, it is included in the final classifier and all covered elements are discarded; if not, the rule is discarded. For classifying an unlabeled element, the minimal rule according to $<_{CBA}$ is used to predict its class label.

Table 4 CBA/CMAR illustrative example

Elements	     
π_1	 
π_2	     
π_3	   

To illustrate this, consider the small example in Table 4: $\blacksquare\blacksquare \Rightarrow \text{Dark}$ predicts the dark class with a confidence of 1.0 and is therefore ranked first. $\blacksquare \Rightarrow \text{Dark}$ and $\blacksquare \Rightarrow \text{Light}$ have the same confidence (0.66) albeit for different classes, and since \blacksquare ’s support is higher, it is ranked before \blacksquare .

CBA will select $\blacksquare\blacksquare$ as the highest-ranked pattern and remove the elements of the data set it covers ($\blacksquare\blacksquare$ and $\blacksquare\blacksquare$) from future consideration. The

second pattern \blacksquare still covers and correctly predicts elements of the data set and thus gets selected, removing *all* elements it covers, i.e. all remaining elements. Since this leaves no elements that \blacksquare could predict correctly, it is discarded. This also illustrates one of the weaknesses of the CBA approach: after removing \blacksquare and \blacksquare , \blacksquare in fact shows a confidence of 0.5 on the remaining elements and \blacksquare should be selected instead of it. Since CBA estimates support and confidence only once – on the training data – however, it makes a sub-optimal decision.

The usual setting for the support threshold θ_s is 0.01, and for the minimum confidence θ_c 0.5. While the second threshold can be justified as only accepting rules that predict their class label more often than not, the first threshold is somewhat arbitrary (and has been shown empirically not to give the best results) [CL05]. It also has to be pointed out that the selection and classification techniques are rather ad hoc, one-shot techniques. On the other hand, the resulting classifier uses an easily interpretable model – a list of rules ordered according to easily understandable criteria: high confidence characterizes rules that are usually correct in their prediction, high support means that they can be expected not to describe spurious phenomena, and short rules adhere to the principle of *Occam's razor*.

5.2 CMAR

CMAR [LHP01] attempted to improve especially on CBA's ad hoc aspects, as well as somewhat on assembling the set of rules who are considered for inclusion in the classifier in the first place. The mining of CARs is performed essentially in the same way. Rules are also ordered according to $<_{CBA}$, but since confidence alone can be a misleading quality measure for rules, CMAR uses the χ^2 statistic to discard rules that do not correlate positively. To give an intuition what this means, consider \blacksquare from Table 4 which covers *all* elements. While this pattern satisfies the minimum confidence constraint, its χ^2 -score is 0, denoting that there is actually no correlation with the target class.

The same database coverage approach that was used in CBA is also used in CMAR. There is a notable difference however: instead of removing an element once it is covered by a *single* rule that was included in the final classifier, there has to be more than one such rule. This would be expected to make classification of unseen elements of the data set more reliable since not only one rule would match it. It is suggested in [LHP01] that four rules have to cover an element before it is removed from the data set – there is however no discussion of why this would be a suitable threshold value.

Let us revisit the CBA example (Table 4). If the database coverage threshold is set to 2 then selecting \blacksquare does not lead to the exclusion of \blacksquare . More

important, however, is that, as mentioned above, \square would be discarded before the database coverage pruning even commences.

The second difference lies in the actual classification process. Instead of using the minimal rule according to $<_{CBA}$, the order is discarded, and for each unseen element all the rules are collected that cover it, which should lead to the more reliable classification mentioned above. Additionally, if rules disagree on the class label to be assigned, the impact that each particular rule has on the final decision is based on the rule’s quality, measured by trading off its actual χ^2 value against the “maximal” one it could have attained. This weighted voting strategy is however once again chosen ad hoc – based on empirical performance as the authors admit. So for the sake of classification robustness, CMAR replaces CBA’s model with a more complex one, albeit still based on confident, frequent rules. The threshold values used for mining are the same as in CBA and also the same for *all* rules mined.

