Searching Extended Emerging Patterns for Supervised Classification

By

MSc. Milton García Borroto

Dissertation submitted in partial fulfillment of the requirements for the degree of

DOCTOR IN COMPUTER SCIENCE

at the

National Institute for Astrophysics, Optics and Electronics
September 2010,
Tonantzintla, Puebla, México

Advisor:
Dr. José Francisco Martínez Trinidad

Abstract

For many learning tasks, a high accuracy is not the only desired characteristic of a supervised classifier; A classifier should also be easily comprehensible by humans. Although higher classification accuracies are usually obtained at the expense of classification comprehensibility, Emerging Pattern classifiers are both accurate and easy to understand. The main contribution of this dissertation is the introduction of two new kinds of emerging patterns, which are more expressive than traditional definitions: *Extended Crisp Emerging Patterns* and *Fuzzy Emerging Patterns*. The higher expressiveness of the new patterns allows to obtain more accurate classifiers, without sacrificing understandability. Another contribution of this dissertation is a collection of algorithms for mining the new kinds of patterns from a database containing mixed and incomplete data. The classifiers proposed in this dissertation, using the new patterns, attain higher accuracy than traditional emerging pattern classifiers and other comprehensible classifiers, while they are competitive with state-of-the-art non-comprehensible classifiers. The selection of the classifier to be used in a particular problem depends on the type of patterns (crisp or fuzzy) the user wants to obtain, and a tradeoff among accuracy, complexity, and classification speed.

Resumen

Para muchas tareas de aprendizaje, una alta eficacia no es la única característica deseada; el clasificador debe ser fácilmente entendible por los humanos. Aunque una elevada eficacia de clasificación se obtiene usualmente en detrimento de la comprensibilidad, los clasificadores basados en Patrones Emergentes son eficaces y fáciles de entender. La contribución principal de esta disertación es la introducción de dos nuevos tipos de patrones emergentes, más expresivos que los tradicionales: Patrones Emergentes Duros Extendidos y Patrones Emergente Difusos. El mayor nivel de expresividad de estos patrones permite obtener clasificadores más eficaces. Otra contribución de este trabajo es una colección de algoritmos para extraer los nuevos tipos de patrones a partir de una base de datos que contiene datos mezclados e incompletos. Los clasificadores basados en los patrones propuestos en esta disertación alcanzan mayor eficacia que los clasificadores tradicionales basados en patrones emergentes y que otros clasificadores comprensibles, siendo además competitivos con otros clasificadores del estado del arte que no son comprensibles. La selección del clasificador a utilizar en un problema en particular depende del tipo de resultado que el usuario desee obtener, así como del compromiso deseado entre eficacia, complejidad y velocidad de clasificación.

Acknowledgments

I would like to express my deep and sincere gratitude to my supervisor, Dr. José Francisco Martínez Trinidad. His knowledge and his logical way of thinking have been of great value for me. His understanding, encouraging and personal guidance have provided a good basis for the present thesis.

I wish to express my warm and sincere thanks to Dr. José Ruiz Shulcloper, Head of the Advanced Technologies Application Center, Havana, Cuba, who introduced me to the field of pattern recognition. His ideals and concepts have had a remarkable influence on my entire career in this field.

I would like to thanks the members of my revision committee for their detailed review, constructive criticism and excellent advice during the preparation of this thesis. Thanks to Dr. Eduardo Morales Manzanares, Dr. Jesús Ariel Carrasco Ochoa, Dr. Manuel Montes y Gómez, Dra. María del Pilar Gómez Gil, and Dr. Gouzhu Dong.

I also wish to thank Mrs. Dania Yudith Suárez Abreu for revising the English of my manuscript and publications.

During this work I have collaborated with many colleagues and friends for whom I have great regard, and I wish to extend my warmest thanks to all those who have helped me with my work in the Computer Science Laboratory, Bioplants Center, Ciego de Ávila, Cuba.

I owe my loving thanks to my wife Yalily Talabera Díaz and my daughters Ana Flavia and Mariana. They have lost a lot due to my research abroad. Without their encouragement and understanding it would have been impossible for me to finish this work. My special gratitude is due to my mother, fathers, and my families for their loving support. My loving thanks are due to José Alberto and his family, Alejandro, and Toño. They let me own a happy family in Mexico.

I also want to acknowledge the extraordinary help provided by all the INAOE staff members.

Finally, I thank to the National Institute of Astrophysics, Optics and Electronics and to the CONACyT (doctoral scholarship 25275) for their support during the doctoral studies.

Contents

Chapter 1

Introduction

1.1 Introduction

Supervised classification (or just "classification") is the branch of Pattern Recognition[1] that finds the relations between object characteristics and a predefined set of classes, in order to predict the class of unseen objects. In this dissertation, we deal with problems where objects are described by *mixed and incomplete data* [49], because they can be simultaneously described by categorical and numerical features, with some feature values missing.

Because of the high diversity in pattern recognition problems, there is a large collection of techniques to find these relations, and a huge amount of class prediction algorithms based on them. Frequently, for a given problem, the user has to test different techniques and algorithms to select the most accurate. Nevertheless, there are some problems where a high accuracy is not the only desired characteristic of a supervised classifier; the classifier should be also easily understandable by the user.

In many domains, the lack of comprehensibility is an important drawback that may cause a reluctance to use certain algorithms. For example, when credit has been denied to a customer, the Equal Credit Opportunity Act of the US requires the financial institution to provide the reasons why the applica-

[1] An introduction to Pattern Recognition can be found in Chapter 2.

tion was rejected; indefinite and vague reasons for denial are illegal [48]. In some other fields, like medical diagnosis and mineral prospection, clarity and explainability are key user requirements.

An important family of understandable classifiers are the emerging pattern classifiers [59]. An emerging pattern (EP) is a discriminative pattern [2] whose support [3] increases significantly from a class to the remaining problem classes.

Example 1.1. *The pattern* $\left[PetalLength > 1.9\right] \wedge \left[PetalWidht > 1.6\right]$ *is an emerging pattern of the class* ***iris-virginica*** *in the popular Iris database, because its support in the iris-virginica class is significantly higher than its support in the remaining two classes.*

Although emerging pattern classifiers are widely used, showing a competitive behavior with respect to other classifiers [27], they have some serious drawbacks.

First, before mining the emerging patterns, these classifiers apply *data discretization* over all the numerical features. Data discretization is the process of transforming numerical features into a finite set of intervals, causing minimal loss of information [39]. However, discretizing individual features to extract later emerging patterns could have the following problems:

- Emerging patterns are combinations of values that must appear simultaneously, thus discretizing individual features could deteriorate the quality of the mined patterns. An example of this behavior appears in Table 1.1, where for *tae* and *wpbc* databases, the SJEP classifier (SJEPC) [27] transforms most numerical features into a single-valued categorical feature. This way, those features are virtually discarded because they can not appear in any pattern, and SJEPC cannot find enough patterns to accurately classify query objects.

- Discretization usually defines crisp boundaries, therefore it is possible that the object $(3, 5)$ matches a pattern, while $(3.001, 5)$ does not match the same pattern.

[2] For a comprehensive introduction to patterns and discriminative patterns see Section 2.6.
[3] The support of a pattern is the amount of objects that fulfill the pattern.

Table 1.1: SJEPC accuracy vs. 3-Nearest Neighbor accuracy in some UCI Databases

Database Name	SJEPC accuracy	3-Nearest Neighbor accuracy
autos	12.31	68.24
glass	20.43	69.03
iris	75.33	96.59
labor	55.33	90.67
lymph	43.86	85.90
tae	0	68.75
wpbc	0	72.30

Second, fast pattern miners use a simplified pattern representation. This simplification usually results in huge amounts of patterns, even for problems with few objects and features. Finally, filtering procedures are based on pattern properties like minimality [4] or covering objects in a single class, which may have two undesirable effects: deletion of important patterns and selection of useless patterns.

1.2 Description of the problem

Emerging patterns can capture useful contrasts among the problem classes [19], therefore they can be used to predict the class of unseen objects. Patterns covering a query object provide the classification support, in a simple and easy to understand language. For this reason, emerging pattern classifiers are valuable tools to solve real problems in fields like Bioinformatics [9, 56, 54], streaming data analysis [2], intruder detection [14], human activity recognition [33], anomaly detection in network connection data [11], rare event forecasting [31], and mining spatio-temporal relationships [12].

There are two families of Emerging Pattern-based classifiers, according to the stage where the pattern extraction takes place. The first family, introduced by Li *et al.* [43], searches in the training sample the patterns that match the query object during the classification stage. This way, no global pattern searching process takes place during the training stage. The second family, introduced by Dong and Li [19], searches all the patterns in the training stage.

[4] Minimality is defined over the subset inclusion relation.

Classifiers in the first family are appropriate to handle dynamic databases, because the addition or deletion of an object to the database could completely change all existing patterns. Finding all the emerging patterns in large databases could take a prohibitive amount of time, so these classifiers are also more scalable than those in the second family. Nevertheless, they have some serious drawbacks:

- Repeating the searching process for every query object could make the classification stage prohibitively slow. Additionally, the classifier cannot apply any global pattern filtering or selection strategy.
- For numerical features, this family of classifiers finds items with the form $Feature \in [value - \alpha, value + \alpha]$, being α a global threshold specified by the user. The determination of the α value, which is the same for all features [43], is a complex and difficult task.

Because of their drawbacks, the classifiers in the first family are not considered in this dissertation. Therefore, we will focus on the second family.

Extracting emerging patterns from a training sample is challenging, due to the following reasons:

1. The downward closure property[5] used for frequent itemset discovery does not hold for EPs, so algorithms like Apriori cannot be used.
2. In high-dimensional databases, there are many potential emerging pattern candidates. The problem of finding all emerging patterns is proved to be an NP-hard problem [65].
3. Continuous features cannot be rationally compared by the equality operator. For example, the values 3, 2.999 and 3.001 are not equal, but they can be probably found in the same pattern. On the other hand, a global discretization of numerical features could seriously degrade the classification accuracy, because very similar values could be assigned to different discrete values.

[5] A property fulfills the downward closure when, for every item X, if X does not fulfill the property, all other items more particular than X do not fulfill the property either. Section 2.6 contains more details about pattern mining and the downward closure.

4. The algorithms for extracting emerging patterns have a high sensitivity to the minimal support threshold value, therefore it could be very hard for the user to select a good value. The minimal support threshold is the minimal amount of objects that should support a pattern to be considered useful.

Most algorithms for emerging pattern mining have the goal of finding the patterns that satisfy a desired property: being supported by a single class, minimality over subset inclusion, or tolerance to noisy objects [27]. These algorithms have the following general steps:

1. Selection of the minimal support threshold μ. The following question arises:
   – *How can we infer a good μ value from the data?*

2. Global discretization of numerical features. The following question arises:
   – *How can we avoid the global discretization of numerical features?*

3. Representation of the transformed objects using a data structure[6] and traversing the structure to find all the emerging patterns. This is the slowest step in the algorithm, because the more the objects, the more complex and larger the structure. The following question arises:
   – *How can we extract a representative collection of high quality emerging patterns without a time consuming structure traversal?*

4. Pattern filtering. Common pattern filtering criteria include pattern minimality[7], pattern maximality, and reduced redundancy in object coverage[8]. The following question arises:
   – *How can we filter the pattern collection to obtain a high-quality non-redundant subset of patterns, without sacrificing classifier accuracy?*

5. Classification of unseen objects. The following question arises:
   – *How can we create more accurate classifiers based on the mined pattern?*

[6] Usually a tree-like representation of the objects in the training set.
[7] Over the support subset inclusion.
[8] Objects covered by the patterns.

1.3 Thesis proposal

Algorithms for mining emerging pattern use a very simplified language to represent the patterns: $\bigwedge [Feature = value]$. That is why all numerical features are discretized a priori, since they will be compared using the equality operator. Using this reduced language allows to use all the tools from frequent pattern discovery, a well formalized and mature branch of Data Mining. However, doing so could degrade the classifier accuracy, as the examples shown in Table 1.1.

As the hypothesis of this dissertation, we claim that it is possible to extract emerging patterns from a collection of diverse decision trees induced from the training data. This method avoids the global discretization step, allowing the patterns to be expressed with a richer set of properties that include operators like $<, \geq, \neq$. Patterns using an extended set of properties are more *expressive*[9], because they can express more selective properties than those used in previous emerging pattern classifiers. Classifying with these more expressive patterns is more accurate than previous comprehensible classifiers, and at least as accurate as state-of-the-art non-comprehensible classifiers.

In order to demonstrate our hypothesis we introduce two new types of emerging patterns: Extended Crisp Emerging Patterns and Fuzzy Emerging Patterns. For mining these patterns, we introduce a family of mining methods, which can be described by the following steps:

1. Induce a diverse decision tree.

2. Extract patterns from the induced decision tree. Each pattern corresponds to the conjunction of the properties from the root node to a leaf node.

3. If stop condition is not met, return to Step 1.

4. Merge the patterns extracted from all induced decision trees.

5. Filter patterns.

[9] A formal definition of pattern language expressiveness can be found in Section 4.2.

Our mining methods do not include a global discretization step, because they discretize only feature values appearing in the objects that belong to each tree node. The proposed mining method have the following parameters:

- Type of decision tree to be built: fuzzy or crisp.
- Induction algorithm to build the decision trees.
- Method to obtain diverse decision trees. Classical methods to induce decision trees obtain a single tree, which is not enough to find a representative collection of patterns.
- Stop condition. This condition evaluates if the patterns mined so far are representative enough for the database.

Traditional emerging pattern miners are able to find all the emerging patterns in a database. Nevertheless, decision tree-based miners do not usually find all the emerging patterns, but commonly obtain a good collection of high-quality patterns. This is supported by the following reasons:

- In databases containing numerical features, there is a finite number of traditional emerging patterns, but an infinite number of extended emerging patterns. Then, it is impossible to mine all the patterns.
- Decision trees split the database using the most discriminative properties first. If the method for obtaining diversity follows this rule, the patterns mined are the most discriminant among all the patterns. So, they are the best patterns for classification.
- The experimental results presented in this dissertation show that decision tree-based miners are more accurate than traditional miners over a significant database collection.

The first algorithm introduced in this dissertation is LCMine (Logical Complexes mining), which uses a collection of C4.5 [58] decision trees[10]. To generate diversity among the trees, LCMine selects different candidate splits at each tree level. LCMine filters the mined patterns using a novel algorithm, which significantly reduce the redundancy in the patterns coverage.

[10] An introduction to decision tree induction can be found in Section 2.8.

The second introduced algorithm is an enhanced version of LCMine, named CEPM (Crisp Emerging Pattern Mining). The main improvement in CEPM is to use the patterns mined so far to guide the construction of subsequent decision trees. To attain this, CEPM weights all the objects in the training sample after each iteration. Higher weights are assigned to objects which are not covered by the current set of patterns, while weights close to zero are assigned to objects covered by many patterns. The decision tree built on each iterations is the one that better represents the objects with higher weights.

One of the key parameters of every pattern based classifier is the minimal support threshold μ; This threshold indicates the minimal amount of objects that must support a pattern in order to be considered as a useful pattern. Finding a good value for μ is a hard task for the user, because the quality of the patterns is very sensitive to the μ value. That is why CEPM finds an accurate estimation of the minimal support threshold μ, by testing different values decrementally. The rationale of this procedure holds in the following property: if we classify using patterns mined with lower μ values, we obtain less accurate classifiers, but having lower abstention levels[11]. Following this property, CEPM infers two values:

- μ_{ini}: a high enough value, such that any classifier built with $\mu > \mu_{ini}$ is inaccurate.
- $minAbst$: minimal abstention level we can expect to attain with the current dataset.

Then, CEPM starts testing μ values decrementally, starting from μ_{ini}, until the built classifier attains an abstention level less than or equal to $minAbst$. As a consequence of this procedure, CEPM returns a set of emerging patterns with the highest support value associated with the lowest expected abstention level.

Additionally, we also introduce CascadeCEPM, a new method for building cascades of CEPM classifiers. This cascade is as accurate as CEPM classifier, but it is faster for most objects. In CascadeCEPM, the topmost classifier[12] is built with the highest μ value, while the bottommost classifier is built using

[11] A definition of classifier abstention can be found in Section 2.3.

[12] In a classifier cascade, the topmost classifier is the first one to be evaluated.

the lowest μ value. This way, this ensemble combines the higher accuracy of classifying with patterns having high support thresholds with the lower levels of abstention of classifying with patterns mined using low support thresholds.

The new Fuzzy Emerging Patterns (FEP) are patterns formed by fuzzy selectors $[Feature \in FuzzySet]$, joined by a fuzzy AND operator. This way, an object satisfies a given pattern to a certain extent, according to the degree the object feature values satisfy the property expressed in the FEP. To efficiently extract fuzzy emerging patterns from a database, we use a set of fuzzy decision trees, induced with a new algorithm that includes the use of linguistic hedges. These hedges modifies the initial fuzzy discretization to satisfy the semantics of the classes in the training sample [32]. Finally, we propose a new classifier based on fuzzy emerging patterns, which uses a novel aggregation mechanism for the pattern votes.

1.4 Goals

To answer the questions enunciated in the description of the problem and to demonstrate the hypothesis, this dissertation has the following goals:

General Goal

To develop methods to mine more expressive emerging patterns from a dataset with mixed and incomplete data, in order to obtain pattern-based classifiers more accurate than other comprehensible classifiers and as accurate as non-comprehensible state-of-the-art classifiers.

Specific goals

G1. To extend the concept of emerging pattern in order to make the patterns more expressive, considering both crisp and fuzzy cases.

G2. To develop algorithms for extracting extended crisp emerging patterns from databases with mixed and incomplete data.

G3. To develop an algorithm for inferring a good value of the minimal support threshold.

G4. To develop an ensemble of emerging pattern based classifiers.

G5. To develop an algorithm to extract fuzzy emerging patterns from a database with mixed and incomplete data.

G6. To develop classifiers more accurate than traditional pattern-based classifiers, using the mined emerging patterns.

1.5 Main contributions

The main contribution of this dissertation is the introduction of two kinds of emerging patterns, which are more expressive than the traditional patterns. The higher expressiveness of the patterns allows to express more selective properties, obtaining more accurate pattern-based classifiers. We also introduce a formal definition for *expressiveness* of a family of emerging patterns.

We introduce two new extended emerging pattern mining algorithms without global discretization of numerical features. Both extract patterns from a collection of decision trees, using a special pattern mining procedure during the tree induction. The first algorithm, LCMine, generates a fixed amount of diverse decision trees, while the second, CEPM, uses a novel object weighting scheme. For CEPM, we introduce an algorithm to calculate a good value of the minimal support threshold.

We also propose a cascade of emerging pattern classifiers. This cascade combines the higher accuracy of classifying with patterns mined using higher support thresholds with the lower levels of abstention of classifying with patterns mined using lower thresholds.

Additionally, we introduce the concept of Fuzzy Emerging Pattern, and a new algorithm for mining fuzzy emerging patterns from a database with crisp classes. This algorithm extracts patterns from a set of fuzzy decision trees, induced with a new algorithm that includes the use of linguistic hedges. We propose a new classifier based on fuzzy emerging patterns, which includes a novel mechanism for the aggregation of single pattern votes.

The classifiers built using both crisp and fuzzy emerging patterns attain higher accuracy than traditional emerging pattern classifiers and other comprehensible classifiers. They are also competitive with state-of-the-art classifiers that are not comprehensible. In a real problem, the selection between fuzzy and crisp patterns depends on the type of model desired by the user and a tradeoff between accuracy, complexity, and classifier training speed.

