Pattern Recognition 43 (2010) 3025–3034

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

LCMine: An efficient algorithm for mining discriminative regularities and its application in supervised classification ☆

Milton García-Borroto [a,b], José Fco. Martínez-Trinidad [b], Jesús Ariel Carrasco-Ochoa [b], Miguel Angel Medina-Pérez [a], José Ruiz-Shulcloper [c,*]

[a] Centro de Bioplantas, Carretera a Moron km 9, Ciego de Avila, Cuba
[b] Instituto Nacional de Astrofísica, Óptica y Electrónica, Luis Enrique Erro No. 1, Sta. María Tonanzintla, Puebla, C.P. 72840 México, Mexico
[c] Advanced Technologies Application Center, 7a #21812 e/ 218 y 222, Siboney, Playa, Habana, Cuba

ARTICLE INFO

Article history:
Received 13 January 2009
Received in revised form
3 April 2010
Accepted 9 April 2010

Keywords:
Discriminative regularities
Emerging patterns
Mixed incomplete data
Comprehensible classifiers

ABSTRACT

In this paper, we introduce an efficient algorithm for mining discriminative regularities on databases with mixed and incomplete data. Unlike previous methods, our algorithm does not apply an a priori discretization on numerical features; it extracts regularities from a set of diverse decision trees, induced with a special procedure. Experimental results show that a classifier based on the regularities obtained by our algorithm attains higher classification accuracy, using fewer discriminative regularities than those obtained by previous pattern-based classifiers. Additionally, we show that our classifier is competitive with traditional and state-of-the-art classifiers.

1. Introduction

A supervised classifier finds the relationship between a query object and a given set of predefined classes, based on a training sample, often using certain similarity or dissimilarity measure between object descriptions. Objects appear in the training sample described in terms of numerical features, non-numerical features, or both. In some applications, we also need to include a special symbol (usually "?") to represent missing values. Some authors [1,2] use the term *mixed and incomplete data* for domains with mixed feature descriptions and missing values.

For many learning tasks, a high accuracy is not the only sought after characteristic of a supervised classifier; the classifier should also be easily understood by humans. "Symbolic" learning systems [3–6] are usually much more amenable to human comprehension than classifiers that use complex mathematical models, like neural networks and support vector machines [7]. In many domains the lack of comprehensibility is an important drawback, that may cause a reluctance to use the model. For example, when credit has been denied to a customer, the equal credit opportunity act of the US requires that the financial institution provide the reasons why the application was rejected; indefinite and vague reasons for denial are not legal [8]. In some other fields, like medical diagnoses and mineral prospection, clarity and explainability are key constraints.

Higher classification accuracies are usually obtained at the expense of classification comprehensibility:

- In neural networks (NN) [9], after training a classifier, the user obtains the connection weights, but those weights do not have a clear interpretation in terms of features or feature value relations. Moreover, when classifying with the trained NN, the level of output neurons brings almost no support of the result. So, neural networks are limited in this respect, since they are usually difficult to interpret after training [7].
- In *k* Nearest Neighbors [10], the user could know the objects that determine the classification, but he needs a deep understanding of the distance function and the representation space in order to obtain a result interpretation.
- In a support vector machine classifier [11] is described as a complex mathematical function, which is rather incomprehensible for humans [8].

In most comprehensible classifiers [3–6], the user can understand the model found by the classifier in the training stage. This understanding is very useful to explain the classification results. For example, in a decision tree [12], any path from the root to a leaf

☆ This work is partly supported by National Council of Science and Technology of México under the Project CB2007-83947-Y and Grant 252752.
* Corresponding author. Tel.: +53 7 271 4787; fax: +53 7 273 0045.
E-mail addresses: mil@bioplantas.cu (M. García-Borroto), fmartine@inaoep.mx (J.F. Martínez-Trinidad), ariel@inaoep.mx (J.A. Carrasco-Ochoa), migue@bioplantas.cu (M.A. Medina-Pérez), jshulcloper@cenatav.co.cu (J. Ruiz-Shulcloper).

0031-3203/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2010.04.008

determines a conjunction of properties appearing mostly in the leaf-associated class, which explains the classification. Therefore, the disjunction of the properties, determined by all paths from the root to the leaf nodes of a given class, forms an empirical characterization of the class. This empirical characterization of classes is the key for classifying query objects. A similar reasoning is applicable to decision forests [4], where each tree gives support to a certain class, and the integration strategy provides a joint explanation.

KORA-3 [3] introduces another way to obtain the empirical characterization of classes. In this algorithm, the key idea is the frequency in which certain feature value combinations (subdescriptions) appear in the objects of a class in the training sample and not in its complement.

From emerging pattern classifiers [5] and KORA type classifiers [3], which are other comprehensible classifiers, the user might obtain for each class a list of patterns or complex features, respectively. Based on this list, the classifier assigns a class to a query object, using the patterns or complex features present.

In general, many comprehensible classifiers use *discriminative regularities* for classification and classification support. *Regularities* are feature value combinations that describe a sample subset. In a domain partitioned in classes, *discriminative regularities* are regularities that describe enough objects in a class, and they describe few objects in any other class. There are different ways to represent discriminative regularities in classifiers, although regularities may be implicit in some classifiers. In a decision tree or forest, the paths from the root to the leaves are implicit discriminative regularities expressed in conjunctive forms. In a rule-based system, the antecedents of the rules that imply the class are discriminative regularities. Emerging patterns and KORA's complex features are explicit discriminative regularities, expressed as conjunctions of simpler properties.

The practical problem is how to find the empirical characterizations of classes. It is important to underline that "empirical" implies not only the dependency with respect to the training sample, but also that the obtained characterization of classes is not the grand truth. Every algorithm for this task obtain an approximation of the truth based on training data and a particular procedure to obtain the characterization.