5.3 CTC

Considering the order that is imposed on patterns (rules) in both the CBA and CMAR techniques, a notion of importance or desirability emerges: high confidence is valued, leading to rules that are probably useful for classification, but so is large support, leading to rules that can be expected to hold not only on the training data.

Also, as seen in the case of the CMAR approach, a pattern having (relatively) high confidence in connection with a class label does not necessarily correlate positively with the class, if said class label is rather frequent in the first place. Similarly, a frequent rule does not automatically translate into a significant one, especially if the class predicted is the majority class. In CMAR, found rules were subjected to evaluation by the χ^2 statistic and only those accepted that correlated positively.

The χ^2 -statistic trades off support against confidence, so to speak, valuing less confident rules highly, if they have only enough support, combining in this way the two criteria of importance expressed in the order used by CBA. An interesting question arising at this point is “Why use minimum-support, minimum-confidence rules at all?”, especially if they are not used in a winner-takes-all way, as in CBA, but by weighted voting. Using the principles explained in Section 3 it is easily possible to *directly* mine strongly class-correlating predictive patterns, without the detour of mining (and pruning) frequent patterns and assessing their significance *after* enumerating them. The CTC [ZB05] – correlating tree patterns for classification – approach does just that, using the pattern language of labeled rooted trees³, mining

³ The extension to other pattern domains is straightforward.

χ^2 -quantified patterns instead of ad hoc decided-upon support and confidence thresholds.

There are several important differences between CBA and CMAR on the one hand, and CTC on the other hand.

In CBA and CMAR, first *all* rules are mined that satisfy certain minimum thresholds, and then the database coverage step is used to select patterns. CTC combines these two steps. To achieve this, an order similar to $<_{CBA}$ is used, with a slight change in significance measure:

1. Let w.l.o.g. $\chi_{\mathcal{D}}^2(|\pi_1(\mathcal{D}^+)|, |\pi_1(\mathcal{D}^-)|) > \chi_{\mathcal{D}}^2(|\pi_2(\mathcal{D}^+)|, |\pi_2(\mathcal{D}^-)|)$ then $\pi_1 <_{CTC} \pi_2$
2. If $\chi_{\mathcal{D}}^2(|\pi_1(\mathcal{D}^+)|, |\pi_1(\mathcal{D}^-)|) = \chi_{\mathcal{D}}^2(|\pi_2(\mathcal{D}^+)|, |\pi_2(\mathcal{D}^-)|)$, let w.l.o.g. $|\pi_1| < |\pi_2|$ then $\pi_1 <_{CTC} \pi_2$

Support has been replaced by χ^2 as a significance measure and *confidence* is not referenced at all anymore. For a two-class problem, obviously if w.l.o.g. $|\pi(\mathcal{D}^+)| \geq |\pi(\mathcal{D}^-)|$ then necessarily $confidence(\pi \Rightarrow +) \geq 0.5$, and the strength of prediction is traded off against coverage of the pattern in the significance measure already.

Folding the two criteria into a single one has an interesting side-effect. The choices for the support threshold θ_s and the confidence threshold θ_c interact to have an effect on the number of rules mined. Increasing the minimum support but lowering the minimum confidence can lead to more rules, for example. It is not clear, however, how many rules will be mined, which gets exacerbated by the use of the data set coverage threshold in CMAR. Using χ^2 , on the other hand, allows to make this explicit – CTC takes a single parameter with a clear meaning, k , the number of rules, instead of two or three more opaque ones.

CTC uses the principles described in Section 3 to compute the top- k patterns in this order, for example the 1000 highest-scoring patterns. Hence, the selection is not performed *after* but *during* mining. This set is used directly for classification, without further pattern selection.