1.6 Thesis organization

Chapter 2 introduces a background of the main topics that are necessary to understand the contents of this dissertation.

Chapter 3 contains a critical review of the state of the art about emerging pattern mining and classification, which is presented in three parts. The first part reviews the different kinds of emerging patterns, the second part reviews the algorithms for mining the patterns, and the last section reviews emerging pattern classification algorithms.

Chapter 4 presents five novel results. First, the formalization of the concept of emerging pattern language and emerging pattern language expressiveness. Second, the Logical Complexes Mining algorithm (LCMine), an initial solution for mining more expressive patterns from a database with numerical and categorical features. Third, the Crisp Emerging Pattern Mining algorithm (CEPM), an improved version of LCMine. Fourth, the novel algorithm for computing an accurate value for the minimal support threshold used in CEPM. Fifth, a new cascade of emerging pattern classifiers, which combines the higher accuracy of classifying with patterns mined using higher support thresholds with the lower levels of abstention of classifying with patterns mined using lower thresholds. This chapter ends with the experimental results, where we compare our classifiers against state-of-the-art comprehensible and non-comprehensible classifiers.

Chapter 5 introduces the concept of Fuzzy Emerging Pattern (FEP) and presents a novel algorithm for mining these patterns from a database with crisp classes. Additionally, this chapter contains a novel pattern based classifier, based on a graph organization of the mined FEPs. This chapter ends with the experimental results, where we compare the FEP classifier against state-of-the-art comprehensible and non-comprehensible classifiers.

Chapter 6 enunciates the conclusions of this dissertation and gives future directions of research.

Chapter 2

Background

2.1 Basic concepts

Pattern recognition is about assigning labels to objects, which is known as classification [42]. Usually, objects are described by a set of measurements called features or attributes. Although there are many types of features used in pattern recognition problems, the most common are numerical (integer or real) and categorical (ordinal[1] or cardinal.). Some authors [60, 49] use the term "Mixed Data" in those cases where objects are described by more than one type of feature. A collection of objects belonging to a pattern recognition problem is usually named *dataset* or *database*.

According to the knowledge we have a priori about the object labels, there are three families of classification algorithms:

Supervised The user has an object collection with known labels or classes. Based on this dataset, named training sample, the classifier must create an internal structure that represents the relationships between the feature values and the known class. This structure is later used to predict the label of unseen objects.

[1] A feature is considered ordinal if its values are ordered. Usually they represent different degrees of fulfillment of a property, like 'high', 'medium', and 'low'.

Unsupervised The user wants to group a collection of objects in clusters according to some predefined criteria. That is why this process is frequently named *clustering*. The name *unsupervised* emphasizes that there is no previous knowledge about the labels of any object in the dataset.

Partially supervised In some supervised classification problems with large databases, the user knows the label of a small subset of the training sample. Using this information, the task involves to infer the correct class of most training objects, using later this knowledge to infer the class of unseen objects.

In this thesis, we focus on supervised classification algorithms. Most supervised classification algorithms has two stages:

- Training. During the training the classifier uses the training sample to create internal structures containing a representation of the relations between feature values and object labels. This structures could be as diverse as the whole training sample (Nearest neighbor classifiers [17]), a tree-like structure (Decision trees [58]), a weighted graph (Neural networks [36]), or a projection of the feature tuple into a higher dimensional space (Support vector machines [15]).
- Classification. During the classification the internal structures are used to infer the class of unseen objects.

In general, according to the type of the internal structures and the way they are built, we have different classification paradigms. There is a general consensus that there is no globally best paradigm. The selection of the best paradigm for a given dataset is still an open problem. Frequently, the user tests different algorithms on a testing sample, and selects the one that performs better.

2.2 Classifier performance

The performance of a classifier is a compound characteristic, whose most important characteristics are classification speed and classification accuracy.

Classification speed is measured in both stages. On static databases[2], training speed is not crucial, because this stage is applied only once. On the contrary, on dynamic databases[3] training speed is usually very important.

Classification accuracy is somehow harder to calculate, because we cannot test a classifier with all possible inputs. That is why we need to estimate how accurate a classifier is, usually based on counting the errors that the classifier makes on a sample database. This database is usually named *testing database*. Selecting a testing database is a complex task due to the following reasons:

- The user usually has a small collection of objects, compared with the total number of objects in the universe.
- The user needs to use as much data as possible to train the classifier, and also as many objects as possible to test the performance, for an accurate error estimation.
- Using the same data for training and testing could make the estimation to be optimistically biased. This way, we can find a classifier to be accurate, because it could perfectly learn the training data, but the classifier might fail on many unseen objects[4].

There are many sampling methods to select the training and testing samples, with their respective strong and weak points:

Resubstitution It uses the whole database for training and testing. As we have seen before, the resubstitution error could be very optimistically biased.

Data shuffle The database is randomly split into training and testing samples, using a fixed percent for each one. The classifier is trained with the training sample and evaluated using the testing sample. This process is repeated and the obtained accuracies are finally averaged. Accuracy results using data shuffle are usually pessimistically biased [42].

[2]Databases where objects cannot be inserted or deleted after training.
[3]Databases where objects can be inserted or deleted after training.
[4]This problem is known as overfitting.

Cross validation The user chooses an integer k (usually 10) and randomly divides the database in k subsets of equal size. Each subset is iteratively used for testing, while the union of the remaining $k-1$ subsets is used for training. Finally, the obtained accuracies are averaged. This method is the most commonly used for comparisons among classifiers.

Leave-one-out A variant of cross validation, using $k = |Database|$. This way, a single object is tested on each iteration. This method is not frequently used because it is very time consuming, and its results could be optimistically biased.

2.3 Classifier abstention

There are some situations where a classifier cannot safely assign a class to a query object; we name this situation as *classifier abstention*. There are two different types of classifier abstention:

Abstention by tie The classifier has the same amount of evidences for two or more different classes, and it cannot break the tie. A common example of this behavior can be found in a nearest neighbor classifier [17] when the query object has the same amount of neighbors from two different classes, or in a neural network [36] when the level of the output neurons for two classes is the same.

Abstention by lack of evidence The classifier has no evidences to infer the query object class. This behavior can appear in a decision tree classifier [57], when the query object has a feature value different to those assigned to child nodes in a decision node.

Abstention by tie is so frequent that most classifiers can have it. It is frequently solved by returning a random class among the tied classes. Abstention by lack of evidence is less frequent, and it can be found mostly in pattern based classifiers, like decision trees and emerging pattern classifiers [59]. It can be solved returning the majority class, or a random class among all the problem classes.

Common solutions to classifier abstention usually hide the classifier inability to classify some objects in the domain, and introduces a random com-

ponent in classifier comparisons. On the other hand, it is very hard to convince a user about using these solutions in a real problem. A more rational approach, in our opinion, is to report the abstention, so that the user can use a different classifier to obtain an accurate result.

2.4 Missing data, challenges

There are cases where datasets contain feature values that are unknown or missing. Common examples include ”Don’t know” and ”Refused” values, unintelligible answers to written questionaries, and unavailable medical test results.

Missing data[5] arise many difficulties in scientific research because most data analysis procedures were not designed for them [61]. The most common approach to handle missing data is data editing, which lends an appearance of completeness. Unfortunately, estimating a missing value may do more harm than good, producing answers that are biased, inefficient, and not reliable [61].

Another common strategy deletes objects and features containing missing values. This strategy can be hard to apply to some problems where most objects contain missing values. Medical diagnose databases, for example, contain the results of the tests applied to each patient. Unfortunately, different tests can be applied to different patients, and many test results are missing. Another important drawback of deletion is that the discarded values frequently contain useful information.

2.5 Understanding classification results

For many learning tasks, a high accuracy is not the only desired characteristic of a supervised classifier; a classifier should also be easily understandable by the users. “Symbolic” learning systems like decision trees or emerging patterns are usually much more amenable to human comprehension than

[5] Also named ‘incomplete data’ by some authors.

classifiers that use complex mathematical models, like Neural Networks and Support Vector Machines [16].

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 the financial institution to provide the reasons why the application was rejected; indefinite and vague reasons for denial are illegal [48]. In some other fields, like medical diagnosis and mineral prospection, clarity and explainability are key constraints.

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

- In neural networks [36] (NN), after training the 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 no information about why the object was assigned to the resultant class. So, neural networks are limited in this respect, since they are usually difficult to interpret after training [16].
- In k-Nearest Neighbor classifiers [17], the user could know the objects that determine the classification, but it is necessary a deep understanding of the distance function and the representation space in order to obtain a meaningful interpretation of the result.
- In a support vector machines [15], the classifier is described as a complex mathematical function, which is rather incomprehensible for humans [48].

In most comprehensible classifiers like KORA-3[8], emerging patterns [20], decision trees [58], and decision forest [37], 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 [41], any path from the root to a leaf 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 the 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 [37], where each tree gives support to a certain class, and the integration strategy provides a joint explanation. It is worth to mention that using a large number of trees can degrade understandability.

KORA-3 classifier [8] introduces another way to obtain an empirical characterization of classes. This algorithm exhaustively searches the database finding subdescriptions[6] of three feature values that appear only in a single class. The subdescriptions appearing in a query object are evidences about the object class, and they are aggregated to obtain the final classification. Emerging pattern classifiers [20] use similar subdescriptions, but they are mined with optimized procedures.

The practical problem is how to find an empirical characterization of the classes in a supervised classification problem. 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 the classes is not a universal truth. Every algorithm for this task obtains an approximation of the truth based on the training data, using a particular procedure to obtain this characterization.

2.6 Pattern vs Discriminative Pattern Mining

A *pattern* is an expression, defined in a language, which describes a collection of objects [55] . Patterns are usually expressed as combinations of feature values, like $(Color = green, Sex = male, Age = 23)$ or as logical properties, like $[Color = green] \wedge [Sex = male] \wedge [Age > 23]$. We say that the pattern P *covers* the object x, or the object x *supports* the pattern P, if the object fulfills the property expressed by the pattern. In this work, we denote this relationship as $P \Subset x$[7]. A useful characteristic of a pattern P is the amount of objects from a collection X that supports P, which is called the pattern *support* and it is denoted by $\text{support}(P, X)$.

In a supervised classification problem, we say that a pattern is *discrimi-*

[6] Bongard uses the term subdescription to name this partial object descriptions.

[7] If the patterns are expressed as itemsets, the "covers" relation becomes the traditional subset relation.

native if it includes properties which helps to differentiate between classes[8]. In general, many comprehensible classifiers use discriminative patterns for classification and classification support [30].

There are different ways to represent discriminative patterns in classifiers, although they could be implicit in some classifiers. In a decision tree [57] or forest [37], the paths from the root to the leaves are implicit discriminative patterns expressed in conjunctive forms. In a rule-based system [34], the antecedents of the rules that imply the class are discriminative patterns. KORA's complex features [8] are explicit discriminative patterns, expressed as conjunctions of simple properties.

An important type of discriminative patterns is the *emerging pattern*. A discriminative pattern is an emerging pattern if its support is significantly larger on a class than on the others [19]. Additionally, the support of a discriminative pattern on its class must be greater than a certain minimal support threshold μ. The intuition behind using a minimal support is that an emerging pattern with low support can be noisy or casual, which could be harmful for the classification [27].

There are many algorithms to search patterns in a dataset. Most of them are based on restrictions that alleviate the computational complexity of using exhaustive methods. The most important restriction reported is the *downward closure* [73]. A property X satisfies the downward closure restriction if, for every pattern P, if P fulfills X, then any pattern more particular than P also fulfills X.

Example 2.1. *The property X = "P has support below n" satisfies the downward closure, because if a given pattern P fulfills X, a more particular pattern P' has a lower support, so it also fulfills X. For example, if $P = [Age > 20]$ has support n, the more particular patterns $P' = [Age > 30]$ and $P'' = [Age > 20] \bigwedge [Sex = male]$ must have support lower or equal to n, so P' and P'' also fulfills X.*

Searching discriminative patterns in a training sample is the key procedure in many comprehensible classifiers, even though it may be implicit. If a pattern is discriminative for a single class, it can cover at most a limited amount of objects in other classes. A more particular pattern covers more objects in its class, but it also covers more objects in other classes and the

[8] Discriminative patterns were named *discriminative regularities* in [30].

limit could be exceeded. That is why discriminative patterns do not fulfill the downward closure and they cannot be mined using algorithms like Apriori [34]. In addition, there could be too many candidates in high dimensional databases and exhaustive solutions may be too costly because of the size of the search space [20].

2.7 Measuring the Quality of a Discriminative Pattern Collection

After training, the classifier quality is proportional to the quality of the internal structure it uses to represent the relationships found in the training sample. In the case of discriminative pattern classifiers, the classifier quality depends tightly on the mined pattern quality.

There is no generally accepted methodology to measure the quality of a discriminative pattern collection. Nevertheless, there are some desired properties the collection should fulfill:

Discriminability Every pattern should cover a significant amount of objects in a class, and few objects in the remaining classes.

Simplicity There should be few patterns, at least an order of magnitude less than the number of objects. Violating this property could seriously degrade the classifier comprehensibility.

Non-redundancy Each pattern should contain some new knowledge, with respect to the remaining patterns. Redundant patterns could present redundant evidence on a query object, biasing the classification towards a single class.

Generality Patterns covering large amount of objects are usually less noisy. On the contrary, too particular patterns could exist due to chance.

Algorithms for mining discriminative patterns follow two different strategies for obtaining a high quality pattern collection:

- Extract patterns belonging to a particular family:

  - Patterns covering objects in a single class, like *jumping emerging patterns*, used in the DEEPs classifier [44].
  - Minimal patterns with respect to the subset inclusion of their respective properties, used in the JEPC classifier [27].

- Filter a large collection of patterns, looking for a subset with the desired properties. This strategy is used in the LCMineC classifier [30], described later in this dissertation.

One of the most popular approaches to measure the quality of a pattern subset, followed in this dissertation, is the accuracy of a classifier built using this subset. Nevertheless, it is important to remember that the accuracy of the classifier is affected by many other parameters, like the support aggregation mechanism and the pattern organization.

2.8 Decision tree induction

In this section, we present an introduction to decision tree induction algorithms, because the algorithms introduced in this dissertation are based on them.

A *tree* is a directed acyclic graph, where all nodes have a single ancestor, except a distinctive node named root, which has no ancestors. Trees are very popular data structures used for representing hierarchical information. In pattern recognition, there is a supervised classifier named *decision tree*, which uses a particular type of tree as internal structure. Figure 2.1 contains an example of a decision tree, built using a database with three features ($Length$ – numerical, $Color$ – categorical, and Fly – boolean) and two classes (**Bad** and **Good**).

To classify a query object a decision tree traverses the inner tree, starting in the root node towards the leaf nodes. A decision tree contains two types of nodes:

Decision nodes All nodes except the leaf nodes. These nodes contain a property assigned to each child node. The child node is selected according to the property fulfilled by the query object.

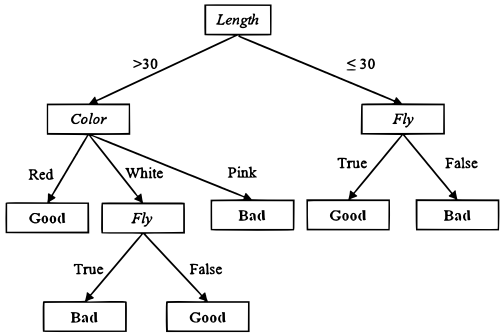

Figure 2.1: Example of a decision tree with three features and two classes: **Good** and **Bad**

Class nodes Each leaf node contains a class, which is assigned to the query object if the traverse ends in this node.

Example 2.2. *Suppose we want to use the decision tree of Figure 2.1 for classifying the following object:*

$$o = (Length = 50, Color = White, Fly = true)$$

We start by traversing the tree in the root node, testing the properties $Length > 30$ *in the left child and* $Length \leq 30$ *in the right child. As o has* $Length = 50$*, we select the left child as the next node in the traversing. Likewise we choose consecutive properties* $Color = White$ *and* $Fly = true$*, arriving to the leaf node with class* ***Bad****.*

There are two main strategies to build decision tress[9], which are known as bottom-up and top-down. In the first iteration, the bottom-up strategy starts grouping similar objects in nodes. In later iterations a common parent is assigned to similar nodes, until a single parent node is built. This way we obtain a decision tree, containing a hierarchical collection of clusters. Bottom-up strategies are not considered in this dissertation.

The top-down strategy starts with the whole collection of objects as a single group, and tries to find the best property to split this collection into subsets (usually two), according to a predefined criterion. The obtained subsets are recursively split with the same algorithm, until certain stopping criterion is

[9]Building decision trees is commonly known as *inducing* decision trees.

met. The property hierarchy is then used to build the decision tree. A general algorithm to build a decision node appears in Algorithm 2.1.

Data: T – objects
Result: N – new tree node created
if *Stopping criterion is met* **then**
  return $N \leftarrow$ new class node, with the majority class in T
end
$CandidateSplits \leftarrow$ All properties that split T in different subsets ;
$BestSplit \leftarrow$ Split in $CandidateSplit$ that maximize the quality criterion ;
$N \leftarrow$ New decision node using property $BestSplit$;
$T_i \leftarrow$ Subsets of T according to property $BestSplit$;
foreach T_i **do**
  $BuildNode(T_i)$
end
return N;

Algorithm 2.1: Pseudocode of the algorithm **BuildNode** to recursively build a decision tree

The following parameters have to be defined in order to build a decision tree:

Stopping criterion Stopping criterion is usually composed of different simple criteria like the following:

- The node is pure enough to be considered as a class node.
- The node has too few objects, so it is worthless to continue analyzing it.

Candidate split generation Candidate splits are created based on the feature values of the objects in T. To consider a feature, it must have at least two different values in the node. According to the feature types, the following candidate splits can be generated, among others:

- Each categorical value v generates a binary candidate split, using the properties $value = v$ and $value \neq v$.
- Each categorical feature with k values v_i generates an n-ary split with the properties $value = v_1, value = v_2, \ldots, value = v_k$.
- Each numerical feature can generate many binary candidate splits with properties $value < v_i$ and $value \geq v_i$, according to the collection of cut points v_i. There are many ways to find a good cut point,

but it is usually the midpoint between two values belonging to objects of different classes.

Quality criterion to evaluate splits This criterion assigns each split a quality rating according to some desired characteristics like node pureness and tree balancing. The following equation is used in CART decision trees [10] for evaluating binary splits:

$$\Delta i(N) = i(N) - P(N_1) \cdot i(N_1) - P(N_2) \cdot i(N_2) \tag{2.1}$$

where $i(N)$ is the impurity of the tree node N and $P(N_i)$ is the probability of an object to be assigned to node N_i, which is estimated as the division of the amount of objects in N_i by the amount of objects in N. The impurity of a node N measures how close the class distribution of the objects in N is to be a pure distribution. A common way to measure this impurity is the *entropy*:

$$i(N) = -\sum_{j=1}^{c} P_j \log P_j$$

where c is the number of classes and P_j is the probability of a random object in node N to belong to class Cj.

2.8.1 Handling Missing Values in Decision Tree Induction

Properly handling missing values is a key component of a decision tree induction algorithm. A straightforward solution is, for each candidate split using feature X_i, to ignore the objects with this feature value missing. Nevertheless, this solution can make a candidate split to be selected, even if it includes a property containing a feature that is missing in most objects. The resultant tree is unable to classify a large amount of objects[10].

There are basically two strategies to handle missing values in the evaluation of a candidate split [58], which penalize splits that contain features with missing values:

[10]There are techniques to classify an object with a missing value in the feature contained in a decision node. Nevertheless, theses techniques are frequently less accurate than building the decision tree using properties with less missing values.