Searching discriminative regularities in a training sample is the key procedure in many comprehensible classifiers, even though it may be implicit. This search is a challenge because the downward closure [13] property no longer holds true for discriminative regularities, and there could be too many candidates in high dimensional databases. In addition, exhaustive solutions may be too costly because of the dimensions of the search space [5].

Despite the fact that some discriminative regularity-based classifiers are widely used, and they show a competitive behavior with respect to other classifiers [14], they have one or more of the following drawbacks:

- There are no efficient pattern finding procedures. For example, KORA type classifiers use an exhaustive search algorithm.
- A simplified pattern representation is necessary to apply fast searching algorithms. An *a priori* discretization of all numerical features and simple "equal" relations with non-numerical features are commonly used. Nevertheless, this simplification usually results in huge amounts of patterns, even for problems with few objects and features.
- The filtering procedures based on properties like minimality over subset inclusion or covering objects in a single class may have two undesirable effects: deletion of important patterns and selection of useless patterns.

The main contribution of this paper is a novel algorithm for mining discriminative regularities. This algorithm creates a collection of diverse decision trees and then extracts patterns from them. Finally, it applies a pattern-filtering step, obtaining a reduced set of high quality discriminative regularities. It is important to note that the resulting regularities are useful by themselves, and they could help the user to discover new knowledge about the problem domain.

In this paper, we introduce a new algorithm named Logical Complex Mine (LCMine) for efficiently finding discriminative regularities on training samples with mixed and incomplete data. LCMine has three distinctive characteristics:

- It does not apply an a priori discretization on numerical features, unlike most algorithms for mining discriminative regularities. It is important to highlight that each numerical attributes is frequently discretized independently. Therefore, discretizing a numerical attribute without considering the values of other features could hide important relations in the objects of a class, causing an important information loss. An example of this undesirable behavior appears in Table 5, where an emerging pattern based classifier (SJEPC), applying an a priori discretization, is unable to find patterns in the *wpbc* database.
- It uses an extended representation for the regularities. This representation allows us to find regularities with a wider set of operators (including $\neq$, $\leq$, $\geq$ and $\in$), while most previous methods use only the "$=$"operator. This way, LCMine patterns can express properties using less regularities.
- It uses a new filtering strategy to eliminate redundant regularities.

Based on the patterns obtained by LCMine, we build a classifier, which attains higher classification accuracy than previous methods, while using fewer patterns. Additionally, our classifier attains very good accuracy results, compared with traditional and state-of-the-art classifiers.

In the next section, we present a review about discriminative regularities in some supervised classifiers. We critically analyze pattern representation and search algorithms for some of the most successful classifiers using implicit or explicit discriminative regularities. Section 3 presents LCMine, the proposed algorithm for finding discriminative regularities. Section 4 presents the results of an experimental comparison using some popular classifiers. Finally, we expose our conclusions and future work.

2. Discriminative regularities in supervised classification

One of the first classifiers based on discriminative regularities is KORA-3, introduced by Bongard in 1963 [3]. The author defines the concept of complex feature for a two-class problem, as a combination of three values from three features appearing at least in $\mu_{\min}$ objects in one class, and not appearing in the other class. A complex feature appears in an object if the object has the same values that the complex feature has in the respective features. As KORA-3 uses Boolean features, a complex feature is a discriminative regularity formed by a combination of three values from three Boolean features. For example, consider the two-class problem in Table 1, containing four Boolean features.

If $\mu_{\min}=2$, we have the complex feature (*tall*=*true*), (*fever*=*true*), (*male*=*true*) in class *sick*, and the complex feature (*fever*=*true*), (*male*=*false*), (*pain*=*false*) in class *healthy*, because for both complex features their three values appear in their respective class in at least two objects, and do not appear in any object of the other class.

KORA-3 finds the complex features in the training process by an exhaustive search over all subsets of three features, and it classifies a query object by majority voting over the complex features appearing in the query object.

De la Vega-Doria et al. extend KORA-3 to KORA-Ω [15], which uses complex features formed by a combination of values for a set of features (not necessarily three) appearing enough in a class and not so much in the others. This extension allows handling multiple-class problems with mixed and incomplete data, but the search procedure of the complex features is still exhaustive. KORA-Ω performs the search in a predefined collection of feature subsets, v.gr. all subsets of three features or the power set of the set of features.

Decision trees and decision forest [4] are very popular classifiers used in many real world applications. Although there are many types of decision trees, generally internal nodes have associated properties that guide the process of classification. Leaves represent a class, which determines the final classification. To classify a query object the classifier follows a path from the root node to a leaf according to the associated properties. The class assigned to the query is the class of the leaf node at the end of the path. This way, the conjunction of the properties in each path is one of the discriminative regularities in the respective class. Moreover, every tree contains an implicit set of hierarchically related regularities. For example, the tree appearing in Fig. 1 implicitly contains the hierarchy shown in Fig. 2.

As you can see, in a decision tree, discriminative regularities appear as conjunctions of properties. It is important to underline that the structure of properties depends on the type of tree and the induction algorithm. Most algorithms for tree induction try to find balanced trees with few levels, so they usually find regularities with approximately the same small number of

Table 1
Two-class problem with four Boolean features, six objects, and two classes.