This has two advantages:

1. Far fewer rules are mined. In fact, most frequent, confident rules do not turn out to be significant.
2. A less complex voting scheme is necessary. CMAR’s voting approach is outperformed by the comparably simple *average strength*, i.e. confidences for each class are added up, and a simple majority vote, i.e. each rule predicts its majority class.

In other words, compared to CBA and CMAR, both the heuristic pruning scheme and the ad hoc (or empirically found) classification technique are replaced by more straight-forward and arguably better-founded solutions.

6 Integrated Pattern Mining

A common feature of the approaches discussed till now is that they assume that a set of patterns is computed once, either based on a threshold, or on the size of the resulting pattern set, such as in *top- k* mining. There is no strong interaction between which patterns are mined and how the model is constructed afterwards. The alternative is to perform *integrated pattern mining*, i.e. patterns are mined, potentially refined or re-mined, *while* the classifier or pattern set is constructed, interleaving the mining and the model formation step, without creating an initial pattern set first.

In this section, we will first describe two updates of techniques discussed in the previous section to perform integrated mining; subsequently, we discuss techniques for building one particular type of model, i.e. a decision tree, using integrated mining techniques.

6.1 FITCARE

Until now we mostly illustrated methods on binary prediction problems. Good performance on binary problems does not always imply a good performance on multi-class problems however. Turning multi-class problems into binary ones usually strongly increases the computational resources needed. Either a number of *1-vs-all* settings has to be addressed that is equal to the number of classes in the data, or an even greater amount of *1-vs-1* settings. The FITCARE algorithm, on the other hand, was developed specifically for good performance on multi-class problems.

We observed in Section 5 that high confidence is not necessarily a good measure for class correlation if the class is a majority class in the first place. The FITCARE algorithm [CGSB08] takes this one step further. It extends the observation that CAR-miners usually focus on one-against-all settings, i.e. the confidence of a rule has to be higher w.r.t. the target class than w.r.t. the *union* of all other classes where it applies, towards the problem of badly skewed data sets. In such data sets, high-confidence, high-support rules will be rules correctly classifying the majority class – yet still covering and effectively misclassifying instances from minority classes. Instead of the usual global minimum frequency and confidence thresholds, a new definition of interesting CARs is proposed based on a distinct support threshold for *every* class. Given this vector of thresholds $(\theta_{s_1}, \dots, \theta_{s_{|c|}})$, a CAR $\pi \Rightarrow c$ is interesting if

1. $\frac{|\pi(\mathcal{D}^c)|}{|\mathcal{D}^c|} \geq \theta_c$
2. $\forall c' \neq c : \frac{|\pi(\mathcal{D}^{c'})|}{|\mathcal{D}^{c'}|} < \theta_{c'}$
3. $\forall \pi \subset \pi', \exists c' \neq c : \frac{|\pi(\mathcal{D}^{c'})|}{|\mathcal{D}^{c'}|} \geq \theta_{c'}$

The advantage of this technique lies in the fact that the vector of thresholds allows for far better fine-tuning of the differentiating power of a CAR between *any* two classes, finer than *emerging* patterns or *class-correlating* patterns can. The drawback is however, that for any given class y_i , $|\mathcal{C}|$ parameters have to be adjusted – the minimum threshold on the class itself and the $|\mathcal{C}| - 1$ maximum thresholds for the other classes. These $O(|\mathcal{C}|^2)$ parameters make up the threshold matrix T whose entries have to be estimated, making the process more expensive than the approaches we have seen so far. This is however traded off against having to break multi-class problems down into several binary problems, as explained above. Using several constraints and a hill-climbing approach, in which pattern mining is repeated, it is possible to estimate these values efficiently. The final matrix is used to extract a set of CARs for each target class in turn. The resulting CARs are

1. highly discriminative between classes, therefore making strongly conflicting predictions unlikely, and
2. highly probable to cover all instances of each target class, thus making default classifications less common.