- Grouping the objects with missing values in a virtual node[11] with maximum impurity. According to (2.1) this virtual node penalizes the quality criterion proportionally to the amount of objects in the virtual node.

- Penalizing the quality criterion evaluation using the probability of finding a non-missing value. This way, the more missing values present in the selected attribute, the lower value of the quality criterion index.

2.9 Fuzzy logic

Fuzzy logic [72] is a form of multi-valued logic to deal with approximate reasoning rather than precise. In contrast with "crisp logic", fuzzy logic variables are not constrained to two truth values as classical propositional logic; the degree of truth of a fuzzy statement can range between 0 (false) and 1 (true) [53].

Fuzzy set theory is a generalization of traditional set theory, using fuzzy logic to define the classical operators like *belong*, *union*, and *intersection*. In this section, we present some concepts of fuzzy set theory, containing the elements used in this dissertation.

Let us consider a universe set $\mathbb{U}$. A fuzzy set F is characterized by a membership-degree function $f : \mathbb{U} \rightarrow [0, 1]$ that associates to every object $o \in \mathbb{U}$ the degree of membership to the fuzzy set F. Fuzzy sets help us to model concepts with some degree of uncertainty.

Example 2.3. *Consider, in the universe of temperatures, the set of hot temperatures. Using crisp set theory we must select a cut point, and consider any temperature above that point as hot, and any temperature below that point as not hot. This is clearly contrary to human interpretation of the concept of Hot, because a very small variation, can make a temperature abruptly change from hot to not hot.*

Figure 2.2 shows the fuzzy sets Hot, Warm, and Cool. As we can see, small variations in the temperature value are associated with small variations in the membership values. For example, temperatures below 0 °C are completely cool,

[11] Virtual nodes are discarded after tree induction, so they are not nodes in the resultant decision tree.

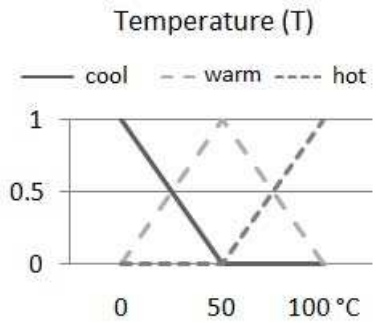

Figure 2.2: Fuzzy sets Hot, Warm, and Cool defined in the universe of temperatures

so they have membership equal to 1 for the cool fuzzy set; temperatures above $50\ ^{\circ}C$ *are not cool at all, so they have membership equal to zero for the cool fuzzy set; when the temperature moves from* 0 *to* $50\ ^{\circ}C$ *, the membership softly decreases from* 1 *to* 0*.*

2.9.1 Fuzzy Set Operators

Fuzzy set operators UNION, INTERSECTION, and COMPLEMENT have similar semantics compared to their crisp versions, but using a fuzzy versions of logical operators OR, AND, and NOT, respectively:

UNION The union of two sets $F \bigcup G$ contains all the elements that belong to F OR to G. Fuzzy OR operators (or T-Conorms) assign to each object a membership to $F \operatorname{OR} G$ which is not smaller than the memberships to F and G. The most popular fuzzy OR operators are the following:

- $\Big(F \operatorname{OR} G\Big)(x) = \max\Big\{F(x), G(x)\Big\}$
- $\Big(F \operatorname{OR} G\Big)(x) = \min\Big\{F(x) + G(x), 1\Big\}$

INTERSECTION The intersection of two sets $F \bigcap G$ contains all the elements that belong to F AND to G. Fuzzy AND operators (or T-Norms) assign to each object a membership to $F \operatorname{OR} G$ which is not greater than the memberships to F and G. Popular fuzzy AND operators are the following:

- $\Big(F \text{ AND } G\Big)(x) = \min\Big\{F(x), G(x)\Big\}$
- $\Big(F \text{ AND } G\Big)(x) = F(x) \cdot G(x)$

COMPLEMENT The complement of a set $\neg F$ contains all the elements that are NOT in F. Fuzzy NOT operators assign to each object a membership complementary to the membership to F. The most popular NOT operator is $\Big(\text{NOT } F\Big)(x) = 1 - F(x)$.

It is easy to see that some properties of crisp set theory, inherited from Boolean logic, do not hold for fuzzy sets:

- $F \bigcup \neg F \neq \mathbb{U}$
- $F \bigcap \neg F \neq \emptyset$

2.9.2 Linguistic hedges

Linguistic hedges modify fuzzy sets in the same way that adverbs modify adjectives in English. It is important to highlight that a new fuzzy set is obtained after applying a linguistic hedge to a fuzzy set.

Example 2.4. *Figure 2.3 shows the fuzzy set Warm modified with linguistic hedges very and somewhat. Like the equivalent adverb, the hedge very concentrates a concept. This way, the membership of a temperature to the new concept very(warm) cannot be greater than the membership to the concept warm. For example, the temperature value* $40\ ^{\circ}C$ *has membership* 0.8 *to the fuzzy set Warm, but only* 0.5 *to the fuzzy set very(Warm). On the other hand, the linguistic hedge somewhat dilates a fuzzy set.*

Linguistic hedges are commonly used in learning algorithms to dynamically fix the fuzzy discretization of continuous features. They can be very useful because they can help to fix a wrong initial discretization of numerical features, which could become a strong limitation for the classification quality [32].

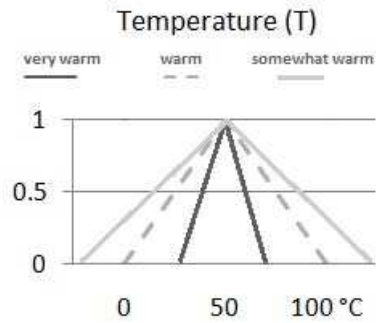

Figure 2.3: Fuzzy set Warm modified with linguistic hedges *very* and *somewhat*

2.10 Fuzzy Decision Tree Induction

Fuzzy decision tree induction has been an active research area for decades [13, 70, 67]. Some fuzzy decision trees are extensions of crisp decision trees to the fuzzy domain, like Fuzzy ID3 [66] and its variations [67]. On the other hand, some authors have created novel schemes of fuzzy decision tree induction specially designed for the fuzzy case. LR-COG [38], for example, uses an induction procedure based on the trapezoid center of gravity, using an information entropy minimization heuristic.

Fuzzy decision trees are similar to their crisp counterparts, but using fuzzy logical operators and properties. The main differences are the following:

- In a crisp decision tree, each child node of a node N is associated to a crisp property. To classify an object, all these properties are tested, and the single one which is true is used to choose which child node to follow. In a fuzzy decision tree, an object can fulfill simultaneously all the child properties, eventually with different degrees.

- In the crisp version, a query object q travels from the root node towards a leaf node. In each recursive call (see Algorithm 2.1,) the query object is located in a single tree node. As a consequence of fulfilling simultaneously all the child properties with different degrees, q would be simultaneously located on many different nodes in the fuzzy tree, but belonging to each node with a different degree. This degree is understood as the level of membership of q to each tree node. To calculate

this membership, we must take into account also the membership of q to the parent node, maybe using a fuzzy AND operator.

- Because the query object arrives to different class nodes, we need an aggregation mechanism to find the resultant class. This mechanism must take into account, for each leaf node, the node class and the membership of q to this node.

Inducing a fuzzy decision tree is similar to the crisp case, but with the following differences:

- Candidate splits use fuzzy properties. To obtain fuzzy properties, we must first fuzzyfy the original features, transforming each one in a collection of fuzzy sets. For example, Figure 2.2 shows the fuzzyfied Temperature feature. Then, fuzzy properties have then the following form:

$$Feature \text{ IS } FuzzySet$$

  where $FuzzySet$ is one of the fuzzy sets in the fuzzyfied $Feature$.

- The quality criterion for split evaluation must take into account that in the fuzzy version each object belongs to different tree nodes with different membership values. This way, object counting strategies are not appropriate, and should be substituted by membership summation strategies.

- The stop criterion must use membership values.

Chapter 3

Related work

This chapter presents a critical revision of previous works on emerging pattern mining and classification. For a better understanding, we split the content in three sections: Section 3.1 introduces different types of emerging patterns, Section 3.2 presents algorithms for mining them, and Section 3.3 reviews different emerging patterns classification algorithms.

3.1 Types of emerging patterns

Most papers on emerging pattern mining use a transactional representation of the objects. This way, an object is represented as a collection of items, or itemset. An item is a pair $(Feature, value)$, where $value$ belongs to the $Feature$ domain. If the original database contains numerical features, they are discretized using some methods like the Entropy algorithm [28]. Considering this transactional representation, an emerging pattern is also an itemset, and the relation $P \Subset x$ becomes the traditional subset inclusion $P \subset x$.

Dong and Li [20] introduced in 1999 the ρ-emerging pattern for two class problems, which is an emerging pattern with $GrowthRate \geq \rho$. The $GrowthRate$ (3.1) measures how frequent a pattern is in its own class C_P with respect to

its frequency in the other class C.

$$GrowthRate(P) = \begin{cases} 0, & if\ \text{support}(P,C) = 0 \wedge \text{support}(P,C_P) = 0 \\ \infty, & if\ \text{support}(P,C) = 0 \wedge \text{support}(P,C_P) > 0 \\ \frac{\text{support}(P,C_P)}{\text{support}(P,C)}, & otherwise \end{cases} \tag{3.1}$$

An important class of emerging patterns are those that cover objects in a single class, which are named Jumping Emerging Patterns (Definition 3.1). The jumping emerging patterns have been widely used in emerging pattern classifiers, because they have a strong predictive power. Jumping emerging patterns represent properties that are only present in a single class, so they should be distinctive.

Definition 3.1 (Li *et al.* [43])**.** *A* Jumping Emerging Pattern *(JEP) is an emerging pattern with infinite growth rate.*

Fan and Ramamohanarao introduced in 2006 [27] the *Strong Jumping Emerging Pattern* (Definition 3.2). These patterns have an infinite growth rate, but they are also minimal with respect to the subset inclusion.

Definition 3.2 (Fan and Ramamohanarao [27])**.** *P is a* Strong Jumping Emerging Pattern *(SJEP) if it satisfies the following conditions:*

1. *P has infinite growth rate.*
2. *No proper subset of P satisfies condition 1.*

Condition 2 means that a strong jumping emerging pattern is the *minimal* JEP satisfying the support constraint. According to Fan and Ramamohanarao [27], non minimal JEPs are useless for classification, and they can be harmful to the accuracy when aggregating many of them to make decisions. SJEPs are also known as essential JEPs (eJEPs) [25].

On the contrary, Wang *et al.* [68] considered that aggregating many minimal EPs may implicitly cause duplicate counting of the EP's contribution, which leads to lower accuracy (see Example 3.1).

Example 3.1. *Suppose we have the properties A, B, C, D, E, and F. Now the patterns ABCD, ABCE, ABCF are all minimal, but counting their contribution as individual pattern can make the pattern ABC to be counted three times.*

To solve the duplicate counting Wang introduced the *Maximal Emerging Pattern* (Definition 3.3).

Definition 3.3 (Wang *et al.* [68])**.** *A* Maximal Emerging Pattern *(MaxEPs) is an emerging pattern whose supersets are not emerging patterns.*

We summarize the pros and cons of using minimal and maximal patterns for classification, pointed out in the literature, as follows:

- Using only minimal, more general patterns:
  - If less features can distinguish between two classes, using more features may not help and may even add noise [25].
  - It can speed up the searching process, saving computing [25].
  - Large growth rate ensures EP's sharp discriminating power; large supports, which means enough coverage on the training dataset, makes EPs more resistant to noise [26].
  - Minimal EPs have higher support, so unknown instances are easier to match [68].
  - Aggregating many minimal EPs may implicitly cause duplicate counting of individual EP's contribution, which could lead to lower accuracy [68, 5].
- Using only maximal, more specific patterns:
  - They provide more information about the higher order interactions between features [74].
  - They minimize duplicated EP contribution [68].
  - Maximal patterns are harder to find in the query object, so the classifier may have fewer patterns to decide about the classification [68].

No matter the advantages of using jumping emerging patterns, they cannot capture useful properties if there is noise in the data. Noise appears frequently in real-world data due to sensor or user errors. To make emerging patterns tolerant to noise, a small but not strictly zero support in other classes must be allowed (Definition 3.4).

Definition 3.4 (Fan and Ramamohanarao [27])**.** *A* Noise-tolerant *emerging pattern (NEP) is a minimal pattern P that satisfies:*

1. $\text{support}(P, C_P) \geq \delta_2$
2. $\text{support}(P, C) \leq \delta_1$

where C_P is the class of the pattern, $C \neq C_P$ is any other problem class, and $\delta_2 \gg \delta_1$ are two positive integer thresholds.

Other varieties of emerging patterns have been defined incorporating appropriate constraints, such as *Chi Emerging Patterns* [59], *Constrained Emerging Patterns* [7], and Emerging Patterns with Occurrence Count [40]. Nevertheless, they are specific to some applications, and they are not frequently used in other domains.

3.2 Algorithms for mining emerging patterns

In 1999 Dong and Li [20] introduced the concept of emerging pattern. They found that the number of EPs in a problem could be too large, so they developed a simplified representation, using the subset-closedness: they represented all the EPs as a collection of minimal and maximal patterns over the subset inclusion relation, called borders. In their paper, they showed that borders can be efficiently found for many commonly used repository databases.

ConstEPMiner [76] introduced a set of constraints to prune the search space of EPs and save computations. The authors created an algorithm to apply these constraints to get a subset with good predictive power and no redundancies. Only patterns that are more general (those with top growth rate) remain and they filter patterns with the same support, considering them as redundant. Although these constraints are the basis of many posterior filtering methods, the algorithm could delete important patterns, with a direct impact in the classifier accuracy.

Bailey *et al.* [5] introduced the first tree based approach for fast JEPs mining. They adapted the frequent pattern tree FP-tree [35] algorithm to deal with datasets structured in classes. Additionally, the authors presented a

study of the influence of the selection of the minimal support threshold in the classification accuracy. They found that, in many databases, it is worth to use higher values because of the substantial decrement in computational time, at the expense of little accuracy degradations. Bailey *et al.* [5] also showed the impact of mining only patterns with length below a given threshold [5], arguing that small (more general) patterns are the best for classification. Although they found a significant increase in extraction speed, they also found accuracy degradation in some databases. They explained this behavior by the fact that classification accuracy is not only dependent upon the representation of the classes, but also upon the discriminative power of each class with respect to other classes.

Li *et al.* [45] introduced the following important modifications to the adapted FP-tree, in order to speed up the process:

- Grouping mined patterns in equivalence classes, according to the objects described. This allows to reduce redundant patterns, and to simplify the task of calculating sophisticated statistics, which can be used to select the most useful patterns.
- Suppressing too frequent and rare items, because they are less prone to appear in emerging patterns.
- In multi-class problems, the algorithm simultaneously mines patterns from all classes. Previous methods handle multiple classes one by one, using a single class and the complement on each iteration.

A different tree, named contrast pattern tree (CP-tree), was introduced by Fan and Ramamohanarao [25, 27]. A CP-tree is an ordered multiway tree structure, where all the objects in the training sample are represented. The mining algorithm searches depth-first the CP-tree to discover the patterns. For speeding up the process, only the strong JEP were considered. Nevertheless, the authors claimed to obtain higher accuracies than previous methods, using fewer patterns.

An adaptive version of CP-tree based mining [63] gradually raises the minimum support threshold during mining. The algorithm tries to find the top-K patterns, so the threshold is raised based on the number of patterns mined so far with the current threshold value. This optimization boosts the mining speed, since more tree branches are pruned earlier.

Bailey *et al.* [6] introduced a fast algorithm for computing hypergraph transversals and applied it to mining emerging patterns. This algorithm is based on a guided partitioning heuristic, which seems to work fine in some databases with thousands of objects.

Fan and Ramamohanarao [26] created the first post-processing pattern filtering. They extracted SJEPs from the training sample, ranked them, and iteratively selected those that covered at least a new object. The ranking considers the pattern support and the pattern length, discarding the growth rate information. According to the authors, EPs have implicitly large growth rate, and it does not make sense to compare between their values.

Loekito and Bailey [47] used Zero-Suppressed Binary Decision Diagrams (ZBDDs) [52] as the core data structure for mining emerging patterns. First, itemsets are represented as a n-bit binary vector, where each Boolean value represents the presence/absence of the particular item. Then, some binary operators like *set-union*, *set-difference*, and *set-intersection* are performed in order to find the emerging patterns. ZBDDs works like CP-trees and FP-trees, but with better performance.

In cases where data are scattered in multiple tables of a relational database, it is not necessary to do costly joins to mine the emerging patterns. Appice *et al.* [4] created Mr-EP, an algorithm that can capture the differences between objects of two classes. Mr-EP can extract emerging patterns whose properties are spanned in separated data tables. A recent approach for this task uses local projections of the database [64].

Most of the algorithms for mining emerging patterns described in this section have the following general steps:

1. Selection of the minimal support threshold μ.
2. Global discretization of numerical features.
3. Representation of the transformed objects using a particular structure.
4. Traversing the structure to find emerging patterns.
5. Pattern filtering.

Using these conventional steps has two important drawbacks:

Table 3.1: Databases where SJEPC gets significant accuracy degradations

DBName	3-NN	J48	SJEPC
balance-scale	85.4	77.6	16.0
breast-cancer	70.3	73.4	44.5
cleveland	82.5	78.2	77.9
haberman	70.6	68.0	0.0
hayes-roth	71.4	89.3	0.0
heart-h	83.6	79.6	46.3
heart-statlog	79.3	79.3	64.8
iris	96.0	94.0	66.7
liver-disorders	65.5	68.7	0.0
lung-cancer	43.3	35.0	0.0
lymph	85.9	78.5	51.5
shuttle-landing-control	60.0	60.0	0.0
spect	64.7	66.8	0.0
trains	70.0	90.0	0.0
wine	96.1	92.7	55.1

1. Global discretization of numerical features can drastically degrade the classifier accuracy, since an emerging pattern relates a combination of feature values with a class. Therefore, discretizing a numerical feature without considering the values of other features could hide important relations.

   In Table 3.1, we can see that SJEPC [27], one of the most accurate emerging pattern classifiers, obtains very poor accuracies compared with Nearest Neighbors [17] and C4.5 [58] classifiers. In some databases, SJEPC is unable to extract even a single pattern, because most numerical features are discretized into a single categorical value. Although SJEPC uses the Entropy discretization method [28], other discretization methods have similar behaviors, perhaps in different databases.

2. High sensitivity to the minimal support threshold value μ. The accuracy of the classifier could have a serious degradation on small variations of the minimal support value. For example, in *chess* and *census* databases (from the UCI repository [50]), the accuracy drops 3% with a difference of 2 in μ [5]. We must point out that the adaptive modification of μ [63] does not solve the problem, because it discards most patterns with low support. Discarding all the patterns with low support usually increases dramatically the classifier abstention level.

3.3 Classification using emerging patterns

Given a query object x, a pattern based classifier assigns to x the class with the highest score. The score is computed based on the patterns contained in x. Usually the scoring function is designed taking into account the characteristics of the kind of emerging pattern used [59]. Emerging pattern classifiers can be grouped in categories, according to the type scoring function used. In this section, we present the most used single categories and the combined approaches.

3.3.1 Aggregation of support

Classifiers on this category aggregate the individual discriminating power of the patterns contained in the query object. The first classifier using aggregation of support was CAEP [21], which can use patterns with support in other classes. CAEP normalizes the votes per class using a base score per class, which is one of the most commonly used approaches [25, 27].

