Efficient Mining of Interesting Emerging Patterns and Their Effective Use in Classification

Hongjian FAN

Submitted in total fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of

Computer Science

May 2004
(Amended July 2004)
Produced on acid-free paper

The Department of Computer Science and Software Engineering
Faculty of Engineering

THE UNIVERSITY OF
MELBOURNE
Victoria, Australia

Thesis advisor
Professor Ramamohanarao (Rao) Kotagiri

Author
Hongjian FAN

Efficient Mining of Interesting Emerging Patterns and Their Effective Use in Classification

Abstract

Knowledge Discovery in Databases (KDD), or Data Mining is used to discover interesting or useful patterns and relationships in data, with an emphasis on large volume of observational databases. Among many other types of information (knowledge) that can be discovered in data, patterns that are expressed in terms of features are popular because they can be understood and used directly by people. The recently proposed Emerging Pattern (EP) is one type of such knowledge patterns. Emerging Patterns are sets of items (conjunctions of attribute values) whose frequency changes significantly from one dataset to another. They are useful as a means of discovering distinctions inherently present amongst a collection of datasets and have been shown to be a powerful method for constructing accurate classifiers.

In this doctoral dissertation, we study the following three major problems involved in the discovery of Emerging Patterns and the construction of classification systems based on Emerging Patterns:

1. How to efficiently discover the complete set of Emerging Patterns between two classes of data?

2. Which Emerging Patterns are interesting, namely, which Emerging Patterns are novel, useful and non-trivial?

3. Which Emerging Patterns are useful for classification purpose? And how to use these Emerging Patterns to build efficient and accurate classifiers?

We propose a special type of Emerging Pattern, called Essential Jumping Emerging Pattern (EJEP). The set of EJEPs is the subset of the set of Jumping Emerging Patterns (JEPs), after removing those JEPs that potentially contain noise and redundant information. We show that a relatively small set of EJEPs are sufficient for building accurate classifiers, instead of mining many JEPs.

We generalize the "interestingness" measures for Emerging Patterns, including the minimum support, the minimum growth rate, the subset relationship between EPs and the correlation based on common statistical measures such as the chi-squared value. We show that those "interesting" Emerging Patterns (called Chi EPs) not only capture the essential knowledge to distinguish two classes of data, but also are excellent candidates for building accurate classifiers.

The task of mining Emerging Patterns is computationally difficult for large, dense and high-dimensional datasets due to the "curse of dimensionality". We develop new tree-based pattern fragment growth methods for efficiently mining EJEPs and Chi EPs.

We propose a novel approach to use Emerging Patterns as a basic means for classification, called Bayesian Classification by Emerging Patterns (BCEP). As a hybrid of the EP-based classifier and Naive Bayes (NB) classifier, BCEP offers the following advantages: (1) it is based on theoretically well-founded mathematical model as in Large Bayes (LB); (2) it relaxes the strong attribute independence assumption of NB; (3) it is easy to interpret, because typically only a small number of Emerging Patterns are used in classification after pruning.

Real-world classification problems always contain noise. A reliable classifier should be tolerant to a reasonable level of noise. Our study of noise tolerance of BCEP shows that BCEP generally handles noise better in comparison with other state-of-the-art classifiers.

We conduct extensive empirical study on benchmark datasets from the UCI Machine Learning Repository to show that our EP mining algorithms are efficient and our EP-based classifiers are superior to other state-of-the-art classification methods in terms of overall predictive accuracy.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD except where indicated in the Preface,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Hongjian Fan
B.E. (Zhengzhou University, China, 1999)
May 2004

Contents

List of Figures

List of Tables

List of Algorithms

Preface

The following list of publications arises from this thesis.

- **Chapter 4** was presented in a preliminary form at the Sixth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD2002).

  Hongjian Fan and Ramamohanarao Kotagiri. An efficient single-scan algorithm for mining essential jumping emerging patterns for classification. In *Proceedings of 6th Pacific-Asia Conference Knowledge Discovery and Data Mining (PAKDD2002), Taipei, Taiwan, May 6 - 8, 2002, pages 456 - 462.*

- **Chapter 5** was published in a preliminary form at the Fourth International Conference on Web-Age Information Management (WAIM2003).

  Hongjian Fan and Ramamohanarao Kotagiri. Efficiently Mining Interesting Emerging Patterns. In *Proceedings of 4th International Conference on Web-Age Information Management (WAIM2003), Chengdu, China, August 17 - 19, 2003, pages 189 - 201.*

- **Chapter 6** was presented in a preliminary form at the Fourteenth Australasian Database Conference (ADC2003).

  Hongjian Fan and Ramamohanarao Kotagiri. A Bayesian approach to use emerging patterns for classification. In *Proceedings of 14th Australasian Database Conference (ADC2003), Adelaide, Australia, February 4 - 7, 2003, pages 39 - 48.*

- **Chapter 7** was presented in a preliminary form at the Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD2004).

  Hongjian Fan and Ramamohanarao Kotagiri. Noise Tolerant Classification by Chi Emerging Patterns. In *Proceedings of 8th Pacific-Asia Conference Knowledge Discovery and Data Mining (PAKDD2004), Sydney, Australia, May 26 - 28, 2004, pages 201 - 206.*

Acknowledgments

The past few years have been a wonderful and often overwhelming experience. Completing this doctoral work is due to the support of many people.

First and foremost, I wish to express my most sincere gratitude to my supervisor, Prof. Kotagiri Ramamohanarao (Rao) for his tireless and prompt help. This thesis would not exist without his guidance and support. I have always been stimulated and excited by his constant flow of good ideas. While he allows me complete freedom to define and explore my own directions in research, he also knows when and how to give me a little push in the forward direction when I need it.

I would like to thank many staff in our department. I thank Prof. Peter Stuckey and Dr James Bailey for their efforts in serving on my advisory committee and their insightful comments. Special thanks go to James for his invaluable feedback on this thesis and many useful discussions. It is both enjoyable and enriching to work as a tutor with Prof. Steve Bird, Dr Chris Leckie, Dr Egemen Tanin, Kathleen Keogh and Antonette Mendoza. I thank Cameron Blackwood, Adam Hendrix, Liam Routt, Ling Shi and Thomas Weichert, for their technical assistance. I also thank administrative staff, Jon Callander, Pinoo Bharucha, Julien Reid and Cindy Sexton, for their help.

Thank Prof. Guozhu Dong from Wright State University, USA, and Dr Jinyan Li from Institute for Infocomm Research, Singapore, for their help with my study. Thank Dr Xiuzhen (Jenny) Zhang for interesting discussions.

Special thanks go to Raymond Wan for providing valuable feedback and proofreading many drafts that I wrote as a non-native English speaker. Anh Ngoc Vo and Bernard Pope deserve thanks for their technical assistance and helpful comments. I am grateful to Thomas Christopher Manoukian for his valuable aid as proofreaders and many constructive discussions. I benefit a lot from the friendship with Ce (William) Dong and Zhou (Joanne) Zhu. Acknowledgment also goes to Yugo Kartono Isal, Ben Rubinstein, Roger Ting, Lily Sun, Qun Sun, Zhou Wang, Lei Zheng and many other fellow students.

The University of Melbourne and the Department of Computer Science and Software Engineering have provided me with much appreciated financial support during my degree. I have been supported by Melbourne International Fee Remission Scholarship and Melbourne International Research Scholarship. They have also provided a Melbourne Travel

Scholarship and travel funds for me to attend conferences.

My thanks also go to the other side of the world - the northern hemisphere.

I appreciate the help from Dr Yangdong Ye, who shared many of his life experiences with me during his stay in Melbourne.

Thanks go to my friends from Zhengda, Feng Shao, Jie Sun, Dayong Zhou. Thanks go to Zhitao Cheng, Shuang Deng, Wenjia Ding, Yue Fang, Mi Gu, Min Li, Guotong Qi, Kun Su, Zheng Wang, Qi Yang, Yingge Zhang, Zhe Zhang, Shengli Zhao, Jiang Zhu, and other 95.1 classmates. Thanks go to Wei Li, Purui Shu, Zhijie Yang, Jianping Zheng, and other classmates from Institute of Software, Chinese Academy of Sciences.

My fascination with computer science is undoubtedly due to the influence of my father, who always encourage me to pursue a higher education. I love to thank my parents and my wife, Lei (Julie). Their unconditional love, support and encouragement are always there, through both the highs and lows of my time in the graduate school.

Dedicated to my parents,
and my wife, Lei Zhu.

Chapter 1

Introduction

> Ignorance is the curse of God, knowledge the wing wherewith we fly to heaven.
> – William Shakespeare

We now live in the information age. "Data owners" such as scientists, businesses and medical researchers, are able to gather, store and manage previously unimaginable quantities of data due to technological advances and economic efficiencies in sensors, digital memory and data management techniques. In 1991 it was alleged that the amount of data stored in the world doubles every twenty months (Piatetsky-Shapiro & Frawley 1991). At the same time, there is a growing realization and expectation that data, intelligently analyzed and presented, will be a valuable resource to be used for a competitive advantage. To cross the growing gap between data generation and data understanding, there is an urgent need for new computational theories and tools to assist humans in extracting useful knowledge from the huge volumes of data. These theories and tools are the subject of the emerging field of Knowledge Discovery in Databases (KDD), or Data Mining (DM), which sits at the common frontiers of several fields including Database Management, Artificial Intelligence, Machine Learning, Pattern Recognition, and Data Visualization.

Most data mining applications routinely require datasets that are considerably larger than those that have been addressed by traditional statistical procedures. The size of the datasets often means that traditional statistical algorithms are too slow for data mining problems and alternatives have to be devised. The volume of the data is probably not a very important difference: the number of variables or attributes often has a much more

profound impact on the applicable analytical methods. The number of variables may be so large that looking at all possible combinations of variables is computationally infeasible. As been emphasized from the beginning of KDD, one of its distinguished features from other fields is that KDD researchers have to deal with not only the very large size of the database, but also the very high dimensionality of the data.

The entire Knowledge Discovery Process encompasses many steps, from data acquisition, cleaning, preprocessing, to the discovery step, to postprocessing of the results and their integration into operational systems. Although some researchers view data mining as an essential step of knowledge discovery, we follow a popular view which regards data mining as a synonym of KDD.

Data mining is at best, a vaguely defined field; its definition largely depends on the background and views of the definer. Here are some definitions taken from the data mining literature:

> Data mining is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data. – Fayyad (Fayyad, Piatetsky-Shapiro & Smyth 1996)
>
> Data mining is the process of extracting previously unknown, comprehensible, and actionable information from large databases and using it to make crucial business decisions. – Zekulin (Friedman 1997)
>
> Data mining is a set of methods used in the knowledge discovery process to distinguish previously unknown relationships and patterns within data. – Ferruzza (Friedman 1997)
>
> Data mining is a decision support process where we look in large data bases for unknown and unexpected patterns of information. – Parsaye (Friedman 1997)
>
> Data mining is ...
> Decision Trees
> Neural Networks
> Rule Induction
> Nearest Neighbors
> Genetic Algorithms
> – Mehta (Friedman 1997)
>
> Data mining is the process of discovering advantageous patterns in data. – John (John 1997)

In summary, data mining can be seen as algorithmic and database-oriented methods that search for previously unsuspected structure and patterns in data. The data involved is often (but not always) massive in nature. Data mining to date has largely focused on computational and algorithmic issues, rather than the more traditional statistical aspects of data analysis. Some of the primary themes in current research in data mining include scalable algorithms for massive datasets, discovering novel patterns in data, and analysis of text, Web, and related multi-media datasets.

Ever since the first KDD workshop in 1989, there has been widespread research on KDD, as evidenced by research published by the mainstream computer science conferences, such as the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), the IEEE International Conference on Data Mining (ICDM), the European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), and the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD). KDD has also attracted a great deal of attention from the Information Technology (IT) industry. Data mining products such as IBM Intelligent Miner, SGI MineSet, and SAS Enterprise Miner have been brought to the market.

Due to the efforts of researchers from academia and industry, there have been many successful data mining applications in areas such as customer profiling, fraud detection, telecommunications network monitoring and market-basket analysis (Fayyad et al. 1996). However, on the commercial front, the huge opportunity has not yet been met with adequate tools and solutions; on the technical front, there are many problems and challenges for researchers and applied practitioners in KDD.

1.1 Statement of the Problem

Common data mining tasks fall into a few general categories: exploratory data analysis (e.g., visualization of the data), pattern search (e.g., association rule discovery), descriptive modelling (e.g., clustering or density estimation), and predictive modelling (e.g., classification or regression). In this thesis, we address two problems: pattern search and predictive modelling.

"*Pattern* is an expression in some language describing a subset of the data" (Piatetsky-Shapiro & Frawley 1991). Patterns in the data can be represented in many different forms, including classification rules, association rules, clusters, sequential patterns, time series, amongst others. The recently proposed Emerging Pattern (EP) is a new type of knowledge pattern that describes significant changes (differences or trends) between two classes of data (Dong & Li 1999). Emerging Patterns are sets of items (conjunctions of attribute values) whose frequency changes significantly from one dataset to another. Like other patterns or rules composed of conjunctive combinations of elements, Emerging Patterns can be easily understood and used directly by people. Related concepts to Emerging Patterns include version spaces, discriminant rules and contrast sets (see Chapter 3 Section 3.1.4 for details).

The problem of mining Emerging Patterns can be stated as follows: Given two classes of data and a growth rate threshold, find all patterns (itemsets) whose growth rates – the ratio of their frequency between the two classes – are larger than the threshold.

Typically, the number of patterns generated is very large, but only a few of these patterns are likely to be of any interest to the domain expert analyzing the data. The reason for this is that many of the patterns are either irrelevant or obvious, and do not provide new knowledge. To increase the utility, relevance, and usefulness of the discovered patterns, *interestingness measures* are required to reduce the number of patterns. The development of interestingness measures is currently an active research area in KDD. Interestingness measures are broadly divided into objective measures (based on the structure of discovered patterns) and the subjective measures (based on user beliefs or biases regarding relationships in the data) (Silberschatz & Tuzhilin 1996). Both objective and subjective interestingness measures are needed in the context of problems related to Emerging Patterns.

After defining interestingness measures for Emerging Patterns, the problem of mining Emerging Patterns turns into the problem of mining *interesting* Emerging Patterns (called Chi EPs). Instead of mining a huge set of Emerging Patterns first and then find interesting ones among them, it would be better to mine only those Chi Emerging Patterns directly, without generating too many uninteresting candidates.

Classification is the process of finding a set of models that describe and distinguish between two or more data classes or concepts. The model is derived by analyzing a set of training data that have been explicitly labelled with the class that they belong to. The model is then used to predict the class of objects whose class labels are unknown. Because Emerging Patterns represent interesting factors that differentiate one class of samples from a second class of samples, the concept of Emerging Patterns is well suited to serving as a classification model.

In summary, we address the following three major problems involved in the EP-based discovery and classification systems:

1. How to efficiently discover the complete set of Emerging Patterns satisfying the pre-defined thresholds between two classes of data?

2. Which Emerging Patterns are interesting, namely, which Emerging Patterns are novel, useful, and non-trivial to compute?

3. Which Emerging Patterns are useful for classification purposes? How does one use these Emerging Patterns to build efficient and accurate classifiers?

1.2 Motivation

The introduction of Emerging Patterns opens new research opportunities including the efficient discovery of Emerging Patterns and the construction of accurate EP-based classifiers. Several EP mining algorithms have been developed and a few EP-based classifiers have been built. It has been shown that Emerging Patterns are not only useful as a means of discovering distinctions inherently present amongst a collection of datasets, but also a powerful method for constructing accurate classifiers. Inspired by the usefulness of Emerging Patterns, we further explore EP-related problems.

The task of mining Emerging Patterns is very difficult for large, dense and high-dimensional datasets, because the number of patterns present may be exponential in the worst case. What is worse, the Apriori anti-monotone property – every subset of a frequent pattern must be frequent, or in other words, any superset of an infrequent item set cannot

be frequent – which is very effective for pruning the search space, does not apply to mining Emerging Patterns. The reason is as follows. Suppose a pattern X with k items is not an EP. This means its growth rate – the support ratio between two data classes – does not satisfy the growth-rate threshold. Consider Y, a super-pattern of X with $(k+1)$ or more items. Y will usually have decreased support in both classes, but its growth rate (the support ratio) is free to be any real value between 1 and ∞. So a superset of a non-EP may or may not be an EP.

Previous EP mining methods include the border-based approach and ConsEP-Miner (see Chapter 3 Section 3.2 for details). In the border-based approach (Dong & Li 1999), borders are used to represent candidates and subsets of Emerging Patterns; the border differential operation is used to discover Emerging Patterns. Note that some Emerging Patterns may not be mined using the method. The border-based approach depends on border finding algorithms such as Max-Miner (Bayardo Jr. 1998). In fact, the task of mining maximal frequent patterns is very difficult, especially when the minimum support is low (e.g. 5% or even 0.1%). Furthermore, the process of extracting the embodied Emerging Patterns with supports and growth rates from the borders and selecting the useful ones is very time-consuming. To reduce the cost of mining Emerging Patterns, ConsEP-Miner (Zhang, Dong & Ramamohanarao 2000a) was developed. This method follows an level-wise, candidate generation-and-test approach and mines Emerging Patterns satisfying several constraints including the growth-rate improvement constraint. Nevertheless, ConsEPMiner is not efficient when the minimum support is low.

Recently, the merits of a pattern growth method such as FP-growth (Han, Pei & Yin 2000), have been recognized in the field of frequent pattern mining. The pattern growth method adopts a divide-and-conquer philosophy to project and partition databases based on currently discovered patterns and grow such patterns to longer ones in the projected databases. It may eliminate or substantially reduce the number of candidate sets to be generated and also reduce the size of the database to be iteratively examined. The pattern growth methodology injects many new ideas and provides new directions on how to efficiently mine Emerging Patterns.

Data classification has been studied substantially in statistics, machine learning, neural networks, and expert systems and is an important theme in data mining. The family of EP-based classifiers is relatively new. Previous EP-based classifiers include Classification by Aggregating Emerging Patterns (CAEP) (Dong, Zhang, Wong & Li 1999) and the JEP-Classifier (JEPC) (Li, Dong & Ramamohanarao 2001). They aggregate each individual EP's sharp differentiating power to compute aggregate scores for each class in order to make decisions. However, these EP-based classifiers have the following weaknesses.

1. They almost always depend on a huge number of EPs, which makes the resulting classifiers very complex. Although individual EP is easy to understand, users are overwhelmed by tens of thousands of EPs and do not know what kinds of EPs play important roles in the classification decision.

2. Their scoring function is somewhat intuitive and lacks theoretical support.

The first weakness motivates us to find interestingness measures for Emerging Patterns. The second leads us to design a new scoring function that combines the power of Emerging Patterns and Bayesian theory.

1.3 Contributions of Thesis

Figure 1.1 shows various types of Emerging Patterns, their relationships, advantages and disadvantages. Emerging Patterns, in the most primitive format, are ρ-EPs, where only growth rates (ρ - the growth rate threshold) are concerned. The first step is the introduction of Jumping Emerging Patterns (JEPs) (Li, Dong & Ramamohanarao 2001), which are Emerging Patterns with infinite growth rates. We further extend JEPs and propose Essential Jumping Emerging Patterns (EJEPs). These are shown in the left stream of Figure 1.1. Going back to ρ-EPs, we take another direction and add several additional interestingness constraints for Emerging Patterns. We call Emerging Patterns that satisfy these constraints Chi Emerging Patterns (Chi EPs in short).

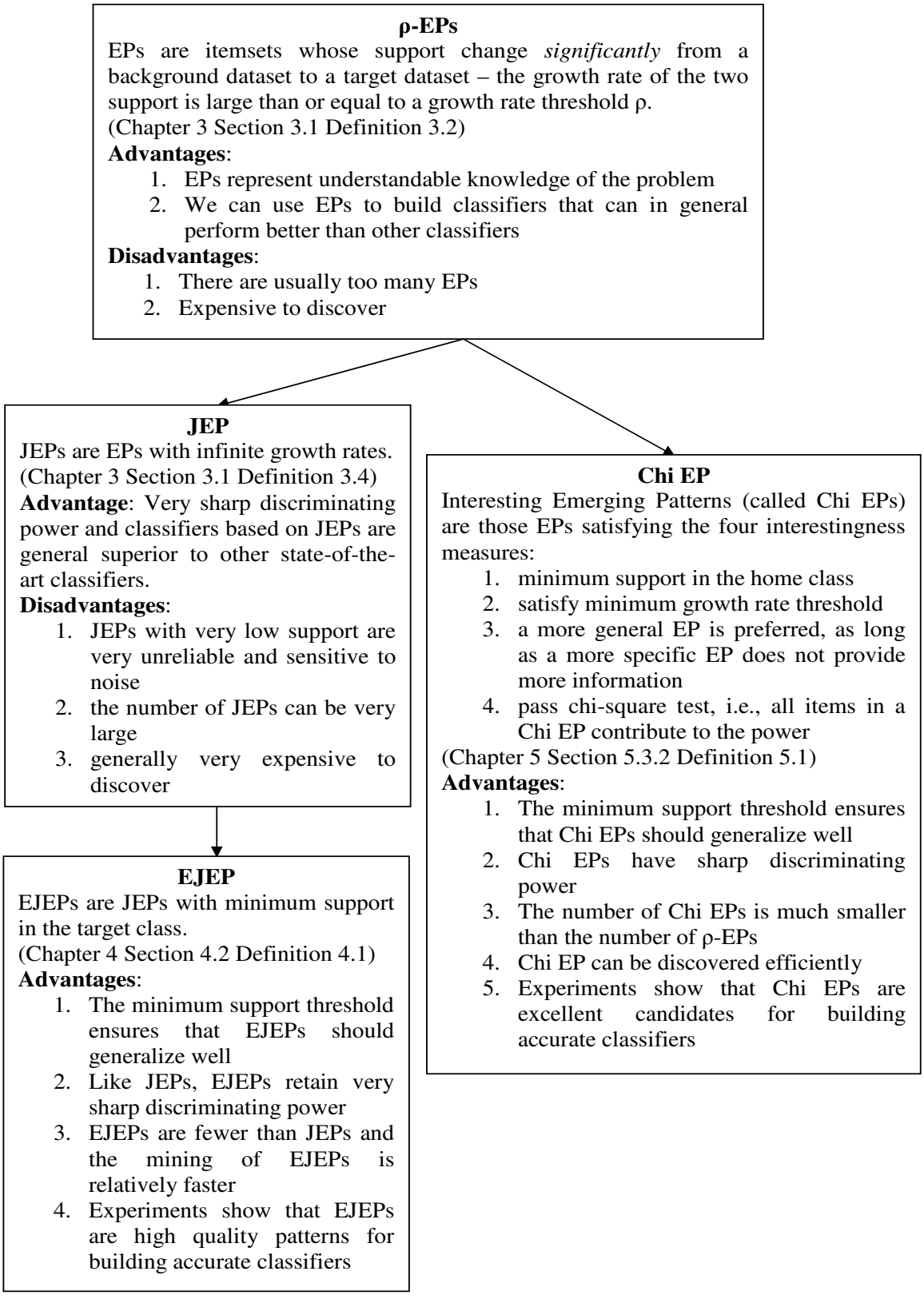


Figure 1.1: Evolution diagram for Emerging Patterns (EPs)

This thesis makes the following contributions to the art and science of KDD:

1. We propose a special type of Emerging Patterns, called Essential Jumping Emerging Patterns (EJEPs) and show that they are high quality patterns for building accurate classifiers.