The resulting rules are once again combined using a rather complex weighted voting scheme that takes into account the reliability of rules when it comes to their contribution to the final prediction. While this is a technique we have seen both in CMAR and in CTC, the focus is now neither on fulfilling a rule's nor on its global confidence but rather on its relative support in the target class. This should, given well-estimated parameters, lead to very small contributions of rules in classes that are not their target class, even if they have globally high support.

6.2 Mining Maximally Informative Pattern Sets Directly

In Section 4.3 we discussed methods for selecting a subset of k patterns that maximize a global optimization criterion. In these methods it was assumed that we start the search from a set of patterns. However, it is also possible to find such sets without first having to mine an initial set of patterns.

The main trick is to change the greedy step in the greedy algorithm of Section 4.3. Instead of iteratively picking from a pre-computed set the pattern which maximizes the optimization criterion, we use branch-and-bound search to determine the pattern that locally optimizes the measure. This branch and bound search employs similar ideas as those used to find correlated patterns (see Section 3).

We will illustrate this for the example of entropy. Assume we have a pattern set with entropy $H(P)$, then we are looking for the pattern π which maximizes $H(P \cup \{\pi\})$, or equivalently, $H(P \cup \{\pi\}) - H(P)$. $H(\pi|P) = H(P \cup \{\pi\}) - H(P)$

is known as the conditional entropy of π given P , and can be written as

$$H(\pi|P) = \sum_{\mathbf{b} \in \{0,1\}^{|P|}} p(\mathbf{b})H(\pi|P = \mathbf{b})$$

where $H(\pi|P = \mathbf{b}) = \sum_{a \in \{0,1\}} -p(\pi = a|P = \mathbf{b}) \log p(\pi = a|P = \mathbf{b})$ and $p(\pi = a|P = \mathbf{b})$ denotes the fraction of elements of the data set characterized by \mathbf{b} also having $\pi(\mathbf{x}) = a$.

The challenge when searching for a pattern that maximizes this score is to determine a bound on the scores of refinements of a pattern; such a bound could allow us to prune parts of the search space that are not promising.

In the case of entropy, we can use the observation that overall entropy is maximized when we maximize the entropy $H(\pi|\mathbf{b})$ in each bin \mathbf{b} . Given a pattern π , what is the highest entropy we can achieve in this bin for a pattern π' that is a refinement of π ? We can distinguish two cases.

- the pattern covers more than half of the elements in the bin \mathbf{b} ; then the highest entropy we might obtain for this bin is obtained by covering half of the elements. Hence, the highest entropy is 1.
- the pattern covers less than half of the elements, the best we can hope for is not to lose any of these elements by refining the pattern. Hence, the best we can hope to obtain is $H(\pi|P = \mathbf{b})$.

Combining these observations we achieve the following bound on the quality of any refinement π' of a pattern π :

$$H(\pi'|P) \leq \sum_{\substack{\mathbf{b} \in \{0,1\}^{|P|} \\ p(\pi = 1|P = \mathbf{b}) \geq 0.5}} p(\mathbf{b}) + \sum_{\substack{\mathbf{b} \in \{0,1\}^{|P|} \\ p(\pi = 1|P = \mathbf{b}) < 0.5}} p(\mathbf{b})H(\pi|P = \mathbf{b}).$$

This bound can be used to prune unpromising branches of a pattern search and makes it possible to find patterns directly without having to post-process a pre-calculated set of patterns.

Combining this result with that of Section 4.3, it follows that we can find a provably good maximally informative pattern set by a combination of branch and bound search and a greedy algorithm.

Similar observations also apply to other measures; for instance, it was also applied in [BYT⁺08] for the supervised $Q(P)$ measure (see Section 4.3).