A different approach consists in aggregating the support of the matching EPs by using the compact summation method [43]. The compact summation of the support set in the class C_i is the percentage of instances in the class that contain one or more of the patterns. This method avoids counting duplicate contributions of training instances, and it is commonly used in classifiers that mine patterns during the classification stage [44].

3.3.2 Information-based classifiers

A variant of CAEP is iCAEP [74], which uses the minimum encoding inference approach to classify an object, instead the aggregation of support. iCAEP uses each matching pattern as a message indicating a class bias of the pattern class. Then, the class that can be encoded with the minimum message length is assigned to the query object. The authors select patterns with many items, claiming that they provide more information about the higher order interactions between features.

3.3.3 Bayesian classifiers

The Bayesian Classification based on Emerging Patterns (BCEP) [26] is a hybrid classifier of the EP and Naive Bayes. It uses essential Jumping Emerging Patterns to relax the strong feature independence assumption. BCEP uses the patterns extracted in the training stage to derive a product approximation for each class probability. To obtain such approximation, the matching patterns are combined using the chain rule of probability.

A version of BCEP using maximal emerging patterns is MaxEPs [68]. As we mentioned in Section 3.1, classifying with maximal EPs reduces duplicate pattern contribution, avoiding accuracy reduction. Nevertheless, maximal patterns are not frequently found in query objects and classifying with few patterns can be unreliable. That is why MaxEPs uses the intersection between the maximal patterns and the query object. However, there is no guarantee that the intersection is an emerging pattern, and this fact can degrade the classifier accuracy.

3.3.4 Combined classifiers

Li *et al.* [46] combined a pattern based classifier in cascade with a k-NN classifier [17]. First, if the query object has objects nearby, the k-NN classifier assigns the classifier result. Otherwise, distance is considered as inaccurate for classification, and an emerging pattern classifier [44] returns the classification result. This way, the resultant classifier combines the higher accuracy of the NN classifier in databases with numerical features, with the effectiveness of the EP classifiers in categorical domains.

Because they are complex and stable classifiers, emerging Pattern classifiers are not frequently used in ensembles. Nevertheless, they have been used as base classifiers in ensembles using Bagging and Boosting [42] methods. The boosted emerging pattern classifier [62] is the application of AdaBoost to the CAEP classifier. The Bagged Emerging Pattern [24] introduced a new scoring function for EP classifiers, to make them unstable. Additionally, the authors modified the classical voting scheme to allow classifier abstention.

There is no general theory that can help us to select the appropriate pat-

terns and classifier for a given problem. Actually, the only solution for users is to test all the available classifiers over some data samples, selecting the one with the highest accuracy. The two most commonly used schemes are using minimal patterns with aggregation of support classifiers and maximal patterns with Bayesian classifiers.

3.4 Summary

This chapter presents a critical revision of previous works on emerging pattern mining and classification. This revision shows that existing methods for mining emerging patterns have two important drawbacks: they include a global discretization step, and they have high sensitivity to the value of the minimal support threshold. Both drawbacks could have a significant impact in the classification accuracy.

The last section shows that there are several emerging pattern-based classifiers, but none of them shows a clear superiority compared to the others. It is also unclear which kind of patterns should be used to attain the highest accuracy: minimal, maximal, non-redundant, or other subsets.

Chapter 4

Extended crisp emerging pattern mining

4.1 Introduction

This chapter presents five novel results. First, Section 4.2 formalizes the concepts of emerging pattern language and emerging pattern language expressiveness. These concepts are very important to understand why using more expressive patterns allows our classifiers to be more accurate than traditional emerging pattern classifiers. Second, Section 4.3 introduces the Logical Complexes Mining algorithm (LCMine), an initial solution for mining more expressive patterns from a database with mixed and incomplete data. Third, Section 4.4 presents the Crisp Emerging Pattern Mining algorithm (CEPM), an improved version of LCMine. Fourth, Section 4.4.1 introduces a novel algorithm for computing an accurate value for the minimal support threshold. Fifth, Section 4.6 proposes a new method for building cascades of emerging pattern classifiers. This method combines the higher accuracy of classifying with higher support thresholds with the lower levels of abstention of classifying with lower thresholds. Finally, Section 4.7 presents a comparison of the proposed classifiers versus state-of-the-art comprehensible and non-comprehensible classifiers.

4.2 Pattern expressiveness

Before introducing the concept of pattern expressiveness, let us start with some basic definitions and notations.

Definition 4.1. *A* Pattern Language Λ *is the set of all possible conjunctive ways to aggregate the selectors*[1] $[Feature \# ValueSet]$*, where* $Feature$ *is any feature,* $\#$ *is a relational operator like* $\leq, \in, =$ *or* $\neq$*, and* $ValueSet \subset domain(Feature)$*. Pattern languages are represented by the tuple* $(\{Feature\}, \{\#\})$*, formed by the feature set and the operator set.*

Example 4.1. *Let us consider a database with two attributes:* $X_1 = \{1, 2, 3\}$*, and* $X_2 = \{a, b\}$*. If* $\#$ *is the set* $\{=, \neq\}$*, the pattern language includes all the following elements:* $[X_1 = 1]$*,* $[X_1 \neq 1]$*,* $[X_1 = 2] \ldots$*,* $[X_2 = a]$*,* $[X_2 = b]$*,* $[X_1 = 1] \wedge [X_2 = a]$*,* $[X_1 \neq 1] \wedge [X_2 = a]$*,* $[X_1 = 2] \wedge [X_2 = b]$*,* $\ldots$

It is important to highlight that each emerging pattern mining algorithm extracts patterns represented in a determinate language. For example, previous algorithms for mining emerging patterns discretize numerical features, so they return patterns expressed in the following language:

$$\Lambda_{Traditional} = (\{Feature\}, \{=\})$$

A very important property of a pattern language is its *expressiveness*, with respect to a given universe of objects $\mathbb{U}$:

$$e(\Lambda, \mathbb{U}) = \frac{\left|\{S \subset \mathbb{U} : \exists P \in \Lambda, supp(P) = S\}\right|}{2^{|\mathbb{U}|}}$$

where $supp(P)$ is the set of objects in $\mathbb{U}$ containing the pattern P.

The expressiveness of a pattern language measures how many subsets of the universe can be exactly represented by a single pattern from the language. This way, if a pattern language Λ_1 is more expressive than the pattern language Λ_2, we can characterize most subsets in the universe using fewer patterns in Λ_1 than using patterns in Λ_2.

All the algorithms introduced in this chapter are expressed as l-complexes.

[1] The term selector was introduced by Michalski [51].

Definition 4.2. *A logical product of selectors* $\bigwedge_{i \in I}[Feature_i \# ValueSet_i]$ *is called a* logical complex *(l-complex) [51], 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*

Example 4.2. *Given the features Color, Age, Height and Weight, the 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 that it satisfies [51]. For instance, the last l-complex in the Example 4.2 represents those objects whose height is different from tall and their age is greater or equal to 20. If an object has some missing values in the features associated with an l-complex, then the l-complex does not cover the object.

L-complexes use the following pattern language:

$$\Lambda_{Extended} = (\{Feature\}, \{=, \neq, \leq, >\})$$

Theorems 4.1 and 4.2 express the relationship between the expressiveness of the language used in traditional pattern extraction algorithm and the language used in the algorithms introduced in this chapter.

Theorem 4.1. *For all universes* $\mathbb{U}$*, the language* $\Lambda_{Extended}$ *is at least as expressive as* $\Lambda_{Traditional}$*. That is,* $e(\Lambda_{Extended}, \mathbb{U}) \geq e(\Lambda_{Traditional}, \mathbb{U})$.

Proof. To prove the theorem, we show that any pattern in $\Lambda_{Traditional}$ can be expressed as a pattern in $\Lambda_{Extended}$. Let be $P = \bigwedge[F_i = S_i]$ a pattern from $\Lambda_{Traditional}$. We can construct a new pattern $Q = \bigwedge[G_i \#_i T_i]$ from $\Lambda_{Extended}$, expressing an identical property. For each item in $[F_i \# S_i]$, we construct the following items for Q:

- if F_i is a nominal feature we construct $[G_i = T_i]$ with $G_i = F_i$ and $T_i = S_i$.

- If F_i is a discretization of a numerical feature, we create two new items for Q using the non-discretized feature G_i: $[G_i >= v_1]$ and $[G_i < v_2]$, where v_1 and v_2 are the left and right bounds of S_i.

□

Theorem 4.2. *There exist some universes* $\mathbb{U}$ *where the language* $\Lambda_{Extended}$ *is more expressive than* $\Lambda_{Traditional}$*. That is,* $e(\Lambda_{Extended}, \mathbb{U}) > e(\Lambda_{Traditional}, \mathbb{U})$.

Proof. Let us consider the universe $\mathbb{U} = \{1, 2, 3, 4, 5, 6\}$ with a single numerical feature X, discretized into two subsets $S_1 = [0, 3]$ and $S_2 = (3, 6]$. Now $\Lambda_{Traditional}$ has the patterns $[X = S_1]$ and $[X = S_2]$, while $\Lambda_{Extended}$ has the patterns $[X \leq 1], [X > 1], \ldots, [X \leq 6], [X = 1], [X \neq 1], \ldots$.

It is easy to see that only two subsets of $\mathbb{U}$ can be exactly covered by patterns in $\Lambda_{Traditional}$, so $e(\Lambda_{Traditional}, \mathbb{U}) = \frac{2}{2^6}$. Nevertheless, there are much more subsets covered exactly by patterns in $\Lambda_{Extended}$. There are the subsets {1}, {2}, …, {6}, {1, 2}, {1, 3}, …, {1, 2, 3, 4, 5, 6}. This way, $e(\Lambda_{Extended}, \mathbb{U}) > e(\Lambda_{Traditional}, \mathbb{U})$. □

Theorems 4.1 and 4.2 show that the emerging patterns mined in this dissertation are more expressive than traditional emerging patterns. This higher expressiveness allows us to find properties that differentiate better objects in different classes. This is an important element to explain why our classifiers outperform traditional emerging pattern classifiers in most databases (Sections 4.7 and 5.5).

4.3 Logical Complex Mining

In this section, we introduce the Logical Complex Mine (LCMine) algorithm for extracting emerging patterns, expressed as logical complexes, from databases with mixed and incomplete data. LCMine has two main components: a searching procedure and a filtering procedure. We also propose a classifier based on the mined emerging patterns.

4.3.1 Searching Procedure

The searching procedure extracts emerging patterns 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 classical methods do in order to look for properties that better describe the training sample in terms of accuracy and simplicity. In this dissertation, we understand simplicity as the number of conjunctions and features involved in an emerging pattern. Therefore, the fewer conjunctions and features the pattern has, the simpler the pattern is.

To guarantee diversity among the 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 former option is unique, and the last one is hard to apply to nontrivial problems because of its time complexity. Our *BuildTree* algorithm, unlike traditional methods for building decision trees[2], expands a subset of the best candidate splits at each tree level. The number of splits to expand decreases from the root node to the leaves, starting from a user-defined value k to 1. 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.

LCMine expands the best k splits on the root, $k-1$ splits on root children, and so on, generating $k!$ diverse decision trees. Each decision tree can be described by the indexes $\{i_{level}\}$, representing which of the best candidate splits to use at each level.

Example 4.3. *The tree* $\{1,1,\ldots,1\}$ *is built using the best split available on each node, in a similar way than traditional decision tree induction algorithms like ID3 and C4.5 [57, 58] do.*

Example 4.4. *Using* $k=5$*, LCMine generates 120 decision trees, starting from* $(1,1,1,1,1)$ *to* $(5,4,3,2,1)$*. Then, the tree* $(2,1,3,2,1)$ *is built using the second best split in the root node, the best split in the second level, the third best split in the third level and so on.*

The LCMine (Algorithm 4.1) algorithm calls the BuildTree procedure (Algorithm 4.2) for each combination of values $\{i_{level}\}$, obtaining a family of di-

[2]Traditional methods for building decision trees expand the single best candidate split at each tree level.

verse trees. BuildTree generates all the candidate splits according to the type of each feature. The split type corresponds to the type of selectors we include in our patterns, in the following way:

- For categorical features:
  - If the feature has exactly two values, BuildTree creates a single split with two children with the properties $Feature = v_1$ and $Feature = v_2$ respectively.
  - Otherwise, BuildTree creates all these candidate splits:
    - For each value v_i, a 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, it groups 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 searches the candidate cut-points $\{v_i\}$, which are the midpoints between two values belonging to objects of different classes. For each v_i, BuildTree creates a division with the properties $Feature \leq v_i$ and $Feature > v_i$ respectively.

We handle missing values introducing a penalizing factor to calculate the information gain. During the construction of a tree, when a split is evaluated, all the objects having a missing value in the feature associated to this split are grouped in a virtual child node with maximum entropy. This way, it is unlikely that a feature with several missing values can be selected as a good candidate.

LCMine extracts all the patterns from the trees, which are the conjunctions of properties in all the paths from the root node to the leaves. For example, from the decision tree in Figure 4.1, we can extract six patterns, like $(Length > 30) \wedge (Color = red)$ from class **Good** and $(Length \leq 30) \wedge (Fly = false)$ from class **Bad**.

Data: T – object collection to build the tree, k – diversity control parameter
Result: $ResultEP$
$EP \leftarrow \emptyset$;
// The i_j values control which of the best candidate splits is selected for expanding the tree nodes, according to each node level
for $i_1 \leftarrow 1$ **to** k **do**
  for $i_2 \leftarrow 1$ **to** $k-1$ **do**
    for $i_3 \leftarrow 1$ **to** $k-2$ **do**
      ...
      $Tree \leftarrow BuildTree(T, 1, \{i_1, i_2, i_3, \ldots\})$;
      $EP \leftarrow EP \cup ExtractPatterns(Tree)$
    end
  end
end
$EP \leftarrow RemoveDuplicates(EP)$;
$EP \leftarrow Simplify(EP)$;
$ResultEP \leftarrow EPS(T, EP)$

Algorithm 4.1: 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 [58];
  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

Algorithm 4.2: BuildTree Algorithm

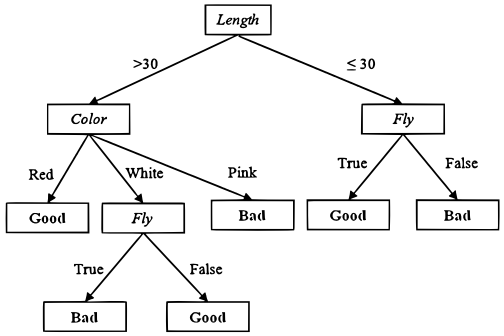

Figure 4.1: Example of a decision tree with three features and two classes: **Good** and **Bad**

After removing duplicated patterns, we simplify each pattern to obtain more compact patterns, 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 $Age \in (30, 50]$

4.3.2 Filtering Strategy

LCMine filters the resulting patterns using procedure EPSF (Algorithm 4.3). LCMine assigns to each pattern P a weight w_P equal to the amount of objects that it supports in its own class. Filtering emerging patterns is a challenge because of two main reasons:

1. Subjective nature of the definition of a “good” pattern.
2. Total amount of possible pattern subsets is 2^n where n is the number of patterns.

This section introduces a filtering strategy based on *redundancy* reduction in the computed emerging pattern set. We compute the *redundancy* of each pattern as the amount of objects covered in the training dataset that are also covered by another(other) pattern(s) defined over the same feature set.

Having highly redundant patterns might cause biased classifiers; for example, classifiers that usually tend to classify most of the objects in the same class.

The following example shows why the introduced strategy takes into account patterns defined over the same feature set. Let $P_1 = ([x_1 = large] \wedge [x_3 \leq 0.4] \wedge [x_4 > 3])$ and $P_2 = ([x_2 > 27] \wedge [x_5 \neq white] \wedge [x_7 > 6.8])$ be two patterns, covering the same objects in the training dataset. Notice that these patterns 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 P_1 is redundant with respect to P_2 and consequently we remove P_1, this removal might be a wrong decision. For example, the object $x = (large; 25; 0.1; 7; blue; true; 5)$ might be now misclassified because it does not satisfy P_2. Nevertheless, it satisfies P_1, which was removed from the final emerging pattern set. A proper algorithm, like the one that we propose, can avoid this undesirable behavior taking into account the redundancy in patterns defined over the same feature set.

The proposed algorithm follows the hypothesis that a good emerging pattern subset must reduce redundancy as much as possible. The Emerging Pattern Selection by Feature set (EPSF, Algorithm 4.3) algorithm applies the Emerging Pattern Selection (EPS, Algorithm 4.4) algorithm on each pattern group, defined over the same feature set. EPS selects a small size pattern subset that covers the same objects as the whole group does. EPS scores each pattern using (4.1).

$$score_P = \frac{|S(P, class(P), T)|}{\max\limits_{\substack{z \in Z \\ z \neq class(P)}} \{|S(P, z, T)|\}} \tag{4.1}$$

where Z is the set of classes, $S(P, z, T)$ is the subset of objects from T with class label z that satisfies the pattern P, and $|A|$ is the cardinality of the set A. This weighting scheme favors patterns covering more objects of their class and fewer objects from different classes. Finally, EPS builds the resultant pattern subset adding the best patterns one by one, only if the pattern covers a yet uncovered object.

4.3.3 Emerging Pattern-Based Classifier

In order to test the quality of the emerging patterns found by LCMine, it is necessary to build a supervised classifier. Like traditional Emerging Pattern-

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

Algorithm 4.3: Emerging Pattern Selection by Feature set algorithm(EPSF)

Data: T – Training sample, L – set of emerging patterns
Result: L' – selected emerging patterns
foreach *pattern* $P \in L$ **do**
  calculate $score_P$
end
Sort L in descending order according to $score_P$;
$L' \leftarrow \emptyset$;
foreach *pattern* P *in the sorted set* L **do**
  if *there is any object that supports* P *and does not support any pattern already in* L' **then**
    $L' \leftarrow L' \cup \{P\}$
  end
end

Algorithm 4.4: Emerging Pattern Selection algorithm(EPS)

based and KORA type classifiers, our classifier uses a scoring function. Given a query object q this function computes the total score for class C_i aggregating the weight of the emerging patterns contained in q. We compute the score per class using (4.2).

$$score(q, C_i) = \sum_{\substack{P \in P_i \\ q \in P}} w_P \tag{4.2}$$

where w_P is the weight of the pattern P (assigned by LCMine as the amount of objects that support P on its own class), and P_i is the collection of all emerging patterns for class C_i.

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

4.4 Crisp Emerging Pattern Mining

Crisp Emerging Pattern Mining (CEPM) is an enhanced version of LCMine. CEPM is faster and more accurate than LCMine, because it includes the following improvements:

- CEPM uses a novel weighting scheme for mining diverse patterns and it uses a stop criterion based on pattern coverage. This way, it does not have to generate a fixed amount of trees like LCMine does.
- CEPM does not need a pattern filtering post-processing. Nevertheless, it obtains fewer and more accurate patterns than LCMine does.
- CEPM assigns weights to the objects according to the support they have with the current mined patterns. This information is used in the generation of the subsequent decision trees. This way, CEPM prioritizes new patterns covering unsupported objects or objects supported in a wrong class.
- CEPM uses a novel algorithm for estimating the minimal support threshold.