2. We develop a new efficient algorithm for mining EJEPs of both data classes (both directions) in a single pass.

3. We generalize the interestingness measures for Emerging Patterns, including the minimum support, the minimum growth rate, the subset relationship between Emerging Patterns and the correlation based on common statistical measures such as chi-squared value.

4. We develop tree-based pattern growth algorithms for mining only those interesting Emerging Patterns (called Chi EPs). We show that our mining algorithm maintains efficiency even at low supports on data that is large, dense and has high dimensionality.

5. We propose a novel approach to use Emerging Patterns as a basic means for classification, Bayesian Classification by Emerging Patterns (BCEP). As a hybrid of the EP-based classifier and Naive Bayes (NB) classifier, it provides several advantages. First, it is based on theoretically well-founded mathematical models as NB and Large Bayes (LB). Second, it extends NB by using essential Emerging Patterns to relax the strong attribute independence assumption. Lastly, it is easy to interpret, as many unnecessary Emerging Patterns are pruned based on data class coverage.

6. We systematically study the noise tolerance of our BCEP classifier, in comparison to other state-of-the-art classifiers. We show that BCEP deals with noise better due to its hybrid nature.

1.4 Outline of Thesis

This thesis is organized as follows. In Chapter 2, we present an overview of KDD. We focus on the problem of association rules and classification.

In Chapter 3, we review previous works on Emerging Patterns, including algorithms for mining Emerging Patterns and approaches to build EP-based classifiers.

In Chapter 4, we first define Essential Jumping Emerging Patterns (EJEPs). A new single-scan algorithm is presented to efficiently mine EJEPs of both data classes (both directions) in one single pass. We then build classifiers based exclusively on EJEPs and compare them to the JEP-classifier and other state-of-the-art classifiers. Experimental results show that mining of EJEPs is much faster than mining JEPs and that EJEP based classifiers use much fewer patterns than the JEP-classifier, while achieving the same or higher accuracy. We conclude that EJEPs are high quality patterns for building accurate classifiers.

In Chapter 5, we first generalize the interestingness measures for Emerging Patterns, including the minimum support, the minimum growth rate, the subset relationship between Emerging Patterns and the correlation based on common statistical measures such as chi-squared value. We then present an efficient algorithm for mining only those interesting Emerging Patterns (called Chi EPs), where the chi-squared test is used as a heuristic to prune the search space. Our experimental results show that the algorithm maintains high efficiency even at low supports on data that is large, dense and has high dimensionality. They also show that the heuristic is admissible, because only unimportant Emerging Patterns with low supports are ignored.

In Chapter 6, we first introduce the idea of data class coverage to prune many unnecessary or redundant Emerging Patterns. We then detail our Bayesian approach to use those selected Emerging Patterns for classification. We also discuss the differences between BCEP and Large Bayes (LB). We present an extensive experimental evaluation of BCEP on popular benchmark datasets from the UCI Machine Learning Repository and compare its performance with Naive Bayes (NB), the decision tree classifier C5.0, Classification by Aggregating Emerging Patterns (CAEP), Large Bayes (LB), Classification Based on Association (CBA), and Classification based on Multiple Association Rules (CMAR).

In Chapter 7, we systematically compare the noise resistance of our BCEP classifier with other major classifiers, such as Naive Bayes (NB), the decision tree classifier C4.5, Support Vector Machines (SVM) classifier, and the JEP-C classifier, using bench-

mark datasets from the UCI Machine Learning Repository that have been affected by three different kinds of noise. The empirical study shows that our method can handle noise better than other state-of-the-art classification methods.

In Chapter 8 we conclude our work and discuss some future research problems.

Chapter 2

Literature Review

In this chapter, we first provide an overview of Knowledge Discovery in Databases (KDD), and survey KDD from a number of perspectives. Specifically, we discuss the relationship between KDD and three key fields of research, i.e., Database, Machine Learning and Statistics. We then describe the frequent pattern mining problem, which is related to the Emerging Pattern mining problem studied in this work. Our review ends with a presentation of a number of state-of-the-art classifiers, ranging from classifiers based on decision trees, Bayesian classifiers, association rules based classifiers, support vector machines, to neural networks.

2.1 Knowledge Discovery and Data Mining: an Overview

Recently, scientific, commercial and social applications have accumulated huge volumes of high-dimensional data, stream data, unstructured and semi-structured data, and spatial and temporal data. Large-scale data-collection techniques have emerged in the biomedical domain. For example, DNA microarrays allow us to simultaneously measure the expression level of tens of thousands of genes in a population of cells (Piatetsky-Shapiro & Tamayo 2003). Digital geographic data also grow rapidly in scope, coverage and volume, because of the progress in data collection and processing technologies. For example, data is constantly acquired through high-resolution remote sensing systems installed in NASA's Earth-observation satellites, global position systems and in-vehicle navigation systems (Han,

Altman, Kumar, Mannila & Pregibon 2002). Moreover, the U.S. National Spatial Data Infrastructure makes those large space-related datasets available for analysis worldwide.

Although large databases of digital information are ubiquitous, the datasets themselves in raw form are of little direct use. What is of value is the knowledge that can be inferred from the data and put to use. For example, in the telecommunication industry, a significant stream of call records are collected at network switches. Except for billing, these records, in raw form, have no other values. However, by "mining" calling patterns from the vast quantities of high-quality data, the discovered knowledge can be used directly for toll-fraud detection and consumer marketing, which may save the company millions of dollars per year (Han, Altman, Kumar, Mannila & Pregibon 2002). The explosive growth of data renders the traditional manual data analysis impractical. The Knowledge Discovery in Databases (KDD) and data mining field draws on findings from statistics, database, and artificial intelligence to construct new techniques and tools that can intelligently and automatically transform the data into useful knowledge. In other words, KDD enables us to extract useful reports, spot interesting events and trends, support decisions and policy based on statistical analysis and inference, and exploit the data to achieve business, operational, or scientific goals.

The terminology associated with the task of finding useful information in data, varies in different research communities. Knowledge Discovery in Databases (KDD), data mining, knowledge extraction, information discovery, information harvesting, data archeology, data analysis are some of the most common expressions used. Out of these different names, "KDD" and "data mining" are particularly prominent. The term "KDD" is often used to refer to the overall process of discovering useful knowledge from data, while "data mining" stresses the application of specific algorithms for extracting patterns or models from data. In principle, data mining is not specific to one type of media or data. Instead, data mining can be applied to various kinds of information repositories, including relation databases, data warehouses, transactional databases, object-oriented databases, spatial databases, temporal and time-series databases, text and multimedia databases, heterogeneous and legacy databases, as well as the World Wide Web (WWW).

2.1.1 The KDD process

In general, a KDD process is interactive and iterative (with many decisions made by the user), comprising the following steps:

- data cleaning, which handles noisy, erroneous, missing, or irrelevant data;
- data integration, where multiple data sources, often heterogeneous, may be combined in a common source;
- data selection, where data relevant to the analysis task are retrieved from the data collection;
- data transforming, also known as data consolidation, where data is transformed into forms appropriate for mining;
- data mining, which is a crucial step where intelligent techniques are applied to extract potentially useful patterns;
- pattern evaluation, which aims to identify the truly interesting patterns representing knowledge based on some interestingness measure;
- knowledge presentation, where visualization and knowledge representation techniques are used to help the user understand and interpret the discovered knowledge.

Although the other steps are equally important if not more important for the successful application of KDD in practice, the data mining step has received by far the most attention in the literature.

2.1.2 Data Mining Tasks

Data mining involves fitting models to, or determining patterns from observed data. The kinds of patterns that can be discovered depend upon the data mining tasks employed. By and large, there are two types of data mining tasks: descriptive and predictive. Descriptive data mining tasks describe a dataset in a concise and summary manner and present interesting general properties of the data; predictive data mining tasks construct

one or a set of models, perform inference on the available set of data, and attempt to predict the behavior of new data objects.

A brief discussion of these data mining tasks follows:

- Concept or Class description. Class description provides a concise and succinct summarization of a collection of data and distinguishes it from others. The summarization of a collection of data is called class characterization, which produces characteristic rules; whereas the comparison between two or more collections of data is called comparison or discrimination, which produces discriminant rules.

- Association. Association is the discovery of association relationships or correlations among a set of items, which are commonly called association rules. An association rule in the form of $X \Rightarrow Y$ is interpreted as "database tuples that satisfy X are likely to satisfy Y".

- Classification. Classification, also known as supervised classification, analyzes a set of training data (i.e., a set of objects whose class labels are known) and constructs a model for each class based on the features in the data.

- Prediction. Prediction refers to the forecast of possible values of some missing data or the value distribution of certain attributes in a set of objects.

- Clustering. Clustering, also called unsupervised classification, involves identifying clusters embedded in the data, where a cluster is a collection of data objects that are "similar" to one another. Many clustering methods are based on the principle of maximizing the similarity between objects in a same class (intra-class similarity) and minimizing the similarity between objects of different classes (inter-class similarity).

- Evolution and deviation analysis. Data evolution analysis describes and models regularities or trends for objects whose behavior change over time. Although this may include characterization, discrimination, association, classification or clustering of time-related data, distinct features of evolution analysis include time-series data analysis, sequence or periodicity pattern matching, and similarity-based data analysis. Devia-

tion analysis considers differences between measured values and expected value and attempts to find the cause of the deviation.

- Outlier analysis. Outliers, also known as exceptions or surprises, are data elements that cannot be grouped in a given class or cluster. While outliers can be considered noise and discarded in some applications, they can reveal important knowledge in other domains.

From another angle, data mining techniques can be divided into two general classes of tools, according to whether they are aimed at model building or pattern detection. In model building, one is trying to produce an overall summary of a set of data, to identify and describe the main features of the shape of the distribution. Examples of such "global" models include a cluster analysis partition of a set of data, a regression model for prediction, and a tree-based classification rule. In contrast, in pattern detection, one is seeking to identify small (but nonetheless possibly important) departures from the norm, to detect unusual patterns of behavior. Examples include unusual spending patterns in credit card usage (for fraud detection) and objects with patterns of characteristics unlike any others.

In this thesis, we focus on discovering a type of knowledge patterns called Emerging Patterns that describe differences between two classes of data, and classification using Emerging Patterns. The problem of association rule mining is further discussed in Section 2.2, where we discuss frequent pattern mining in detail because it is the most important step in association rule generation. We formally state the problem of classification in Section 2.3, and review a number of different classifiers. The task of prediction and clustering are out of the scope of this thesis, although they are closely related to classification. An excellent survey of clustering can be found in (Jain, Murty & Flynn 1999). For an overview of time-series data mining, please refer to (Last, Klein & Kandel 2001).

2.1.3 Interestingness

A data mining system has the potential to generate thousands or even millions of patterns, or rules. A very large number of patterns may lead to a data-mining problem of the second-order, i.e., the interpretation and evaluation of the discovered patterns could be

a highly resource-consuming exercise. Typically, only a small fraction of these generated patterns would actually be of interest to any given user. What makes a pattern interesting? The notion of *interestingness* is usually taken as an overall measure of pattern value, combining validity, novelty, usefulness, and simplicity. In other words, a pattern is interesting if it is

1. easily understood by humans;
2. valid on new or test data with some degree of certainty;
3. potentially useful;
4. novel.

A pattern is also interesting if it validates a hypothesis that the user sought to confirm. An interesting pattern represents knowledge. Interestingness functions can be defined explicitly or can be manifested implicitly through an ordering placed by a KDD system on discovered patterns or models.

The development of good measures of interestingness of discovered patterns is one of the central problems in the field of KDD. The measures of interestingness are divided into the objective measures and the subjective measures (Silberschatz & Tuzhilin 1996, Freitas 1999). The objective measures are those that depend only on the structure of a pattern and the underlying data used in the discovery process. In the context of association rule mining, objective measures can be itemset support (the frequency with which combinations of items appear in sales transactions) and rule confidence (the conditional probability of some item being purchased given that a set of items were purchased) (Agrawal & Srikant 1994). Other objective measures include lift, strength and conviction, to name a few (Bayardo Jr. & Agrawal 1999).

Although objective measures help identify interesting patterns, they are insufficient unless combined with subjective measures that reflect the needs and interests of a particular user. The subjective measures are those that also depend on the beliefs or biases of the users who examine the pattern or relationships in the data. These measures regard patterns as interesting if they are *unexpected* (contradicting a user's belief) or *actionable*

(offering strategic information on which the user can act). Patterns that are *expected* can be interesting if they confirm a hypothesis that the user wishes to validate, or resemble a user's hunch.

It is often unrealistic and inefficient for data mining systems to generate all possible patterns. Instead, user-specified constraints and interestingness measures should be used to focus the search. As an optimization problem, generating only interesting patterns remains a challenging issue in data mining.

2.1.4 KDD Viewed from Different Perspectives

A Database Perspective on Knowledge Discovery

Traditionally, Database Management Systems (DBMS) were used to support business data processing applications. Much DBMS research addresses issues such as efficiency and scalability in the storage and handling of large amounts of data. Recently, Data Warehousing (DW) or Online Analytical Processing (OLAP) addresses the issue of storing and accessing information useful for high-level Decision Support Systems (DSS), rather than for low-level operational (production) purposes (Chaudhuri & Dayal 1997). A data warehouse is a "subject-oriented, integrated, time-varying, non-volatile collection of data that is used primarily in organization decision making" (Chaudhuri & Dayal 1997). OLAP and data mining tools enable sophisticated data analysis on these enterprise data warehouses, which are usually hundreds of gigabyte to terabyte in size.

Data mining queries pose some unusual problems.

1. They tend to involve aggregations of huge amounts of data.

2. They tend to be ad hoc, issued by decision makers who are searching for unexpected relationships.

3. In applications such as trading of commercial instruments, there is a need for an extremely fast response, and the figure of merit is total elapsed time, including the writing, debugging, and execution of the query.

4. Often, the user cannot formulate a precise query and his real question is "Find me something interesting?"

Thus in the database community, data mining research is concerned about the following:

- Optimization techniques for complex queries, such as those involving aggregation and grouping.
- Techniques for supporting "multidimensional" queries where the data is organized into a "data cube" consisting of a quantity of interest broken down into "dimensions".
- Optimization techniques involving tertiary storages.
- Very high-level query languages and interfaces that support nonexpert users making ad-hoc queries.

Although data mining builds upon the existing body of work in statistics and machine learning, it provides completely new functionalities. The key new component is the ad hoc nature of KDD queries and the need for efficient query compilation into a multitude of existing and new data analysis methods. Most current KDD systems and OLTP tools offer isolated discovery features using tree inducers, neural networks, rule discovery and other data mining algorithms. In fact, these techniques of data mining would appropriately be described as "file mining" since they assume a loose coupling between a data-mining engine and a database. It is argued that the development of querying tools for data mining is one of the big challenges for the database community (Imielinski & Mannila 1996).

Machine Learning in Knowledge Discovery

Humans excel at tasks such as learning, or gaining the ability to perform tasks from examples and training. Artificial intelligence is concerned with improving algorithms by employing problem solving techniques used by human beings (Russell & Norvig 2003). As an sub-branch of artificial intelligence, machine learning involves the study of how machines and humans can learn from data (Mitchell 1997). In more recent years (since the early 1980's), much research in machine learning has shifted from modelling how humans learn

to the more pragmatic aims of constructing algorithms which learn and perform well on specific tasks (such as prediction).

Data mining is regarded as the confluence of machine learning techniques and the performance emphasis of database technology (Agrawal, Imielinski & Swami 1993a). In fact, machine learning plays such an important role in KDD that some claim that knowledge discovery is simply machine learning with large datasets, and that the database component of the KDD is essentially maximizing performance of mining operations running on top of large persistent datasets and involving expensive I/O.

Although machine learning shares some research questions with data mining, it is also concerned with others. Data mining can be regarded as *empirical learning*, where learning relies on some form of external experience. Machine learning also studies *analytical learning*, where learning involve no interaction with an external source of data. *Analytical learning* focuses on improving the speed and reliability of the inferences and decisions that computers perform. An example is *explanation-based learning*, where it remembers and analyzes past searches to make future problems be solved quickly and with little or no search.

The Relationship between Statistics and Data Mining

From a statistical perspective, data mining can be viewed as computer automated exploratory data analysis of usually large complex datasets (Friedman 1997).

Many techniques that are popular in data mining have their roots in applied statistics. Methods for prediction includes decision trees, nearest neighbor models, naive Bayes models, and logistic regression. K-means and mixed models using Expectation-Maximization (EM) for clustering and segmentation are also prevalent in Statistics. A statistician might argue that data mining is not much more than the scaling up of conventional statistical methods to very large datasets and it is just a large-scale "data engineering" effort.

However, there are several contributions that have arisen primarily from work within computer science rather than from conventional statistics. These include: flexible predictive modelling methods, the use of hidden variable models for large-scale clustering

and prediction problems, finding patterns rather than global models (pattern-mining algorithms do not attempt to "cover" all of the observed data, but rather focus on "local" pockets of information in a data-driven manner), the data engineering aspects of scaling traditional algorithms to handle massive datasets, analyzing heterogeneous structured data such as multimedia (images, audio, video) data and Web and text documents.

There are other general distinctions between data mining and statistics. Statisticians care about experimental design, the construction of an experiment to collect data to test a specific hypothesis. In contrast, data mining is typically concerned with observational retrospective data, i.e., data that has already been collected for some other purpose.

Although there can be significant differences between the views from data mining and statistics, there are numerous well-known examples of symbiosis at the computer science/statistics interface. Research on essentially the same idea is first carried out independently within each field, and later integrated to form a much richer and broader framework. Success stories include neural networks, graph-based models for efficient representation of multivariate distributions, latent (hidden) variable models, decision trees, and boosting algorithms.

Statistical and algorithmic issues are both important in the context of data mining. Statistics is an essential and valuable component for any data mining exercise. Data mining and statistics have much in common. Data mining can prosper by cultivating and harvesting ideas from statistics (Glymour, Madigan, Pregibon & Smyth 1997).

2.2 Frequent Pattern Mining Problem

2.2.1 Preliminaries

The frequent pattern mining problem was first introduced as mining association rules between sets of items (Agrawal, Imielinski & Swami 1993b). It is formally stated as follows. Let $I = \{i_1, i_2, \cdots, i_m\}$ denote the set of all items. A set $X (X \subseteq I)$ of items is also called an itemset. Particularly, an itemset with l items is called an *l-itemset*. A transaction $T = (tid, X)$ is a tuple where tid is a transaction ID and X is an itemset. $T = (tid, X)$ is said to contain itemset Y if $Y \subseteq X$. A transaction database TDB is a set of transactions.

The count of an itemset X in TDB, denoted as $count_{TDB}(X)$ or simply $count(X)$ when TDB is clear, is the number of transactions in TDB containing X. The support of an itemset X in TDB, denoted as $sup_{TDB}(X)$ or simply $sup(X)$ when TDB is clear, is the proportion of transactions in TDB containing X, i.e.,

$$sup_{TDB}(X) = \frac{count_{TDB}(X)}{|TDB|},$$

where $|TDB|$ is the total number of transactions in TDB.

Definition 2.1 Given a transaction database TDB and a minimum support threshold ξ, an itemset X is called a frequent itemset or frequent pattern if $sup(X) \geqslant \xi$.

The problem of mining frequent patterns is to find the complete set of frequent patterns in a transaction database with respect to a given support threshold.

Definition 2.2 An association rule is an implication of the form $X \Rightarrow Y$, where X and Y are itemsets and $X \cap Y = \emptyset$. The left-hand side (LHS) X is called the antecedent of the rule, and the right-hand side (RHS) Y is called the consequent of the rule. The support of the rule in a transaction database TDB is $sup_{TDB}(X \cup Y)$; the confidence of the rule in TDB is

$$\frac{sup_{TDB}(X \cup Y)}{sup_{TDB}(X)}.$$

An association rule indicates an affinity between the antecedent itemset and the consequent itemset, where frequency-based statistics such as support and confidence describe the relationship.

The problem of association rule mining from a transaction database is to find the complete set of association rules that have support and confidence no less than the user-specified thresholds. Association rule mining can be divided into two steps.

1. find all frequent itemsets with respect to the support threshold;

2. construct rules which exceed the confidence threshold from the frequent itemsets in step 1.

In practice, the first step is the most time-consuming and the second step is simple and quite straightforward. Therefore, much research is devoted to methods for generating all frequent itemsets efficiently (Agrawal et al. 1993b, Agrawal & Srikant 1994, Park, Chen & Yu 1995, Savasere, Omiecinski & Navathe 1995, Toivonen 1996, Brin, Motwani, Ullman & Tsur 1997, Bayardo Jr. 1998, Bayardo Jr. & Agrawal 1999, Bayardo, Agrawal & Gunopulos 2000, Han et al. 2000, Wang, He & Han 2000, Pei, Han, Lu, Nishio, Tang & Yang 2001, Burdick, Calimlim & Gehrke 2001, Agarwa, Aggarwal & Prasad 2001, Liu, Pan, Wang & Han 2002, Han, Wang, Lu & Tzvetkov 2002, Zou, Chu, Johnson & Chiu 2002).

2.2.2 Apriori Algorithm for Generating Frequent Patterns

A naive algorithm for mining frequent itemsets would simply consider every possible combinations of items. If there are 1,000 items, which is common in market basket analysis, 2^{1000} (that is approximately 10^{300}) candidate itemsets will require examination. Such an extremely large search space renders the naive algorithm computationally intractable. To overcome the difficulties, an anti-monotonic property of frequent itemsets, called the Apriori heuristic, was identified in (Agrawal et al. 1993b, Agrawal & Srikant 1994).

Property 2.1 Any superset of an infrequent itemset can not be frequent. In other words, every subset of a frequent itemset must be frequent.