Object	Tall	Fever	Male	Pain	Class
o1	True	True	True	False	Sick
o2	True	False	False	True	Sick
o3	True	True	True	True	Sick
o4	True	True	False	False	Healthy
o5	False	True	False	False	Healthy
o6	False	True	False	True	Healthy

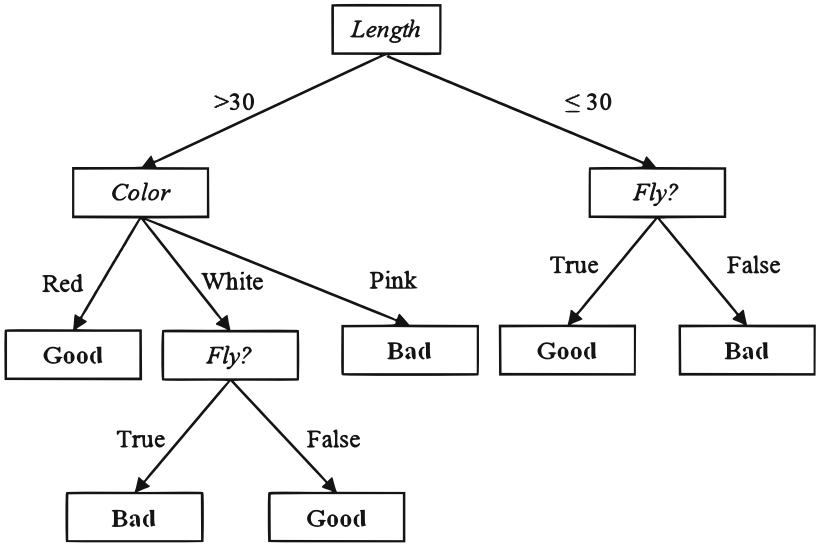

Fig. 1. Example of a decision tree.

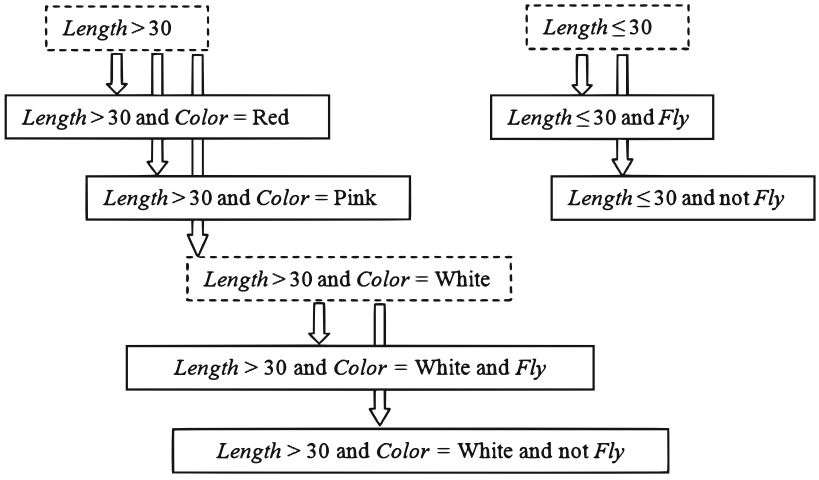

Fig. 2. Hierarchy of the regularities implicit in the tree of Fig. 1. Non-discriminative regularities appear with dotted lines.

properties. This reasoning is similar for a forest, but we obtain a different family of regularities from each tree.

In 1999 Dong and Li [5] introduce a new family of supervised classifiers, based on the concept of *emerging pattern* (EP). An *emerging pattern* is a combination of feature values that appears significantly more in a class than in the remaining classes. Although the authors did not mention it, this concept is similar to the complex features used in KORA-Ω [15].

There are two families of emerging pattern-based classifiers. The first family, introduced by Li et al. [16], searches patterns that match the query object in the training sample during the classification stage. They use border-processing operators (MaxSelector, IntersecOperation and DiffOperation) created by the authors to speed up the pattern searching. A border is a two-bound structure $\langle L,R\rangle$, which succinctly represents all the emerging patterns in the query object. Finally, they add the score of the patterns per class to determine the winning class. This way, no global pattern searching process takes place during the training stage. In [16] the authors consider this strategy as optimal, because it can handle dynamic databases and it is more scalable than the full search strategy. Nevertheless, it has some serious drawbacks. First, repeating the search process for every query object makes the classification stage slow, and we cannot apply any global EPs filtering or selection strategy. Second, in general, this whole family finds patterns expressed using two kinds of items. For numerical features, the items have the form $feature \in [value-\alpha, value+\alpha]$, being α a global threshold specified by the user. This makes the determination of the α value—which is the same for all features—a difficult task. For non-numerical features, the items have the form ($feature = value$).

The second emerging pattern-based classifier family, introduced by Bailey et al. [17] in 2002, makes a global emerging pattern search in the training stage. They make a previous discretization of all numerical features, so resulting patterns use only items with the structure ($feature = value$). The discretized objects are ranked using the information of their attribute values, and they are represented in a multi-value tree structure, which is a tree-based representation of the complete training sample. Finally, the authors make a depth-first traverse of the tree to extract patterns, using a subtree merging algorithm to allow extracting all the patterns.

The a priori discretization allows us to find all the emerging patterns efficiently. However, it may not be a feasible solution due to information loss and limited possibility of transformed data interpretation [18]. Moreover, not all numerical features are reasonably discretizable. A clear example appears when the similarity between two different feature values is given in terms of the absolute value of their difference. Consequently, discretizing this kind of feature might have the undesirable result of including two highly similar values in different discretized subsets.

Fan and Ramamohanarao [14] introduced in 2006 a new type of emerging pattern that allows handling some level of noise in the training data. The *noise tolerant emerging pattern* can support objects in other classes if the number of objects is below a predefined threshold, and these objects are considered noisy objects (a similar solution was previously introduced for KORA [19]). They present a wide experimental comparison of a classifier built using those patterns against many popular classifiers, attaining better results in accuracy. They also show that an EP-based classifier needs many patterns to work, even in small databases. For example, the *annealing* database [20] has 998 objects and 5891 *jumping emerging patterns* (JEPs), while the *cleveland* database [20] has 303 objects and 8281 JEPs. A *jumping emerging pattern* is an emerging pattern with no support in other classes.