The main improvement in CEPM is to use the patterns mined so far to guide the construction of subsequent decision trees. To attain this, CEPM weights all the objects in the training sample after each iteration. Higher weights are assigned to objects which are not covered by the current set of patterns, while weights close to zero are assigned to objects covered by many patterns. The decision tree built on each iteration is the one that best represents the objects with higher weights, so CEPM induces decision trees using the *weighted information gain*. The weighted information gain is similar to the classical information gain (4.3) but there is a weight associated to each object.

$$IG(N) = Imp(N) - \sum_{N_{ch} \in childern(N)} P_{N_{ch}} \cdot Imp(N_{ch}) \tag{4.3}$$

where

$$Imp(N) = - \sum_{class \in ClassesN} (P_{class} \cdot \log(P_{class})) \tag{4.4}$$

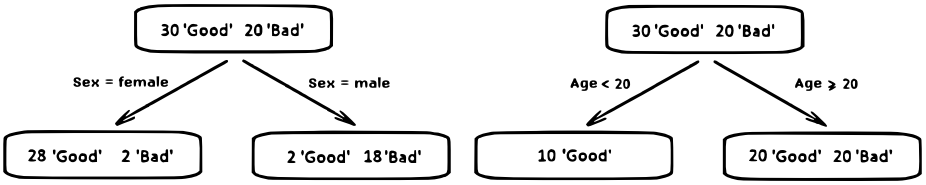

Figure 4.2: Example of an emerging pattern appearing in a non-optimal candidate split

The weighted information gain uses a modified class probability on a node P_{Class} and a modified child probability P_{child} (4.5). Note that objects with weight close to 0 have low influence in the determination of the best split. CEPM handles incomplete data with the same procedure used by LCMine (Section 4.3).

$$P_{Class} = \frac{\sum_{o \in Class} w_o}{\sum w_o}, \quad P_{child} = \frac{\sum_{o \in child} w_o}{\sum w_o} \quad . \tag{4.5}$$

CEPM extracts emerging patterns during the induction procedure; every tree node that, (I) has at least μ objects in a class, and (II) has at most one object in the complement of that class [3] , generates a new emerging pattern. Additionally, CEPM extracts patterns while evaluating the splits, even if a split has not the highest gain; any tree node that fulfills (I) and (II) generates an emerging pattern. For example, Figure 4.2 shows two candidate splits, using different properties. Although the first one has the highest information gain, the second contains the emerging pattern $(Age < 20)$. So, this pattern is extracted although the split is discarded.

CEPM iteratively induces several decision trees, updating the object weights after each induction. The algorithm updates the weights using (4.6).

$$w_o = \frac{\operatorname{arccot}\left(5 \cdot \frac{Support_o}{averageSupport}\right)}{\pi} \quad . \tag{4.6}$$

where

[3] Traditional emerging pattern classifiers allow certain amount of objects in the complement of the class. The classifiers introduced in this dissertation allow only a single object in the complement of the class, which is considered as noise.

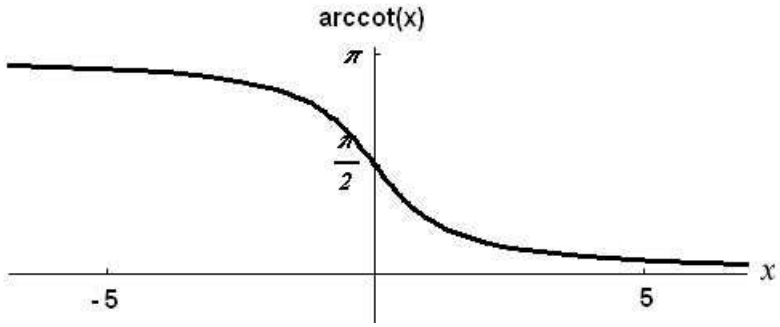

Figure 4.3: The *arccot* function

- $Support_o$ is the sum of the support of such patterns contained in o. If a pattern belongs to a different class than the o class, its support is multiplied by -1.

- $averageSupport$ is the average support of the patterns in the database, which is estimated based on the patterns found in the first built tree.

Equation 4.6 ranges from 0 to 1, because arccot ranges between 0 and π (Figure 4.3). An object obtains a weight close to 1 if it has a negative support lower than -5. In this case, it is necessary to mine more patterns to support the object to its own class. On the contrary, a weight close to 0 means the object has support above 5. In this case, no more patterns are necessary for this object, and it is virtually ignored in information gain calculations. As you can see in Figure 4.3, values 5 and -5 are both distinctive in the arccot function, because those are the points where the function values start to be close to the asymptotes. That is why we use value 5 in equation 4.6.

It is important to mention that the property in the topmost node in each tree is not allowed to appear in any other node in further generated decision trees. This property is usually highly discriminant therefore it is present in most decision trees, so the pattern miner would extract many useless non-minimal sub-patterns. Discarding such properties helps to obtain more diverse patterns.

The pseudocode of CEPM appears in Algorithm 6. It is worth to mention that CEPM returns a set of the most general emerging patterns with support greater or equal to μ. A pattern P is more general than a pattern Q if the set of objects described by Q is strictly contained in those described by P,

Data: T - training sample, μ - minimum support threshold, *maxIter* - maximum number of iterations
Result: EPS - Mined Patterns, *abstentionRatio* - Abstention ratio of EPS with respect to T
forall $o \in T$ **do** $w_o \leftarrow 1$;
/* *InduceTree* stores the topmost property in a *NotAllowed* list */
$NewTree \leftarrow InduceTree(T, w, \mu)$;
/* Simplify procedure deletes duplicated and less general patterns */
$EPS \leftarrow Simplify(ExtractPatterns(NewTree))$;
$averageSupport \leftarrow \frac{\sum_{ep \in EPS} support(ep)}{|EPS|}$;
$i \leftarrow 0$;
repeat
  $AbstentionCount \leftarrow 0$;
  // Weight recalculation
  foreach $o \in T$ **do**
    $EPc \leftarrow$ {Patterns that support o, belonging to its class};
    $EPnc \leftarrow$ {Patterns that support o, belonging to different classes};
    $support = \sum_{ep \in EPc} support(ep) - \sum_{ep \in EPnc} support(ep)$;
    $w_o = \text{arccot}\left(5 \cdot \frac{support}{averageSupport}\right)/\pi$;
    if *no pattern supports o* **then**
      $AbstentionCount \leftarrow AbstentionCount + 1$
  end
  $EPS \leftarrow Simplify(EPS \bigcup ExtractPatterns(InduceTree(T, w, \mu)))$;
  $i \leftarrow i + 1$
until $i =$ *maxIter OR no new pattern was added in this iteration* ;
return $(EPS,\ abstentionRatio = \frac{AbstentionCount}{|T|})$

Algorithm 4.5: Pseudocode of the algorithm CEPM, which extracts a representative collection of patterns from T

considering all the objects in the universe.

The CEPM classifier (CEPMC) uses the same decision rule than the LCMine classifier does: it assigns the query object q to the class with the maximum value of the total votes given by the patterns supported by q. Every pattern supported by q votes for its own class with its total support. If no pattern supports q or there is a tie in the votes, the classifier refuses to classify q, counting this abstention as an error.

4.4.1 Estimating the Minimal Support Threshold for CEPM

Selecting the minimal support threshold for an emerging pattern classifier is a difficult task; a classifier using patterns with higher μ value is a more accurate classifier, but it could reject to classify more objects. On the contrary, a classifier using patterns with lower μ values might contain many useless patterns, which could degrade its classification accuracy.

CEPM finds an accurate estimation of the minimal support threshold μ, testing different values decrementally. The rationale of this procedure holds in the following property: if we classify using patterns with lower μ values, we obtain less accurate classifiers with lower abstention levels[4]. To use this property, CEPM infers two values:

- μ_{ini} is a high enough value, so any classifier built with $\mu > \mu_{ini}$ is inaccurate.
- $minAbst$ is the minimal abstention level we can expect to attain with the current dataset.

Then, CEPM decrementally tests the μ values, starting from μ_{ini}, until the built classifier attains an abstention level smaller than or equal to $minAbst$. As a consequence of this procedure, CEPM returns a set of emerging patterns with the highest support value associated with the lowest expected abstention rate.

[4] A definition of classifier abstention can be found in Section 2.3.

4.5 LCMine, CEPM, and Over-fitting

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]. Following this, as complex models tend to be overtrained, it may seem that LCMine and CEPMC, using the mined emerging patterns, would be overfitted. Nevertheless, some characteristics of the proposed classifier avoid this drawback:

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

4.6 Cascading CEPM-based classifiers

A cascade classifier is a type of ensemble where a single classifier is active at each time [42]. To classify a query object, the first classifier is activated and returns the class. If a classifier is not sure enough about the correct classification, it passes the query object to the next classifier in the chain. The cascading classifier approach is better than multi-expert methods [5] when the topmost classifiers can handle most objects with higher accuracy, but they are unable to classify some other objects [3].

Emerging pattern classifiers with different minimal support μ are good candidates for cascading; a classifier using patterns with higher μ values, is more accurate but could reject to classify more objects. Then, to build a cascade of emerging pattern classifiers, the first classifier should be built with a high μ value. The remaining classifiers in the cascade should have a

[5] In multi-expert methods all the classifiers are activated at once, and the final classification is obtained aggregating the individual votes. Majority class, bagging and boosting ensembles are examples of this approach.

Data: T - training sample
Result: $Classifiers$ - Cascade of classifiers. The first classifier is the one built with the highest μ value
$(\mu_{ini}, MinAbst) \leftarrow InferParams(T)$ `// in Algorithm 4.7`
$currentSupport \leftarrow \mu_{ini}$;
$step \leftarrow \max\left\{\frac{currentSupport}{10}, 1\right\}$;
while $currentSupport > 1$ *AND* $currentAbstention > MinAbst$ **do**
  $(patterns, currentAbstention) \leftarrow CEPM(T, maxIter = 120, \mu = currentSupport)$;
  if $currentAbstention < 0.5$ **then**
    Add a new classifier using *patterns* to $Classifiers$
  $currentSupport \leftarrow currentSupport - step$;
  if $currentSupport = 0$ *AND* $abstentionRatio >= MinAbst$ **then**
    $currentSupport \leftarrow step - 1$;
    $step \leftarrow \max\left\{\frac{currentSupport}{10}, 1\right\}$;
  end
end

Algorithm 4.6: Pseudocode of the CascadeCEPM algorithm, which builds a cascade of CEPM-based classifiers

Data: T - training sample
Result: μ_{ini} - Maximal μ value used in the classifier ensemble, *minAbst* - Expected minimal abstention level
$(Patterns, minAbst) \leftarrow CEPM(T, maxIter = 20, \mu = 2)$;
`/* Using the highest support value makes` μ_{ini} `to be high enough; checking that at least half of the objects be supported makes` μ_{ini} `to be not too high */`
$\mu_{ini} \leftarrow$ Highest support value such that at least half of the objects in T are supported for at least a pattern in *Patterns* ;
return *(*μ_{ini}*, minAbst)*

Algorithm 4.7: Pseudocode of the InferParams algorithm, used to predict the values of maximal support and maximum abstention rate in CascadeCEPM

μ value lower than the μ value of their predecessors, to allow them to classify uncovered objects.

Our novel cascading creation method, named CascadeCEPM, appears in Algorithm 4.6. It starts inferring the support of the topmost classifier (μ_{ini}) and the minimal expected abstention rate for the lower classifier ($minAbst$). CascadeCEPM creates classifiers starting with $\mu = \mu_{ini}$, decrementing μ for each new classifier until an abstention rate lower than $minAbst$ is achieved or $\mu = 1$. For decrementing μ, CascadeCEPM uses a calculated $Step = \mu_{ini}/10$, because if μ_{ini} is high, decrementing μ with $Step = 1$ might be too costly.

Some important remarks:

1. μ_{ini} and $minAbst$ are inferred like in the CEPM classifier (Section 4.4).
2. $minAbst$ is inferred using $\mu = 2$, so it measures the minimal expected abstention of a pattern based classifier. A maximum iteration value equal to 20 speeds up the procedure (Algorithm 4.7, step 1).
3. We dismiss classifiers with abstention level higher than 0.5, because they are usually inaccurate, according to our experiments.
4. The value 120 (Algorithm 4.6, Step 5) is the maximum number of iterations of the algorithm CEPM. It is necessary to use this value because CEPM converges slowly[6] in some databases. As in each iteration the mined patterns have less quality, stopping the algorithm in a high enough iteration does not significantly alter the classifier quality.
5. The condition $currentAbstention < 0.5$ in Algorithm 4.6 (step 6) discards inaccurate classifiers, having high abstention levels.

CascadeCEPM creates a cascade of emerging pattern classifiers, each one using the highest sum of support as decision rule. Given a query object, the first classifier returns the most supported class; if no pattern supports the object or there is a tie, the classifier refuses to classify and activates the next classifier in the cascade. If the last classifier cannot classify the query object, the whole cascade refuses to return a classification, counting this abstention

[6]CEPM always converge, because there is a finite amount of emerging patterns in any dataset. It is easy to prove this affirmation, because even numerical features have a finite amount of candidate cut-points.

Table 4.1: Description of the databases used in the experiments

DBName	Objects	#Attr	#Class	DBName	Objects	#Attr	#Class
autos	205	25	4	balance-scale	625	4	3
breast-cancer	286	9	2	breast-w	699	10	2
cleveland	303	13	2	credit-screening	690	15	2
cylinder-bands	540	39	2	diabete	768	8	2
glass	214	9	6	haberman	306	3	2
hayes-roth	160	4	3	heart-c	303	13	2
heart-h	294	13	2	heart-statlog	270	13	2
hepatitis	155	19	2	ionosphere	351	34	2
iris	150	4	3	labor-neg	57	16	2
liver	345	6	2	lymph	148	18	4
monkproblem1	556	6	2	monkproblem2	601	6	2
monkproblem3	554	6	2	mushroom	8124	22	2
sick	4744	29	2	sonar	208	60	2
spect	267	22	2	tic-tac-toe	958	9	2
vote	433	16	2	wdbc	569	30	2
wine	178	13	3	wpbc	198	33	2

as an error. Classifying with this cascade is faster than using CEPMC, because most objects are classified using top classifiers, which uses a reduced subset of patterns[7].

4.7 Experimental Results

To compare the performance of the LCMine, CEPMC, and CascadeCEPM classifiers, we carried out some experiments over 33 well-known databases from the UCI Repository of Machine Learning [50]. We include a representative collection of the databases most commonly used in previous pattern based publications. We also include some databases where pattern based classifiers obtain poor results. The general description of tested databases appears in Table 4.1. In this table, The column Objects contains the number of objects in the database while the columns #Attr and #Class have the number of attributes and classes, respectively.

For comparison, we selected six state-of-the-art classifiers: Nearest Neighbors [17], Bagging and Boosting [42], Random Forest [37], C4.5 [58] and

[7]The number of patterns is proportional to the classification speed, because a query object has to be tested with all the patterns.

Table 4.2: Default parameters used in the Weka classifiers

Classifier	Weka Name	Parameters
3-NN	IBK	-K 3 -W 0
AdaBoost	AdaBoostM1	-P 100 -S 1 -I 10
Bagging	Bagging	-P 100 -S 1 -I 10 -o -W
		REPTree – -M 2 -V 0.0010 -N 3 -S 1 -L -1
C4.5	J48	-C 0.25 -M 2
Random Forest	RandomForest	-I 10 -K 0 -S 1
SVM	SMO	-C 1.0 -E 1.0 -G 0.01 -A 250007
		-L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1

Support Vector Machines [15]. For each classifier, we used the Weka 3.6.1 implementation [29] with its default parameters (Table 4.2). We also tested SJEPC [27], which is one of the most accurate emerging pattern based classifiers, using the minimal support threshold suggested by their authors. To make a fair comparison, our classifiers use the same parameters on all the databases.

We performed 10-fold cross validation, averaging the results. In emerging pattern classifiers LCMine, SJEPC, CEPMC and CascadeCEPM we reported the abstentions as errors. In these objects, the classifier is unable to assign a class; returning the majority or a random class could hide these undesirable cases. Table 4.3 presents the accuracy results, in percentage.

Experimental results show that SJEPC has low accuracy values in many databases, compared to other classifiers. In those databases, most numerical features were discretized into a categorical feature with a single value, so they were useless for mining patterns. CEPMC and CascadeCEPM have higher accuracies than SJEPC does in most databases.

In order to determine if the differences in accuracy are statistically significant, we performed a pairwise comparison between all the classifiers. Each cell in Table 4.4 contains the number of databases where the classifier in the row significantly wins/loses with respect to the classifier in the column. We detected ties using a two-tailed T-Test [18] with significance of 0.05.

The pairwise comparison shows that, in the tested databases, our classifiers are more accurate than other understandable classifiers, while being competitive with Nearest Neighbors and Support Vector Machines classifiers. On the other hand, CEPMC is more accurate than LCMine, and slightly more

Table 4.3: Accuracy results of the tested classifiers in the selected databases. The highest accuracy per database is bolded

DBName	3nn	Boost	Bagg	c4.5	RandFor	SVM	SJEPC	LCMine	CEPMC	CascCEPM
autos	68.2	42.4	61.1	81.0	81.0	66.0	12.3	80.1	79.8	**83.5**
balance-scale	**89.3**	71.7	82.6	77.6	79.4	87.5	16.0	75.1	79.5	82.7
breast-cancer	**74.5**	72.4	71.0	73.4	65.8	70.7	44.5	70.7	72.0	72.3
breast-w	96.5	95.6	95.6	95.9	96.5	97.0	96.3	**97.4**	96.0	94.1
cleveland	81.8	84.2	79.9	78.2	78.6	**84.5**	77.9	82.2	81.2	81.5
credit-screening	84.2	86.3	85.9	85.1	85.0	86.3	82.6	86.8	85.9	**87.6**
cylinder-bands	70.2	72.9	60.4	72.2	78.5	80.9	64.3	**81.8**	77.8	75.1
diabete	74.9	75.4	75.7	**76.4**	75.4	74.1	76.2	73.1	75.7	74.0
glass	**69.0**	44.7	73.8	67.7	70.9	57.1	20.4	68.2	63.2	58.4
haberman	72.2	70.9	**72.5**	68.0	67.6	72.5	0.0	70.9	68.6	71.6
hayes-roth	50.0	53.6	75.0	**89.3**	85.7	53.6	0.0	53.2	78.6	75.0
heart-c	81.2	**83.2**	81.9	76.2	80.9	82.8	78.6	81.3	81.2	82.2
heart-h	**83.0**	81.6	79.9	79.6	79.3	82.6	46.3	79.5	81.0	80.2
heart-statlog	81.1	80.7	79.3	79.3	79.3	**83.0**	64.8	79.2	80.0	80.0
hepatitis	**86.0**	83.6	82.0	81.8	82.4	85.2	83.2	79.4	82.5	81.3
ionosphere	85.5	91.4	90.3	90.3	92.6	88.0	**93.5**	91.1	89.5	90.2
iris	96.6	**97.8**	95.8	95.8	95.2	97.0	75.3	96.5	95.3	95.3
labor-neg	**90.7**	87.0	84.0	80.0	86.7	90.7	82.0	80.9	89.0	79.3
liver	62.6	66.1	68.7	68.7	**70.7**	57.7	0.0	70.2	69.3	69.9
lymph	85.9	75.7	77.7	78.5	79.9	**87.9**	43.9	85.5	83.7	82.5
monkproblem1	81.0	75.0	50.0	88.9	75.7	50.0	86.8	95.6	**100.0**	**100.0**
monkproblem2	61.3	60.7	55.1	69.9	65.1	50.5	71.1	75.0	**83.8**	79.2
monkproblem3	88.2	97.2	50.0	96.3	97.2	50.0	93.5	97.5	97.5	**97.5**
mushroom	**100.0**	96.3	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	99.7	**100.0**
sick	96.0	97.2	95.1	**98.7**	98.2	97.6	96.8	96.8	97.8	96.9
sonar	82.3	77.4	76.4	68.4	81.8	82.0	**85.1**	75.7	78.8	77.1
spect	64.7	66.8	61.5	66.8	62.0	67.9	0.0	66.8	78.1	**83.4**
tic-tac-toe	97.9	74.7	85.3	85.3	92.6	95.8	**98.8**	97.0	96.5	94.4
vote	92.2	94.7	95.2	**96.1**	**96.1**	95.9	91.1	94.5	94.0	94.5
wdbc	96.3	92.3	94.4	91.4	94.0	**97.7**	85.1	94.3	95.6	95.3
wine	96.0	87.5	94.3	92.7	97.2	**98.9**	55.1	87.9	93.3	95.0
wpbc	72.3	71.0	78.8	75.5	74.7	75.9	0.0	68.4	79.8	**80.8**

accurate than CascadeCEPM. Nevertheless, CascadeCEPM is faster in the classification stage.