Apriori Algorithm

The Apriori algorithm (Algorithm 2.1) computes the frequent itemsets in the database through several iterations. Iteration l computes all frequent l-itemsets. Each iteration has two steps: *candidate generation* and *candidate counting and selection*. Line 3 in Algorithm 2.1 generates new candidates. Rather than the computationally expensive exercise of testing whether all subsets of a candidate itemset are frequent, the candidate generation process produces a new candidate (a $(k+1)$-itemset) from two of its immediate subsets (k-itemsets) that are both frequent.

Algorithm 2.1: Apriori Algorithm for generating frequent itemsets

input : a transaction database TDB and a minimum support threshold ξ

output: the complete set of frequent patterns in TDB with respect to ξ

```
1 scan TDB once to find L_1 = {frequent length-1 itemsets};
2 for k = 2; L_{k-1} ≠ ∅; k++ do
3 |   C_k = {{x_1, x_2, ..., x_{k-2}, x_{k-1}, x_k} | {x_1, x_2, ..., x_{k-2}, x_{k-1}} ∈
  |   L_{k-1} ∧ {x_1, x_2, ..., x_{k-2}, x_k} ∈ L_{k-1};
4 |   foreach transaction t ∈ TDB do
5 |   |   foreach candidate c ∈ C_k ∧ c ⊆ t do c.count ++;
  |   end
6 |   L_k = {c ∈ C_k | c.count ⩾ ξ};
  end
7 return ⋃_k L_k;
```

Improvements over the Apriori Algorithm

Several refinements over Apriori have been proposed that focus on reducing the number of database scans, the number of candidate itemsets counted in each scan, or both (Park et al. 1995, Savasere et al. 1995, Toivonen 1996, Brin, Motwani, Ullman & Tsur 1997).

DHP (Direct Hashing and Pruning) (Park et al. 1995) is a hash-based, Apriori-like algorithm, with improvement on candidate generation and counting. DHP employs a hash table, which is built in the previous pass (L_{k-1}), to test the eligibility of a k-itemset. DHP adds a k-itemset into candidate set C_k only if that k-itemset is hashed into a hash entry whose value is large than or equal to the minimum support threshold. As a result, the size of C_k can be reduced significantly. DHP also reduces the database size progressively by not only trimming each individual transaction but also pruning the number of transactions in the database.

DIC (Dynamic itemset counting) (Brin, Motwani, Ullman & Tsur 1997) attempts to reduce the number of database scans. Intuitively, DIC works like a "train" carrying candidate itemsets running over the database with stops at intervals M transactions apart, i.e. it reads M transactions at a time and updates the appropriate support counts of its

"passengers" (the candidate itemsets). When the train reaches the end of the database, it has made one scan and it starts over at the beginning for the next scan. The support of a "passenger" is updated each time a transaction containing the itemset is scanned. Instead of counting only l-itemsets in the lth iteration, DIC overlaps the counting of different lengths of itemsets to save some database scans. It also explores efficient support counting to reduce the number of inner loops (line 5 in Algorithm 2.1). It is reported that DIC is faster than Apriori when the support threshold is low (Brin, Motwani, Ullman & Tsur 1997).

There are also works on optimizing Apriori by partitioning (Savasere et al. 1995) and sampling (Toivonen 1996).

Extensions and generalizations to the basic problem of finding frequent itemset have been proposed, such as calendric market basket analysis (Ramaswamy, Mahajan & Silberschatz 1998), mining correlations (Brin, Motwani & Silverstein 1997), causality (Silverstein, Brin, Motwani & Ullman 2000), sequential patterns (Agrawal & Srikant 1995), episodes (Mannila, Toivonen & Verkamo 1997), max-patterns (Bayardo Jr. 1998), constraint-based mining (Srikant, Vu & Agrawal 1997, Ng, Lakshmanan, Han & Pang 1998, Grahne, Lakshmanan & Wang 2000), cyclic association rules (Ozden, Ramaswamy & Silberschatz 1998), and many other patterns.

The Apriori algorithm is very efficient when the data is sparse and there are few frequent itemsets containing a large number of items. Data is considered sparse when each item is relatively infrequent. For dense data, the number of frequent itemsets can greatly outnumber the number of transactions, requiring a lot of computation for the management of candidate frequent itemsets. Thus, Apriori is less efficient when applied to dense data.

2.2.3 FP-growth: A Pattern Growth Method

The frequent pattern growth (FP-growth) method (Han et al. 2000) is an efficient way of mining frequent itemsets in large databases. The algorithm mines frequent patterns without the time-consuming candidate-generation process that Apriori-like algorithms employ.

FP-tree Data Structure

The Frequent-Pattern tree (FP-tree) (Han et al. 2000) is a highly compact structure that stores the complete information of the original transaction database necessary for frequent pattern mining.

Definition 2.3 A Frequent-Pattern tree (FP-tree) is a tree structure defined as:

1. It consists of one root labelled as "null", a set of item-prefix subtrees as the children of the root, and a frequent-item-header table.
2. Each node in the item-prefix subtree consists of three fields: item-name, count and node-link, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this node, and node-link links to the next node in the FP-tree carrying the same item, or null if there is none.
3. Each entry in the frequent-item-header table consists of two fields: (1) item-name; and (2) head of node-link, which points to the first node in the FP-tree carrying the item-name.

Example 2.1 Suppose the transaction database given in Table 2.1 and $\xi = 3$. The constructed FP-tree is shown in Figure 2.1. Note that the header table is built to facilitate tree traversal: every frequent item connects nodes in the FP-tree with its values through node-links, e.g. all p nodes are connected in one list. Those node-links are represented by dashed lines. □

The FP-tree construction algorithm (Algorithm 2.2) is based on the FP-tree definition. The function `insert_tree([p|P],T)` (Algorithm 2.3) is called recursively during the construction of the FP-tree.

FP-growth Method

The FP-growth algorithm mines frequent itemsets using the FP-tree structure. Let us examine the mining process based on the FP-tree shown in Figure 2.1. According

Table 2.1: A small transaction database

TID	Items Bought	Ordered Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

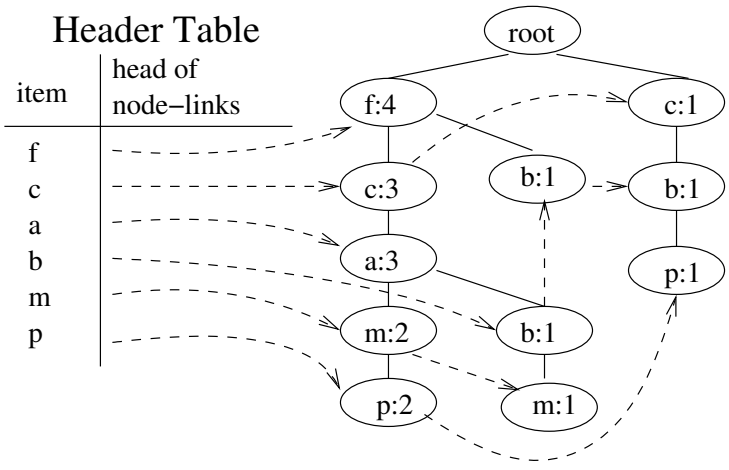


Figure 2.1: The corresponding FP-tree of the small transaction database

Algorithm 2.2: FP-tree construction

input : A transaction database TDB and a minimum support threshold ξ

output: The frequent pattern tree of TDB, FP-tree

/* `The first database scan` */

1 Scan TDB once;

2 Collect the set of frequent items F and their supports;

3 Sort F in support descending order as L, the list of frequent items;

/*`The second database scan` */

4 Create the root of an FP-tree, R, and label it as "null";

5 **foreach** *transaction* t *in* TDB **do**

6 | Select and sort the frequent items in t according to the order of L;

7 | Let the sorted frequent item list in t be $[p|P]$, where p is the first element and P is the remaining list. Call `insert-tree(`$[p|P]$, R`)`;

end

Algorithm 2.3: Function: insert-tree($[p|P], T$) (for mining frequent itemsets)

/* $[p|P]$ `is the sorted frequent item list, where` p `is the first element and` P `is the remaining list` */

/* T `refers to a subtree in the FP-tree` */

1 **if** T *has a child* N *such that N.items-name=p.item-name* **then**

increment N's count by 1

end

2 **else**

create a new node N, and let its count be 1, its parent link be pointed to T, and its node-link be pointed to the nodes with the same item-name via the node-link structure

end

3 **if** P *is nonempty* **then** call `insert-tree(`P, N`)`;

to the list of frequent items, the complete set of frequent itemsets can be divided into six subsets without overlap:

1. frequent itemsets having p (the least frequent item);
2. the frequent itemsets having m but not p;
3. the frequent itemsets with b and without both m and p;
4. the frequent itemsets with a and without b, m and p;
5. the frequent itemsets with c and without a, b, m and p;
6. the frequent itemsets only with f.

Based on node-link connections, we collect all the transactions that p participates in by starting from the header table of p and following p's node-links. Two paths will be selected in the FP-tree: $\{(f:4),(c:3),(a:3),(m:2),(p:2)\}$ and $\{(c:1),(b:1),(p:1)\}$. These two paths of p form p's sub-pattern base, which is called p's *conditional pattern base* (i.e., the sub-pattern base under the condition of p's existence). p's *conditional FP-tree* is constructed on its conditional pattern base. Here p's *conditional FP-tree* contains only one branch $\{(c:3)\}$. Hence only one frequent itemset $\{c,p\}$ (with the support of 3) is derived. The search for frequent patterns associated with p terminates.

The same process is repeated for the remaining five subsets. The final set of frequent patterns for our example is $\{\{c,p\},\{f,c,a,m\}\}$.

Given the FP-tree constructed based on Algorithm 2.2 and a minimum support threshold ξ, the complete set of frequent patterns can be mined by calling the function `FP-growth` (see Algorithm 2.4) recursively. Initially, *Tree* =FP-tree and $\alpha = \emptyset$.

Experiments have shown that the FP-growth algorithm is faster than the Apriori algorithm by about an order of magnitude.

Other Pattern Growth Methods

The TreeProjection method (Agarwa et al. 2001) proposes database projection technique that explores the projected databases associated with different different itemsets.

Algorithm 2.4: FP-growth(Tree, α)

```
1 if Tree contains a single path P then
2     foreach combination(denoted as β) of the nodes in P do
3         generate pattern β ∪ α with support = minimum support of nodes in β;
      end
  end
4 else foreach a_i in the header of Tree do
5     generate pattern β = a_i ∪ α with support = a_i.support;
6     construct β's conditional pattern base;
7     construct β's conditional FP-tree Tree_β;
8     if Tree_β is not empty then call FP-growth(Tree_β, β);
  end
```

It mines frequent patterns by successive construction of the nodes of a lexicographic tree of itemsets. It is reported that the algorithm is up to one order of magnitude faster than other recent techniques in literature (Agarwa et al. 2001).

H-mine, is proposed for mining frequent patterns using a simple, memory-based hyper-structure, H-struct (Pei, Han, Lu, Nishio, Tang & Yang 2001). Using H-struct, H-mine(Mem) is a memory-based, efficient algorithm for mining frequent patterns for the database that can fit in (main) memory. For large databases, H-mine first partitions the database, mines each partition in memory using H-mine(Mem), and then consolidates the global frequent patterns. Performance studies show that H-mine is highly scalable and is faster than Apriori and FP-growth.

The pattern growth techniques have been used in constraint-based mining (Pei, Han & Lakshmanan 2001), where a user expresses his focus for mining by means of a rich class of constraints that capture application semantics.

2.3 Classification

Learning how to classify objects to one of a pre-specified set of categories or classes is a characteristic of intelligence that has been of keen interest to researchers in psychology and computer science (such as artificial intelligence, machine learning, and neural networks). The task is called classification, which has been studied extensively by the machine learning community as a possible solution to the "knowledge acquisition" or "knowledge extraction" problem. Recently, classification has become an important data mining problem (Fayyad et al. 1996, Chen, Han & Yu 1996, Han & Kamber 2000). In general, given a database of records, each with a class label, a classifier generates a concise and meaningful description (or model) for each class in terms of attributes. The model is then used by the classifier to predict class labels of unknown objects. When the model is used to predict numerical values rather than class labels, the problem is often specifically referred to as regression.

Classification is also known as *supervised learning*, as the learning of the model is "supervised" in that it is told which class each training example belongs to. In contrast to supervised learning is *unsupervised learning*, which is sometimes called *clustering* (see Section 2.1.2). Often the goal in unsupervised learning is to decide which objects should be grouped together – in other words, the learner forms the classes itself.

In this thesis, we focus on classification, i.e., supervised learning of a model to predict class labels of future, unlabelled records.

2.3.1 Statement of Classification Problem

In a typical classification problem, data is represented as a table of *records* (also called *instances*, *examples* or *tuples*). Each instance is described by a fixed number of measurements, called *features* or *attributes*. Attributes with numerical domains are referred to as *numerical* or *continuous*, where attribute values are infinite real numbers. Attributes whose domains are not numerical are referred to as *categorical* or *discrete*, where attribute values are members of a finite set of categories[1]. There is one distinguished attribute, called the *dependent attribute*, or the *class label*, which denotes the class membership of a record.

[1]Some authors distinguish between categorical attributes that take values in an unordered set (*nominal* attributes) and categorical attributes having ordered scales (*ordinal attributes*).

The remaining attributes are called *predictor attributes*.

A **classifier** is a function that maps unlabelled instance to a label using internal data structures. An **inducer**, or an induction algorithm, builds a classifier from a given dataset.

In most research on classification, the dataset is divided into a "training set" (a set of *training instances*) and a "test set" (a set of *testing instances*). The training set is used to generate a classifier, while the test set is used to assess the classifier's predictive accuracy. Following (Breiman, Friedman, Olshen & Stone 1984), we refer to a classifier's performance on the training set as *resubsititution error* and *resubstitution accuracy*. The terms *error* and *accuracy* alone refer to performance on the test set; we sometimes refer to them as *predictive* or *generalization* error/accuracy. Usually, datasets from the UCI Machine Learning Repository (Blake & Merz 1998) are used as the benchmark to evaluate the performance of different classifiers, because these datasets represent a wide range of application domains and the size, the number of classes and attributes vary significantly with different datasets (see Appendix A for the summary of datasets used in this thesis).

Example 2.2 Table 2.2 from (Quinlan 1986) shows 14 instances of suitable and unsuitable days for which to play a game of golf. Each instance is a day described in terms of the nominal attributes Outlook, Temperature, Humidity and Windy, along with the class label which indicates whether the day is suitable for playing golf or not. □

Classification has been successfully applied to a wide range of application areas, such as scientific experiments, medical diagnosis, weather prediction, credit approval, customer segmentation, target marketing, and fraud detection (Fayyad et al. 1996, Brachman, Khabaza, Kloesgen, Piatetsky-Shapiro & Simoudis 1996). Many classification models have been proposed in the literature: Neural networks (Bishop & Bishop 1995, Ripley 1996), genetic algorithms (Freitas 2002), Bayesian methods (Cheeseman & Stutz 1996), log-linear models and other statistical methods (Christensen 1997), instance-based learning algorithms (Aha, Kibler & Albert 1991) (such as nearest neighbor classifiers described in (Dasarathy 1991)) and decision trees or classification trees (Breiman et al. 1984, Quinlan 1986, Quinlan 1993, RuleQuest 2000).

Table 2.2: A small training set: Saturday Morning activity

Class P (Play)				Class N (Don't Play)			
Outlook	Temperature	Humidity	Windy	Outlook	Temperature	Humidity	Windy
sunny	mild	normal	true	sunny	hot	high	true
sunny	cool	normal	false	sunny	hot	high	false
overcast	mild	high	true	sunny	mild	high	false
overcast	hot	high	false	rain	mild	high	true
overcast	cool	normal	true	rain	cool	normal	true
overcast	hot	normal	false				
rain	mild	high	false				
rain	cool	normal	false				
rain	mild	normal	false				

2.3.2 Decision Tree Based Classifiers

Decision tree induction algorithms have long been popular in machine learning, statistics, and other disciplines for solving classification and related tasks. This is due in part to their robustness and execution speed, and to the fact that explicit concept descriptions are produced for users to interpret.

A decision tree takes as input an object described by a set of attributes and returns a "decision" – the predicted output value for the input. A decision tree can be used to classify a test (or query) case as follows. Given a query t to classify, a tree is traversed along a path from its root to a leaf node, whose class label is assigned to t. Each internal node contains a *test* that determines which of its subtrees is traversed for t. A test typically evaluates a *feature* used to describe cases, or a (e.g., Boolean or linear) combination of features.

Each path from the root to a leaf represents a *rule* for inferring class membership. The conjunction of tests on the path is the rules's premise (left-hand side or antecedent) and the class label of the leaf is its conclusion (right-hand side or consequent). A rule provides an *explanation* for a query's classification.

Learning Decision Trees

Decision trees are constructed by finding ways to separate the data into two or more groups. We then separate each of these groups in turn, until we have small groups of examples left. Decision tree algorithms are designed to find the best questions to ask so that most or all of the examples in each group belong to one class. A generic decision tree induction algorithm is given in Algorithm 2.5.

Algorithm 2.5: The decision tree learning algorithm

```
function decision-tree-learning(D, A, default )
input  : a training dataset D described by a set of attributes A and the default
         value for the goal predicate default
output: a decision tree
1  if D is empty then return default ;
2  else if all the instances in D have the same classification then
       return the classification
3  else if A is empty then
       return majority-value(D )
4  else
5      best ← choose-attribute(A, D );
6      tree ← a new decision tree with root test best ;
7      m ← majority-value(D );
8      foreach value v_i of best do
9          D_i ← {elements of D with best = v_i};
10         subtree ← decision-tree-learning(D_i, A − best, m);
11         add a branch to tree with label v_i and subtree subtree ;
       end
12     return tree ;
   end
```

The function `choose-attribute` in Algorithm 2.5 attempts to pick "fairly good" (not "really useless") attributes that go as far as possible toward providing an exact clas-

sification of the examples – a perfect attribute will divide the examples into sets that are all positive or all negative. This greedy approach is to minimize the depth of the final tree. The expected amount of *information* provided by an attribute can be used as a measure of the goodness of the attribute. In general, if the possible answer v_i has probabilities $P(v_i)$, then the information content I of the actual answer is given by

$$I(P(v_1), \cdots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) \log_2 P(v_i).$$

An estimate of the probabilities of the possible answers before any of attributes have been tested is given by the proportions of positive and negative examples in the training dataset. Suppose the training dataset contain p positive examples and n negative examples. Then an estimate of the information contained in a correct answer is

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

Any attribute A divides the training dataset D into subsets $D_1, \cdots, D_v$ according to their values for A, where A can have v distinct values. Each subset D_i has p_i positive examples and n_i negative examples. So if we go along that branch, we will need an additional $I(p_i/(p_i+n_i), n_i/(p_i+n_i))$ bits of information to answer the question. A randomly chosen example from the training dataset has the ith value for A with probability $(p_i+n_i)/(p+n)$. Therefore, on average, after testing the attribute A, we will still need

$$\sum_{i=1}^{v} \frac{p_i+n_i}{p+n} \, I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$$

bits of information to classify the example. The *information gain* from the attribute test is the difference between the original information requirement and the new requirement:

$$Gain(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - \sum_{i=1}^{v} \frac{p_i+n_i}{p+n} \, I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}).$$

The function `choose-attribute` in Algorithm 2.5 is implemented to choose the attribute with the largest gain.

The *information* mentioned above can be also measured by *Gini measure*, which is defined as

$$Gini(P(v_1), \cdots, P(v_n)) = 1 - \sum_{i=1}^{n} -P(v_i)^2.$$

Several Decision Tree Classifiers

A number of algorithms for inducing decision trees have been proposed over the years, such as CART (Breiman et al. 1984), ID3 (Quinlan 1986), C4.5 (Quinlan 1993), SLIQ (Mehta, Agrawal & Rissanen 1996), SPRINT (Shafer, Agrawal & Mehta 1996), CLOUDS (Alsabti, Ranka & Singh 1998), BOAT (Gehrke, Ganti, Ramakrishnan & Loh 1999), PUBLIC (Rastogi & Shim 2000), RainForest (Gehrke, Ramakrishnan & Ganti 2000) and so on.

The decision tree classifier SLIQ (Mehta et al. 1996) was designed for large databases, but uses an in-memory data structure that grows linearly with the number of tuples in the training database. This limiting data structure was eliminated by SPRINT, a scalable data access method that removes all relationships between main memory and size of the dataset (Shafer et al. 1996). CLOUDS is a decision tree classifier proposed in (Alsabti et al. 1998), which first samples the splitting points for numeric attributes and then uses an estimation step to narrow the search space of the best split. Their experimental results using a number of real and synthetic datasets show that CLOUDS reduces computation and I/O complexity substantially compared to state of the art classifiers, while maintaining the quality of the generated trees in terms of accuracy and tree size. BOAT (Gehrke et al. 1999) offers performance improvement by an optimistic approach to tree construction in which an initial tree is constructed using a small subset of the data and the tree is further refined to arrive at the final tree. It was claimed in (Gehrke et al. 1999) that BOAT is the first scalable algorithm with the ability to incrementally update a decision tree with respect to both insertions and deletions over the dataset. PUBLIC (Rastogi & Shim 2000) is an MDL[2]-based pruning algorithm for binary trees that is interleaved with the tree growth phase. In the RainForest framework (Gehrke et al. 2000), a generic scalable data access method was proposed for classification tree construction that separates the scalability aspects of algorithms for constructing a tree from the central features that determine the quality of the tree. The framework is easy to instantiate with most split selection methods from the literature. Recent work in (Garofalakis, Hyun, Rastogi & Shim 2000) permits users to specify con-

[2]MDL is the abbreviation for Minimum Description Length.

straints on tree size or accuracy, and then builds the "best" tree (both easy to understand and having good accuracy) that satisfies the constraints. By pushing the constraints into the building phase of classifiers, and pruning early tree nodes that cannot possibly satisfy the constraints, significant performance speedups and reductions in the number of nodes expanded can be achieved, as opposed to applying the constraints after the entire tree is built.

Preventing Decision Tree Classifiers from Overfitting

Decision trees can become cumbersomely large for reasons such as noise. Noise causes *overfitting*, where trees model both the target concept and the inherent noise. Overly large trees can become *fragmented*, having many leaves with only a few cases per leaf. These leaf nodes and the tree paths corresponding to them are sometimes referred to as *small disjuncts* (Quinlan 1991), regions of the problem space with low frequencies of occurrence. There have been a number of approaches to simplify decision trees to produce simpler, more comprehensible trees (or data structures derived from trees) with good classification accuracy. Please see (Breslow & Aha 1997) for an excellent survey of simplifying decision trees. For example, some algorithms perform a pruning phase after the building phase in which nodes are iteratively pruned to prevent overfitting of the training data and to obtain a tree with higher accuracy. Popular pruning strategies include MDL pruning (Quinlan & Rivest 1989, Fayyad & Irani 1993, Mehta, Rissanen & Agrawal 1995), cost-complexity pruning (Quinlan 1987), and pessimistic pruning (Quinlan 1987).