In 2008, Kobylinski and Walczak [18] try to remove the initial discretization introducing a new kind of pattern: emerging pattern with occurrence count. They show significant improvements in classifier accuracy on a particular image classification task, but the number of extracted patterns increases dramatically. For example, in the *flower/food* database the number of patterns increases from 9 to 14444. This way, the resulting model is hard to understand by the user.

Terlecki and Walczak in 2008 [21] introduce two adaptive versions of classifiers based on emerging patterns, which dynamically modify the classifier to achieve better results. Although they present better results in accuracy on several databases, it is not due to better patterns, since they use a traditional pattern finding procedure.

In general, classifiers that use discriminative regularities obtain good accuracies in repository and real world problems [14]. However, as we have pointed out in this section, they all have important drawbacks that might affect their accuracy and/or efficiency. The algorithm introduced in the next section tackles these drawbacks, because it mines discriminative regularities in the training stage without any a priori discretization of numerical features, and it includes a pattern filtering post-processing.

3. The LCMine algorithm

In this section, we introduce the LCMine algorithm (Fig. 3) for extracting discriminative regularities in databases with mixed and incomplete data, which has two main components: a searching procedure and a filtering procedure. We also propose a classifier based on the regularities.

Let us consider a supervised classification problem with n objects distributed in r classes. Each object o is described by $x_1, x_2, \ldots x_t$ features, where x_i is defined in the corresponding domain D_i, $i=1,\ldots,t$. Thus $o \in D_1 \times D_2 \times \cdots D_t$.

In this paper, we consider discriminative regularities represented as *logical complexes* (l-complexes), a concept introduced by Michalski [22] for conceptual clustering. A logical complex is a conjunction of selectors. A selector is a relational statement $[x_i \# R_i]$, where R_i is a value or a set of values belonging to the domain of feature x_i, and # is a relational operator [22]. The relational operators used for non-numerical data are $=, \neq, \in, \notin$ and for numerical data are $>, <, \geq, \leq, \in, \notin$. Some examples of selectors are:

- $[color \in \{red, pink, white\}]$,
- $[age \geq 20]$,
- $[height \neq tall]$,
- $[weight \in (123.2, 170.5)]$.

A logical product of selectors $\bigwedge_{i \in I}[x_i \# R_i]$ is called a *logical complex* (l-complex) [22], where I is a set of feature indexes. An object o satisfies an l-complex—or an l-complex *covers* an object o—if the feature values in o satisfy all the selectors in the l-complex. For example, given the features *color*, *age*, *height* and *weight*, object $o_1 = (red, 34, tall, 150.1)$ satisfies the l-complex

$$[Color \in \{red, pink, white\}] \wedge [age \geq 20]$$

however, it does not satisfy the l-complex

$$[Height \neq tall] \wedge [age \geq 20]$$

This way, an l-complex can be viewed as an exact symbolic representation of the set of objects which satisfies it [22]. For example, the last l-complex in the example represents those objects whose *height* is different from *tall* and whose *age* is greater or equal to 20.

We chose l-complexes to represent the discriminative regularities found by LCMine, because all the reviewed discriminative regularities can be represented using them.

3.1. Searching procedure

The searching procedure extracts l-complexes from a set of different decision trees induced from the training sample. The induction procedure is similar to traditional methods for building decision trees. However, we explore more candidate splits than classic methods do in order to look for properties that better describe the training sample in terms of accuracy and simplicity. We understand simplicity as the number of conjunctions and features involved in a l-complex. Therefore, the fewer conjunctions and features the l-complex has, the simpler the l-complex is.

To guarantee diversity among trees we select a trade-off between the best tree (the one with the highest gain in all splits) and the generation of all possible trees, because the first is unique, and the latter is hard to apply to non-trivial problems given its time complexity. Our algorithm *BuildTree*, unlike traditional methods for building decision trees, expands the best s candidate splits. The value s is decreased according to the node level. This allows higher diversity in upper nodes, where good candidate splits are more frequent, reducing the diversity in lower nodes, where it is common to find fewer good splits. In our experiments, we expanded the best five splits in the root node, the best four in the next level, and so on (5, 4, 3, 2), and we selected the best split in lower levels. This way, we created $5 \cdot 4 \cdot 3 \cdot 2 = 120$ different trees.

The algorithm LCMine (Fig. 3) calls, for each different $\{k_{level}\}$ values, the procedure BuildTree (Fig. 4), obtaining a family of diverse trees. BuildTree generates all the candidate splits

Data: T – object collection to build the tree
Result: $ResultEP$
$EP \leftarrow \emptyset$;
// The k_i values control which of the best candidate splits is selected for expanding the tree nodes, according to each node level
for $k_1 \leftarrow 1$ **to** 5 **do**
  for $k_2 \leftarrow 1$ **to** 4 **do**
    for $k_3 \leftarrow 1$ **to** 3 **do**
      for $k_4 \leftarrow 1$ **to** 2 **do**
        $Tree \leftarrow BuildTree(T, \{k_1, k_2, k_3, k_4\})$;
        $EP \leftarrow EP \cup ExtractPatterns(Tree)$
      end
    end
  end
end
$EP \leftarrow RemoveDuplicates(EP)$;
$EP \leftarrow Simplify(EP)$;
$ResultEP \leftarrow LCS(T, EP)$

Fig. 3. Algorithm LCMine.