The model built by our classifiers is easy to understand in terms of the problem domain, unlike the Nearest Neighbors and the Support Vector Machines models. Each class is described as a collection of discriminative properties, as you can see in the example appearing in Table 4.5.

Table 4.4: Pairwise comparison between the tested classifiers. Each cell shows the number of times the classifier in the row Wins/Losses with respect to the classifier in the column, over the 33 databases

	3nn	Boost	Bagg	c4.5	RandFor	SVM	SJEPC	LCMine	CEPMC	CascCEPM
3nn		13/9	14/6	12/11	12/10	8/9	20/5	8/11	8/10	10/12
Boost	9/13		8/10	5/12	7/13	5/13	22/6	5/10	3/15	4/15
Bagg	6/14	10/8		8/9	5/13	6/15	18/7	8/9	3/12	2/8
c4.5	11/12	12/5	9/8		5/9	10/14	20/4	7/10	3/12	3/16
RandFor	10/12	13/7	13/5	9/5		8/13	21/4	9/9	6/9	8/11
SVM	9/8	13/5	15/6	14/10	13/8		21/7	11/7	10/9	12/9
SJEPC	5/20	6/22	7/18	4/20	4/21	7/21		4/23	3/24	6/23
LCMine	11/8	10/5	9/8	10/7	9/9	7/11	23/4		3/11	5/8
CEPMC	10/8	15/3	12/3	12/3	9/6	9/10	24/3	11/3		6/4
CascCEPM	12/10	15/4	8/2	16/3	11/8	9/12	23/6	8/5	4/6	

Table 4.5: Classifier model built by CEPM for one of the folds in the Iris database

iris-setosa
$[PetalLength \leq 1.90]$
$[PetalWidth \leq 0.60]$
iris-versicolor
$[PetalLength > 1.90] \wedge [PetalLength \leq 4.90] \wedge [PetalWidth \leq 1.60]$
iris-virginica
$[PetalLength > 1.90] \wedge [PetalWidth > 1.60]$
$[PetalLength > 4.90]$

4.8 Summary

In this chapter, we formalized the concept of extended crisp emerging pattern. We introduced two algorithms for mining this kind of patterns from databases with mixed and incomplete data. The first algorithm, named LCMine, uses local discretization of numerical values to solve the global discretization drawback of previous emerging pattern classifiers. LCMine extracts patterns from a collection of decision trees, using a special extraction procedure during the tree induction. To overcome the limitations of LCMine, we introduced an enhanced version, named CEPM. To obtain a collection of representative patterns, CEPM uses a novel object weighting scheme. Furthermore, this chapter proposed an algorithm for accurately estimating the minimal support threshold for CEPM.

We proposed also a cascade of emerging pattern classifiers. This cas-

cade combines the higher accuracy of classifying with patterns having higher support thresholds with the lower levels of abstention of classifying with patterns having lower support thresholds.

Experimental results showed that CEPMC and CascadeCEPM are more accurate than one of the most accurate emerging pattern classifiers, in the majority of tested databases. A pairwise comparison revealed that both classifiers are more accurate than other understandable classifiers, and as accurate as Nearest Neighbors and Support Vector Machines, while the classification model built by them is easy to understand in terms of the problem domain. LCMine, on the other hand, is less accurate than the CEPMC and CascadeCEPM classifiers. To select which classifier to use in a particular problem we must consider that CEPMC is more accurate, but CascadeCEPM is faster in the classification stage.

In order to improve some of the limitations of our algorithm, the following approaches can be studied:

- Speeding up the algorithm to estimate the minimal support threshold, which is the slowest component of CEPM.
- Speeding up the algorithm to estimate the support threshold for the classifiers in the CascadeCEPM cascade.
- Studying the impact of the weighting recalculation function in the quality of the patterns and the convergence of the algorithms, to propose new weighting schemes.

Chapter 5

Fuzzy emerging pattern mining

5.1 Introduction

Fuzzy sets [71] offers a solution to discretization problems in many domains. Fuzzy sets have contributed to improve many mining tasks, like mining fuzzy association rules from uncertain data [69]. This chapter introduces a new kind of emerging pattern, named *Fuzzy Emerging Pattern* (FEP). Fuzzy emerging patterns are patterns formed by fuzzy selectors with the structure $[Feature \in FuzzySet]$, joined by fuzzy AND operators. This way an object satisfies a given pattern in certain degree, according to the degree the object feature values satisfy the property expressed in the pattern. Fuzzy patterns alleviate some drawbacks of crisp patterns:

- Crisp boundaries in numerical features are eliminated. Let us consider the pattern $[Weight \geq 24]$. Note that an object with $Weight = 23.99$ does not fulfill the pattern at all, having a very small difference with respect to the pattern.
- The pattern fulfillment relation is more flexible, because objects can fulfill a fuzzy pattern in certain degree. This way, classifier abstention for lack of evidence is less frequent.
- Fuzzy patterns are easier to read and to explain in terms of the problem domain. Using them we can find patterns like “Temperature is high and

Pressure is above normal", where "high", "above", and "normal" are defined by the user, according to his background knowledge.

To efficiently extract fuzzy emerging patterns from a database, we use a set of fuzzy decision trees, induced using a new algorithm that includes linguistic hedges. Linguistic hedges allow modifying the semantics of the initial fuzzy discretization to satisfy the semantics of the different classes in the training sample [32]. We also propose a new classifier based on fuzzy emerging patterns, which includes a novel mechanism for the aggregation of pattern votes. This classifier outperforms many state-of-the-art classifiers in most of the tested databases.

5.2 Fuzzy Emerging Patterns

Fuzzy emerging patterns are an extension of crisp emerging patterns, where the objects support the patterns in a certain degree. First, we introduce the concept of *Fuzzy Pattern*.

Definition 5.1. *A* Fuzzy Pattern *is a conjunction (using a T-Norm*[1]*) of* selectors $[Feature \in FuzzySet]$*, where* $\in$ *is the membership of the* $Feature$ *value to* $FuzzySet$*; in this paper, these selectors will be named* F-selectors*.*

Example 5.1. $[Temperature \in hot] \wedge [Humidity \in normal]$ *and* $[Outlook \in sunny] \wedge [Windy \in true]$ *are fuzzy patterns. Additionally, any number of linguistic hedges can modify each* $FuzzySet$*, like in* $[Temperature \in very(hot)] \wedge [Humidity \in somewhat(normal)]$.

Instead of the phrase "an object supports a pattern", commonly used with the classical emerging patterns, each object supports every fuzzy pattern in a certain *degree* between zero and one. If the degree is one, the object fully supports the pattern; if the degree is zero, the object does no support the pattern. The individual fuzzy support of a fuzzy pattern fp with respect to an object o, denoted as $fsup(o, fp)$, can be calculated as the minimum value (or another T-norm) of the membership μ of every feature value of o to the fuzzy set in the respective F-selector fs of the fuzzy pattern fp:

$$fsup(o, fp) = \min_{fs \in fp} \{\mu_{fs}(o)\}$$

[1] An introduction to fuzzy set operators can be found in Section 2.9.1.

In a problem with multiple classes, each fuzzy pattern fp has a different support in every class C_i, which is calculated as the sum of the individual fuzzy support for all objects in C_i.

$$FSup(fp, C_i) = \sum_{o \in C_i} fsup(o, fp)$$

In order to measure the relevance of each fuzzy pattern for classification, we introduce the concept of *Trust*.

Definition 5.2. *The* trust *of a fuzzy pattern fp is:*

$$Trust(fp) = \frac{\max_{C_i} FSup(fp, C_i)}{\sum_{C_i} FSup(fp, C_i)}$$

The $Trust$ of a fuzzy pattern measures the support ratio of fp in the class with the highest support, with respect to the total support of fp in all the classes. We can understand the $Trust$ as the membership degree of a given pattern to the fuzzy set of "good patterns for classification". It is a measure that evaluates how good a pattern is to discriminate between classes, and we will use it to select fuzzy emerging patterns.

Definition 5.3. *A* Fuzzy Emerging Pattern *(FEP) is a fuzzy pattern with* $Trust > 0.5$.

A $Trust$ above 0.5 means the pattern has higher support in the highest supported class than in the combined remaining classes. That is why we use this value as cutting-point to determine which patterns are good for supervised classification. It is worth to mention that a pattern P whose support in a class is significantly higher than its support in the remaining classes has a higher $Trust$ than a pattern Q with lower differences; since the $Trust$ is used as voting weight, P has a higher influence in the final classification of the objects than Q.

5.3 Mining Fuzzy Emerging Patterns

Extracting fuzzy patterns from a given training sample is the first step for classification using fuzzy emerging patterns. Initially, we fuzzify all features,

according to their types. For each categorical feature, we create a collection of singleton fuzzy sets, i.e. for each different value, we create a fuzzy set having membership 1 for that value, and 0 for the remaining values. For each numerical feature, we apply a traditional fuzzification algorithm, which transforms the feature in a fuzzy Ruspini partition [2]. Next, we extract fuzzy emerging patterns from a set of diverse fuzzy decision trees induced from the training sample. The induction algorithm is a variant of the ID3 method [57] for the fuzzy case, with the following diferences:

- Candidate splits use the following fuzzy sets and their fuzzy negation:
  1. All the fuzzy sets obtained in the fuzzyfication step.
  2. Fuzzy sets in the previous item, modified by all different predefined hedges. The hedges must be defined by the user according to the problem domain.
- In classical ID3, each object in a decision node is assigned to a single child node. In the fuzzy version, each object is assigned to all child nodes with different membership value. This way, every object belongs to all the nodes in the fuzzy tree with a different membership value[3]. To calculate the object membership in a child node, our algorithm applies a fuzzy AND between the object membership in the parent node and the membership of the object to the fuzzy set associated to the child node.
- If the membership of an object to a particular node is below a given threshold μ_{min}, the object is deleted from the node.
- In the fuzzy version, we use the following stopping criteria:
  - The node is pure, i.e. all the objects in the node belong to the same class.
  - The node is empty.
  - No split provides an improvement in the quality function.

[2] In a fuzzy Ruspini partition the total membership of every value in the domain is equal to one. For example, the fuzzy sets in Figure 5.1 are Ruspini partitions of the domains of Temperature, Rain, and Humidity respectively.

[3] In our mining method, if an object belongs to a node with membership below 0.05, it is discarded.

- The use of *fuzzy information gain* (fig) of a node N. The fig is an extension of the ID3 information gain for the fuzzy case [22]:

$$fig(N) = fimp(N) - \sum_{Nc \in child(N)} fimp(Nc) \cdot \frac{\sum_{o \in Nc} \mu_{Nc}(o)}{\sum_{o \in N} \mu_N(o)} \tag{5.1}$$

  where $\mu_N(o)$ refers to the membership of the object o to the node N, and the *fuzzy impurity* $fimp(N)$ is defined as:

$$fimp(N) = - \sum_{C \in classes(N)} \frac{\sum_{o \in C} \mu_N(o)}{\sum \mu_N(o)} \cdot \log\left(\frac{\sum_{o \in C} \mu_N(o)}{\sum \mu_N(o)}\right)$$

In the tree construction process, missing data are handled like in LCMine algorithm (Section 4.3).

To guarantee diversity among the trees used to extract fuzzy emerging patterns, we use the same searching procedure as for the LCMine algorithm (Section 4.3.1). The pseudocode of the pattern mining procedure appears in Algorithm 5.1.

Data: T - training sample, k - diversity control parameter
Result: $ResultFEP$
$FEP \leftarrow \emptyset$;
forall $o \in T$ **do**
  set $\mu_T(o) = 1$
end
// The i_j values control which of the best candidate splits is selected for expanding the tree nodes, according to each node level
for $i_1 \leftarrow 1$ **to** k **do**
  for $i_2 \leftarrow 1$ **to** $k-1$ **do**
    for $i_3 \leftarrow 1$ **to** $k-2$ **do**
      $\cdots$
      $Tree \leftarrow BuildNode\big(T, \{\mu_T(o)\}, \{i_1, i_2, i_3, \ldots\}\big)$;
      $FEP \leftarrow FEP \cup ExtractPatterns(Tree)$
    end
  end
end
$ResultEP \leftarrow RemoveDuplicates(FEP)$;

Algorithm 5.1: MinePatterns Algorithm

Example 5.2. *For example, Table 5.1 contains the description of three objects, using three numerical features: Temperature, Rain and Humidity. Figure 5.1*

Data: T – object collection to build the tree, $\{\mu_T(o)\}$ - associated membership of the object o to the current node, l – level in the tree of the resultant node, $\{k_{level}\}$ – set of k values for each level
Result: N – decision node
while *T has objects in more than one class* $\wedge$ $\exists C_i \in T : \sum_{o \in C_i} \mu_T(o) > \mu_{min}$ **do**
  Generate all candidate splits S_i using all fuzzy sets and hedges;
  Calculate $fig(S_i)$, the fuzzy information gain of every split S_i;
  Sort S_i in descending order according to $fig(S_i)$;
  Find S', the $k_l{}^{th}$ element of the sorted S_i collection;
  Construct the left and right child node fuzzy sets FS_{left} and FS_{right} according to S'. FS_{left} corresponds to the fuzzy set associated to S', while FS_{right} corresponds to the negated fuzzy set, using a fuzzy negation operator;
  Find the child node subsets T_{left} and T_{right}, according to the split S'.
  Calculate $\{\mu_{T_{left}}(o)\}$ and $\{\mu_{T_{right}}(o)\}$, as the product of its current value and the membership of the object to FS_{left} and FS_{right} respectively;
  $N_{left} \leftarrow BuildNode\Big(T_{left}, \{\mu_{T_{left}}(o)\}, l+1, \{k_{level}\}\Big)$;
  $N_{right} \leftarrow BuildNode\Big(T_{right}, \{\mu_{T_{right}}(o)\}, l+1, \{k_{level}\}\Big)$;
  Construct the decision node N, with child nodes N_{left} and N_{right} respectively;
end

Algorithm 5.2: BuildNode algorithm

shows the membership functions generated using a fixed-bin fuzzy discretization, with three bins.

To generate the tree $i_j = (2, 1, 3, 2, 1)$ the algorithm expands the root node, generating all candidate splits (Figure 5.2). Next, it calculates the fuzzy information gain of each candidate split (Figure 5.3) using (5.1), sorts the splits according to their fig (Figure 5.4), and selects the second highest value (Figure 5.5). Finally, we built a new decision node using the fuzzy sets associated with the selected split (Figure 5.6), and recursively apply the whole procedure to each child node until we find a leaf node. Figure 5.7 shows the complete fuzzy decision tree generated by the BuildNode algorithm (Algorithm 5.2).

From each tree, we extract all its fuzzy emerging patterns, which are the conjunctions of the properties in the paths from the root node to the leaves. Each pattern is assigned to the class with the highest fuzzy support. Finally, we remove duplicated patterns and patterns with a $Trust$ below 0.5, because

Table 5.1: Description of the objects used in the example

Object	Class	Temperature (^{o}C)	Rain (mm)	Humidity (%)
o1	Bolded	95	120	5
o3	Bolded	50	10	40
o2	Non-bolded	70	150	15

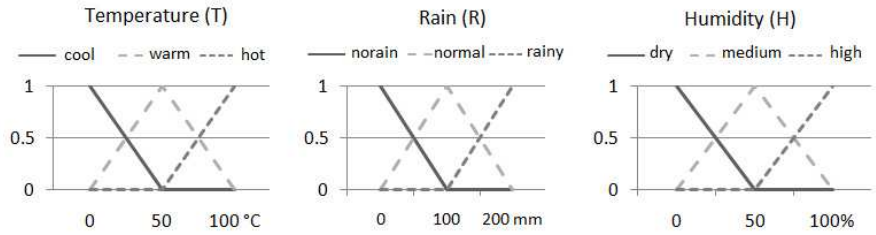

Figure 5.1: Fuzzy membership functions per feature used in the example

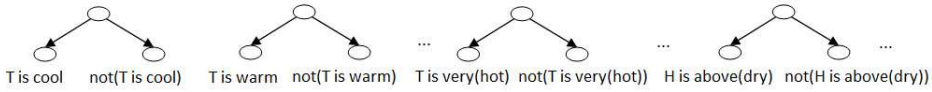

Figure 5.2: Candidate splits in the example

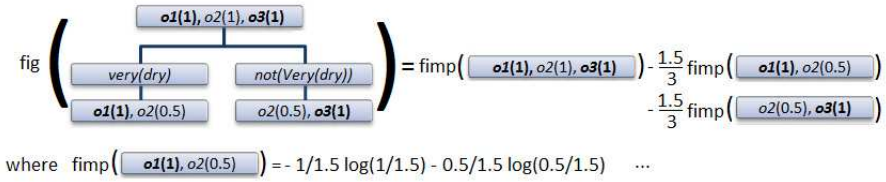

Figure 5.3: Evaluating the *fuzzy information gain* (fig) in a candidate split example

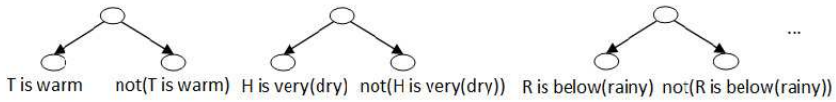

Figure 5.4: Sorting candidate splits in the example according to the fig value

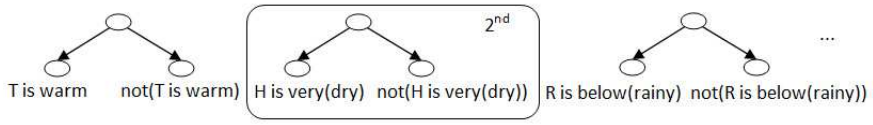

Figure 5.5: Selecting the second candidate split in the example

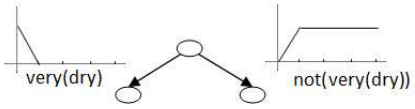

Figure 5.6: Creating a new decision node and its related fuzzy sets

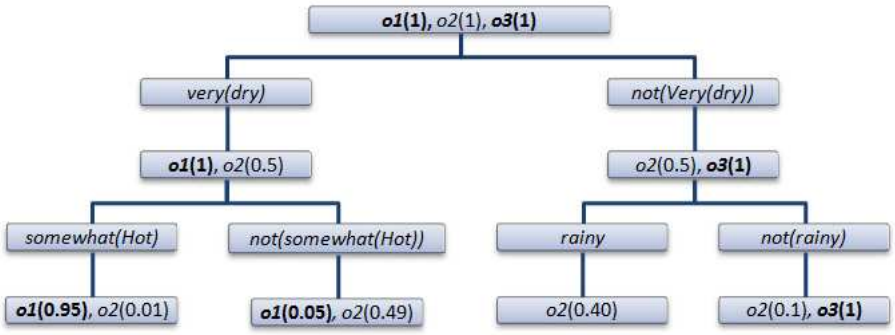

Figure 5.7: Fuzzy tree generated in the example. The membership of the objects to each node appears between parenthesis

Table 5.2: Fuzzy emerging patterns extracted from the tree in Figure 5.7

Class	FEP	$Trust$
Bolded	$very(dry) \wedge somewhat(hot)$	0.99
	$not(very(dry)) \wedge not(rainy)$	0.91
Non-bolded	$very(dry) \wedge not(somewhat(hot))$	0.91
	$not(very(dry)) \wedge rainy$	1.00

they are not fuzzy emerging patterns.