Advantages of Decision Trees

Decision trees are attractive in data mining. This is due to the following reasons.

- First, the resulting classification model is easy to assimilate by humans due to their intuitive representation (Breiman et al. 1984).

- Second, decision trees do not require any parameter setting from the user and thus are especially suited for exploratory knowledge discovery.

- Third, decision trees can be constructed relatively fast compared to other models (Quinlan 1993, Mehta et al. 1996, Shafer et al. 1996, Alsabti et al. 1998, Gehrke et al. 1999, Rastogi & Shim 2000, Gehrke et al. 2000).

- Last, the accuracy of decision trees is comparable or superior to other classification models.

2.3.3 Bayesian Methods

Bayes Theorem

Let T be an instances whose class label is unknown. Let H be some hypothesis, such as that T belongs to a specified class C. For classification problems, we want to determine $P(H|T)$, the probability that the hypothesis H holds given the observed instance T. $P(H|T)$ is the *posterior probability* of H conditioned on T; while $P(H)$ is the *prior probability* of H. The posterior probability, $P(H|T)$, is based on more information (such as background knowledge) than the prior probability, $P(H)$, which is independent of T. Similarly, $P(T|H)$ is the *posterior probability* of T conditioned on H; and $P(T)$ is the *prior probability* of T.

Bayes theorem is

$$P(H|T) = \frac{P(T|H)P(H)}{P(T)}. \tag{2.1}$$

$P(T)$, $P(H)$ and $P(T|H)$ can be estimated from the training data. $P(H|T)$ can be calculated by the above Bayes theorem (Equation 2.1) from $P(T)$, $P(H)$ and $P(T|H)$.

Bayesian Belief Networks

A Bayesian Belief Network is a graph structure that captures the probabilistic dependencies (and independencies) among a set of random variables. Each node in the graph has a associated probability distribution. From these individual distributions, the joint distribution of the observed data can be computed. Bayesian Belief Networks provide a graphical model of causal relationships, on which learning can be performed. These networks are also known as *belief networks*, *Bayesian networks*, and *probabilistic networks*.

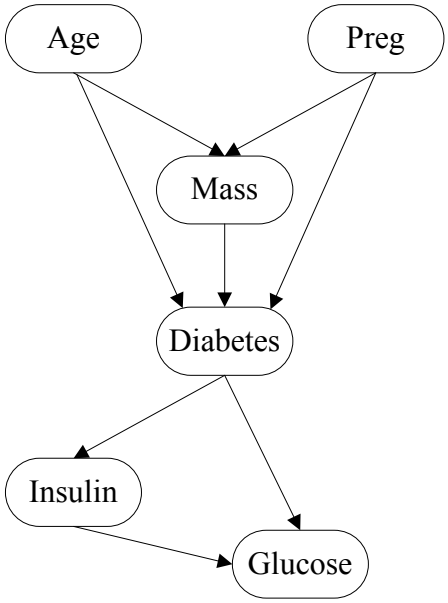


Figure 2.2: A Bayesian Belief Network for diabetes diagnosis

Figure 2.2 is an example of a Bayesian Belief Network that might be used to diagnose diabetes, adapted from (Dietterich 1998). Given a set of training instances, a learning algorithm can compute the actual probability values based on the observed data. For example, $P(\texttt{Age=26-50})$ is the fraction of training instances that have `Age=26-50`. $P(\texttt{Mass=51-100kg} \mid \texttt{Age=0-25})$ is the fraction training instances with `Mass=51-100kg` that have `Age=26-50`.

The process of learning Bayesian Belief Networks consists of three steps:

1. choose the graphical structure;

2. specify the form of the probability distribution at each node in the graph;

3. fit the parameters of those probability distributions to the training data.

Usually, the first two steps are performed by a human user (although there are some works trying to automate these two steps), while the third step is performed by a learning algorithm on a computer. More details of the above process can be found in (Mitchell 1997).

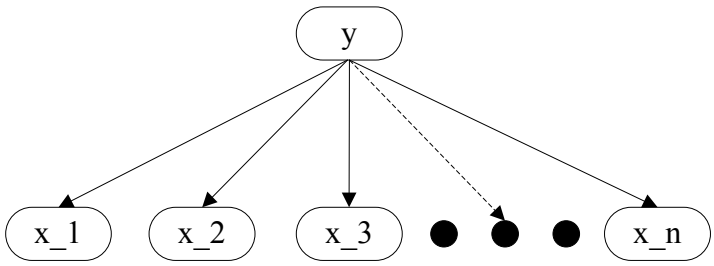


Figure 2.3: A probabilistic network for the Naive Bayesian Classifier

The Naive Bayesian (NB) Classifier

In the Naive Bayesian (NB) approach, the training instances are assumed to be produced by the probabilistic network shown in Figure 2.3, where the class variable is y, and the features (attributes) are $x_1, x_2, \cdots, x_n$. Naive Bayes is one of the most efficient and effective inductive learning algorithms for machine learning and data mining. It is time efficient, for its time complexity is only linear of the training data. It is space efficient, because, after discretization, it builds up a frequency table in size of the product of the number of attributes, number of class labels, and the number of values per attribute. It does not need to store the training dataset in memory when it builds the frequency table, but just scans the dataset once from the disk.

Example 2.3 Suppose a Naive Bayes classifier has been trained using the dataset in Table 2.2 and we wish to use it to determine whether the following day is suitable for a game of golf: the outlook is sunny, the temperature is hot, the humidity is normal, and there is no wind.

An example frequency table for attribute "Outlook" is like:

Outlook	Play	Don't Play
sunny	2	3
overcast	4	0
rain	3	2

Frequency tables for other attributes are created in a similar way.

p(Don't Play | sunny, hot, normal, false) = p(Don't Play) $\times$ p(sunny | Don't Play) $\times$ p(hot | Don't Play) $\times$ p(normal | Don't Play) $\times$ p(false | Don't Play) = $\frac{5}{14} \times \frac{3}{5} \times \frac{2}{5} \times \frac{1}{5} \times \frac{2}{5} =$ 0.0069.

$p(\text{Play} \mid \text{sunny, hot, normal, false}) = p(\text{Play}) \times p(\text{sunny} \mid \text{Play}) \times p(\text{hot} \mid \text{Play}) \times p(\text{normal} \mid \text{Play}) \times p(\text{false} \mid \text{Play}) = \frac{9}{14} \times \frac{2}{9} \times \frac{2}{9} \times \frac{6}{9} \times \frac{6}{9} = 0.0141$.

$p(\text{Don't Play} \mid \text{sunny, hot, normal, false}) < p(\text{Play} \mid \text{sunny, hot, normal, false})$.

So this day is good for playing games. □

Despite the fact that the NB model is very simple, it performs surprisingly well. It is shown that NB is very robust to violations of the assumption that the attributes are generated independently (Domingos & Pazzani 1996). However, by relaxing the strong independence assumptions of NB, researchers have developed several extensions that achieve higher classification accuracy than NB (Kononenko 1991, Kohavi 1996, Friedman, Geiger & Goldszmidt 1997, Webb & Pazzani 1998, Zheng & Webb 2000, Meretakis & Wuthrich 1999). We will discuss these NB extensions in Chapter 6. Among them, Large Bayes (LB) (Meretakis & Wuthrich 1999), a recently proposed extension of NB using long itemsets to approximate probabilities, is closely related to our Bayesian Classification by Emerging Patterns (BCEP). LB is further discussed and compared with BCEP in chapter 6.

2.3.4 Association Rules Based Classifiers

Classification Based on Association (CBA)

CBA (Liu, Hsu & Ma 1998) adopts an Apriori-like candidate set generation-and-test method to mine a complete set of association rules satisfying both minimum support and minimum confidence thresholds, and then uses database coverage concept to select a small set of rules to form a classifier. CBA defines association rules of the form $X \Rightarrow y$, where X is a set of attribute value pairs and y is a class label, as *Class Association Rules* (CARs). Because the total number of CARs generated is very large, CBA uses a heuristic pruning method based on a defined rule rank and database coverage to obtain a small set of high-rank rules that cover all tuples in the database. To classify a test instance, CBA searches the selected rule set starting from the highest rank and finds the first rule that matches the test. It is reported that CBA outperforms C4.5 in terms of average accuracy on 26 UCI datasets.

Association-Based Decision Tree (ADT)

ADT (Wang, Zhou & He 2000) first generates a complete set of association rules satisfying only minimum confidence threshold, and then prunes overfitting rules on an accuracy-driven basis. ADT abandons the ad-hoc minimum support and employs association rules satisfying only the minimum confidence, called *confident rules*, to build a classifier. To find confident rules without examining all rules, it proposes a confidence-based pruning technique that exploits a certain monotonicity of confidence. During the pruning process, ADT first removes a specific rule if there exists at least one, more general rule with a higher rank; it then further filters rules based on the estimated accuracy. ADT stores the remaining rules in a so called *Association-Based Decision Tree* (ADT) and uses the *pessimistic estimation method* as used in C4.5 to estimate the error. ADT works in a similar way as CBA to classify an unknown instance. It is shown that ADT outperforms C4.5 in terms of average accuracy on 21 UCI datasets.

Classification based on Multiple Association Rules (CMAR)

CMAR (Li, Han & Pei 2001) extends the FP-tree by adding class distribution information to each tree node and uses the FP-Growth method to generate a complete association rule set. It stores those rules in a compressed data structure, called the *Compressed Rule tree* (CR-tree) to facilitate post-processing. It prunes specialized rules with low confidence and then prunes the remaining rules based on correlation analysis and database coverage. When classifying a test instance, instead of relying on a single rule to make a decision, as in CBA and ADT, it takes all matched rules into consideration to predict its class label. The motivation here is that all matched rules collectively can provide a global view about the test. To overcome the difficulty where those matched rules do not agree on the class label, CMAR uses a weighted-χ measurement to evaluate *multiple* matches rules. It is found that compared to C4.5 and CBA, CMAR improves the accuracy on average for more than twenty UCI datasets; CMAR also consumes less memory and has better scalability than CBA.

Classification based on Predictive Association Rules (CPAR)

CPAR (Yin & Han 2003) inherits the basic idea of FOIL in rule generation and integrates the features of associative classification in predictive rule analysis. On one hand, CPAR generates and tests more rules than traditional rule-based classifiers. When generating rules, instead of selecting only the best literal, CPAR selects all the close-to-the-best literals to avoid missing important rules. On the other hand, CPAR generates a much smaller set of high-quality predictive rules directly from the dataset than association classification. It avoids producing redundant rules by considering the set of "already-generated" rules. Lastly, to avoid overfitting, CPAR uses "expected accuracy" to evaluate each rule and uses the *best k rules* in prediction. The experimental results and performance study show that CPAR is much more time-efficient in both rule generation and prediction and achieves as high accuracy as association classification.

Discussions

Although these works show the success of integrating classification and association rule discovery, there are crucial differences between the association-rule discovery and the classification task, and these differences involve the key notion of prediction. The classification task can be regarded as an ill-defined, non-deterministic task, in the sense that in general, using only the training data, one cannot be sure that a discovered classification rule will have a high predictive accuracy on unseen data in the training phase[3]. Note the following well-known facts about classification: a classification algorithm must have an *inductive bias*; any bias has a domain-dependent effectiveness. Therefore, the performance of a classification algorithm strongly depends on the application domain. In contrast, the association task, which is stated in the well-known support-confidence framework in (Agrawal et al. 1993b), can be considered well-defined, deterministic, relatively simple task, because the goal is to discover *all* association rules having support and confidence greater than user-specified thresholds.

[3]There are, however, theoretical bounds on test set error for some classifiers under certain conditions.

2.3.5 Neural Networks

Neural network (Bishop & Bishop 1995, Ripley 1996) uses a model of biological systems to perform classification. Neural networks are characterized by highly connected networks which are trained on a set of data in the hope that the network will correctly classify future examples. Neural network learning methods provide a robust approach to approximating real-valued, discrete-valued, and vector-valued target functions.

A neuron is a cell in the brain whose principal function is the collection, processing, and dissemination of electrical signals. Artificial Neural Networks (ANN) are mathematical models that are inspired by the architecture of the human brain, whose information-processing capacity is thought to emerge primarily from *networks* of such neurons. The field of ANN also goes by other names, such as connectionism, parallel distributed processing and neuron-computing. A Neural Network is typically composed of a large number of highly interconnected processing elements (neurons), working in unison to solve specific problems. Each neuron is linked to certain neighbors with varying coefficients (weights) of connectivity that represent the strengths of these connections. Every neuron performs a certain computation, such as calculating a weighted sum of its input connections, and sends the result as an output signal to neighbor neurons. Learning is accomplished by adjusting these weights to cause the overall network to output appropriate results.

The fact that the "knowledge" of a neural network has a distributed representation, spreading around a large number of connection weights, contributes to its robustness to resist noise. On the negative side, a neural network behaves like a "black box" whose output cannot be explained adequately, making it difficult to associate the network's weights with simple if-then rules.

2.3.6 Support Vector Machines

Support Vector Machines (SVMs) were introduced in the early 1990s and the topic on the entire family of kernel-based learning methods (KMs) has developed into a very active field of Machine Learning research. The main reason for interest in Support Vector and Kernel Methods is their flexibility and remarkable resistance to overfitting, their

simplicity and theoretical elegance, all appealing to practitioners as well as theoreticians. Their modular design (a general purpose learning module fitted with a problem specific kernel function) makes them simple to analyze and use. Consequently, they are very flexible tools, which have been applied to diverse fields, delivering state of the art performance in applications from analysis of DNA microarray data to text categorization, from handwritten digits recognition to protein homology detection.

SVMs combine two key ideas. The first is the concept of an *optimum margin classifier*. An optimum margin classifier is a linear classifier; it constructs a separating hyperplane which maximizes the distance to the training points. The important point is maximization of the margin, which turns the under-specified learning problem into an optimization problem (without local optima) and has been shown to give very good generalization performance. Margin maximization provides a useful trade-off with classification accuracy and it avoids overfitting of the training data (which leads to poor performance on test data). This makes SVMs well-suited to deal with learning tasks where the number of attributes is large with respect to the number of training examples. In general, the optimum margin hyperplane will be a linear combination of the input vectors; *support vectors* are those training instances which obtain a non-zero coefficient, i.e., the ones that lie closest to the separating hyperplane.

The second key concept underlying SVMs is a *kernel*. In its simplest form, a kernel is a function which calculates the dot product of two training vectors. Intuitively, this dot product expresses the similarity of the two training instances in terms of the given attributes. If we use feature transformation techniques to reformulate the input vectors in terms of new features and find a way to calculate dot products in this feature space, we can leave the linear classifier unaffected. Generally, the kernel can be thought of as a non-linear similarity measure and kernel functions are inner products in some feature space (potentially very complex).

Due to the similarity between the optimum margin classifier and the perceptron algorithm, SVMs are often seen as an extension of neural networks. However, SVMs offer a much more sophisticated mechanism to incorporate domain knowledge by means of the kernel. SVMs can also deal with non-numerical, symbolic data. The kernel methods can

use existing algorithms and need to be engineered to fit the underlying domain.

2.3.7 Evolutionary Computation

The evolutionary algorithms paradigm consists of stochastic search algorithms that are based on abstractions of the processes of Neo-Darwinian evolution. The basic idea is that each "individual" of an evolving population encodes a candidate solution (e.g. a predictive rule) to a given problem (e.g. classification). Each individual is evaluated by a fitness function (e.g. the predictive accuracy of the rule). Then these individuals evolve towards better and better individuals via operators based on natural selection, i.e. survival and reproduction of the fittest, and genetics, e.g. crossover and mutation operators. The crossover operator swaps genetic material between two individuals, while the mutation operator changes the value of a gene to a new random value. Both crossover and mutation are stochastic operators, applied with user-defined probabilities. Mutation happens in a much lower probability than crossover, but it is necessary to increase the genetic diversity of individuals in the population.

An important characteristic of evolutionary algorithms is that they perform a *global* search (Freitas 2002). Because they work with a population of candidate solutions rather than a single candidate solution at a time, and because they use stochastic operators to perform their search, the probability that they will get stuck in local maxima is reduced and the probability that they will find the global maxima is increased.

In KDD, evolutionary algorithms can be used for rule discovery (Freitas 2002). In contrast to the local, greedy search performed by often-used rule induction and decision tree algorithms, evolutionary algorithms tend to cope well with attribute interactions as a consequence of their global search. Evolutionary algorithms can also be used in attribute selection and as a wrapper to optimize parameters of several other kinds of KDD algorithms.

2.4 Onwards

We have presented our view of Knowledge Discovery in Databases (KDD), which integrates the views from researchers in Database, Machine Learning and Statistics. As

a relatively new type of KDD pattern, Emerging Patterns (EPs) (Dong & Li 1999) represents discriminating knowledge between two classes of data. We have described the frequent pattern mining problem in detail, because the basic definitions - such as `attributes`, `attribute values`, `items`, `itemsets`, `counts` and `supports` - serve as a background to our Emerging Pattern mining problem. New terms specific to Emerging Patterns will be defined upon those basic terms later. A number of approaches for mining frequent patterns have been reviewed, including Apriori-like algorithms (described in Section 2.2.2) and FP-growth (Han et al. 2000) (described in Section 2.2.3). It is worth mentioning that FP-growth uses the FP-tree data structure and pattern growth methodology, making it able to avoid the expensive candidate generation-and-test. The success of FP-growth offers the chance of looking at the Emerging Pattern mining problem from a new angle.

Classification is an important data mining problem. We have reviewed a great diversity of classification methods, including decision tree based classifiers, Bayesian classifiers, association rules based classifiers, neural networks, support vector machines and Evolutionary Computation. Later in this thesis, we will present our new approaches to build highly accurate and efficient classifiers, which provides new choices for people to use for real-world classification tasks.

Chapter 3

Problem Statement and Previous Work on Emerging Patterns

In this chapter, we formally state the problem of discovering Emerging Patterns and classification by Emerging Patterns. We first give the definitions and concepts that will be used in later chapters. We then review previous algorithms for mining Emerging Patterns and classification approaches based on Emerging Patterns. Lastly, we introduce several tools used throughout this thesis.

3.1 Emerging Patterns

In Chapter 2, we have seen that as a descriptive data mining task, the discovery of class comparison or discrimination information - i.e., the comparison between two or more collections of data - is an important theme in the field of data mining. The recently introduced concept of Emerging Patterns (EPs) (Dong & Li 1999), defined as multivariate features (i.e., itemsets) whose supports (or frequencies) change *significantly* from one class to another, are very useful as a means of discovering distinctions inherently present between different classes of data. For example, by using Emerging Patterns, in (Li & Wong 2002b, Li, Liu, Ng & Wong 2003), the authors identified good diagnostic genes or genes groups from gene expression data of the ALL/AML dataset (Golub, Slonim, Tamayo, Huard, Gaasen-

beek, Mesirov, Coller, Loh, Downing, Caligiuri, Bloomfield & Lander 1999) and the colon tumor dataset (Alon, Barkai, Notterman, Gish, Ybarra, Mack & Levine 1999); simple rules are also found underlying gene expression profiles of more than six subtypes of acute lymphoblastic leukemia (ALL) patients (Li, Liu, Downing & A. Yeoh 2003).

The new type of knowledge pattern, Emerging Patterns (EPs) (Dong & Li 1999) can also serve as a classification model. By aggregating the differentiating power of EPs/JEPs, the constructed classification systems (Li, Dong & Ramamohanarao 2001, Dong, Zhang, Wong & Li 1999) are usually more accurate than other existing state-of-the-art classifiers. EP-based classifiers are also successful in real life applications, such as classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling (Yeoh, Ross, Shurtleff, William, Patel, Mahfouz, Behm, Raimondi, Reilling, Patel, Cheng, Campana, Wilkins, Zhou, Li, Liu, Pui, Evans, Naeve, Wong & Downing 2002). The success is highlighted by the newly-developed, award-wining gene chip-based analysis (where EP is the core technology) that helps diagnose acute lymphoblastic leukaemia in children quickly, easily and cheaply.

3.1.1 Terminology

To help formally define Emerging Patterns, we first give some preliminary definitions. Suppose that a dataset $\mathcal{D}$ is defined upon a set of attributes $\{A_1, A_2, \cdots, A_n\}$. For each attribute A_i, there is a set of permitted values, called the domain of that attribute, denoted as $domain(A_i)$. Attributes can be either categorical or continuous. For a continuous attribute, we assume that its value range is discretized into intervals. For example, attribute *sex* is categorical and $domain(sex) = \{male, female\}$; attribute *age* is continuous, $domain(age) = [0, 150]$, and it can be discretized into intervals [0, 18], [18, 60] and [60, 150]. After discretization, $domain(age) = \{[0, 18], [18, 60], [60, 150]\}$. We call each (attribute, categorical-value) or (attribute, continuous-interval) pair an *item*. $(sex, male)$ and $(age, [18, 60])$ are two examples of items. By aggregating all the domain categorical-values and continuous-intervals across all attributes, we obtain the set of all items in $\mathcal{D}$, denoted as I, where $I = \{domain(A_1) \cup domain(A_2) \cup \cdots \cup domain(A_n)\}$. For convenience, we map all items from I including (attribute, categorical-value) and (attribute, continuous-interval)

pairs to consecutive positive integers, i.e., we use 1 to represent the first item in I, 2 to the second item, and so on. By doing this, the original dataset can be treated as a transaction database, which is commonly used in the mining of frequent patterns (Chapter 2 Section 2.2.1).

A set X of items is also called an itemset, which is defined as a subset of I. We say any instance S contains an itemset X, if $X \subseteq S$. The support of an itemset X in a dataset $\mathcal{D}$, $supp_D(X)$, is $count_D(X)/|\mathcal{D}|$, where $count_D(X)$ is the number of instances in $\mathcal{D}$ containing X.

3.1.2 Definitions

We first define the growth rate of an itemset with respect to two classes of data.

Definition 3.1 Given two different classes of datasets $\mathcal{D}_1$ and $\mathcal{D}_2$, the **growth rate** of an itemset X from $\mathcal{D}_1$ to $\mathcal{D}_2$ is defined as

$$GrowthRate(X) = GR(X) = \begin{cases} 0 & \text{if } supp_1(X) = 0 \text{ and } supp_2(X) = 0 \\ \infty & \text{if } supp_1(X) = 0 \text{ and } supp_2(X) > 0 \\ \frac{supp_2(X)}{supp_1(X)} & \text{otherwise} \end{cases}$$

Emerging Patterns are those itemsets with large growth rates from $\mathcal{D}_1$ to $\mathcal{D}_2$.