6.3 DL8

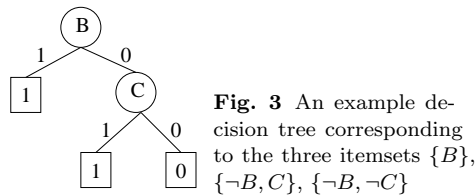
One of the most popular predictive models is the decision tree. A decision tree is a tree in which each internal node is labeled with a test on an attribute and each leaf is labeled with a prediction [Mit97]. A prediction for a particular

element (\mathbf{x}, y) can be obtained by sorting it down the tree starting from the root. The left-hand branch of a node is taken if the specified test on the element of the data set is true; otherwise the right-hand branch is taken. Note that if all attributes are binary, it suffices to label internal nodes with attributes; an element of the data set will be sorted down the left-hand branch of a node labeled with attribute i if its value for \mathbf{x}_i is true; otherwise it is sorted down the right-hand branch.

Many algorithms have been developed for learning decision trees from training data. Most of these algorithms employ the principle of heuristic top-down tree construction [Mit97, Qui93]: starting from an empty tree, iteratively a leaf of the tree is replaced with a test node. A test is chosen by using a heuristic such as information gain. The advantage of this method is that it is fast and usually obtains sufficiently good results. However, it is not guaranteed to be optimal in many ways: given a bound on tree size, the heuristic method may not find the tree that is either most accurate or most cost-effective, in a setting of cost-based learning. In [NF07, NF10] an algorithm, called DL8, was proposed that addresses the problem of finding optimal decision trees by exploiting a connection between pattern sets and decision trees.

The DL8 algorithm is based on exploiting the relationships between paths in decision trees and itemsets. To make this relationship clear we need to extend traditional itemsets to include *negative items*. Traditionally, an itemset $I \subseteq \{1, \dots, n\}$ occurs in an element \mathbf{x} of length n iff for all $j \in I$: $\mathbf{x}_j = 1$. Assume now that $I \subseteq \{1, \dots, n, -1, \dots, -n\}$. Then we can define that an itemset occurs in an element iff for all positive $j \in I$: $\mathbf{x}_j = 1$ and for all negative $-j \in I$: $\mathbf{x}_j = 0$.

This extension allows us to represent every path in a decision tree as an itemset. For example, consider the decision tree in Figure 3. We can determine the leaf to which an element belongs by checking which of the itemsets $\{B\}$, $\{\neg B, C\}$ and $\{\neg B, \neg C\}$ matches. We denote the set of the itemsets corresponding to the leaves of a tree T with $leaves(T)$. Similarly, the itemsets that correspond to paths in the tree are denoted with $paths(T)$. In this case, $paths(T) = \{\emptyset, \{B\}, \{\neg B\}, \{\neg B, C\}, \{\neg B, \neg C\}\}$. A further illustration of



the relation between itemsets and decision trees is given in Figure 4. In this figure, every node represents an itemset; an edge denotes a subset relation. Highlighted is one possible decision tree.

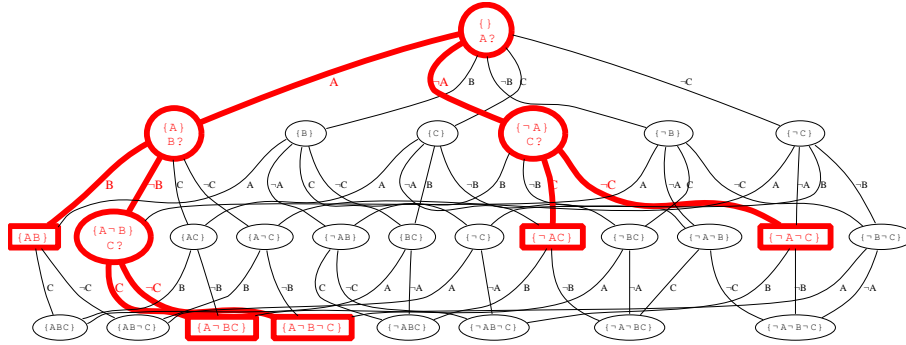


Fig. 4 An itemset lattice for items $\{A, \neg A, B, \neg B, C, \neg C\}$.