Data: T – object collection to build the tree, l – level in the tree of the resulting node (1 is the default value), $\{k_{level}\}$ – set of k values for each level
Result: N – decision node
while T *has more than one object in the non-majority classes* **do**
  Generate all candidate splits S_i as explained before;
  Calculate $ig(S_i)$, the information gain using entropy [6];
  Sort S_i in descending order according to its information gain $ig(S_i)$;
  Find S', the $k_l{}^{th}$ element of the sorted S_i collection;
  Find the child node subsets T_{ch}, according to the split S';
  Create a decision node N using S', with the appropriate number of children;
  For each child node ch call $BuildTree(T_{ch}, l+1, \{k_{level}\})$;
end

Fig. 4. Algorithm BuildTree.

according to the type of feature. The split types correspond to the different types of selectors we want to include in our patterns, in the following way:

- For non-numerical features:
  - If the attribute has exactly two values, BuildTree creates a single split with two children with the properties $feature = v_1$ and $feature = v_2$, respectively.
  - Else, BuildTree creates all these candidate splits:
    - For each value v_i, an split $feature = v_i$ and $feature \neq v_i$.
    - A single split with a child for each feature value with the property $feature = v_i$.
    - A split with a child per class. In the child node corresponding with class C, group in V_C all the values that appear more in class C than in its complement. An extra node groups the values with no correlated class. Each child node uses the property $feature \in V_C$.
- For numerical features BuildTree sorts the values and splits them as in [6], creating a division for each value v with the properties $feature \leq v$ and $feature > v$, respectively.

Each split represents a simple one-feature property we can use to describe a subset of the objects in the training sample. The splits selected in each level are those which usually lead to smaller and more balanced trees, so the extracted l-complexes are simpler.

We handle missing values introducing a penalizing factor in the information gain calculation. During the construction of a tree, when a split is evaluated, all the objects having a missing value in the features associated to this split are grouped in a virtual child node with maximum entropy. This way it is not likely that a feature with many missing values is selected as a good candidate.

LCMine extracts all the l-complexes from the trees, which are the conjunctions of properties in all the paths from the root node to the leaves. After removing duplicated patterns, we simplify each l-complex to obtain structures that are more compact, by joining redundant selectors and deleting duplicated selectors. Examples of redundant selectors are the following:

- $age > 30$ and $age > 50$, which are simplified to $age > 50$,
- $age > 30$ and $age \leq 50$, which are simplified to $age \in (30,50]$.

Finally, LCMine filters the resulting patterns using procedure LCSF (Fig. 5). It assigns to each l-complex α a weight w_α equal to the amount of objects that it covers in its own class.

3.2. Filtering strategy

The problem of filtering l-complexes is a challenge because of two main reasons:

(1) subjective nature of the definition of a "good" l-complex,
(2) total amount of possible l-complex subsets is 2^p where p is the number of l-complexes.

In this paper, we introduce a filtering strategy based on *redundancy* reduction in the l-complexes. We compute the *redundancy* of each l-complex as the amount of objects covered in the training dataset that are also covered by other l-complex(es), defined over the same feature set. Having highly redundant l-complexes might cause biased classifiers; for example, classifiers that usually tend to classify most of the objects in the same class.

In order to show why we take into account l-complexes defined over the same feature set we will make use of the following example. Suppose two l-complexes $\alpha_1 = ([x_1 = large] \wedge [x_3 \leq 0.4] \wedge [x_4 > 3])$ and $\alpha_2 = ([x_2 > 27] \wedge [x_5 \neq white] \wedge [x_7 > 6.8])$, covering the same objects in the training dataset. Notice that these l-complexes are not defined over the same feature set ($\{x_1,x_3,x_4\} \neq \{x_2,x_5,x_7\}$). If we consider that α_1 is redundant with respect to α_2 and consequently we remove α_1, this removal might be a wrong decision. For example, the object $e = (large; 25; 0.1; 7; blue; true; 5)$ might be now misclassified because it does not satisfy α_2. Nevertheless it satisfies α_1, which was removed from the final l-complex set. A proper algorithm, like the one that we propose, can avoid this undesirable behavior taking into account the redundancy in l-complexes defined over the same feature set.

We based the proposed algorithm on the hypothesis that a good l-complex subset must reduce the redundancy as much as possible. The L-Complex Selection by Feature set (LCSF, Fig. 5) algorithm applies the L-Complex Selection (LCS, Fig. 6) algorithm on each l-complex group, defined over the same feature set, selecting a small size l-complex subset that covers the same objects as the whole group does. LCS scores each l-complex using the following equation:

$$Score_\alpha = \frac{|S(\alpha, class(\alpha), T)|}{\max_{\substack{z \in Z \\ z \neq class(\alpha)}} \{|S(\alpha, z, T)|\}} \quad (1)$$

where Z is the set of classes, $S(\alpha,z,T)$ is the subset of objects from T with class label z that satisfies the l-complex α, and $|A|$ is the cardinality of the set A. This weighting scheme favors l-complexes covering more objects of its class and fewer objects from different classes.

Data: T – Training sample, L – set of l-complexes
Result: L' – selected l-complexes
$L' \leftarrow \emptyset$
Split L in groups G_i, where l-complexes in the same group are defined over the same feature subset ;
foreach $G \in G_i$ **do**
  foreach C *in* G **do**
    $L'' \leftarrow \{\alpha \in G : class(\alpha) = C\}$;
    $L' \leftarrow L' \cup LCS(T, L'')$;
  end
end

Fig. 5. L-complex selection by feature set algorithm (LCSF).

Data: T – Training sample, L – set of l-complexes

Result: L' – selected l-complexes

foreach *l-complex* $\alpha \in L$ **do**
  calculate $score_\alpha$
end

Sort L in descending order according to $score_\alpha$;

$L' \leftarrow \emptyset$;

foreach *l-complex* α *in the sorted set* L **do**
  if *there is any object that satisfies* α *and does not satisfy any l-complex already in* L' **then**
    $L' \leftarrow L' \cup \{\alpha\}$
  end
end

Fig. 6. L-complex selection algorithm (LCS).