Definition 3.2 Given a growth rate threshold $\rho > 1$, an itemset X is said to be a **ρ-Emerging Pattern** (**ρ-EP** or simply **EP**) from a background dataset D_1 to a target dataset D_2 if GrowthRate(X)$\geqslant \rho$.

When D_1 is clear from the context, an EP X from D_1 to D_2 is simply called an EP of D_2 or an EP in D_2. The support of X in D_2, $supp_2(X)$, denoted as $supp(X)$, is called the support of the EP. The background dataset D_1 is also referred to as the *negative* class, and the target dataset D_2 as the *positive* class.

An EP with high support in its home class and low support in the contrasting class can be seen as a strong signal indicating the class of a test instance containing it. The strength of such a signal is expressed by its supports in both classes and its growth rate.

Definition 3.3 The **strength** of an EP X is defined as

$$strength(X) = \frac{GR(X)}{GR(X)+1} * supp(X)$$

A Jumping Emerging Patterns (JEP) is a special type of Emerging Pattern and also a special type of discriminant rule (Han, Cai & Cercone 1992, Han, Cai & Cercone 1993).

Definition 3.4 A **Jumping Emerging Pattern (JEP)** from a background dataset D_1 to a target dataset D_2 is defined as an Emerging Pattern from D_1 to D_2 with the growth rate of ∞.

Note that for a JEP X, $strength(X) = supp(X)$.

Example 3.1 Table 2.2 contains a training dataset for predicting whether the weather is good for some activity. If the minimum growth rate threshold is $\rho = 2$, we have some EPs as follows.

- Itemset $\{(Outlook, sunny)\}$ is an EP of Class N, with support 2/9 in Class P, 3/5 in Class N, and growth rate of 2.7.
- Itemset $\{(Outlook, sunny), (Windy, false)\}$ is a EP of Class N, with support 1/9 in Class P, 2/5 in Class N, and growth rate of 3.6.

It can be seen that longer EPs tend to have lower support but higher growth rates.

We also point out two interesting JEPs.

- Itemset $\{(Outlook, overcast)\}$ is a JEP of Class P, with support 4/9 in Class P and 0 in Class N.
- Itemset $\{(Outlook, sunny)$,$(Temp, mild)$, $(Humidity, normal)$, $(Windy, true)\}$, which is just the first instance of Class P, is a JEP of Class P with support 1/9 in Class P and 0 in Class N.

It can be seen that the second JEP (appearing once in Class P but zero time in Class N) is not useful for classification, especially when there is much noise present in the data. □

Table 3.1: A simple gene expression dataset

Tissue ID	Cell type	gene_1	gene_2	gene_3	gene_4
1	Normal	0.10	1.20	-0.70	3.25
2	Normal	0.20	1.10	-0.83	4.37
3	Normal	0.30	1.30	-0.75	5.21
4	Cancerous	0.40	1.40	-1.21	0.41
5	Cancerous	0.50	1.00	-0.78	0.75
6	Cancerous	0.60	1.10	-0.32	0.82

Example 3.2 Table 3.1 shows a small, hypothetical dataset taken from (Li & Wong 2002b) containing gene expression data, which records expression levels of genes under specific experimental conditions. There are 6 tissues samples in total: 3 normal and 3 cancerous tissues. Each tissue sample is described by the 4 genes (namely, gene_1, gene_2, gene_3 and gene_4).

We call gene_j@$[l, r]$ an *item*, meaning the values of gene_j is limited inclusively between l and r. We point out the following EPs/JEPs.

- The pattern {gene_1@[0.1, 0.3]} is a JEP as it has a frequency of 100% in the sub-dataset with normal cells but 0% with cancerous cells.
- The pattern {gene_1@[0.4, 0.6], gene_4@[0.41, 0.82]} is a JEP of cancerous cells with a support of 100% as it has a frequency of 0% in the sub-dataset with normal cells.
- The pattern {gene_2@[1.2, 1.4]} is an EP from cancerous to normal cells with a growth rate of 2 (its support in normal is 2/3 and its support in cancerous is 1/3).

Here an EP between normal and cancerous tissues represents a group of genes that have certain ranges of expression levels *frequently* in one type of tissue but *less frequently* in another. □

3.1.3 The Border Representation for Emerging Patterns

We first give some preliminary definitions.

Definition 3.5 Given two itemsets I_1 and I_2, I_1 is **more general** than I_2 if $I_1 \subset I_2$; it is also said that I_2 is **more specific** than I_1.

It can be seen that I_1 covers more instances than I_2 when I_1 is *more general* than I_2.

Definition 3.6 Given a collection S of itemsets, an itemset $X \in S$ is said to be **maximal** in S if there is no proper superset of X in S. Similarly, an itemset $Y \in S$ is said to be **minimal** in S if there is no proper subset of Y in S.

It is easy to see that the minimal itemsets are the most general in the collection. Similar to the most general rules used for classification, minimal EPs are more useful than non-minimal EPs due to their larger coverage on the training data.

Definition 3.7 A collection S of sets is said to be a **convex space**, if

$$\forall X, Y, Z \ (X \subseteq Y \subseteq Z) \wedge (X, Z \in S) \Longrightarrow Y \in S.$$

If a collection is a convex space, we say that it holds convexity or it is interval closed. If X and Z are in collections S of sets that are interval closed and Y is a set such that $X \subseteq Y \subseteq Z$, then Y is in S.

A collection S of sets is called an *anti-chain* if X and Y are incomparable sets (i.e., neither $X \subseteq Y$ nor $Y \subseteq X$ is true) for all $X, Y \in S$.

Definition 3.8 A **border**, denoted as $\langle L, R \rangle$, is defined as an ordered pair of two bounds L and R such that L and R are two anti-chain collections of sets satisfying

- $\forall X \in L$, $\exists Y \in R$ such that $X \subseteq Y$,
- $\forall Y \in R$, $\exists X \in L$ such that $X \subseteq Y$

In this work, bounds are sets of itemsets. Semantically, the border $\langle L, R \rangle$ represents a collection which contains itemsets Z such that $\exists X \in L$ and $\exists Y \in R$, $X \subseteq Z \subseteq Y$. In (Li, Ramamohanarao & Dong 2000), it is proved that the space of all JEPs with respect to two classes of data is a convex space, which has a unique border $\langle L, R \rangle$, where L is the collection of the most general sets in the space and R is the collection of the most specific sets in the space.

3.1.4 Concepts Related to Emerging Patterns

Version Space

Given a set of positive and a set of negative training instances, a *version space* is the set of all *generalizations* that each match every positive instance and no negative instance in the training set (Mitchell 1997). Similarly, *EP spaces* can be defined as the set of all EPs with respect to both sets of positive and negative training instances. Both EP spaces and version spaces are convex spaces which can be concisely represented (Li et al. 2000). However, the consistency restrictions with the training data are significantly different between EP spaces and version spaces. A JEP space is the set of all patterns that each match one or more (not necessarily every) positive instances and no negative instance in the training data. A general EP space relaxes the restrictions further: it removes the requirement of matching no negative instance; instead, it uses a finite growth-rate threshold to regulate the frequencies on both sets.

Discriminant Rules

A discrimination rule is an assertion that discriminates a concept of the class being learned (the target class) from other classes (called contrasting classes) (Han et al. 1992, Han et al. 1993). An attribute-oriented induction approach is used to extract discrimination rules by generalizing the data in both the target class and the contrasting class synchronously and excluding properties that overlap in both classes in the final generalized rules.

There are important differences between discrimination rules and Emerging Patters.

- Emerging Patters must satisfy a growth rate threshold, but discrimination rules do not have such constraint.

- Discrimination rules are usually represented at high abstraction levels in the concept hierarchy. For example, a student's status can be "freshman", "sophomore", "junior" and "senior" at low level; the high level concept of the attribute can be "undergraduate", which is more general than the previous four status. In this case, discrimination

rules are concerned with "undergraduate", but not "freshman" or "sophomore" etc. In contrast, Emerging Patters aim to find distinguishing properties at low conceptual levels, which may provide new insights into the problem, e.g., the differences between "freshman" and "sophomore", although both of them belong to the same high level concept "undergraduate".

- We note that a JEP can be regarded as a special discrimination rule at some high conceptual level.

Contrast Sets

Contrast Sets are conjunctions of attributes and values that differ meaningfully in their distribution across groups (Bay & Pazzani 2001). Formally, a contrast set (`cset`) is a conjunction of attribute-value pairs defined on groups $G_1, \cdots, G_n$, where

$$\exists ij P(cset = True|G_i) \neq P(cset = True|G_j)$$

$$\max_{ij} |support(cset, G_i) - support(cset, G_j)| \geqslant \delta$$

and δ is a user defined threshold called the *minimum support difference*.

Although both contrast sets and Emerging Patterns are intended for detecting differences between contrasting groups from observational multivariate data, there are important differences between the two concepts.

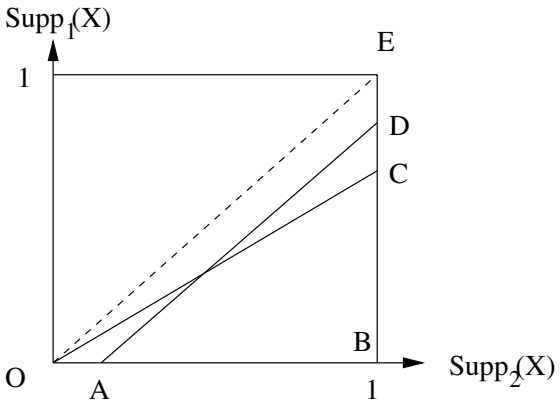
- More than two groups (classes) of data are involved in contrast sets, while Emerging Patterns are defined on two classes of data.
- Contrast sets measure absolute differences of supports among groups, but Emerging Patterns consider the supports in both classes and the growth rate.
- The above two differences lead to quite different algorithms for mining them, using different data structures, different search strategies and different heuristics.

3.1.5 The Landscape of Emerging Patterns

Figure 3.1 illustrates the support plane for Emerging Patterns. JEPs occupy the x-axis from O to B; EPs occupy the triangle $\triangle OBC$, where the slope of the line OC equals

the reciprocal of the growth rate, i.e., $\tan \angle COB = \frac{1}{\rho}$, where $\rho > 1$. Suppose we consider contrast sets with respect to two groups. The contrast sets occupy the triangle $\triangle ABD$, where the line AD parallels the diagonal of the square box, and the length of OA equals δ.

The number of Emerging Patterns is extremely large. Recall that ρ-EPs are only concerned with growth rates, not supports. It is well recognized that when supports are low, there are an exponential number of frequent patterns. A pattern with low supports in both classes, can still satisfy the growth-rate threshold. Our aim is to find a small number of useful or interesting EPs from the huge space of patterns. Therefore, in this thesis, we propose various subtypes of Emerging Patterns that can be defined by the incorporation of different constraints, in order to capture the most important and useful Emerging Patterns. Examples include Essential Jumping Emerging Patterns (EJEPs) discussed in Chapter 4, and Chi Emerging Patterns (Chi EPs) discussed in Chapter 5.



ρ is the minimum growth rate threshold for Emerging Patterns, and

$$\tan \angle COB = \frac{1}{\rho}, (\rho > 1)$$

δ is the minimum support difference threshold for Contrast Set, and

$$|OA| = \delta$$

Figure 3.1: The support plane: Emerging Patterns vs. Contrast Set

3.2 Algorithms for Mining Emerging Patterns

3.2.1 Border-based Approach

To efficiently discover the border of the JEP space with respect to a positive dataset and a negative dataset, border manipulation algorithms are proposed (Li et al. 2000). There are three algorithms: Horizon-Miner (Algorithm 3.1), Border-Diff (Algorithm 3.2) and JEP-Producer (Algorithm 3.3). The Horizon-Miner is used to derive the horizontal border of both the positive and the negative dataset. The basic idea behind Horizon-Miner is to extract the maximal itemsets from all instances in $\mathcal{D}$. With these two discovered horizontal borders as arguments, JEP-Producer outputs one border as a concise representation of the JEP space. Border-Diff is a core subroutine in JEP-Producer and it aims to derive the differential between a pair of borders of a special form: $\langle\{\emptyset\}, \{U\}\rangle$, and $\langle\{\emptyset\}, R_1\rangle$, where $\langle\{\emptyset\}, R_1\rangle = \langle\{\emptyset\}, \{S_1, S_2, \cdots, S_k\}\rangle$. Note that $\{U\}$ is a singleton set containing U only.

Borders are a powerful representation mechanism for large collections. The border-based algorithms achieve high efficiency by manipulating only the itemsets in the borders and avoiding the tedious process of enumerating all the individual JEPs.

Algorithm 3.1: Horizon-Miner (Li, Dong, Ramamohanarao 2001)

input : a dataset $\mathcal{D}$

output: the horizontal border of $\mathcal{D}$

/* `assume that the instances are arbitrarily ordered into` $T_1, T_2, \cdots, T_n$ */

1 RightBound $\leftarrow \{T_1\}$;

2 **for** *i from 2 to n* **do**

3 | **if** T_i *is not a subset of any element in* RightBound **then**

4 | | add T_i into RightBound ;

5 | | remove all S in RightBound such that $S \subset T_i$;

| **end**

end

6 return $\langle$ {$\emptyset$}, RightBound $\rangle$;

Algorithm 3.2: Border-Diff (Dong & Li 1999)

input : a pair of borders $\langle\{\emptyset\}, \{U\}\rangle$, and $\langle\{\emptyset\}, \{S_1, S_2, \cdots, S_k\}\rangle$

output: $\langle L, \{U\}\rangle$, such that $\langle L, \{U\}\rangle = [\langle\{\emptyset\}, \{U\}\rangle] - [\langle\{\emptyset\}, R_1\rangle]$

initialize L to $\{\{x\}|x \in (U - S_i)\}$;

for *i from 2 to k* **do**

  $L \leftarrow \{X \cup \{x\}|X \in L, x \in (U - S_i)\}$;

  remove all Y in L that are not minimal;

end

return $\langle L, \{U\}\rangle$;

Algorithm 3.3: JEP-Producer (Li, Dong, Ramamohanarao 2001)

input : the horizontal border of $\mathcal{D}_1$, $\langle\{\emptyset\}, \{A_1, A_2, \cdots, A_{k_1}\}\rangle$, and the horizontal border of $\mathcal{D}_2$, $\langle\{\emptyset\}, \{B_1, B_2, \cdots, B_{k_2}\}\rangle$

output: $\langle L, R\rangle$ such that $[L, R] = [\{\emptyset\}, \{A_1, A_2, \cdots, A_{k_1}\}] - [\{\emptyset\}, \{B_1, B_2, \cdots, B_{k_2}\}]$

$L \leftarrow \{\}, R \leftarrow \{\}$;

for *j from 1 to* k_1 **do**

  if *some* B_{k_i} *is a super set of* A_j **then continue**;

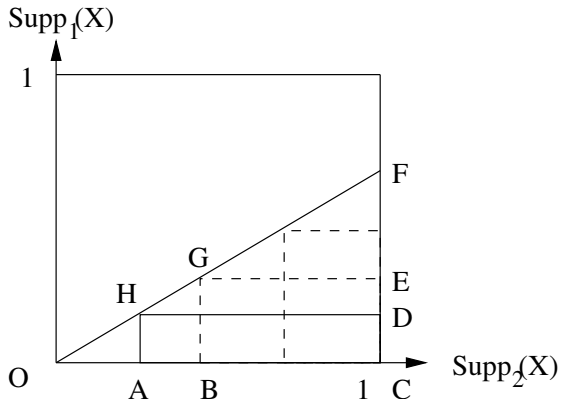
  border = `Border-Diff`$(\langle\{\emptyset\}, \{A_j\}\rangle, \langle\{\emptyset\}, \{B_1, B_2, \cdots, B_{k_2}\}\rangle)$;

  $R = R \cup$ right bound of border ;

  $L = L \cup$ left bound of border ;

end

return $\langle L, R\rangle$;



$\tan \angle FOC = \frac{1}{\rho}, |AH| = \xi_1, |OA| = \xi_2$

Figure 3.2: Using a series of rectangles to cover the whole area of $\triangle FOC$

These border-based algorithms can also be used to discover finite-growth-rate EPs, not just JEPs. Given a minimum growth rate threshold $\rho > 1$, suppose we want to find EPs from $\mathcal{D}_1$ to $\mathcal{D}_2$ with growth rates more than ρ. We first fix the minimum support threshold ξ_1 for $\mathcal{D}_1$, and use a border-discovery algorithm such as Max-Miner (Bayardo Jr. 1998) to find the border for $\mathcal{D}_1$. We then obtain the minimum support threshold ξ_2 for $\mathcal{D}_2$, where $\xi_2 = \xi_1 \times \rho$; we again use the border-discovery algorithm to find the border for $\mathcal{D}_2$, which essentially consists of the maximal frequent patterns with respect to ξ_2. After both borders for $\mathcal{D}_1$ and $\mathcal{D}_2$ are available, the border of Emerging Patterns can be quickly derived by the border differential procedure, Border-Diff (Algorithm 3.2). In Figure 3.2, suppose $\tan \angle FOC = \frac{1}{\rho}, |AH| = \xi_1, |OA| = \xi_2$. Note that the discovered EPs occupy the rectangle $ACDH$, which is part of the whole EP space $\triangle FOC$. To discover the complete set of EPs satisfying growth rate ρ, we need to use a series of rectangles (e.g., rectangle $ACDH$ and other rectangles in dashed lines shown in Figure 3.2) to cover the whole area of $\triangle FOC$. The difficulty is that when the minimum support threshold is very low, finding maximal frequent patterns to derive large borders is a very difficult task.

3.2.2 Constraint-based Approach

The constraint-based EP miner (ConsEPMiner) (Zhang et al. 2000a) utilizes two types of constraints for effectively pruning the search space: external constraints and internal constraints. The external constraints include user-given minimums on support, growth rate and growth-rate improvement – they are used to confine the resulting EP set; the internal constraints including the same subset support, top growth rate and the same origin are derived from the properties of data – they are solely for pruning the search space and saving computation. ConsEPMiner can efficiently mine *all* EPs satisfying those constraints at low support on large, high-dimensional datasets.

Algorithm 3.4 gives the sketch of ConsEPMiner. It uses the breadth-first search strategy over the SE-tree (Rymon 1993): given a level of surviving groups G, groups at the next level are generated by expanding sruviving groups in G (`Prune-Gen-Next`). Pruning occurs in three stages, namely, when generating groups at a new level – stage 1 (line 12), when groups are processed partially – stage 2 (line 5), and when groups are processed completely – stage 3 (line 11). ConsEPMiner always tries to first prune group tails with less costly constraints; further pruning is applied by computing the bounds of groups. Only groups with the possibility of deriving valid EPs are kept; and only tail items that can produce promising child groups are kept. This can reduce the dataset scanning time considerably. EPs satisfying the given constraints are accumulated in E (line 9-10). The algorithm terminates when no more groups are available.

The superior performance of ConsEPMiner derives from its direct application of both external and inherent constraints at the mining stage. ConsEPMiner is orders of magnitude faster than Apriori-based approaches that exploit only the support constraint for pruning (Zhang et al. 2000a).

3.2.3 Other Approaches

Concurrent to my PhD work, there has been recent progress on Emerging Patterns from different directions and emphases (Bailey, Manoukian & Ramamohanarao 2002, Bailey, Manoukian & Ramamohanarao 2003a, Bailey, Manoukian & Ramamohanarao 2003b).

Algorithm 3.4: ConsEPMiner (Zhang, Dong, Ramamohanarao 2000)

input : a backgroud dataset D_1 and a target dataset D_2

output: the set of EPs E from D_1 to D_2

/* τ_s, τ_g `and` τ_i `are the thresholds on support, growth rate and growth-rate improvement` */

```
1  E = ∅;
2  G = Generate-Initial-Groups(D1, D2);
3  while G ≠ ∅ do
4  |  scan D2 to process the groups in G;
5  |  Prune-Groups(G);
6  |  scan D1 to process the groups in G;
7  |  foreach g ∈ G do
8  |  |  foreach i ∈ t(g) do
9  |  |  |  if h(g) ∪ {i} satisfies τs, τg and τi then E = E ∪ {h(g) ∪ {i}};
   |  |  end
   |  end
10 |  Prune-Groups(G);
11 |  G = Prune-Gen-Next(G);
   end
```

In (Bailey et al. 2002), a fast algorithm for mining Jumping Emerging Patterns (JEPs) is proposed, which is typically 5-10 times faster than the earlier border-based approach. The algorithm constructs tree structures to target the likely distribution of JEPs.

In (Bailey et al. 2003a), Constrained Emerging Patterns (CEPs) are proposed and demonstrated to have improved classification power in comparison to previous types of Emerging Patterns. A CEP is an itemset X whose support in one data class is more than a given threshold α while whose support in another class is less than another threshold β, namely

$$supp_1(X) \geqslant \alpha \wedge supp_2(X) \leqslant \beta.$$

CEPs are more valuable for classification due to its flexibility to specify thresholds for each classes of data individually. It is also shown in (Bailey et al. 2003a) that classification accuracy for CEPs can be significantly improved for multi class problems by the use of a round robin technique (Furnkranz 2002).

In (Bailey et al. 2003b), a new algorithm for computing hypergraph transversals is developed. It outperforms previous approaches by a factor of 9-29 times shown by experiments on a number of large datasets. The hypergraph minimal transversal problem is particularly significant from a data mining perspective and its close connection to the mining of Emerging Patterns is further highlighted in (Bailey et al. 2003b). Indeed, the algorithmic complexity of mining maximal frequent itemsets and minimal infrequent itemsets is closely linked to the complexity of computing minimal hypergraph transversals (Gunopulos, Mannila, Khardon & Toivonen 1997).

3.3 Classifiers Based on Emerging Patterns

3.3.1 A General Framework of EP-based Classifiers

Algorithm 3.5 shows a generic eager EP-based classifier. CAEP and the JEP-Classifier belong to this framework; while DeEP is a lazy EP-based classifier.

Although EPs are defined on only two classes of data, EP-based Classifiers are able to handle datasets containing more than two classes. A training dataset containing n classes is usually partitioned into n training datasets according to the class label. Line 1 in

Algorithm 3.5 produces the *pair-wise features*, which consists of the following sets of EPs: the first set of EPs from $(\bigcup_{j=2}^{n} D_j)$ to D_1, the second set of EPs from $(\bigcup_{j\neq 2}^{n} D_j)$ to D_2, $\cdots$, the nth set of EPs from $(\bigcup_{j=1}^{n-1} D_j)$ to D_n. For example, if $n = 3$, the *pair-wise features* contains three sets of EPs: EPs from $D_2 \cup D_3$ to D_1, EPs from $D_1 \cup D_3$ to D_2, and EPs from $D_1 \cup D_2$ to D_3. The *pair-wise features* can be discovered by any EP mining algorithm, e.g., the bordered-based algorithms and ConsEPMiner discussed above. The discovery of *pair-wise features* corresponds to line 1 in Algorithm 3.5.