Given this correspondence, learning a decision tree can be seen as finding a set of class association rules, where the rules should include both positive and negative items and the set of rules should fulfill properties that ensure that it can be represented as a tree.

In the basic setting, the DL8 algorithm can be seen as a post-processing algorithm that can be applied on a lattice of itemsets. For the problem of finding a tree T which minimizes error $error_{\mathcal{T}}(T)$ on a set of examples \mathcal{T} , the main property that is exploited by the algorithm is that the error of a decision tree equals the sum of the errors of the left-hand and right-hand subtree of the root of this tree. Hence, we can solve the problem of finding an accurate decision tree by independently and recursively searching the best left-hand and right-hand subtrees of each possible root. By storing the best tree for every itemset, we can avoid that we need to consider every itemset more than once, and the computation is linear in the size of the itemset lattice.

In [NF07, NF10] several extensions are discussed of this general idea:

- the use of condensed representations to limit the number of itemsets that need to be considered;
- how to deal with other constraints and optimisation criteria than minimum support and error, for instance, cost-based constraints;
- how to integrate the decision tree construction with the pattern mining.

This last point is important, as by integrating the pattern mining in the tree construction, a smaller number of patterns needs to be considered and less information of the lattice needs to be stored. Overall, the integrated pattern miner searches for patterns once, but does so as guided by the decision tree construction procedure.

6.4 TREE²

In the preceding discussion on DL8, decision trees were described that iteratively split data on a binary attribute to build an effective classifier. If data is described in terms of attribute values or binary attributes denoting, for instance, an item’s presence or absence, this is a straightforward way of building a decision tree. Once more complex data such as trees or graphs needs to be analysed a simple split based on, for instance, the presence or absence of an atom in a molecule will lead to unwieldy classifiers which are unlikely to perform well.

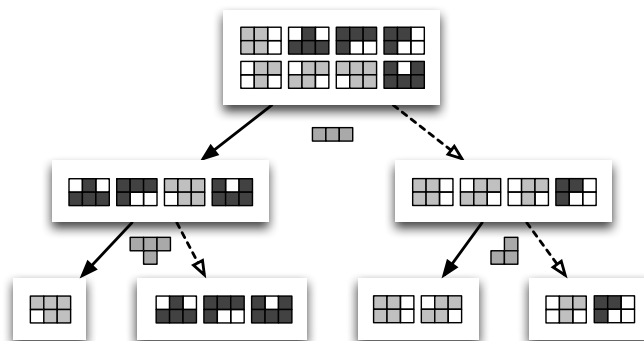


Fig. 5 A decision-tree to separate light and dark structures based on the shapes.

A possible alternative lies in mining a set of patterns, encoding data in terms of their presence or absence and building the decision tree from this re-encoded data. However, we would need to select a constraint under which such patterns need to be mined. Instead of choosing this constraint ad-hoc, we can also integrate the pattern mining step in the decision tree learning algorithm, such that we directly mine for the pattern that the decision tree learning algorithm would select as the best test in post-processing.

To illustrate this, consider the example in Figure 5: mining $\blacksquare\blacksquare$ allows to split the data set into two subsets which each consist 75% of one class, a clear improvement over the original 50–50 split. On these subsets, two more patterns can be mined, \blacksquare and $\blacksquare\blacksquare$, resulting in three pure leaves out of four in total. Especially mining $\blacksquare\blacksquare$ on the full data would probably require very lenient constraints since it matches only one of eight elements, leaving the unmatched subset still very “impure”.

This is the approach taken in the TREE² algorithm [BZ05]. TREE² employs top-1 mining to find the best class-correlating subtree in each iteration, splits the data in covered and uncovered parts, and re-iterates pattern mining for each of these two sets of data points recursively. In this way, ad-hoc thresholds are avoided, and compact trees of truly meaningful patterns can be induced,

improving on the post-processing method of building decision trees from a set of pre-mined patterns.