3.3. Discriminative regularity-based classifier

In order to test the quality of the discriminative regularities found by LCMine, we used them to build a supervised classifier. Like traditional emerging pattern-based and KORA type classifiers, we used a scoring function. Given a query object q this function computes the total score for class C_i aggregating the weight of the l-complexes that cover q. We computed the score per class using the following equation:

$$score(q,C_i) = \sum_{\substack{\alpha \in D_i \\ q \subseteq \alpha}} w_\alpha \tag{2}$$

where w_α is the weight of regularity α (assigned by LCMine as the amount of objects that α covers on its own class), D_i is the collection of all discriminative regularities for class C_i and $\subseteq$ represents the satisfiability relation.

Finally, the class with the highest score is the output of the classifier. If votes tie, or no vote exists, the classifier refuses to classify, and such abstentions count as errors.

3.4. LCMine and overfitting

While a complex model may allow a very good classification of the training samples, it is likely to assign wrong classes to unseen objects. This situation is known as overfitting [23]. As complex models tend to be overtrained, it may seem that an LCMine classifier, using the mined l-complexes, would be overfitted. Nevertheless, some characteristics of the proposed classifier avoid this drawback:

- Despite decision trees, where a single property (branch) assigns the class, in the LCMine classifier different l-complexes influence the final classification result. This way, the votes of too specific l-complexes are usually lower than the votes of more general l-complexes, which are more likely to appear in a query object.
- Using a wider set of operators allows us to find more general l-complexes, which cover more objects in the universe. This way, many unseen objects are correctly classified using this kind of l-complexes.

4. Experimental results

To show the performance of LCMine, we conducted a set of experiments on 13 databases from the UCI Repository of Machine Learning [20]. The selected databases, described in Table 2, have different characteristics in size (column *#obj*), class distribution (column *class distrib*), feature types (columns *#numerical features* and *#non-numerical features*) and percentage of objects with missing values (column *Missing values*).

Table 2
Description of the databases used for the experiments.

Database	#Obj	Class distrib (%)	# Features		Missing values
			Numerical	Non-numerical	
breastwis	699	65/35	–	9	<1%
cmc	1473	43/23/34	2	7	–
cleveland	303	54/46	6	7	<1%
creditscr	690	45/55	6	9	5%
cylinder	512	61/29	20	20	10%
hepatitis	155	79/21	6	13	6%
horsecolic	368	63/37	7	15	23.8%
iris	150	33/33/33	4	–	–
mp1	432	50/50	–	6	–
mp2	432	62/38	–	6	–
mp3	432	50/50	–	6	–
vote	435	61/39	–	16	5.63%
wpbc	198	76/24	32	–	<1%

For comparisons, we selected some popular classifiers, belonging to different paradigms: Two k Nearest Neighbors classifiers [10] with different values of k, Bagging and Boosting [24], Random Forest [4], C4.5 [6] and support vector machine (SVM) [11]. We conducted all experiments in a Compaq Presario V3000 (AMD Sempron 3600+2 GHz, 1 Gbyte RAM) running Microsoft Windows XP. We calculated the accuracy using 10 fold cross validation. Accuracy values appearing in tables are the average accuracy over the 10 folds. For all methods except ours, we used the Weka [25] implementations, with their default parameter values. It is worth mentioning that the reader can find different accuracies, reported in the literature, for these classifiers in the same databases. This is mainly because the strong random component of the cross validation, or because some experts fine-tuned the parameters of a particular classifier.

Table 3 presents accuracy results of the compared classifiers on the tested databases. As the reader can see, the proposed classifier achieves the best accuracy on most databases. Table 4 contains a pairwise comparison of the tested classifiers. Each cell contains the number of databases where the classifier in the *row* wins, loses or ties with respect to the classifier appearing in the *column*. For detecting ties (statistically similar results), we use a pairwise T-Test with a significance level of 0.05 [26].

Table 3
Accuracy results of the compared classifiers on the tested databases.

Database	3nn	7nn	Boosting	Bagging	RandFor	C4.5	SVM	LCMine
breastwis	96.53	95.71	95.58	95.57	95.93	96.47	96.99	**97.43**
cmc	47.06	48.19	42.52	53.63	50.95	50.90	48.20	**55.58**
cleveland	82.51	81.83	84.15	79.88	78.57	78.19	**84.51**	82.20
creditscr	84.15	86.24	86.28	85.94	85.02	85.08	86.30	**86.77**
cylinder	70.16	71.19	72.86	60.93	72.19	78.51	80.89	**81.79**
hepatitis	**86.04**	85.08	83.58	84.52	81.79	82.37	85.16	79.41
horsecolic	81.78	83.95	80.72	84.77	85.04	**85.60**	78.80	84.28
iris	96.59	97.21	**97.75**	94.00	95.83	95.20	96.00	96.45
mp1	81.02	76.85	75.00	50.00	75.69	88.89	50.00	**95.60**
mp2	61.34	65.28	60.65	55.09	65.05	69.88	50.46	**75.00**
mp3	88.19	92.82	97.22	50.00	97.22	96.30	50.00	**97.45**
vote	91.97	92.2	94.72	95.18	**96.10**	**96.10**	95.87	94.50
wpbc	72.30	75.95	71.02	**78.78**	74.69	75.51	75.88	68.37

The best result for each database appears bolded.

Table 4
Comparative evaluation of the classifier in the row (win/lose/tie) with respect to the classifier in the column.

	3nn	7nn	Boosting	Bagging	RandFor	C4.5	SVM	LCMine
3nn		1/5/7	3/4/6	6/4/3	3/7/3	2/8/3	4/4/5	2/8/3
7nn	5/1/7		4/3/6	5/3/5	2/3/8	3/6/4	4/3/6	2/6/5
Boosting	4/3/6	3/4/6		6/3/4	1/4/8	2/6/5	3/3/7	2/5/6
Bagging	4/6/3	3/5/5	3/6/4		3/4/6	3/4/6	4/2/7	2/6/5
RandFor	7/3/3	3/2/8	4/1/8	4/3/6		0/3/10	5/3/5	2/5/6
C4.5	8/2/3	6/3/4	6/2/5	4/3/6	3/0/10		5/3/5	2/5/6
SVM	4/4/5	3/4/6	3/3/7	2/4/7	3/5/5	3/5/5		3/5/5
LCMine	8/2/3	6/2/5	5/2/6	6/2/5	5/2/6	5/2/6	5/3/5	

Table 5
Accuracy results of SJEPC and LCMine on the tested databases.