When EPs of each class are available, usually a selection process is used to filter EPs that are not important or not useful for classification, according to some criteria. This corresponds to line 2 in Algorithm 3.5. We point out that this step is optional, although it may dramatically reduce the number of EPs for classification.

Up to now, we have already had a classification model built from the training data, where the model is represented by n sets of EPs, one set per class. The model can be used to classify unknown instances in the future. We stress that the building of the model, which is equivalent to the discovery of EPs, needs to be done only once, during the training phase.

In the testing phase, the EP-based classifier is used to classify every testing instance (Algorithm 3.5 line 3-5). For a testing instance t, we derive n scores for it, one score per class, by feeding the EPs of each class into a scoring function (Algorithm 3.5 line 4). Usually the scoring function is carefully designed to make good use of the characteristics of different kinds of EPs. The class with the highest score is assigned to t as its class label (Algorithm 3.5 line 5). Ties are broken in favor of the class with more instances in the training dataset. When the scores are exactly same or very close (i.e., the absolute difference is less than a given threshold), it will predict the majority class.

3.3.2 CAEP: Classification by Aggregating Emerging Patterns

CAEP is the first application of Emerging Patterns for classification. The approach is based on the following idea:

- Each EP can sharply differentiate the class membership of a (possibly small) fraction of instances containing the EP, due to the large difference between its support in the

Algorithm 3.5: A generic eager EP-based Classifier

input : a set of training instances D (contaning n classes of data, i.e. $D = D_1 + D_2 + \cdots + D_n$) and a set of testing instances T

output: the classification for each testing instance $t \in T$

/* `the training phase` */

1 **foreach** $1 \leqslant i \leqslant n$ **do** mine the set E_i of EPs from $(\bigcup_{j=1}^{n} D_j - D_i)$ to D_i;

2 post-process the discovered EPs;

/* `the testing phase` */

3 **foreach** *testing instance* $t \in T$ **do**

4 | **foreach** *class* i **do** compute a score $S(i,t)$ based on `Scoring-Function` ;

5 | assign the class with the highest score to t;

end

two opposing classes;

- such strong differentiating power of an EP is roughly proportional to its growth rate and its support in the target class.

In the training phase, CAEP employs ConsEPMiner (Zhang et al. 2000a) to mine EPs for each class. In order to classify a test instance T, it derives an aggregate score for each class C_i, by summing the differentiating power of all EPs of C_i that are subsets of T.

Definition 3.9 Given a test instance T and a set $E(C_i)$ of EPs of data class C_i, discovered from the training data, the **aggregate score (or score)** of T for the class C_i is defined as

$$score(T, C_i) = \sum_{X \subseteq T, X \in E(C_i)} \frac{growth\_rate(X)}{growth\_rate(X) + 1} * supp_{C_i}(X),$$

where $supp_{C_i}(X)$ is the support of X in class C_i, and $growth\_rate(X)$ is $supp_{C_i}(X)$ divided by the X's support in non-C_i class.

To reduce the negative effect of unbalanced distribution of EPs among the classes – a class with many more EPs in comparison to other classes tends to have higher scores, even for a test which indeed belongs to the other class – the score for C_i is then "normalized" by dividing it by some base score (e.g., the median score of all scores obtained using the training

instances of C_i as "tests"). The score, per class, measures the weight of that class as an appropriate label for the test. By sorting these scores, the order of them is generally a good indication of class membership for the test.

A variant of CAEP is iCAEP (Zhang, Dong & Ramamohanarao 2000b). iCAEP is an information-based approach to aggregate EPs for classification – classification by Minimum Message Length (MML) inference. The idea is to use EPs appearing in a test instance as symbols encoding the instance. If the instance has the highest probability of belonging to a class, its encoding cost should be the minimum given the class. When classifying a test, iCAEP selects a smaller but more representative subset of EPs. Experiments on many benchmark datasets show that iCAEP, when compared to CAEP, has better predictive accuracy and shorter time for training and classification (Zhang et al. 2000b).

Classifiers based on aggregating EPs, such as CAEP and iCAEP, compute scores for each class to make a decision. However, the skewed distribution of EPs among classes and intricate relationship between EPs sometimes make the decision by directly comparing scores unreliable. A Score Behavior Knowledge Space (SBKS) is used to record the behavior of the training instances on scores (Zhang, Dong & Ramamohanarao 2001). Classification decisions are drawn from SBKS using statistical techniques. Extensive experiments on real-world datasets show that SBKS frequently improves the performance of both EP-based classifiers, especially on datasets where they perform relatively poorly (Zhang et al. 2001).

3.3.3 JEPC: JEP-Classifier

The JEP-Classifier (Li, Dong & Ramamohanarao 2001) uses the large support JEPs (the most discriminating and expressive knowledge) to maximize its collective discriminating power when making predictions. In its learning phase, the most expressive JEPs are discovered by simply taking the left bounds of the borders derived by JEP-Producer (Algorithm 3.3). Its classification decision is based on the collective impact contributed by the most expressive pair-wise features.

To classify a test instance T, the JEP-Classifier evaluates the collective impact of only the most expressive JEPs that are subsets of T.

Definition 3.10 Given a test instance T and a set $E(\mathcal{D}_i)$ of the most expressive JEPs of a data class $\mathcal{D}_i$ discovered from the training data, the **collective impact** in favor of class $\mathcal{D}_i$, contributed by the most expressive JEPs of $\mathcal{D}_i$ is defined as

$$\sum_{X \subseteq T, X \in E(\mathcal{D}_i)} supp(X).$$

3.3.4 DeEPs: Decision-making by Emerging Patterns

The EP-based classifiers discussed so far are eager, i.e., they do a once only mining of EPs and then aggregate the power of these EPs that are contained within a given test instance. In contrast, the Decision-making by Emerging Patterns (DeEPs) classifier (Li, Dong, Ramamohanarao & Wong 2004) computes EPs in a lazy fashion, based on knowledge of the test.

Assume that a classification problem has a set of positive training instances ($D_p = \{P_1, P_2, \cdots, P_m\}$), a set of negative instances ($D_n = \{N_1, N_2, \cdots, N_m\}$), and a set of test instances. In order to classify a test T, DeEPs performs the following steps.

1. Take the intersection of each training instance with T, namely, $T \cap P_1, T \cap P_2, \cdots, T \cap P_m$ and $T \cap N_1, T \cap N_2, \cdots, T \cap N_m$.

2. Select the maximal itemsets from $\{T \cap P_1, T \cap P_2, \cdots, T \cap P_m\}$ to form a new set of positive instances (max_D_p); similarly, the maximal itemsets from $\{T \cap N_1, T \cap N_2, \cdots, T \cap N_m\}$ form a new set of negative instances (max_D_n).

3. Discover two JEP borders that represent JEPs of max_D_p and of max_D_n using the border-based algorithms.

4. Select the JEPs in the left bound and calculate classification scores for both classes based on the frequencies of those JEPs.

This first step is equivalent to removal of irrelevant training values, thus, all zero-frequency subsets of T are removed from the training data. A neighborhood-based intersection is used to deal with continuous attributes. All continuous attribute values can be

normalized into the range of $[0, 1]$ if its domain is not $[0, 1]$. Suppose A is a continuous attribute. Given a training instance S, $T \cap S$ will contain $T[A]$, if $T[S] \in [T[A] - \alpha, T[A] + \alpha]$, where $T[A]$ and $S[A]$ denote the value of T and S on attribute A. The parameter α is called the neighborhood factor, which can be used to adjust the length of the neighborhood. Experiments have shown 0.12 to be a suitable value in practice.

In the last step, the compact summation method (Definition 3.11) is used to aggregate the frequencies of individual JEPs to compute scores for all classes.

Definition 3.11 The **compact summation** of the frequencies in a class of data D_i with respect to a collection of JEPs is defined as the percentage of instances in D_i that contain at least one of these JEPs. The percentage is called the compact score of T for class i, that is,

$$compactScore(i) = \frac{count_i(E)}{|D_i|},$$

where E is the selected boundary JEPs and $count_i(E)$ is the number of instances in D_i that contain at least one JEP from E.

3.3.5 Discussion

The emphasis of the research in the machine learning and statistics community has been on improving the accuracy of a classifier on unseen test cases. For many practical applications, it is also desirable that the classifier "provide insight and understanding into the predictive structure of the data", as well as explanations of its individual predictions (Breiman et al. 1984). The EP-based classifiers are attractive in that the resulting classification model is easy to assimilate by humans, because EPs, which are basically conjunctions of simple conditions (where each conjunct is a test of the value of one of the attributes), are like if-then rules.

Decision-Tree Based Classifiers usually arrive at a classification decision by making a sequence of micro decisions, where each micro decision is concerned with one attribute only. EP-based classifiers such as CAEP and JEP-Classifier, and Association Rules Based Classifiers such as CBA adopt a new approach by testing groups of attributes in each micro

decision. While CBA uses one group at a time, EP-based classifiers use the aggregation of many groups of attributes.

Studies have shown that so far no algorithm uniformly outperforms all other algorithms in terms of quality (Lim, Loh & Shih 2000). The family of classifiers based on Emerging Patterns can provide new alternatives for people to use when other classification systems do not work well.

3.4 Tools

3.4.1 Discretization for Continuous Values

EP mining algorithms and EP-based classifiers require discrete values. Using discrete values has a number of advantages.

1. Intervals of numbers are more concise to represent and specify;
2. Discrete values are easier to use and comprehend as they are closer to a knowledge-level representation than continuous values;
3. Many induction tasks can benefit from discretization: rules with discrete values are normally shorter and more understandable; discretization can lead to improved predictive accuracy.

There are numerous discretization methods available in the literature - please see (Liu, Hussain, Tan & Dash 2002) for a systematic survey. In this thesis, we use the "discretize" utility from $\mathcal{MLC}$++ (Kohavi, John, Long, Manley & Pfleger 1994, Kohavi, Sommerfield & Dougherty 1997) to discretize continuous values into intervals. We adopt the Entropy method described in (Fayyad & Irani 1993), because it is the recommended discretization method by a number of studies (Fayyad & Irani 1993, Dougherty, Kohavi & Sahami 1995).

3.4.2 Weka: Machine Learning Software in Java

Weka[1] (http://www.cs.waikato.ac.nz/ml/weka/) is a collection of machine learning algorithms for solving real-world data mining problems. The abbreviation WEKA, stands

[1]Found only on the islands of New Zealand, the weka is a flightless bird with an inquisitive nature.

for "Waikato Environment for Knowledge Analysis". The algorithms can either be applied directly to a dataset or called from users' own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes. Weka is open source software issued under the GNU General Public License. Many classification algorithms are included in WEKA, such as the Naive Bayesian (NB) classifier, the decision tree based classifier C4.5, the Support Vector Machines (SVM) classifier, and etc.

WEKA machine learning software uses ARFF[2] (Attribute-Relation File Format) files, which are ASCII text files that describe a list of instances sharing a set of attributes. ARFF files have two distinct sections: the first section is the *Header* information, which is followed the *Data* information. The Header of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types. An example header of the UCI Weather dataset looks like this:

```
@relation golfWeatherMichigan_1988/02/10_14days

@attribute outlook {sunny, overcast rainy}
@attribute windy {TRUE, FALSE}
@attribute temperature real
@attribute humidity real
@attribute play {yes, no}
```

The Data of the ARFF file looks like the following:

```
@data
sunny,FALSE,85,85,no
sunny,TRUE,80,90,no
overcast,FALSE,83,86,yes
rainy,FALSE,70,96,yes
rainy,FALSE,68,80,yes
```

Lines that begin with a % are comments. The @RELATION, @ATTRIBUTE and @DATA declarations are case insensitive.

To compare the performance of WEKA classifiers with our new EP-based classifier, we use the following command lines, where exactly the same training (`train_db`) and testing (`test_db`) datasets are used in all WEKA classifiers and EP-based classifiers.

[2]ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato.

```
java weka.classifiers.bayes.NaiveBayes -t train_db -T test_db

java weka.classifiers.trees.j48.J48 -t train_db -T test_db

java weka.classifiers.functions.supportVector.SMO -t train_db  -T test_db
```

3.4.3 Classification Procedure

To evaluate the performance of different classifiers, we use the procedure shown in Algorithm 3.6 to perform experiments. Our procedure follows the ten fold cross-validation (CV-10) methodology, which is usually used to measure the predictive accuracy of classifiers and compare the performance of classifiers (Kohavi 1995, Salzberg 1997).

Algorithm 3.6: Classification procedure

/* `Following ten fold cross-validation (CV-10) methodology` */

1 Download the original datasets (denoted as $\mathcal{D}$) from the UCI website or $\mathcal{MLC}$++ website;

2 Suppose there are c classes in $\mathcal{D}$. Partition $\mathcal{D}$ into $\mathcal{D}_1$, $\mathcal{D}_2$, $\cdots$, $\mathcal{D}_c$ according to the class labels of instances;

3 **foreach** $i = 1, 2, \cdots, c$ **do**

4 | Shuffle randomly $\mathcal{D}_i$, using the function `random_shuffle()` from the Standard Template Library (STL);

end

5 **foreach** *iteration* $fold = 1, 2, \cdots, 10$ **do**

6 | **foreach** D_i $(i = 1, 2, \cdots, c)$ **do**

7 | | Select the fold-th 10% of instances as testing data (`test_db_i`) and the remaining 90% as training data (`train_db_i`). If the number of instances in D_i is less than 10, select one instance as testing data and the remaining instances as training data;

| | **end**

8 | Let `train_db` $= \bigcup_i^c$ `train_db_i`;

9 | Let `test_db` $= \bigcup_i^c$ `test_db_i`;

10 | **if** *the dataset contains continuous attributes* **then**

11 | | Discretize `train_db` by using the "discretize" utility from $\mathcal{MLC}$++. Use the mapping obtained solely from `train_db` to discretize `test_db`;

| | **end**

12 | Evaluate the performance of different classifiers, using the training dataset (`train_db`) and the testing dataset (`test_db`);

end

13 Average the performance in ten folds, such as accuracy, number of rules/patterns used for classification, training time, classification time, and so on;

Chapter 4

Essential Jumping Emerging Patterns (EJEPs)

As we have already seen, Emerging Patterns (EPs) (Dong & Li 1999) serve well as a classification model, because EPs are multivariate features (i.e., itemsets) describing significant differences between two classes of data. In this chapter, we present a special type of Emerging Pattern, called Essential Jumping Emerging Patterns (EJEPs), which removes (potentially) useless Jumping Emerging Patterns (JEPs) while retaining those with discriminating power. Previous algorithms such as border-based algorithms (Dong & Li 1999) and consEPMiner (Zhang et al. 2000a) can not directly mine these EJEPs. We present a new "single-scan" algorithm to efficiently mine EJEPs of both data classes (both directions) at the same time in case of two-class problem. Experimental results show that EJEPs are high quality patterns with the most differentiating power and thus EJEPs are sufficient for building accurate classifiers.

In Chapter 5, we will discuss Chi Emerging Patterns (Chi EPs), which are more general than EJEPs and have less strict constraints. Another new, heuristic algorithm will also be discussed, which is able to mine most of the interesting EPs without losing many important ones.

4.1 Motivation

Recall that a Jumping Emerging Pattern (JEP) is a special type of Emerging Pattern, which is defined as an itemset whose support increases abruptly from zero in one dataset, to non-zero in another dataset – the ratio of support-increase being infinite. Actually, JEPs represent knowledge which discriminates between different classes more strongly than any other type of Emerging Pattern. For example, the pattern (itemset) {(Odor=none), (Gill_Size=broad), (Ring_Number=one)} is a JEP, which occurs frequently (63.9%) in edible mushroom but zero time in poisonous mushroom. To classify an unknown type of mushroom, if the test contains such a JEP, then we can say with a very high confidence that the mushroom is edible. By aggregating *the most expressive* JEPs, JEP-Classifier (JEP-C) (Li, Dong & Ramamohanarao 2001) achieves surprisingly higher accuracy than other state-of-the-art classifiers such as C4.5 (Quinlan 1993) and CBA (Liu et al. 1998). However, JEP-Classifier suffers from some weakness.

On one hand, although the number of *the most expressive* JEPs is much smaller than the number of all JEPs, sometimes JEP-Classifier has to use a huge number of JEPs to make predictions, even for small datasets. For example, from Table 2 in (Li, Dong & Ramamohanarao 2001), we can see that JEP-Classifier uses 32,510 JEPs for the UCI German dataset, which consists of 1000 instances described by 20 attributes; it uses 13,050 JEPs for the UCI Sonar dataset, which consists of 208 instances described by 60 attributes. As we will see later, we can use 1,137 patterns for the UCI German dataset and 1,391 patterns for the UCI Sonar dataset to achieve classification accuracy very close to or equal to JEP-Classifier, i.e., lose of accuracy is less than 1%. Therefore, many of those JEPs are redundant for classification.

On the other hand, although the border-based algorithms used in JEP-Classifier are efficient for discovering concise border representations of all JEPs, mining many unnecessary JEPs prolongs the learning phases of JEP-Classifier. For instances, it is reported in (Li, Dong & Ramamohanarao 2001) that JEP mining can take up to 2 hours for the UCI datasets such as Mushroom, Sonar, German, Nursery and Ionosphere. Even the most efficient algorithms for mining JEPs, recently proposed in (Bailey et al. 2002), are still not

fast enough. It is reported in (Bailey et al. 2002) that for the UCI Waveform dataset, which consists of 5000 instances by 21 attributes, it took up to four hours to mine 4,096,477 JEPs. In contrast, our algorithm can mine 7,083 EJEPs in about 20 minutes from the Waveform dataset, and these EJEPs are sufficient for building accurate classifiers, as experimental results show that our accuracy on the Waveform dataset is a little higher than JEP-Classifier.

Many redundant JEPs not only make JEP-Classifier complex and hard to understand, but also prolong both the training and testing phases of JEP-Classifier. Therefore, we are interested in those JEPs which represent the essential knowledge to discriminate between different classes. We propose Essential Jumping Emerging Patterns (EJEPs) to capture the crucial difference between a pair of data classes. EJEPs are defined as *minimal* itemsets whose supports in one data class are zero but in another are above a given support threshold ξ. An EJEP covers at least a predefined number of training instances. EJEPs are also minimal or shortest: any proper subset of an EJEP is no longer an EJEP. If all the itemsets satisfying supports in one class are 0 but in another above ξ are to be represented by border description (Dong & Li 1999), EJEPs are those minimal in the left bounds. We believe that EJEPs are more useful in classification because:

- The set of EJEPs is the subset of the set of JEPs, after removing JEPs containing noise and redundant information. JEPs have been proved to have sharp discriminating power. EJEPs maintain such power by having infinite growth rates, and improve JEPs by having a minimum coverage on training data.

- EJEPs are minimal itemsets, i.e., itemsets consisting of least items (attribute-values). If less attributes can distinguish two data classes, using more may not help and may even add noise.

To get useful new insights and guidance from training data, the minimum support threshold ξ should be set reasonably low (e.g. 1%). Since useful Apriori property no longer holds for EJEPs and there are usually too many candidates, naive algorithms are too costly.

Inspired by FP-tree (Han et al. 2000), a successful structure to mine frequent patterns without candidate generation, we propose to use a tree structure called Pattern

tree (P-tree) as the basis for mining Essential Jumping Emerging Patterns. A P-tree is an extended prefix-tree structure storing the quantitative information about EJEPs. The counts of an item for both data classes are registered. Items with larger support ratios are closer to the root. So EJEPs, itemsets with infinite growth rate made up of more likely items with large support ratio, will appear near the root. From the root to search EJEPs, we will always find the shortest ones first. We develop a P-tree based pattern fragment growth mining method. It searches a P-tree depth-first to discover EJEPs from the root, which is completely different from FP-Growth (Han et al. 2000). While searching, nodes are merged, which ensures the complete set of EJEPs are generated. The pattern growth is achieved via concatenation of the prefix pattern with the new ones at deeper level. Since we are interested in the shortest EJEPs, the depth of the search is not very deep (normally 5-10). Another advantage of the method is that it is a "single-scan" algorithm, which can mine EJEPs from $\mathcal{D}_1$ to $\mathcal{D}_2$ and those from $\mathcal{D}_2$ to $\mathcal{D}_1$ at the same time. Previous approaches such as border-based algorithms (Dong & Li 1999) and consEPMiner (Zhang et al. 2000a) will call the corresponding algorithm twice using $\mathcal{D}_1$ and $\mathcal{D}_2$ as target databases separately.

We build classifiers based on EJEPs to measure their quality in classification. Experimental results show that our classifier uses much fewer EJEPs than the JEPs used in JEP-Classifier, while maintaining the accuracy very close to JEP-Classifier. Thus, EJEPs are sufficient for building accurate classifiers.

Organization: The remaining of this chapter is organized as follows. In Section 4.2, we introduce the notation of EJEP. In Section 4.3, we present the P-tree structure and explore its properties. Section 4.4 presents our algorithm for mining EJEPs from P-tree. In Section 4.5, we discuss the procedure of classification based on EJEPs . In Section 4.6, we provide experimental results using a number of benchmark databases from the UCI Machine Learning Repository (Blake & Merz 1998). Section 4.7 discusses related work. Finally, remarks are given in Section 4.8.

4.2 Essential Jumping Emerging Patterns (EJEPs)

Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two different classes of datasets.

Definition 4.1 Given $\xi > 0$ as a minimum support threshold, an **Essential Jumping Emerging Pattern (EJEP)** from $\mathcal{D}_1$ to $\mathcal{D}_2$, is an itemset X that satisfies the following conditions:

1. $supp_{\mathcal{D}_1}(X) = 0$ and $supp_{\mathcal{D}_2}(X) \geqslant \xi$, and
2. Any proper subset of X does not satisfy condition 1.

When $\mathcal{D}_1$ is clear from context, an EJEP X from $\mathcal{D}_1$ to $\mathcal{D}_2$ is simply called an EJEP of $\mathcal{D}_2$. The support of X in $\mathcal{D}_2$, $supp_{\mathcal{D}_2}(X)$, is called the support of the EJEP, denoted as $supp(X)$.