7 Conclusions

In this chapter we provided an overview of methods we recently proposed for using patterns in classification tasks. We showed that there exists a large variety in methods, ranging from strict step-wise approaches to approaches in which pattern mining and model construction is integrated. We put these methods in context by providing extensive descriptions of related methods.

Despite the amount of work we reported on, this overview is far from complete. Providing a detailed overview of all approaches for pattern-based classification is beyond the scope of this chapter, and is left as future work. Other possibilities for future work include a more detailed investigation of the merits of algorithms for pattern-based classification. We are not aware that a systematic experimental comparison has been carried out for all pattern-based classification methods. Finally, most methods until now concentrate on rule-based classification. An interesting question is for instance how patterns can be used in graphical models.

References

- [AMS⁺96] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.
- [BP99] Stephen D. Bay and Michael J. Pazzani. Detecting change in categorical data: Mining contrast sets. In *KDD*, pages 302–306, 1999.
- [Bri09] Björn Bringmann. *Mining Patterns in Structured Data*. PhD thesis, K.U.Leuven, September 2009. De Raedt, Luc (supervisor).
- [BYT⁺08] Karsten Borgwardt, Xifeng Yan, Marisa Thoma, Hong Cheng, Arthur Gretton, Le Song, Alex Smola, Jiawei Han, Philip Yu, and Hans-Peter Kriegel. Combining near-optimal feature selection with gSpan. In Samuel Kaski, S.V.N. Vishwanathan, and Stefan Wrobel, editors, *MLG*, 2008.
- [BZ05] Björn Bringmann and Albrecht Zimmermann. Tree² - decision trees for tree structured data. In Alípio Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama, editors, *PKDD*, volume 3721 of *Lecture Notes in Computer Science*, pages 46–58. Springer, 2005.
- [BZ09] Björn Bringmann and Albrecht Zimmermann. One in a million: picking the right patterns. *Knowl. Inf. Syst.*, 18(1):61–81, 2009.
- [BZRN06] Björn Bringmann, Albrecht Zimmermann, Luc De Raedt, and Siegfried Nijssen. Don't be afraid of simpler patterns. In Fürnkranz et al. [FSS06], pages 55–66.
- [CGSB08] Loïc Cerf, Dominique Gay, Nazha Selmaoui, and Jean-François Boulicaut. A parameter-free associative classification method. In Il-Yeol Song, Johann Eder,