In the tree appearing in Figure 5.7 we have four leaves, so we extract the four fuzzy emerging patterns shown in Table 5.2. It is important to highlight the expressivity and easiness to understand of the extracted patterns.

5.3.1 Computational complexity

The computational cost of the proposed mining procedure is a constant ($k!$) multiplied by the cost of inducing each fuzzy decision tree. Extracting the fuzzy emerging patterns does not increase the complexity, because the patterns can be directly extracted from the trees during the induction procedure.

The algorithm we use to induce fuzzy decision trees has similar complexity than crisp decision tree induction algorithms; the following are the main differences:

- A fixed amount of splits are used for each numerical feature, instead of dynamically calculating the cut points, as in the crisp version.
- In the crisp version, every object is assigned to a single child, so in the case of balanced trees you get $\log(n)$ levels. In the fuzzy version, most objects are assigned to both child nodes with different membership values, so the tree has more depth.

Section 5.5.1 presents an experimental study about the scalability of the mining procedure by increasing the number of objects, features and discretized values.

5.4 Classifying with Fuzzy Emerging Patterns

To understand the classification stage of the classifier, we first extend the concepts of more particular pattern and more general pattern to the fuzzy case.

Definition 5.4. *An F-selector* $fs_1 \equiv [Attr_1 \in FS_1]$ *is more particular than another F-selector* $fs_2 \equiv [Attr_2 \in FS_2]$ *if* $Attr_1 = Attr_2$ *and* $FS_1 \subset FS_2$[4].

Definition 5.5. *Let* fep_1 *and* fep_2 *be two fuzzy emerging patterns.* fep_1 *is more particular than* fep_2 *if for all F-selectors in* fep_1, fep_2 *contains an equal or more particular f-selector.*

Definition 5.6. *Let* fep_1 *and* fep_2 *be two fuzzy emerging patterns.* fep_1 *is more general than* fep_2 *if* fep_2 *is more particular than* fep_1.

If two patterns contain linguistic hedges, we use the fuzzy subset inclusion between them to select the more general. This way, the F-selector $very(high)$ is more general than $extremly(high)$, but less general than $high$. The "more

[4]Fuzzy subset relation. A fuzzy set is subset of another fuzzy set if it has lower membership values for all the values in the domain.

general" relation is antisimetric, so a pattern can be unrelated with other patterns. According to Definition 5.6, an object supports a more general pattern with higher degree than a less general pattern.

As we reviewed in Section 3.1, particular patterns reduce duplicate pattern contribution [65] and provide more information about relationships between features [75], but they are harder to appear in query objects, generating abstention [65]. On the other hand, general patterns are more resistant to noise [25, 26] and can be mined with less computational effort [25]. Nevertheless, aggregating many minimal patterns may implicitly cause duplicate counting of individual pattern contributions, which could decrease classification accuracy [5, 65].

In this section, we introduce a novel strategy to build the classifier, which uses all available patterns. In our classifier, we build a pattern graph as follows:

- Nodes are associated with fuzzy emerging patterns.
- Arcs connect nodes with more particular patterns to nodes with more general patterns, according to the antisymmetric relation "is more particular than".

  When an arc A connects two nodes N_1 and N_2, if there is a longer path between both nodes, A is discarded. This procedure avoids considering a more general pattern while a more particular one is still untested.

In this graph, nodes with no ancestors are maximal patterns (more particular), because any other pattern contains it. Similarly, nodes with no successors are minimal patterns (more general). The algorithm for creating such graph is straightforward. We evaluate every possible pair of nodes, evaluating the fulfillment of the relation "is more particular than". If it holds, we create the corresponding arc. Finally, a post-processing step discards longer redundant paths.

To compute the votes per class of a query object, the proposed classifier named FEPC starts evaluating the patterns with no ancestors. If the evaluated pattern matches the query object (with a fuzzy support above a certain threshold) the vote to its class is increased with its $Trust$, while all its successors are discarded. Otherwise, all immediate successors are evaluated in the

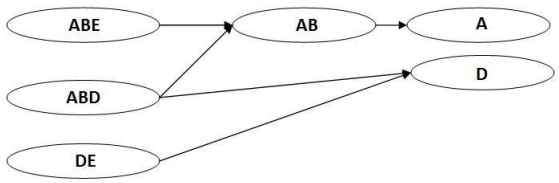

Figure 5.8: Example of pattern tree built by FEPC

same way. The process ends when every single node has been evaluated or discarded. Finally, FEPC assigns to the query object the class with the highest total vote (See Algorithm 5.3).

Data: q – query object to classify, G – fuzzy emerging pattern graph
Result: $Classification$ – class assigned by the classifier to the query object
$Processed \leftarrow \emptyset$;
foreach *class* C **do**
  $Votes_C \leftarrow 0$
end
$Pendant \leftarrow$ FEPs with no ancestors in G ;
foreach $fep \in Pendant$ **do**
  $Processed \leftarrow Processed \cup \{fep\}$;
  if fep *match* q **then**
    $Votes_{class(fep)} \leftarrow Votes_{class(fep)} + Trust(fep)$;
    $Processed \leftarrow Processed \cup \{$ descendants of fep in $G\}$
  else
    $Pendant \leftarrow Pendant \cup \{$ descendants of fep in $G\}$
  end
  if $\exists Class : Votes_{Class} > 0$ **then**
    $Classification \leftarrow \arg\max_C \{Votes_C\}$
  else
    $Classification \leftarrow \emptyset$
  end
end

Algorithm 5.3: FEPC Algorithm, Fuzzy Emerging Pattern Classifier

Example 5.3. *Suppose we have the following patterns* ***ABE****,* ***ABD****,* ***DE****,* ***AB****,* ***A****, and* ***B****, formed by the selectors* ***A****,* ***B****,* ***C****,* ***D*** *and* ***E****. In the training stage, FEPC builds the pattern graph appearing in Figure 5.8. If we want to classify the object* $Q =$ **ABE***, FEPC starts considering the patterns with no ancestors:* ***ABE****,* ***ABD****, and* ***DE****. Any of them matches Q, so FEPC considers their immediate successors:* ***AB*** *and* ***D****. Since Q matches* ***AB****, the pattern* ***A*** *is discarded. Finally, the pattern* ***AB*** *is the only one considered for classification. Following the same algorithm,*

*FEPC classifies **BDE** using the pattern **DE**, and classifies **ADE** considering the patterns **DE** and **A**.*

FEPC tries to classify each object with fuzzy emerging patterns as particular as possible. If a pattern does not match the query object, the classifier uses more general patterns, following the arcs in the graph. This way, we can combine, in a synergic way, particular patterns having lower errors with general patterns with low abstention. FEPC strategy avoids vote duplication, which frequently leads to incorrect classification in problems where there are many similar patterns.

5.5 Experimental Results

In this section, we present the results of our experimental comparison between FEPC and other classifiers. To make the results comparable, we select the same classifiers, evaluation protocol, and databases than in the previous chapter (Section 4.7). From our new classifiers, we select CEPMC because it is the most accurate.

For FEPC, we construct a fuzzy Ruspini partition for each numerical feature. We create d uniformly distributed fuzzy sets, the first and last ones with a trapezoidal shape, and the inner sets with triangular shapes (like the examples in Figure 5.1), which is a commonly used configuration in mining tasks [38, 1]. We experimentally determine that the highest accuracy is reached using $d = 4$, but 3 and 5 achieved similar accuracy results. In a similar way, we set $k = 5$. We use $\mu_{min} = 0.05$ for all threshold values, although we did not find statistically significant differences in the accuracies using values between 0.05 and 0.15. Also, we experimentally found that the shape of the fuzzy set has a low impact in the quality of the mined patterns, compared to other similar shapes (Sigmoid, beta) used in knowledge mining tasks. Nevertheless, the number of bins has a high negative impact in the algorithm efficiency, because using more bins implies evaluating more candidate splits on each iteration of the tree building procedure.

In Table 4.3, we show the accuracy results of the experiments. We perform 10-fold cross validation, averaging the results. It is easy to notice that our classifier outperforms all the tested classifiers in most of the databases.

Table 5.3: Accuracy of the tested classifiers on the selected databases. The highest accuracy result for each database appears bolded

DBName	3nn	Boost	Bagg	c4.5	RandFor	SVM	SJEPC	CEPMC	FEPC
autos	68.2	42.4	61.1	81.0	81.0	66.0	12.3	79.8	**85.4**
balance-scale	**89.3**	71.7	82.6	77.6	79.4	87.5	16.0	79.5	84.5
breast-cancer	**74.5**	72.4	71.0	73.4	65.8	70.7	44.5	72.0	70.6
breast-w	96.5	95.6	95.6	95.9	96.5	97.0	96.3	96.0	**98.8**
cleveland (cleve)	81.8	84.2	79.9	78.2	78.6	**84.5**	77.9	81.2	82.2
credit-screening(crx)	84.2	86.3	85.9	85.1	85.0	86.3	82.6	85.9	**88.9**
cylinder-bands	70.2	72.9	60.4	72.2	78.5	80.9	64.3	77.8	**89.8**
diabete	74.9	75.4	75.7	**76.4**	75.4	74.1	76.2	75.7	74.2
glass	**69.0**	44.7	73.8	67.7	70.9	57.1	20.4	63.2	59.7
haberman	72.2	70.9	**72.5**	68.0	67.6	72.5	0.0	68.6	70.0
hayes-roth	50.0	53.6	75.0	**89.3**	85.7	53.6	0.0	78.6	78.6
heart-c	81.2	83.2	81.9	76.2	80.9	82.8	78.6	81.2	**84.2**
heart-h	**83.0**	81.6	79.9	79.6	79.3	82.6	46.3	81.0	79.5
heart-statlog	81.1	80.7	79.3	79.3	79.3	**83.0**	64.8	80.0	82.2
hepatitis	**86.0**	83.6	82.0	81.8	82.4	85.2	83.2	82.5	85.8
ionosphere	85.5	91.4	90.3	90.3	92.6	88.0	**93.5**	89.5	93.4
iris	96.6	**97.8**	95.8	95.8	95.2	97.0	75.3	95.3	97.7
labor-neg	**90.7**	87.0	84.0	80.0	86.7	90.7	82.0	89.0	90.3
liver	62.6	66.1	68.7	68.7	**70.7**	57.7	0.0	69.3	65.0
lymph	85.9	75.7	77.7	78.5	79.9	87.9	43.9	83.7	**92.0**
monkproblem1	81.0	75.0	50.0	88.9	75.7	50.0	86.8	**100.0**	99.1
monkproblem2	61.3	60.7	55.1	69.9	65.1	50.5	71.1	**83.8**	75.0
monkproblem3	88.2	97.2	50.0	96.3	97.2	50.0	93.5	**97.5**	96.1
mushroom	**100.0**	96.3	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	99.7	**100.0**
sick	96.0	97.2	95.1	**98.7**	98.2	97.6	96.8	97.8	96.4
sonar	82.3	77.4	76.4	68.4	81.8	82.0	**85.1**	78.8	82.2
spect	64.7	66.8	61.5	66.8	62.0	67.9	0.0	**78.1**	66.3
tic-tac-toe	97.9	74.7	85.3	85.3	92.6	95.8	98.8	96.5	**99.4**
vote	92.2	94.7	95.2	**96.1**	**96.1**	95.9	91.1	94.0	91.3
wdbc	96.3	92.3	94.4	91.4	94.0	**97.7**	85.1	95.6	96.4
wine	96.0	87.5	94.3	92.7	97.2	**98.9**	55.1	93.3	98.7
wpbc	72.3	71.0	78.8	75.5	74.7	75.9	0.0	79.8	**83.4**

As we saw in Section 4.7, SJEPC attains poor results in some databases. In those databases, most numerical features were discretized into a categorical feature with a single value, so they were useless for mining patterns.

In order to determine if the differences in accuracy are statistically significant, we performed a pairwise comparison between CEPMC, FEPC, and the other classifiers. Each cell in Table 5.4 contains the number of databases where our classifier significantly Win/Lose to each other classifier. We detected ties using a two-tailed T-Test [18] with significance of 0.05. The results in the pairwise comparison reveal that the FEPC classifier beats in accuracy

Table 5.4: Pairwise comparison between CEPMC, FEPC, and the others. Each cell shows the number of times CEPMC and FEPC Win/Loss with respect to the corresponding classifier over the 33 selected databases

	3nn	Boost	Bagg	c4.5	RandFor	SVM	SJEPC	CEPMC	FEPC
CEPMC	10/8	15/3	12/3	12/3	9/6	9/10	24/3		5/15
FEPC	14/5	18/2	20/4	19/7	20/4	13/5	25/1	15/5	

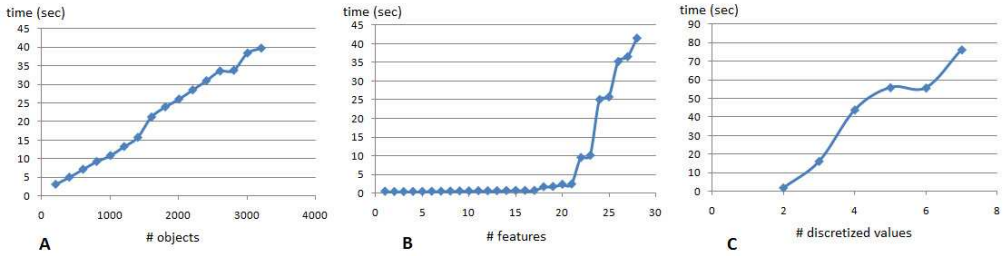

Figure 5.9: Scalability results of FEP miner by increasing the number of objects (A), features (B), and values of the features (C)

all other single classifiers in almost all the tested databases.

Tables 5.3 and 5.4 show that the new classifier performs better than the others in many but not all cases. Although FEPC clearly outperforms other non-metric methods (Bagging, Boosting, c4.5, Random Forest and SJEPC), it frequently ties with metric methods (kNN and SVM). To explain this behavior, we should point out that emerging patterns are unable to capture relations among features [5] , which are easily captured by distance-based methods.

FEPC is more accurate than CEPMC, while CEPMC has a slower training time. To select between them, the user must also consider the type of model he wants to generate, in order to comprehend the classifier results. In a fuzzy model, to understand each single pattern the user needs a deep understanding of each fuzzified value and each hedge present in the pattern. That is why a fuzzy model can be easier to comprehend for some users, but very hard to comprehend for others.

5.5.1 Algorithm Scalability

In order to test the scalability of our mining algorithm by increasing the number of objects, the number of features, and the number of discretized values of the features, we use the database *hypothyroid* [50]. *Hypothyroid* has 3772 objects, 7 numerical features, and 22 categorical features. We evaluated the impact of increasing each parameter, testing different values while keeping the other parameters unaltered. The results were the following:

Number of objects Adding new objects has different levels of impact, depending on characteristics of the objects. An object similar to previous objects in its same class usually does not alter the decision tree. An object very dissimilar to previous objects, or similar to objects in a different class, could force the mining procedure to make more splits. Figure 5.9 shows a linear dependence between the number of objects and the time needed to mine the patterns.

Number of features Adding a new feature increments the number of candidate splits in every node of the tree by $k \cdot NumHedges$, where $NumHedges$ is the amount of hedges considered in the system, and k is the number of values of the feature (4 for numerical features). Figure 5.9 shows an exponential dependence with respect to the number of features.

Number of discretized values Increasing the number of discretized values adds $NumHedges$ candidate splits in every node of the tree. Figure 5.9 shows a dependence close to linear.

Generally speaking, our algorithm scales better to adding new objects than to adding new discretized values, and scales worst to adding new features.

5.6 Summary

In this chapter, we introduced Fuzzy Emerging Patterns, an emerging pattern extension to the fuzzy case, aiming at a better discretization of continuous

[5] Emerging patterns capture relations among feature values and the class. A relation between features like $Attr_1 > Attr_2$ cannot be represented using this kind of patterns.

features. In order to extract these fuzzy patterns from a training sample, we proposed a new procedure based on the induction of several fuzzy decision trees. The induction procedure uses linguistic hedges to fix the initial fuzzy discretization of the continuous features, which is also a contribution of this thesis. Besides, we introduced a measure to test the discriminability of a fuzzy emerging pattern for classification, named *Trust*.

Using the extracted fuzzy emerging patterns, we proposed the FEPC classifier, which uses a novel graph-based strategy for organizing the patterns. This strategy allows to create more accurate classifiers, with lower levels of abstention. In our experiments, FEPC showed significant higher accuracies than some popular and state of the art classifiers. In the pairwise comparison, FEPC beats every other single classifier in the majority of the tested databases.

In order to improve some of the limitations of our algorithm, the following approaches can be studied:

- Inclusion of splits containing more than one feature in the tree induction algorithm, in order to capture relations among features. This is a complex task because it could significantly degrade the mining efficiency, because many more splits would have to be considered in every node. Additionally, finding the splitting hyperplanes can be a long time consuming task.
- Selection of a dynamic number of bins for the fuzzy discretization, according to the feature value distribution.
- Generation of a variable number of fuzzy decision trees, according to the complexity of the database, maybe using a scheme similar to CEPM (Section 4.4.)

Chapter 6

Conclusions and future work

6.1 Conclusions

A useful characteristic of a supervised classifier is that the user can comprehend the classification results in terms of knowledge domain, particularly in those cases where the classification is contradictory with the user expectations. Unfortunately, top accurate classifiers are usually non comprehensible, while most comprehensible classifiers attain lower accuracy in most databases. Emerging Pattern classifiers, on the other hand, build accurate and easy to understand models.

In this dissertation, we introduced two new kinds of emerging patterns, which are more expressive than the traditional ones. The higher expressiveness of the new patterns allows expressing more selective properties, which allows to obtain more accurate classifiers. For each kind of emerging patterns, we introduced novel mining algorithms, which allow extracting the patterns from databases with mixed and incomplete data.

For the crisp case, we introduced LCMine, a new extended emerging pattern mining algorithm without global discretization of numerical features. LCMine extracts patterns from a collection of C4.5 decision trees, using a special pattern mining procedure during the tree induction. Although LCMine is a very accurate classifier, it is slow and very sensitive to the selection of the minimal support threshold. To overcome these limitations, we introduced

an enhanced version, named CEPM. To guarantee that CEPM finds a representative collection of patterns, it uses a novel object weighting scheme. Furthermore, we proposed an algorithm for accurately estimating the minimal support threshold of the CEPM classifier.

We also proposed a new method for building cascades of emerging pattern classifiers, which combines the higher accuracy of classifying with higher support thresholds with the lower levels of abstention of classifying with lower thresholds.

From the experimental results we can conclude that CascadeCEPM and CEPMC are more accurate than other understandable classifiers, while being competitive with Nearest Neighbors and Support Vector Machines classifiers. On the other hand, CascadeCEPM attains similar results than CEPMC, but it is faster.