It is obvious that EJEPs also have infinite growth rates, which indicates they have strong predictive power. Note that the definition of EJEP is different from that of JEP in (Li, Dong & Ramamohanarao 2001). Their JEPs from $\mathcal{D}_1$ to $\mathcal{D}_2$ are the itemsets whose supports in $\mathcal{D}_1$ are zero but in $\mathcal{D}_2$ are non-zero. In condition 1, we further require the supports in $\mathcal{D}_2$ to be above a minimum support threshold ξ, which makes an EJEP cover at least a certain number of instances in a training dataset. We believe that JEPs whose supports in $\mathcal{D}_1$ are zero and whose supports in $\mathcal{D}_2$ are too low, contain too much noise, hence are not suitable to use in classification.

Condition 2 shows that any proper subset of an EJEP is not an EJEP any more, which means EJEPs are the *shortest* JEP. A JEP, by definition, is not necessarily the shortest. In (Li, Dong & Ramamohanarao 2001), they refer to the minimal JEPs as the most expressive JEPs, which are actually patterns in the left bounds of the border description. Consider that JEPs are actually itemsets. A shorter JEP means less items (attributes). If we can use less attributes to distinguish two data classes, adding more attributes will not contribute to classification, and even worse, bring noise when classifying by aggregating JEPs. Supersets of EJEPs are not useful in classification because of the following reason. Let E_1 and E_2 be two different itemsets satisfying condition 1, and $E_1 \subset E_2$. E_1 covers more (at least equal) instances of the training dataset than E_2, because $supp(E_1) \geqslant supp(E_2)$.

Table 4.1: Comparison between EJEP and JEP

Itemset Type	support in $\mathcal{D}_1$	support in $\mathcal{D}_2$	Growthrate	Minimal
JEP	0	> 0	∞	NO
EJEP	0	$> \mu$	∞	YES

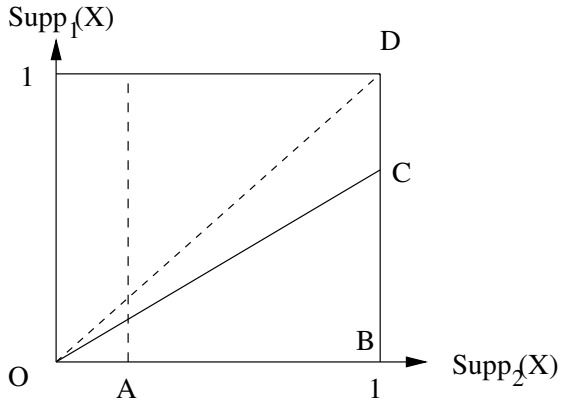


Figure 4.1: The support plane for Emerging Patterns - EJEP vs. JEP

4.2.1 A Comparison of EJEP with JEP

Table 4.1 compares EJEP against JEP in several facets, namely, support in $\mathcal{D}_1$, support in $\mathcal{D}_2$, growth rate, and whether minimal patterns or not. Note that although JEPs are not necessarily minimal patterns by definition, the traditional use of JEPs are all concentrated on minimal ones. In our work, we put the requirement of minimal patterns explicitly in the definition of EJEPs.

We also illustrate the differences among EJEPs, JEPs and other general EPs using the Figure 4.1. Let ρ be the growth rate threshold for EPs and ξ be the minimum support threshold for EJEPs. In the support plane, suppose $\tan \angle COB = \frac{1}{\rho}, (\rho > 1)$ and $|OA| = \xi$. All EPs occupy the triangle $\triangle OBC$, because only the growth rate threshold ρ is concerned. JEPs occupy the x-axis from O to B. EJEPs occupy part of the x-axis from A to B, because they have the minimum support requirement.

Note that if we let the minimum support threshold $\xi = 1$ in absolute occurrence, then the set of EJEPs is exactly the set of minimal JEPs.

Table 4.2: A dataset containing 2 classes

Class 1 ($\mathcal{D}_1$)					Class 2 ($\mathcal{D}_2$)				
a		c	d	e	a	b			
a							c		e
	b			e	a	b	c	d	
	b	c	d	e				d	e

Example 4.1 In the dataset shown in Table 4.2, itemsets $\{a, e\}(1:0)$[1], $\{a, c, e\}(1:0)$, $\{a, d, e\}(1:0)$, $\{a, c, d, e\}(1:0)$, $\{b, e\}(2:0)$, $\{b, c, e\}(1:0)$, $\{b, d, e\}(1:0)$, $\{b, c, d, e\}(1:0)$ and $\{c, d, e\}(2:0)$ are JEPs of class 1; itemsets $\{a, b\}(0:2)$, $\{a, b, c\}(0:1)$, $\{a, b, d\}(0:1)$ and $\{a, b, c, d\}(0:1)$ are JEPs of class 2. The total number of JEPs is 13, and 4 among them are minimal, namely, $\{a, e\}(1:0)$, $\{b, e\}(2:0)$, $\{c, d, e\}(2:0)$ and $\{a, b\}(0:2)$.

Suppose the minimum support for EJEP $\xi = 2$ (in absolute occurrence). $\{a, e\}(1:0)$, which is a JEP occurring once in $\mathcal{D}_1$ but zero time in $\mathcal{D}_2$, is not an EJEP. $\{a, b\}(0:2)$ is a JEP and also an EJEP. There are only 3 EJEPs, namely, $\{b, e\}(2:0)$, $\{c, d, e\}(2:0)$ and $\{a, b\}(0:2)$. □

From Example 4.1, we can see that there are many JEPs, even for a small dataset; but the number of EJEPs are not only much smaller than JEPs, but also smaller than minimal JEPs.

When using JEPs to classify a test instance, only the shortest JEPs with large supports contribute a lot to the decision of classification. EJEPs are such important contributors. By using EJEPs instead of JEPs for classification, we can greatly reduce the complexity of a classifier, and strengthen its resistance to noise in the training data.

Next, we consider how to discover the EJEPs between two classes of data.

4.3 The Pattern Tree Structure

Before we present our algorithm for mining EJEPs, we first define the order that we use to sort itemsets, and then describe the data structure and study the properties and benefits of the structure.

[1] $\{a, e\}(1:0)$ means that itemset $\{a, e\}$ appear 1 time in $\mathcal{D}_1$ and 0 time in $\mathcal{D}_2$. Others can be explained in the same way.

4.3.1 Support Ratio of Individual Item

Assume a training dataset contains only two classes of data: the positive class $\mathcal{D}_1$ and the negative class $\mathcal{D}_2$. Let $I = \{i_1, i_2, \cdots, i_n\}$ be the set of all the items appeared in the training dataset. Note that for an item $i \in I$, we have a singleton itemset $\{i\} \subset I$.

Definition 4.2 Given $\xi > 0$ as a minimum support threshold, the *support ratio* of an item i between $\mathcal{D}_1$ and $\mathcal{D}_2$, denoted as SupportRatio(i), is defined as

$$SupportRatio(i) = \begin{cases} 0 & \text{if } supp_{\mathcal{D}_1}(\{i\}) < \xi \ \wedge \ supp_{\mathcal{D}_2}(\{i\}) < \xi \\ \infty & \text{if } (supp_{\mathcal{D}_1}(\{i\}) = 0 \ \wedge \ supp_{\mathcal{D}_2}(\{i\}) \geqslant \xi) \vee \\ & \ (supp_{\mathcal{D}_1}(\{i\}) \geqslant \xi \ \wedge \ supp_{\mathcal{D}_2}(\{i\}) = 0) \\ \frac{supp_{\mathcal{D}_2}(\{i\})}{supp_{\mathcal{D}_1}(\{i\})} & \text{if } supp_{\mathcal{D}_1}(\{i\}) > 0 \ \wedge \ supp_{\mathcal{D}_2}(\{i\}) \geqslant \xi \wedge \\ & \ supp_{\mathcal{D}_2}(\{i\}) \geqslant supp_{\mathcal{D}_1}(\{i\}) \\ \frac{supp_{\mathcal{D}_1}(\{i\})}{supp_{\mathcal{D}_2}(\{i\})} & \text{if } supp_{\mathcal{D}_2}(\{i\}) > 0 \ \wedge \ supp_{\mathcal{D}_1}(\{i\}) \geqslant \xi \wedge \\ & \ supp_{\mathcal{D}_1}(\{i\}) \geqslant supp_{\mathcal{D}_2}(\{i\}) \end{cases}$$

Definition 4.2 is used to capture individual items with sharp contrast between $\mathcal{D}_1$ and $\mathcal{D}_2$ in either direction. The larger the *support ratio* of an item, the sharper the discriminating power the item has. Usually, the *support ratios* are more than or equal to 1, because we always let the larger support be divided by the smaller support. Only when both the supports in $\mathcal{D}_1$ and $\mathcal{D}_2$ are less than the minimum support threshold ξ, the *support ratio* becomes 0. Items with *support ratio* of 0 are useless for EJEP mining because EJEPs must satisfy the minimum support threshold ξ and EJEPs will never contain items whose supports in $\mathcal{D}_1$ and $\mathcal{D}_2$ are less than ξ. Note that for an item $i \in I$, if SupportRatio(i)=∞, then the singleton itemset $\{i\}$ is an EJEP.

Based on the definition of the *support ratio* (Definition 4.2), we define the order $\prec$ on I. Let $i, j \in I$ be two items. We say $i \prec j$,

- if $SupportRatio(i) > SupportRatio(j)$; or
- if $SupportRatio(i) = SupportRatio(j)$, and $i < j$ (in lexicographical order).

Now any two items can be compared by $\prec$. Any itemset can be transformed to an ordered list like $\{i_1, i_2, \cdots, i_k\}\,(i_1 \prec i_2 \prec \cdots \prec i_k)$. The itemsets we discuss below are such ordered

Table 4.3: Two datasets containing 4 instances each

ID	Class Label	Instances(Itemsets)	ordered itemsets by $\prec$
1	Positive $\mathcal{D}_1$	a,c,d,e	e,a,c,d
2	Positive $\mathcal{D}_1$	a	a
3	Positive $\mathcal{D}_1$	b,e	e,b
4	Positive $\mathcal{D}_1$	b,c,d,e	e,b,c,d
5	Negative $\mathcal{D}_2$	a,b	a,b
6	Negative $\mathcal{D}_2$	c,e	e,c
7	Negative $\mathcal{D}_2$	a,b,c,d	a,b,c,d
8	Negative $\mathcal{D}_2$	d,e	e,d

lists. We also define the order $\prec$ on itemsets. Let $\{a_1, a_2, \cdots, a_m\}$ and $\{b_1, b_2, \cdots, b_n\}$ be two itemsets. We say $\{a_1, a_2, \cdots, a_m\} \prec \{b_1, b_2, \cdots, b_n\}$,

- if $\exists 1 \leqslant i \leqslant m$, when $1 < j < i$, $a_j = b_j$, but $a_i \prec b_i$; or
- $\forall 1 < j < m$, $a_j = b_j$, but $m < n$.

Example 4.2 Table 4.3 shows a simple training dataset containing 4 positive instances and 4 negative instances, where the 8 instances of both classes are subsets of $I = \{a, b, c, d, e\}$. It is easy to calculate the frequency of each item:

$$sup_{\mathcal{D}_1}(a) = 2,\ sup_{\mathcal{D}_1}(b) = 2,\ sup_{\mathcal{D}_1}(c) = 2,\ sup_{\mathcal{D}_1}(d) = 2,\ sup_{\mathcal{D}_1}(e) = 3;$$

$$sup_{\mathcal{D}_2}(a) = 2,\ sup_{\mathcal{D}_2}(b) = 2,\ sup_{\mathcal{D}_2}(c) = 2,\ sup_{\mathcal{D}_2}(d) = 2,\ sup_{\mathcal{D}_2}(e) = 2.$$

Obviously, we have $e \prec a \prec b \prec c \prec d$ after calculating the support ratio of each item. □

4.3.2 Pattern Tree (P-tree)

A pattern tree (P-tree) is an *ordered multiway* tree structure. Each node N of the P-tree has a variable number of items, denoted as N.items[i], where $i = 1, 2, \cdots$, N.item-number, and N.item-number is the number of items in the node N. If N.item-number=k ($k > 0$), N has k positive counts, k negative counts, and at most k branches (child nodes), denoted as N.positive-counts[i], N.negative-counts[i] and N.child-nodes[i],

respectively, where $i = 1, 2, \cdots, k$. For N.items[i] ($1 \leqslant i \leqslant k$), N.positive-counts[i] registers the number of positive instances in $\mathcal{D}_1$ represented by the portion of the path reaching N.items[i], N.negative-counts[i] registers the number of negative instances in $\mathcal{D}_2$ represented by the portion of the path reaching N.items[i], N.child-nodes[i] refers to the subtree with the root of N.items[i] (also called N.items[i]'s subtree). To maintain the branches of N *ordered*, we require that the k items inside N satisfy: N.items[1] $\prec$ N.items[2] $\prec \cdots \prec$ N.items[k]. An example P-tree is illustrated in Figure 4.2.

To help the following discussion, we follow the conventional concepts about trees. The root of P-tree is drawn at the top and defined to be at level 0. A node N that is directly below node M, is called a direct descendant of M. If M is at level i, then N is said to be at level $i + 1$. Conversely, node M is said to be the direct ancestor of N. We sometimes carry the analogy to family trees further and refer to the "parent", "children" or "sibling" of a node.

Let node N be a direct descendant of node M. There is a branch of M to N, which is associated with M.items[i]. We say that M is N's parent, and N is M's child. Specially, N is M.items[i]'s child, and N.items[j] is the j-th child of M or M.items[i]. To make the P-tree ordered, not only all the items inside a node are ordered as described above, we further require that the items between parents and children are also ordered. Let M.items[i] be the parent of N, then $\forall j \leqslant N$.item-number, M.items[i] $\prec$ N.items[j]. That is, for any child node of M.items[i], denoted as N.items[j], we have M.items[i] $\prec$ N.items[j].

We are interested in the paths beginning from the root of the P-tree. Let R be P-tree's root, $P_1 = \langle\ R.\text{items}[i_1],\ M_2.\text{items}[i_2],\ \cdots,\ M_m.\text{items}[i_m]\ \rangle$ and $P_2 = \langle\ R.\text{items}[j_1],\ N_2.\text{items}[j_2],\ \cdots,\ N_n.\text{items}[j_n]\ \rangle$ be two paths from the root R to M_m.items[i_m] and to N_n.items[j_n], respectively. Both paths can be traversed by going through the branches of those items sequentially. We say P_1 is to the left of P_2, if $\exists 1 \leqslant k \leqslant \min(m, n)$, R.items[i_1]=R.items[j_1], M_2.items[i_2]=N_2.items[j_2], $\cdots$, M_{k-1}.items[i_{k-1}]=N_{k-1}.items[j_{k-1}], but M_k.items[i_k] $\prec$ N_k.items[j_k]. That means, P_1 and P_2 overlap the first $(k-1)$ levels from the root, and P_1 is to the left of P_2 at the kth level. When $k > 1$, they have the same prefix $\langle\ R.\text{items}[i_1],\ M_2.\text{items}[i_2],\ \cdots,\ M_{k-1}.\text{items}[i_{k-1}]\ \rangle$.

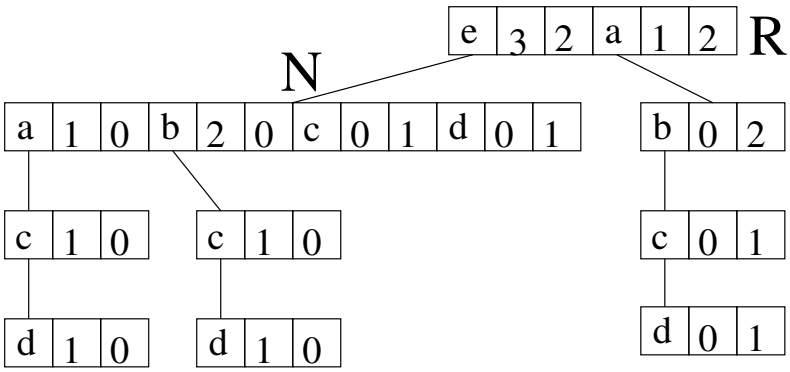


Figure 4.2: The original P-tree of the example dataset

Example 4.3 The P-tree of the example dataset in Table 4.3 is shown in Figure 4.2. Refer to Algorithm 4.1 for the construction of the P-tree. The root $R = \{e(3,2), a(1,2)\}$. This means that

- for those instances beginning with e (instances with ID = 1, 3, 4, 6 and 8), e appears 3 times in the positive class and 2 times in the negative.
- for those instances beginning with a (instances with ID = 2, 5 and 7), a appears 1 time in the positive class and 2 times in the negative. Note that a also appear in some instances beginning with e, e.g., the instance with ID = 1.

We can see that in each node of the P-tree, the items a, b, c, d, e are ordered, i.e., $e \prec a \prec b \prec c \prec d$. Also observe that the items in every path from R to a leaf are ordered, e.g., the left-most branch of the P-tree: $\langle e, a, c, d \rangle$. $\langle e, a, c, d \rangle$ corresponds to an itemset $\{e, a, c, d\}$, which appears once in the positive class and zero time in the negative. □

4.3.3 Properties of Pattern Tree (P-tree)

Several important properties of P-tree can be observed from the above description. The basic property of P-tree is that the items inside any node are *ordered*, and the items between parents and children are *ordered*. We draw some interesting properties of P-tree from this basic property. Let R be P-tree's root, $P_1 = \langle\ R.\text{items}[i_1],\ M_2.\text{items}[i_2],\ \cdots,\ M_m.\text{items}[i_m]\ \rangle$ and $P_2 = \langle\ R.\text{items}[j_1],\ N_2.\text{items}[j_2],\ \cdots,\ N_n.\text{items}[j_n]\ \rangle$ be two paths. P_1

represents itemset $I_1 = \{R.\text{items}[i_1], M_2.\text{items}[i_2], \cdots, M_m.\text{items}[i_m]\}$; P_2 represents itemset $I_2 = \{R.\text{items}[j_1], N_2.\text{items}[j_2], \cdots, N_n.\text{items}[j_n]\}$. Obviously, the map from a path to an itemset is a one-to-one mapping. For example, the left-most branch of the P-tree shown in Figure 4.2 corresponds to itemset $\{e, a, c, d\}$.

Property 4.1 If P_1 is P_2's subpath (prefix), then $I_1 \prec I_2$.

Property 4.2 If P_1 is to the left of P_2, then $I_1 \prec I_2$.

Property 4.1 and 4.2 enable us to produce the complete set of paths (itemsets) systematically through depth-first searches of the P-tree according to the predefined order $\prec$. Additionally, it's easy to prove that $R.\text{items}[1]$ is the minimum of all the items under the order $\prec$ and that no other subtrees of R except the subtree with the root of $R.\text{items}[1]$, contains $R.\text{items}[1]$. So we have another important property.

Property 4.3 All the paths (itemsets) containing $R.\text{items}[1]$ are included in the subtree with the root of $R.\text{items}[1]$.

Property 4.3 enables us to mine the complete set of paths (itemsets) containing $R.\text{items}[1]$ exclusively from the subtree with the root of $R.\text{items}[1]$. However, the same is not true for the subtree with the root of $R.\text{items}[2]$, $R.\text{items}[3]$, $\cdots$, because those paths that are not from the root R may also represent interesting itemsets. For example, itemsets containing $R.\text{items}[2]$ may not only appear in $R.\text{items}[2]$'s subtree, but also appear in $R.\text{items}[1]$'s subtree. Our solution to the problem is to merge nodes (see Section 4.4.3 for details).

4.3.4 The Construction of Pattern Tree (P-tree)

Based on the P-tree definition, we have Algorithm 4.1 to construct P-tree.

Although P-tree and FP-tree are both tree structures, there are many important differences between them, as reflected by the construction algorithm.

- P-tree registers the counts in both positive and negative class; FP-tree has only one count for an item in a node because frequent patterns are defined upon one collection of data.

Algorithm 4.1: P-tree construction (for mining EJEPs)

input : A training dataset D containing two classes of data (positive $\mathcal{D}_1$ and negative $\mathcal{D}_2$) and a minimum support threshold ξ for EJEPs

output: The patter tree of D, P-tree

/* `The first database scan` */

1 Scan the training dataset D once;

2 Collect the set of items whose support ratios are more than zero, denoted as J, and their support ratios;

3 Sort J in support-ratio-descending order as L;

/* `The second database scan` */

4 Create the root of a P-tree, R, with R.item-number=0;

5 **foreach** *instance inst in D* **do**

6 | Select and sort the J items in inst according to the order of L;

7 | Let the sorted item list in inst be $[p|P]$, where p is the first element and P is the remaining list. Call `insert-tree(`$[p|P]$, R`)`;

end

Algorithm 4.2: Function: insert_tree($[p|P], T$) (for mining EJEPs)

/* `The function insert-tree is called by Algorithm 4.1, P-tree construction, preparing the data structure for mining EJEPs` */

search T for T.items[i]=p;
if *T.items[i] is not found* **then**
    /*`This ensures all items in T are always ordered by` $\prec$ */
    insert p at the right place in T, denoted as T.items[i], with both positive and negative counts zero;
    increment T.item-number by 1;
end
switch *the class label of the instance* $[p|P]$ **do**
    case *positive class* increment T.count-positive[i] by 1;
    case *negative class* increment T.count-negative[i] by 1;
end
if *P is nonempty* **then**
    if *T.items[i]'s subtree is empty* **then** create a new node N with N.item-number=0 as T.items[i]'s subtree;
    Let N be T.items[i]'s subtree, call `insert-tree(`$[p|P]$, N`)`;
end

- FP-tree has node-links to facilitate tree traversal; P-tree does not require node-links because we traverse it depth-first from the root. One immediate benefit is less memory space needed.

- P-tree uses support-ratio-descending order to sort items; FP-tree uses support descending order.

- In P-tree, the items inside any node are *ordered* and the items between parents and children are also *ordered*. This implies ordering of input data. In contrast, FP-tree does not have such explicit requirements of the order on items.

4.4 Using Pattern Tree (P-tree) to Mine EJEPs

4.4.1 An Illustrative Example

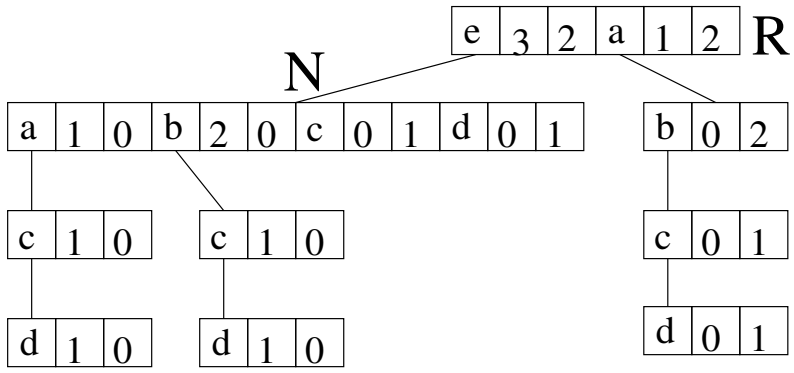
Let us use an example to illustrate the mining process. Let the minimum count (in absolute occurrence) for EJEP be 2. The initial P-tree of the example dataset is shown in Figure 4.2, where R denotes its root and N denotes $R.e$'s subtree. We first merge the subtree N with R, i.e., call $merge(N, R)$ (the reason for doing this will be discussed later this section). After $merge(N, R)$, the modified P-tree is as Figure 4.3. Because N does not change, the nodes in N are not drawn in the figure. Note the following differences between the modified P-tree and the original one.