- and Tho Manh Nguyen, editors, *DaWaK*, volume 5182 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 2008.
- [CL05] Frans Coenen and Paul Leng. Obtaining best parameter values for accurate classification. In *ICDM* [DBL05], pages 597–600.
- [Coh95] William W. Cohen. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.
- [CYHH07] Hong Cheng, Xifeng Yan, Jiawei Han, and Chih-Wei Hsu. Discriminative frequent pattern analysis for effective classification. In *ICDE*, pages 716–725. IEEE, 2007.
- [CYHY08] Hong Cheng, Xifeng Yan, Jiawei Han, and Philip S. Yu. Direct discriminative pattern mining for effective classification. In *ICDE*, pages 169–178. IEEE, 2008.
- [DBL05] *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*. IEEE Computer Society, 2005.
- [DK02] Mukund Deshpande and George Karypis. Using conjunction of attribute values for classification. In *CIKM*, pages 356–364. ACM, 2002.
- [DKWK05] Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. Knowl. Data Eng.*, 17(8):1036–1050, 2005.
- [DL99] Guozhu Dong and Jinyan Li. Efficient mining of emerging patterns: Discovering trends and differences. In *KDD*, pages 43–52, 1999.
- [FSS06] Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors. *Knowledge Discovery in Databases: PKDD 2006, 10th European Conference on Principles and Practice of Knowledge Discovery in Databases, Berlin, Germany, September 18-22, 2006, Proceedings*, volume 4213 of *Lecture Notes in Computer Science*. Springer, 2006.
- [GKL06] Gemma C. Garriga, Petra Kralj, and Nada Lavrac. Closed sets for labeled data. In Fürnkranz et al. [FSS06], pages 163–174.
- [HPY00] Jiawei Han, Jian Pei, and Yiwon Yin. Mining frequent patterns without candidate generation. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *SIGMOD Conference*, pages 1–12. ACM, 2000.
- [KCFS08] Arno Knobbe, Bruno Crémilleux, Johannes Fürnkranz, and Martin Scholz. From local patterns to global models: the LeGo approach to data mining. In Johannes Fürnkranz and Arno Knobbe, editors, *LeGo’08, Proceedings of the ECML PKDD 2008 Workshop ‘From Local Patterns to Global Models’*, pages 1–16, 2008.
- [KH06a] Arno J. Knobbe and Eric K. Y. Ho. Maximally informative k-itemsets and their efficient discovery. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *KDD*, pages 237–244. ACM, 2006.
- [KH06b] Arno J. Knobbe and Eric K. Y. Ho. Pattern teams. In Fürnkranz et al. [FSS06], pages 577–584.
- [KNK⁺06] Jeroen Kazius, Siegfried Nijssen, Joost N. Kok, Thomas Bäck, and Adriaan P. IJzerman. Substructure mining using elaborate chemical representation. *Journal of Chemical Information and Modeling*, 46(2):597–605, 2006.
- [KR01] Stefan Kramer and Luc De Raedt. Feature construction with version spaces for biochemical applications. In Carla E. Brodley and Andrea Pohoreckyj Danyluk, editors, *ICML*, pages 258–265. Morgan Kaufmann, 2001.
- [LHM98] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *KDD*, pages 80–86, 1998.
- [LHP01] Wenmin Li, Jiawei Han, and Jian Pei. Cmar: Accurate and efficient classification based on multiple class-association rules. In Nick Cercone, Tsau Young Lin, and Xindong Wu, editors, *ICDM*, pages 369–376. IEEE Computer Society, 2001.

- [LZO99] Neal Lesh, Mohammed Javeed Zaki, and Mitsunori Ogihara. Mining features for sequence classification. In *KDD*, pages 342–346, 1999.
- [Mit97] T.M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
- [MS00] Shinichi Morishita and Jun Sese. Traversing itemset lattice with statistical metric pruning. In *PODS*, pages 226–236. ACM, 2000.
- [NF07] Siegfried Nijssen and Élisabeth Fromont. Mining optimal decision trees from itemset lattices. In Pavel Berkhin, Rich Caruana, and Xindong Wu, editors, *KDD*, pages 530–539. ACM, 2007.
- [NF10] Siegfried Nijssen and Elisa Fromont. Optimal constraint-based decision tree induction from itemset lattices. *Data Mining and Knowledge Discovery*, 2010. (In press).
- [NK05] Siegfried Nijssen and Joost N. Kok. Multi-class correlated pattern mining. In Francesco Bonchi and Jean-François Boulicaut, editors, *KDID*, volume 3933 of *Lecture Notes in Computer Science*, pages 165–187. Springer, 2005.
- [NWF78] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.
- [Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [RZ07] Luc De Raedt and Albrecht Zimmermann. Constraint-based pattern set mining. In *SDM*. SIAM, 2007.
- [Web08] Geoffrey I. Webb. Layered critical values: a powerful direct-adjustment approach to discovering significant patterns. *Machine Learning*, 71(2-3):307–323, 2008.
- [Wro97] Stefan Wrobel. An algorithm for multi-relational discovery of subgroups. In Henryk Jan Komorowski and Jan M. Zytkow, editors, *PKDD*, volume 1263 of *Lecture Notes in Computer Science*, pages 78–87. Springer, 1997.
- [ZB05] Albrecht Zimmermann and Björn Bringmann. CTC - correlating tree patterns for classification. In *ICDM* [DBL05], pages 833–836.
- [ZPOL97] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. In *KDD*, pages 283–286, 1997.