Database	SJEPC	LCMine	# Pat SJEPC	# Pat LCMine
breastwis	96.28	**97.43**	223	72
cmc	7.00	**55.58**	9	252
cleveland	78.24	**82.20**	733	41
creditscr	82.61	**86.77**	1186	114
cylinder	64.28	**81.79**	46	77
hepatitis	**83.21**	79.41	1695	27
horsecolic	78.80	**84.28**	1656	206
iris	78.00	**96.45**	12	11
mp1	86.81	**95.60**	125	25
mp2	71.06	**75.00**	201	56
mp3	93.52	**97.45**	119	27
vote	93.12	**94.50**	1773	25
wpbc	0	**68.37**	0	21

The best result for each database appears bolded.

The results in Table 4 present that our method defeated every other single classifier used in the experiments in at least five databases, and lost in at most three of them. Finally, we compared our classifier with one of the most accurate pattern-based classifiers, the Strong Jumping Emerging Pattern Classifier (SJEPC) [14], with some optimization and fixes introduced in [27]. The results, appearing in Table 5, reveal that LCMine is a more accurate classifier, using fewer patterns. Since both methods use similar classifiers, the quality difference is mainly due to the quality of the patterns used.

We should note that the SJEPC accuracy in the databases *wpbc*, *iris* and *cmc* are unexpectedly low. This behavior is mainly due to the discretization method used in the algorithm [28], which in those databases converts the values of most of the numerical features into a single non-numerical value.

The last column in Table 5 presents the number of l-complexes used by our classifier. This is at least five times less than the number of objects, so we are building a model simpler than those built by other pattern-based methods, which usually needs more patterns than objects to achieve good results. This is very important, from a user's point of view, because it makes the model easier to understand, and it is possible to explain each classification result as conjunctions of a few properties. For example, in one of the folds of the Iris database, we found the discriminative regularities presented in Table 6. Therefore, nine of them are enough for accurately describing the 150 objects in this database.

5. Conclusions

The main contribution of this paper is an efficient algorithm for finding discriminative regularities in a training sample with mixed and incomplete data for supervised classification. This algorithm (LCMine) is based on diverse decision tree induction and a filtering post-processing stage, which allows us to find a reduced set of high quality discriminative properties for each class. Based on these regularities, we built a classifier, similar to those based on emerging patterns or complex features. Our classifier attains higher accuracy than the most accurate pattern-based classifier, but using fewer patterns, which is desirable in situations when an interpretation of the results is needed. Based on our results and because we used a similar classifier, we can conclude that our classifier attains higher accuracy because it uses more representative patterns of their respective classes.

Additionally, experimental results show that the classifier built using the mined l-complexes attains superior results than other state-of-the-art classifiers in most of the tested databases.

Table 6
Discriminative regularities found by LCMine in the Iris database.

Class	Regularity	Support per class		
		0	1	2
0	$[PetalWidth \leq 0.60]$	45		
1	$[PetalWidth \in [0.60,1.60]] \wedge [PetalLength \leq 4.90]$		44	
	$[SepalWidth > 2.50] \wedge [PetalWidth \leq 1.70] \wedge [PetalLength \in [1.90,5.00]]$		32	
	$[SepalWidth > 3.00] \wedge [PetalLength \in [4.40,5.00]] \wedge [PetalWidth > 1.50]$		3	
	$[PetalLength \in [4.70,5.10]] \wedge [PetalWidth <= 1.70] \wedge [SepalWidth > 2.20]$		4	1
2	$[PetalWidth > 1.70]$		1	39
	$[PetalLength > 5.00]$		1	37
	$[SepalWidth \leq 3.10] \wedge [PetalWidth \in [1.60,1.80]]$		1	10
	$[SepalWidth \leq 3.00] \wedge [PetalLength \in [4.70,5.00]] \wedge [PetalWidth \leq 6.30]$		1	6

In the near future, we will face the problem of high dimensionality databases.

Acknowledgment

The authors want to thank the anonymous reviewers for their valuable suggestions, which significantly improved the quality of this paper.