To reduce the drawbacks of crisp discretization, we introduced the concept of Fuzzy Emerging Pattern, and we proposed a new algorithm for mining fuzzy emerging patterns from a database with crisp classes. This algorithm extracts patterns from a set of fuzzy decision trees, induced with a new algorithm that includes the use of linguistic hedges. We proposed a new classifier based on fuzzy emerging patterns, which includes a novel mechanism for aggregation of single pattern votes. From the experimental results we concluded that the proposed fuzzy emerging pattern classifier is better in accuracy than all other tested single classifiers.

All the classifiers proposed in this dissertation attain higher accuracy than traditional emerging pattern classifiers and other comprehensible classifiers. They are also competitive with other state-of-the-art classifiers that are not comprehensible.

To select which of the introduced classifiers to use in a particular problem, the expert must select first the type of model wanted: a crisp model or a fuzzy model. Fuzzy models are very close to the way humans talk and think, but they are slower and more complex to implement. If the expert chooses a fuzzy model, then it must use FEPC. Otherwise, the selection depends on a tradeoff between complexity and classification speed. While CEPMC is a simpler classifier, because it uses a model simpler to be understood by the users, CascadeCEPMC is faster.

6.2 Future work

Mining extended emerging patterns is computationally more expensive than mining traditional patterns. An important point to alleviate this drawback is to develop more effective mechanisms to early estimate when a pattern subset is representative enough of a training sample. Another interesting point to explore is developing new decision tree induction procedures specialized for the task of mining patterns, even if the induced trees are not good classifiers by themselves.

The new kinds of patterns proposed in this dissertation could improve the user understanding in many application fields. That is why it is interesting to apply the new classifiers in domains where traditional emerging pattern classifiers succeeded.

There are some databases where any emerging pattern classifier works fine. It could be very important to find an automated procedure to detect those cases, based on some database information. This way, we could combine an emerging pattern classifier with distance based classifiers, obtaining a more accurate ensemble.

Finally, emerging pattern based classifiers have been modified to deal with some traditional problems: one-class problems (Like intrusion detection), data flow analysis (Like news and stocks), highly unbalanced datasets, and large datasets. Although the classifiers introduced in this dissertation uses a new mining paradigm, they could also be modified to handle these problems.

6.3 Publications

- Milton García-Borroto et al. LCMine: An efficient algorithm for mining discriminative regularities and its application in supervised classification. Pattern Recognition vol. 43, pp. 3025-3034, 2010.

- Milton García-Borroto et al. Fuzzy Emerging Patterns for Classifying Hard Domains. Knowledge and Information Systems 2010, DOI:10.1007/s10115-010-0324-x.

- Milton García-Borroto et al. A New Emerging Pattern Mining Algorithm and Its Application in Supervised Classification. M.J. Zaki et al. (Eds.): PAKDD 2010, Part I, LNAI 6118, pp. 150-157, 2010.
- Milton García-Borroto et al. Cascading an Emerging Pattern based classifier. J.A. Carrasco-Ochoa et al. (Eds.): MCPR 2010, LNCS 6256, pp. 240-249, 2010.

Bibliography

[1] Jesús Alcalá-Fdez, Rafael Alcalá, María José Gacto, and Francisco Herrera. Learning the membership function contexts for mining fuzzy association rules by using genetic algorithms. *Fuzzy Sets Syst.*, 160(7):905–921, 2009. [cited at p. 76]

[2] Hamad Alhammady. Mining streaming emerging patterns from streaming data. In *IEEE/ACS International Conference on Computer Systems and Applications*, pages 432–436, Amman, 2007. [cited at p. 3]

[3] Ethem Alpaydin and Cenk Kaynak. Cascading classifiers. *Kybernetica*, 34(4):369–374, 1998. [cited at p. 56]

[4] Annalisa Appice, Michelangelo Ceci, Carlo Malgieri, and Donato Malerba. Discovering relational emerging patterns. In *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*, pages 206–217. 2007. [cited at p. 36]

[5] James Bailey, Thomas Manoukian, and Kotagiri Ramamohanarao. Fast algorithms for mining emerging patterns. In *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, volume 2431 of *Lecture Notes in Computer Sciences*, pages 187–208. Springer-Verlag, 2002. [cited at p. 33, 34, 35, 37, 74]

[6] James Bailey, Thomas Manoukian, and Kotagiri Ramamohanarao. A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 485, Washington, DC, USA, 2003. IEEE Computer Society. [cited at p. 36]

[7] James Bailey, Tomas Manoukian, and Kotagiri Ramamohanarao. Classification using constrained emerging patterns. In *Fourth International Conference on*

Web-Age Information Management, pages 226–237, Chengdu, China, 2003. [cited at p. 34]

[8] Mikhail N. Bongard. Solution to geological problems with support of recognition programs. *Sov. Geologia*, 6:33–50, 1963. [cited at p. 18, 19, 20]

[9] Anne-Laure Boulesteix, Gerhard Tutz, and Korbinian Strimmer. A cart-based approach to discover emerging patterns in microarray data. *Bioinformatics*, 19(18):2465–2472, 2003. [cited at p. 3]

[10] Leo Breiman, Jerome. H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth International, Belmont, California, 1984. [cited at p. 25]

[11] Michelangelo Ceci, Annalisa Appice, Costantina Caruso, and Donato Malerba. Discovering emerging patterns for anomaly detection in network connection data. *Lecture Notes in Artificial Intelligence*, 4994:179–188, 2008. [cited at p. 3]

[12] Mete Celik, Shashi Shekhar, James P. Rogers, and James A. Shine. Sustained emerging spatio-temporal co-occurrence pattern mining: A summary of results. In *ICTAI '06: Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, pages 106–115, Washington, DC, USA, 2006. IEEE Computer Society. [cited at p. 3]

[13] Robin L. P. Chang and Theodosios Pavlidis. Fuzzy decision tree algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(1):28–35, 1977. [cited at p. 29]

[14] Lijun Chen and Guozhu Dong. Masquerader detection using oclep: One-class classification using length statistics of emerging patterns. In *WAIMW '06: Proceedings of the Seventh International Conference on Web-Age Information Management Workshops*, page 5, Washington, DC, USA, 2006. IEEE Computer Society. [cited at p. 3]

[15] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. [cited at p. 14, 18, 60]

[16] Mark W. Craven and Jude W. Shavlik. Extracting tree-structured representations of trained networks. In *Advances in Neural Information Processing Systems*, volume 8. MIT Press, Cambridge, MA, 1996. [cited at p. 18]

[17] Belur Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, California, 1991. [cited at p. 14, 16, 18, 37, 39, 59]

[18] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998. [cited at p. 60, 77]

[19] Guozhu Dong and Jinyan 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*, pages 43–52, San Diego, California, United States, 1999. ACM. [cited at p. 3, 20]

[20] Guozhu Dong and Jinyan 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*, pages 43–52, San Diego, California, United States, 1999. ACM. [cited at p. 18, 19, 21, 31, 34]

[21] Guozhu Dong, Xiuzhen Zhang, Limsoon Wong, and Jinyan Li. Caep: Classification by aggregating emerging patterns. In *DS '99: Proceedings of the Second International Conference on Discovery Science*, pages 30–42, London, UK, 1999. Springer-Verlag. [cited at p. 38]

[22] Ming Dong and Ravi Kothari. Look-ahead based fuzzy decision tree induction. *IEEE Transactions on Fuzzy Systems*, 9(3):461–468, 2001. [cited at p. 69]

[23] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, second edition, 2000. [cited at p. 56]

[24] Hongjian Fan, Ming Fan, Kotagiri Ramamohanarao, and Mengxu Liu. Further improving emerging pattern based classifiers via bagging. In W.K. Ng, M. Kitsuregawa, and Jianping Li, editors, *PAKDD 2006*, volume 3918 of *Lecture Notes in Artificial Intelligence*, pages 91–96, 2006. [cited at p. 39]

[25] Hongjian Fan and Kotagiri Ramamohanarao. An efficient single-scan algorithm for mining essential jumping emerging patterns for classification. In *PAKDD '02: Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 456–462, London, UK, 2002. Springer-Verlag. [cited at p. 32, 33, 35, 38, 74]

[26] Hongjian Fan and Kotagiri Ramamohanarao. A bayesian approach to use emerging patterns for classification. In *ADC '03: Proceedings of the 14th Australasian database conference*, pages 39–48, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc. [cited at p. 33, 36, 39, 74]

[27] Hongjian Fan and Kotagiri Ramamohanarao. Fast discovery and the generalization of strong jumping emerging patterns for building compact and accurate classifiers. *IEEE Transactions on Knowledge and Data Engineering*, 18(6):721–737, 2006. [cited at p. 2, 5, 20, 22, 32, 34, 35, 37, 38, 60]

[28] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *13th Int'l Joint Conf. Artificial Intelligence (IJCAI)*, pages 1022–1029, 1993. [cited at p. 31, 37]

[29] Eibe Frank, Mark A. Hall, Geoffrey Holmes, Richard Kirkby, Bernhard Pfahringer, and Ian H. Witten. Weka: A machine learning workbench for data mining. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook: A Complete Guide for Practitioners and Researchers*, pages 1305–1314. Springer, Berlin, 2005. [cited at p. 60]

[30] Milton García-Borroto, José F. Martínez Trinidad, Jesús Ariel Carrasco Ochoa, Miguel Angel Medina-Pérez, and José Ruiz-Shulcloper. Lcmine: An efficient algorithm for mining discriminative regularities and its application in supervised classification. *Pattern Recognition*, 43(9):3025–3034, 2010. [cited at p. 20, 22]

[31] Valeriy V. Gavrishchaka and Valery Bykov. Market-neutral portfolio of trading strategies as universal indicator of market micro-regimes: From rare-event forecasting to single-example learning of emerging patterns. In *ICICIC '07: Proceedings of the Second International Conference on Innovative Computing, Informatio and Control*, page 215, Washington, DC, USA, 2007. IEEE Computer Society. [cited at p. 3]

[32] Antonio González and Raul Pérez. A study about the inclusion of linguistic hedges in a fuzzy rule learning algorithm. *International journal of uncertainty, fuzziness and kenowledge-based systems*, 7(3):257–266, 1999. [cited at p. 9, 28, 66]

[33] Tao Gu, Zhanqing Wu, Xianping Tao, Hung Keng Pung, and Jian Lu. epsicar: An emerging patterns based approach to sequential, interleaved and concurrent activity recognition. In *PERCOM '09: Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications*, pages 1–9, Washington, DC, USA, 2009. IEEE Computer Society. [cited at p. 3]

[34] Wilhelmiina Hämälïnen. Statapriori: an efficient algorithm for searching statistically significant association rules. *Knowledge and Information Systems*, 2009. DOI 10.1007/s10115-009-0229-8. [cited at p. 20, 21]

[35] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004. [cited at p. 34]

[36] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998. [cited at p. 14, 16, 18]

[37] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998. [cited at p. 18, 19, 20, 59]

[38] Dong-Mei Huang. An algorithm for generating fuzzy decision tree with trapezoid fuzzy number-value attributes. In *International Conference on Wavelet Analysis and Pattern Recognition, ICWAPR 08*, volume 1, pages 41–45, Hong Kong, China, 2008. [cited at p. 29, 76]

[39] Ruoming Jin, Yuri Breitbart, and Chibuike Muoh. Data discretization unification. *Knowledge and Information Systems*, 19:1–29, 2009. [cited at p. 2]

[40] Lukasz Kobylinski and Krzysztof Walczak. Jumping emerging patterns with occurrence count in image classification. In Takashi Washio, editor, *PAKDD 2008*, volume 5012 of *Lecture Notes in Artificial Inteligence*, pages 904–909. Springer-Verlag, 2008. [cited at p. 34]

[41] Ravi Kothari and Ming Dong. Decision trees for classification: A review and some new results. In Sankar K. Pal and Amita Pal, editors, *Pattern Recognition. From Cassical to Modern Approaches*, pages 169–184. World Scientific, 2001. [cited at p. 18]

[42] Ludmila I. Kuncheva. *Combining Pattern Classifiers. Methods and Algorithms*. Wiley-Interscience, Hoboken, New Jersey, USA, 2004. [cited at p. 13, 15, 39, 56, 59]

[43] Jinyan Li, Guozhu Dong, and Kotagiri Ramamohanarao. Instance-based classification by emerging patterns. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 191–200. Springer-Verlag, 2000. [cited at p. 3, 4, 32, 38]

[44] Jinyan Li, Guozhu Dong, Kotagiri Ramamohanarao, and Limsoon Wong. Deeps: A new instance-based lazy discovery and classification system. *Machine Learning*, 54(2):99–124, 2004. [cited at p. 22, 38, 39]

[45] Jinyan Li, Guimei Liu, and Limsoon Wong. Mining statistically important equivalence classes and delta-discriminative emerging patterns. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 430–439, New York, NY, USA, 2007. ACM. [cited at p. 35]

[46] Jinyan Li, Kotagiri Ramamohanarao, and Guozhu Dong. Combining the strength of pattern frequency and distance for classification. In *PAKDD '01: Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 455–466, London, UK, 2001. Springer-Verlag. [cited at p. 39]

[47] Elsa Loekito and James Bailey. Fast mining of high dimensional expressive contrast patterns using zero-suppressed binary decision diagrams. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 307–316, New York, NY, USA, 2006. ACM. [cited at p. 36]

[48] David Martens, Bart Baesens, Tony Van Gestel, and Jan Vanthienen. Comprehensible credit scoring models using rule extraction from support vector mahines. *European Journal of Operational Research*, 2007. [cited at p. 2, 18]

[49] José Francisco Martínez-Trinidad and Adolfo Guzmán-Arenas. The logical combinatorial approach to pattern recognition, an overview through selected works. *Pattern Recognition*, 34:741–751, 2001. [cited at p. 1, 13]

[50] Christopher J. Merz and Patrick M. Murphy. Uci repository of machine learning databases. Technical report, University of California at Irvine, Department of Information and Computer Science, 1998. [cited at p. 37, 59, 79]

[51] Ryszard Michalski and Robert Stepp. Revealing conceptual structure in data by inductive inference. In D. Michie, J. E. Hayes, and H. H. Pao, editors, *Machine Intelligence*, volume 10, pages 173–196. Ellis Horwood Ltd, New York, 1982. [cited at p. 42, 43]

[52] Shin-ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *DAC '93: Proceedings of the 30th international Design Automation Conference*, pages 272–277, New York, NY, USA, 1993. ACM. [cited at p. 36]

[53] Vilem Novak, Irina Perfilieva, and Jiri Mockor. *Mathematical principles of fuzzy logic*. Kluwer Academic, 1999. [cited at p. 26]

[54] Nicolas Pasquier, Claude Pasquier, Laurent Brisson, and Martine Collard. Mining gene expression data using domain knowledge. *International Journal of Software and Informatics*, 2(2):215–231, 2008. [cited at p. 3]

[55] Gregory Piatetsky-Shapiro and William J. Frawley. *Knowledge Discovery in Databases*. AAAI/MIT Press, Cambridge, MA, 1991. [cited at p. 19]

[56] John Quackenbush. Computational approaches to analysis of dna microarray data. In *IMIA Yearbook of Medical Informatics*, pages 91–103. 2006. [cited at p. 3]

[57] James Ross Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, 1986. [cited at p. 16, 20, 45, 68]

[58] James Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993. [cited at p. 7, 14, 18, 25, 37, 45, 47, 59]

[59] Kotagiri Ramamohanarao and Hongjian Fan. Patterns based classifiers. *World Wide Web*, 10(1):71–83, 2007. [cited at p. 2, 16, 34, 38]

[60] José Ruiz-Shulcloper and Mongi A. Abidi. Logical combinatorial pattern recognition: A review. In S.G. Pandalai, editor, *Recent Research Developments in Pattern Recognition*, pages 133–176. Transword Research Networks, USA, 2002. [cited at p. 13]

[61] Joseph L. Schafer and John W. Graham. Missing data: our view of the state of the art. *Psychological Methods*, 7(2):147–77, 2002. [cited at p. 17]

[62] Yanmin Sun and Andrew K.C Wong. Boosting an associative classifier. *IEEE Trans. on Knowl. and Data Eng.*, 18(7):988–992, 2006. [cited at p. 39]

[63] Pawel Terlecki and Krzysztof Walczak. Efficient discovery of top-k minimal jumping emerging patterns. In C. Chang, editor, *RSCTC*, volume 5306 of *Lecture Notes in Artificial Intelligence*, pages 438–447, 2008. [cited at p. 35, 37]

[64] Pawel Terlecki and Krzysztof Walczak. Local projection in jumping emerging patterns discovery in transaction databases. In *PAKDD'08: Proceedings of the 12th Pacific-Asia conference on Advances in knowledge discovery and data mining*, pages 723–730, Berlin, Heidelberg, 2008. Springer-Verlag. [cited at p. 36]

[65] Lusheng Wang, Hao Zhao, Guozhu Dong, and Jianping Li. On the complexity of finding emerging patterns. *Theor. Comput. Sci.*, 335(1):15–27, 2005. [cited at p. 4, 74]

[66] Xi Zhao Wang, Bin Chen, Guoliang Qian, and Feng Ye. On the optimization of fuzzy decision trees. *Fuzzy Sets and Systems*, 112:117–125, 2000. [cited at p. 29]

[67] Xi Zhao Wang, Jun Hai Zhai, and Su Fang Zhang. Fuzzy decision tree basen on the important degree of fuzzy attribute. In *2008 International Conference on Machine Learning and Cybernetics*, volume 1, pages 511–516, Kunming, 2008. [cited at p. 29]

[68] Zhao Wang, Hongjian Fan, and Kotagiri Ramamohanarao. Exploiting maximal emerging patterns for classification. In *17th Australian Joint Conference on Artificial Intelligence*, pages 1062–1068, Cairns, Queensland, Australia, 2004. [cited at p. 32, 33, 39]

[69] Cheng-Hsiung Weng and Yen-Liang Chen. Mining fuzzy association rules from uncertain data. *Knowledge and Information Systems*, 2009. DOI 10.1007/s10115-009-0223-1. [cited at p. 65]

[70] Yufei Yuan and Michael J. Shaw. Induction of fuzzy decision trees. *Fuzzy Sets and Systems*, 69:125–139, 1995. [cited at p. 29]

[71] Lofti Zadeh. Fuzzy sets. *Information Control*, 8:338–353, 1965. [cited at p. 65]

[72] Lotfi A. Zadeh. *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers by Lotfi A. Zadeh*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1996. [cited at p. 26]

[73] Mohammed J. Zaki and Ching-Jui Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):462–478, 2005. [cited at p. 20]

[74] Xiuzhen Zhang, Guozhu Dong, and Kotagiri Ramamohanarao. Information-based classification by aggregating emerging patterns. In *IDEAL '00: Proceedings of the Second International Conference on Intelligent Data Engineering and Automated Learning, Data Mining, Financial Engineering, and Intelligent Agents*, pages 48–53, London, UK, 2000. Springer-Verlag. [cited at p. 33, 38]

[75] Xiuzhen Zhang, Guozhu Dong, and Kotagiri Ramamohanarao. Information-based classification by aggregating emerging patterns. In *Proceedings of the Second International Conference on Intelligent Data Engineering and Automated Learning, Data Mining, Financial Engineering, and Intelligent Agents*, pages 48–53. Springer-Verlag, 2000. [cited at p. 74]

[76] Xiuzhen Zhang, Guozu Dong, and Ramamohanarao Kotagiri. Exploring constraints to efficiently mine emerging patterns from large high-dimensional datasets. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 310–314, New York, NY, USA, 2000. ACM. [cited at p. 34]