References

[1] J. Ruiz-Shulcloper, M.A. Abidi, Logical combinatorial pattern recognition: a review, in: S. Pandalai (Ed.), Recent Research Developments in Pattern Recognition, Transword Research Networks, USA, 2002, pp. 133–176.
[2] J.F. Martnez-Trinidad, A. Guzmin-Arenas, The logical combinatorial approach to pattern recognition, an overview through selected works, Pattern Recognition 34 (2001) 741–751.
[3] M.N. Bongard, Solution to geological problems with support of recognition programs, Sov. Geol. 6 (1963) 33–50.
[4] T.K. Ho, The random subspace method for constructing decision forests, IEEE Trans. Pattern Anal. Mach. Intell. 20 (8) (1998) 832–844.
[5] G. Dong, J. Li, Efficient mining of emerging patterns: discovering trends and differences, in: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, San Diego, CA, USA, 1999, pp. 43–52.
[6] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 1993. ISBN: 1558602402.
[7] M.W. Craven, J. Shavlik, Extracting tree-structured representations of trained networks, Advances in Neural Information Processing Systems, vol. 8, MIT Press, Cambridge, MA, 1996.
[8] D. Martens, B. Baesens, T. Van Gestel, J. Vanthienen, Comprehensible credit scoring models using rule extraction from support vector machines, European Journal of Operational Research 183 (3) (2007) 1466–1476.
[9] S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice Hall PTR, 1998.
[10] B.D. Dasarathy, Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques, IEEE Computer Society Press, Los Alamitos, CA, 1991.
[11] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.
[12] R. Kothari, M. Dong, Decision trees for classification: a review and some new results, in: S.K. Pal, A. Pal (Eds.), Pattern Recognition. From Classical to Modern Approaches, World Scientific, Singapore, 2001, pp. 169–184.
[13] M.J. Zaki, C.-J. Hsiao, Efficient algorithms for mining closed itemsets and their lattice structure, IEEE Trans. Knowl. Data Eng. 17 (4) (2005) 462–478.
[14] H. Fan, K. Ramamohanarao, Fast discovery and the generalization of strong jumping emerging patterns for building compact and accurate classifiers, IEEE Trans. Knowl. Data Eng. 18 (6) (2006) 721–737.
[15] L. De la Vega-Doria, J.A. Carrasco Ochoa, J. Ruiz-Shulcloper, Fuzzy kora-omega algorithm, in: Proceedings of Sixth European Congress on Intelligent Techniques and Soft Computing, Aachen, Germany, 1998, pp. 1190–1194.
[16] J. Li, G. Dong, K. Ramamohanarao, Instance-based classification by emerging patterns, in: D.A. Zighed, H.J. Komorowski, J.M. Zytkow (Eds.), Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases, Springer-Verlag, Lyon, France, 2000, pp. 191–200.
[17] J. Bailey, T. Manoukian, K. Ramamohanarao, Fast algorithms for mining emerging patterns, in: Proceedings of the Sixth European Conference on Principles of Data Mining and Knowledge Discovery, Helsinki, Finland, Lecture Notes in Computer Sciences, vol. 2431, Springer-Verlag, 2002, pp. 187–208.
[18] L. Kobylinski, K. Walczak, Jumping emerging patterns with occurrence count in image classification, in: T. Washio (Ed.), Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2008, Osaka, Japan, Lecture Notes in Artificial Intelligence, vol. 5012, Springer-Verlag, London, UK, 2008, pp. 904–909.
[19] A. Keilis-Borok, A. Soloviov, Pattern recognition: general description, in: Workshop in Non-linear Dynamics and Earthquake Prediction, International Center for Science and High Technology, Trieste, Italy, 1991, pp. 1–14.
[20] C. Merz, P. Murphy, Uci repository of machine learning databases, Technical Report, University of California at Irvine, Department of Information and Computer Science, 1998.
[21] P. Terlecki, K. Walczak, Adaptive classification with jumping emerging patterns, in: G. Wang (Ed.), RSKT 2008, Lecture Notes in Artificial Inteligence, vol. 5009, 2008, pp. 39–46.
[22] R.S. Michalski, R. Stepp, Revealing conceptual structure in data by inductive inference, in: D. Michie, J.E. Hayes, H.H. Pao (Eds.), Machine Intelligence, vol. 10, Ellis Horwood Ltd., New York, 1982, pp. 173–196.
[23] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, second ed., Wiley-Interscience, New York, NY, 2000.
[24] L.I. Kuncheva, Combining Pattern Classifiers. Methods and Algorithms, Wiley-Interscience, Hoboken, New Jersey, 2004.
[25] I. Wittn, E. Frank, L. Trigg, M. Hall, G. Holmes, S. Cunnigham, Weka: practical machine learning tools and techniques with java implementations, in: Emerging Knowledge Engineering and Connectionist-based Information Systems, 1999, pp. 192–196.
[26] T.G. Dietterich, Approximate statistical tests for comparing supervised classification learning algorithms, Neural Comput. 10 (7) (1998) 1895–1923.
[27] P. Terlecki, K. Walczak, Efficient discovery of top-k minimal jumping emerging patterns, in: C. Chang (Ed.), RSCTC, Lecture Notes in Artificial Intelligence, vol. 5306, 2008, pp. 438–447.
[28] U. Fayyad, K. Irani, Multi-interval discretization of continuous-valued attributes for classification learning, in: Proceedings of 13th International Joint Conference on Artificial Intelligence (IJCAI), 1993, pp. 1022–1029.

About the Author—MILTON GARCÍA-BORROTO graduated from Las Villas Central University, Cuba, in 2000. He received his M.Sc. degree in 2007 from the National Institute of Astrophysics, Optics and Electronics, Mexico, where he continues his studies toward a Ph.D. degree. His research interests are pattern recognition, intelligent systems, machine learning and biometry.

About the Author—JOSÉ FCO. MARTÍNEZ-TRINIDAD received his B.S. and M.Sc. degrees in Computer Science from the Autonomous University of Puebla, Mexico, in 1995 and 1997, respectively, and his Ph.D. degree in the National Polytechnic Institute, Mexico, in 2000. Professor Martinez-Trinidad edited/authored four books and over fifty papers, on subjects related to pattern recognition.

About the Author—JESÚS ARIEL CARRASCO-OCHOA received his Ph.D. in Computer Science from the Center for Computing Research of the National Polytechnic Institute, Mexico, in 2001. He works as full-time researcher at the National Institute for Astrophysics, Optics and Electronics, Mexico. His current research interests include pattern recognition, feature and prototype selection, and clustering.

About the Author—MIGUEL ANGEL MEDINA-PÉREZ graduated from University of Ciego de Ávila in 2007 and received his M.Sc. degree in Applied Informatics from University of Ciego de Ávila in 2007. Currently, his research interests include pattern recognition and biometry.

About the Author—JOSÉ RUIZ-SHULCLOPER graduated from University of Havana in 1972 and received his Ph.D. degree in mathematics from the Moscow State University in 1978. Since 1978 he is the leader of the research area on logical combinatorial pattern recognition and author of more than 100 scientific publications on pattern recognition and data mining.