- The new root contains more items: b, c, d are added into R.

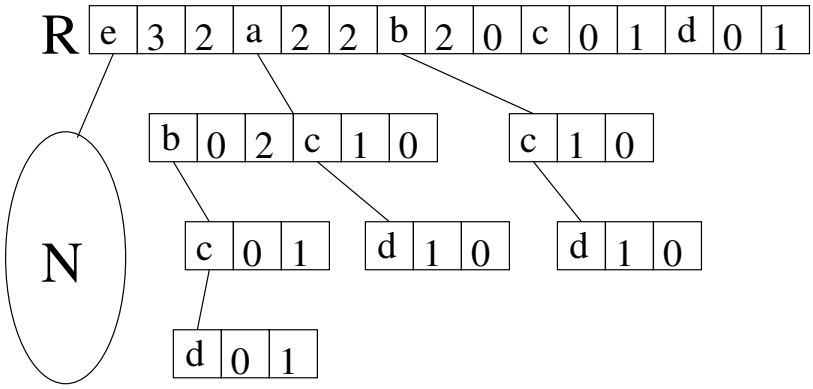
- The counts of some items in R changes: $a(1:2) \Rightarrow a(2:2)$.

- Two new branches from R: $\langle\ a(2:2)\ c(1:0)\ d(1:0)\ \rangle$ and $\langle\ b(2:0)\ c(1:0)\ d(1:0)\ \rangle$.

We then examine both counts of $R.e(3:2)$ and find that itemset $\{e\}$ is not an EJEP. We depth-first search the P-tree. After calling $merge(M, N)$, where M is $N.a$'s subtree, we examine N and find that itemset $\{e, a\}(1:0)$ (1 and 0 are two counts in $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively) is not an EJEP. Because both counts are smaller than the threshold, we do not go down this branch further looking for EJEPs. Instead, we turn to the next item in



The original P-tree



The P-tree after merging $merge(N, R)$

Note: (1) for easy comparison of the original P-tree and the P-tree after $merge(N, R)$, we put the original P-tree again on the top.

Note: (2) because the subtree N does not change, we do not drawn N in detail in the P-tree after $merge(N, R)$.

Figure 4.3: The P-tree after merging nodes ($merge(N, R)$)

N and merge $N.b$'s subtree with N. The search is done recursively and the complete set of EJEPs can be generated, that is $\{e, b\}(2:0)$, $\{a, b\}(0:2)$ and $\{e, c, d\}(2:0)$.

4.4.2 Algorithms for Mining EJEPs

While the P-tree structure is somewhat similar to FP-tree (Han et al. 2000), the mining process is completely different. Because every training instance is sorted by its support ratio between both classes when inserting into P-tree, items with high ratio, which are more likely to appear in an EJEP, are closer to the root. We start from the root to search P-tree depth-first for EJEPs. We have Algorithm 4.3 for mining EJEPs using P-tree. Algorithm 4.3 Line 1 calls the recursive function `mine-tree()` with two arguments: the root of P-tree R and an empty itemset. The empty itemset will grow one item by another when `mine-tree()` is called recursively. Algorithm 4.3 Line 2-3 selects only those minimal JEP by filtering out JEPs that are super sets of another JEP. The remaining minimal ones are EJEPs since they also satisfy the minimum support threshold ξ.

Assume we arrive at node M. When the path from the root to $M.items_k$, which represents an itemset, forms a JEP P, we no longer need to go deeper into $M.childnodes_k$, because those paths are supersets of P. All the JEPs that P-tree contains can be represented by a border description $\langle L, R\rangle$. Although the algorithm can not exclusively generate JEPs in the left-hand bound L, it will not generate all either. Therefore, we usually get a relatively small set of JEPs, we choose EJEPs from them by selecting only the minimal ones.

We stress that our algorithm can discover EJEPs of both $\mathcal{D}_1$ and $\mathcal{D}_2$ from the P-tree at the same time - a "single-scan" algorithm. Previous EP mining methods such as border-based algorithms (Dong & Li 1999) and consEPMiner (Zhang et al. 2000a) have to call the corresponding algorithm twice using $\mathcal{D}_1$ and $\mathcal{D}_2$ as target dataset separately. Unlike those two approaches, we do not need to construct one P-tree for mining EJEPs of $\mathcal{D}_1$, and construct another P-tree for mining EJEPs of $\mathcal{D}_2$.

4.4.3 Why Merge Subtrees?

Now we explain the reason behind the need for calling `merge_tree(`T_1, T_2`)` during the mining process. One itemset can contribute to the counts of a number of patterns by 1.

Algorithm 4.3: Mining Essential Jumping Emerging Patterns (EJEPs) using P-tree

input : a P-tree with the root of R, and a minimum support threshold ξ for EJEPs

output: the complete set of EJEPs

/* R `refers to the root of P-tree and` *null* `means an empty itemset` */

1 Call `mine-tree` $(R, null)$. The result is a set of itemsets, denoted as E;

/* `These patterns in` E `are EPs with infinite growth rates, i.e., they have zero support in one class but minimum support` ξ `in another` */

/* `Filter out non-minimal itemsets from` E */

2 **foreach** *itemset* $e \in E$ **do**

3 **if** $\exists\, f \in E\ (f \subset e)$ **then** remove e from E;

end

4 Output the remaining patterns in E;

Note that $e \prec a \prec b \prec c \prec d$. Consider the left-most branch of the P-tree shown in Figure 4.2, which corresponds to the first instance of the dataset shown in Table 4.3, $\{e, a, c, d\}$. It can contribute to 15 patterns, namely, $\{e\}$, $\{e, a\}$, $\{e, a, c\}$, $\{e, a, c, d\}$, $\{e, a, d\}$ $\{e, c\}$, $\{e, c, d\}$, $\{e, d\}$, $\{a\}$, $\{a, c\}$, $\{a, c, d\}$, $\{a, d\}$, $\{c\}$, $\{c, d\}$, $\{d\}$. We can partition these itemsets into three groups:

1. Group 1 includes the prefix of $\{e, a, c, d\}$: $\{e\}$, $\{e, a\}$, $\{e, a, c\}$, $\{e, a, c, d\}$.

2. Group 2 include the prefix of some suffix of $\{e, a, c, d\}$: $\{a\}$, $\{a, c\}$ and $\{a, c, d\}$ (these 3 patterns are $\{a, c, d\}$'s prefix); $\{c\}$ and $\{c, d\}$ (these 2 patterns are $\{c, d\}$'s prefix); $\{d\}$ (it is $\{d\}$'s prefix);

3. Group 3 includes 5 itemsets: $\{e, a, d\}$, $\{e, c\}$, $\{e, c, d\}$, $\{e, d\}$ and $\{a, d\}$.

When $\{e, a, c, d\}$ is inserted into P-tree initially, only the counts of $\{e\}$, $\{e, a\}$, $\{e, a, c\}$, $\{e, a, c, d\}$ are registered correctly. Note that they are all the prefixes of $\{e, a, c, d\}$, which belong to the Group 1. Let R denote the root of P-tree, $T_2 = R$, and $T_1 = R.next[i]$, where $R.items[i] = e$. After calling `merge_tree`(T_1, T_2), that is, merging the subtree $R.next[i]$

Algorithm 4.4: Function: mine_tree(T, α) (for mining EJEPs)

/* T is a subtree of the P-tree and α is an accumulating itemset */

/* ξ is a minimum support threshold for EJEP */

/* min-positive = $|\mathcal{D}_1| \times \xi$ */

/* $|\mathcal{D}_1|$ = the total number of instances in $|\mathcal{D}_1|$ */

/* min-negative = $|\mathcal{D}_2| \times \xi$ */

/* $|\mathcal{D}_2|$ = the total number of instances in $|\mathcal{D}_2|$ */

1 **foreach** *item of* T, $T.items[i]$ **do**

2 **if** $T.items[i]$*'s subtree* M *is not empty* **then** $merge(M, T)$;

3 $\beta = \alpha \cup T.items[i]$;

/*Note that EJEPs of both $\mathcal{D}_1$ and $\mathcal{D}_2$ are generated at the same time in a ''single scan''. Unlike previous EP mining methods, we do not need to construct one P-tree for mining EJEPs of $\mathcal{D}_1$, and construct another P-tree for mining EJEPs of $\mathcal{D}_2$. */

4 **switch** *conditions* **do**

5 **case** $T.positive\text{-}counts[i]=0 \wedge T.negative\text{-}counts[i] \geqslant$ *min-negative*

generate an EJEP β of $\mathcal{D}_2$ with $supp(\beta)$ = T.negative-counts[i]

6 **case** $T.negative\text{-}counts[i]=0 \wedge T.positive\text{-}counts[i] \geqslant$ *min-positive*

generate an EJEP β of $\mathcal{D}_1$ with $supp(\beta)$ = T.positive-counts[i]

/*Go deeper searching for longer EJEPs */

7 **case** *($T.items[i]$'s subtree N is not empty)* $\wedge$ *($T.positive\text{-}counts[i] \geqslant$ min-positive* $\vee$ $T.negative\text{-}counts[i] \geqslant$ *min-negative)*

call mine-tree (N, β)

end

/*The whole subtree of $T.items[i]$ has been processed and the useful information about item counts has been recorded in the other parts of the P-tree by $merge(M, T)$ */

8 **delete** $T.items[i]$'s subtree;

end

Algorithm 4.5: Function: merge_tree(T_1, T_2) (for mining EJEPs)

```
/* Given two subtrees of P-tree, T1 and T2, the function merges T1's
   nodes into T2 */
/* T2 is updated (including new-node generation and existing-node
   changes, but no nodes deletion), while T1 remains unchanged */
1  foreach item of T1, T1.items[i] do
2     search T2 for T2.items[j] = T1.items[i];
3     if T2.items[j] found then
4        T2.positive-counts[j] = T2.positive-counts[j] + T1.positive-counts[i];
5        T2.negative-counts[j] = T2.negative-counts[j] + T1.negative-counts[i];
      end
      else
         /*This ensures all items in T2 are always ordered by ≺ */
6        insert T1.items[i] with its both positive and negative counts and childnode
         (T1.child-nodes[i]) at the right place in T2, denoted as T2.items[j];
7        increment T2.item-number by 1;
      end
8     if T1.items[i]'s subtree, M is not empty then
9        if T2.items[j]'s subtree is empty then
10          create a new node N with N.item-number=0 as T2.items[j]'s subtree;
         end
11       else N ← T2.items[j]'s subtree;
12       call merge-tree (M, N);
      end
   end
```

with R, $\langle a, c, d\rangle$ is merged to the paths from the root. So the counts of $\{a\}$, $\{a, c\}$ and $\{a, c, d\}$ ($\{a, c, d\}$'s prefix) will become correct when merging recursively. Similarly, the counts of $\{d\}$ and $\{d, e\}$ ($\{d, e\}$'s prefix), and the counts of $\{e\}$ ($\{e\}$'s prefix) will be correct later on. These itemsets belongs to the Group 2. For those itemsets in the Group 3, we use $\{e, d\}$ as an example to illustrate the idea. Not only $\{e, a, c, d\}$, but also $\{e, b, c, d\}$ and $\{e, d\}$ contribute the count of $\{e, d\}$. By merging the subtrees of $R.e.a$, $R.e.b$ and $R.e.c$ into $R.e$, we accumulate all the counts of d, hence deriving the correct count for $\{e, d\}$.

4.5 The EJEP-Classifier: Classification by Aggregating EJEPs

To handle the general case where the training dataset contains more than two classes, we adopt the concept of pair-wise features (Li, Dong & Ramamohanarao 2001). For a dataset $\mathcal{D}$ with q classes: $\mathcal{D}_1$, $\mathcal{D}_2$, $\cdots$, $\mathcal{D}_q$, we discover q groups of EJEPs : those of $\mathcal{D}_1$ over $\cup_{j=2}^{q}\mathcal{D}_j$, those of $\mathcal{D}_2$ over $\cup_{j=1,j\neq 2}^{q}\mathcal{D}_j$, $\cdots$, those of $\mathcal{D}_q$ over $\cup_{j=1}^{q-1}\mathcal{D}_j$.

To classify a test instance T, we let all the EJEPs of a data class $\mathcal{D}_i$ that are subsets of T contribute to the final decision of whether T should be labelled as $\mathcal{D}_i$.

Definition 4.3 Given a test instance T and a set $E(\mathcal{D}_i)$ of EJEPs of a class $\mathcal{D}_i$ discovered from the training data, the aggregate score (or score) of T for ($\mathcal{D}_i$)is defined as

$$score(T, \mathcal{D}_i) = \sum_{e\subseteq T, e\in E(\mathcal{D}_i)} supp(e).$$

Note that the above definition is similar to the scoring function of the JEP-Classifier (Li, Dong & Ramamohanarao 2001) (Chapter 3), except EJEPs rather than JEPs used in aggregating supports.

Usually the numbers of EJEPs for different classes are not balanced, so, if a class of data $\mathcal{D}_1$ contains many more EJEPs than another $\mathcal{D}_2$, a test instance tends to get higher scores for $\mathcal{D}_1$ than for $\mathcal{D}_2$, even when the test really belongs to $\mathcal{D}_2$ (this leads to misclassification of the test). We adopt the solution in (Dong, Zhang, Wong & Li 1999) by "normalizing" the scores. A base score for each class $\mathcal{D}_i$, $base\_score(\mathcal{D}_i)$, is first found from

the training instances of the class. We select $base\_score(\mathcal{D}_i)$ so that exactly 50%-80%[2] of the training instances of $\mathcal{D}_i$ have scores higher than $base\_score(\mathcal{D}_i)$. The normalized score of an instance T for $\mathcal{D}_i$, $norm\_score(T, \mathcal{D}_i)$, is defined as the ratio of the score and the base score, that is

$$norm\_score(T, \mathcal{D}_i) = \frac{score(T, \mathcal{D}_i)}{base\_score(\mathcal{D}_i)}.$$

We let the class with the highest normalized score win. Ties are broken by letting the class with the largest population win.

4.5.1 Feature Reduction for the EJEP-Classifier

Real, large datasets usually contain irrelevant, correlated, and redundant data. Out of a large set of possible attributes, only a small subset is necessary to model the performance of a data-mining algorithm accurately. Extraneous features (attributes) can make it hard to detect insightful patterns. So the technique of feature subset selection is usually used to identify and remove as much irrelevant and redundant information as possible, allowing learning algorithms to operate faster and more effectively (Hall 1999). The resulting classifier is a high-fidelity, yet simple and comprehensible model.

Our P-tree based algorithm is able to generate the complete set of EJEPs. However, since EJEPs are proposed for classification, the completeness for EJEP-Classifier is not so important here. Accuracy and speed are two important factors for classification. So, our aim is to make the EJEP-Classifier run faster in both training and testing phases, without any decrease in accuracy.[3] Recall that the *support ratio* (see Definition 4.2) evaluates how sharp power individual items have between $\mathcal{D}_1$ and $\mathcal{D}_2$. After discretization, items can be regarded as features or attributes. Because EJEPs are intended to catch differences between classes of data, intuitively, EJEPs made up of items with larger *support ratio* are generally more important and more reliable. Consider an extreme case that an item appears frequently in one class but does not appear in another class at all, thus having infinite *support ratio*. The singleton itemset containing only the item with infinite support ratio is

[2] The percentage is called *normalization percentage*. The value of 50%-80% is suggested by (Dong, Zhang, Wong & Li 1999).

[3] In our experiments, we have observed both reduced time and improved accuracy on some datasets.

an EJEP.

We adopt a simple feature reduction technique. Since we are interested in EJEPs containing items with large support ratios, we select only those items before performing EJEP mining. In other words, we select a certain amount of top ranking items (features), where the ranking is measured by the support ratios (Definition 4.2) of items - the larger the support ratio, the higher the ranking. Suppose that originally we have 100 items. We select a subset of those items, after removing 10 items with the lowest rankings. This reduces the search space from 2^{100} down to 2^{90}, which is 1024 times smaller. An algorithm that searches the reduced space may potentially be 1024 times faster.

We use the following procedure to perform feature reduction for EJEP-Classifier.

1. After the first scan of the training database (containing two classes[4]), calculate the support ratios for each item in the dataset.

2. Sort all the item in the support-ratio-descending order ($\prec$).

3. Select the top k (e.g., 90%) items. Denote the support ratio of the k-th item as `min-ratio`.

4. During the construction of P-tree, filter those items whose support ratios are below `min-ratio` (only top k items is retained)[5]. Note that because every instance is sorted by $\prec$ before inserting into P-tree, such filter is easy to perform, i.e., just to truncate the instance by cutting off uninteresting items in the rear.

How to determine k? Because different datasets contain different number of items (after discretization), it is not wise to use a constant k for them all. Equivalently, we specify what percentage of the items we keep. We call the percentage `item-reduction-percentage`. A value of 90% means that we keep the top 90% items while removing the lowest 10%. We will in Section 4.6.1 discuss the setting of `item-reduction-percentage` as well as the minimum support threshold ξ.

[4]Note that multi-class problems can be transformed into two-class problems by the technique of pair-wise features discussed in Section 4.5.

[5]Basically, this equals to the **project** operation in relational algebra.

Experimental results show that we grasp most of the "good" EJEPs if not all, while "bad" EJEPs which we discard play no role or little role in classification. Therefore, the above feature reduction approach is reasonable.

4.6 Experimental Results

In this section, we report the experimental results. We first study the properties of our EJEP-based classification approach, EJEP-Classifier (EJEP-C). We then compare EJEP-C against three popular classification methods: Naive Bayes (NB), C4.5 (Quinlan 1993) and CBA (Liu et al. 1998), and two JEP/EP-based classifiers: CAEP (Dong, Zhang, Wong & Li 1999) and JEP-Classifier (JEP-C) (Li, Dong & Ramamohanarao 2001). All the experiments were performed on a 500Mhz Pentium III PC with 512Mb of memory. The accuracy was obtained by using the methodology of ten-fold cross-validation. For datasets containing continuous attributes, they are discretized using the Entropy method described in (Fayyad & Irani 1993). The code is taken from the $\mathcal{MLC}$++machine learning library (Kohavi et al. 1994).

4.6.1 Parameter Setting for EJEP-Classifier (EJEP-C)

Our EJEP-Classifier(EJEP-C) has two parameters: the minimum support threshold ξ and the percentage of top ranking items used for mining EJEPs, `item-reduction-percentage`.

It is relatively easy to determine ξ. On one hand, if ξ is too small, we may not be able to eliminate useless JEPs. Particularly, if $\xi = 1$ in absolute count, then EJEPs become the minimal JEPs, some of which are not useful due to small supports. On the other hand, if ξ is too large, we may not be able to find enough EJEPs to cover[6] most of the training dataset. From Figure 4.4, we can see that the coverage on the training dataset remains stable once ξ drops to 1%, i.e., EJEPs with smaller supports can not cover more training instances. So we set $\xi = 1\%$, or $\xi = 5$ (in absolute count)[7], whichever is larger. This setting

[6] Here we say an EJEP covers a training instance if the instance contains the EJEP.

[7] If a class of training instances has less than 500 instances, then the minimum support threshold for this class will be less than 5, which is regarded statistically unreliable.

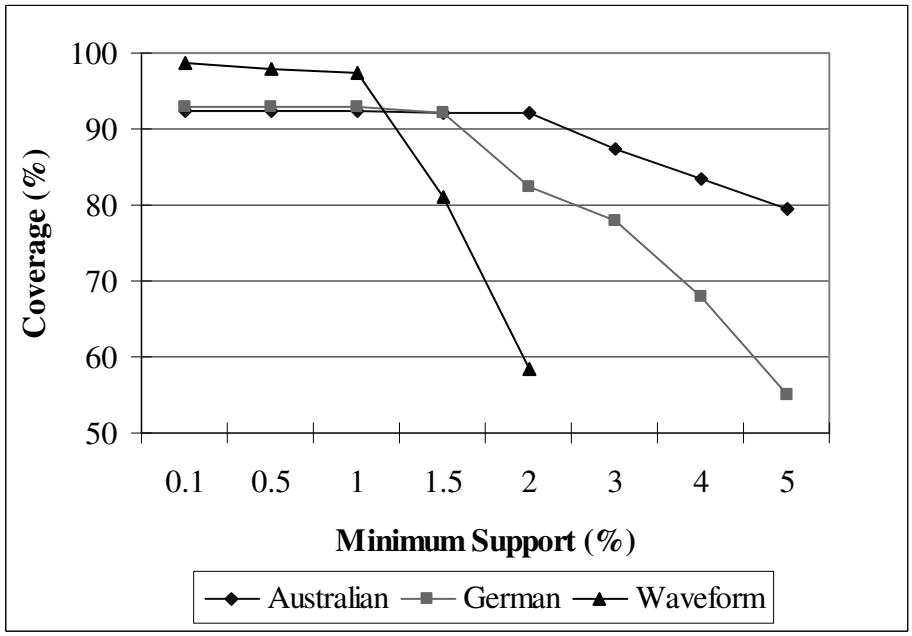


Figure 4.4: EJEP-C: the effect of minimum support vs. the coverage on training data

for the minimum support threshold is similar to (Liu et al. 1998), which sets 1% as the the minimum support threshold for class association rules.

How to choose `item-reduction-percentage`, or equivalently `min-ratio`? We conduct experiments on the UCI German, Ionosphere, Sonar and Waveform datasets to see the effect of `item-reduction-percentage` on classification accuracy. The results are shown in Figure 4.5. It can be seen that (1) accuracy goes up when `item-reduction-percentage` goes up; (2) after a certain point of percentage, accuracy remains relatively stable when `item-reduction-percentage` goes up. It suggests that 90% is a good choice in general. `min-ratio` is chosen in the way that around 90% items are above `min-ratio` and 10% items below it. In practice, `min-ratio` is between 1.1 and 1.5.

In our experiments, we have observed less training time and fewer EJEPs by setting `item-reduction-percentage`=90%. For example, for the UCI Waveform dataset, it takes about 40 minutes to mine EJEPs before applying feature reduction, while only around 20 minutes after removing 10% items with the lowest ranking. The resulting EJEP-based classifier for Waveform is also less complex: the number of EJEPs drops from 8,324

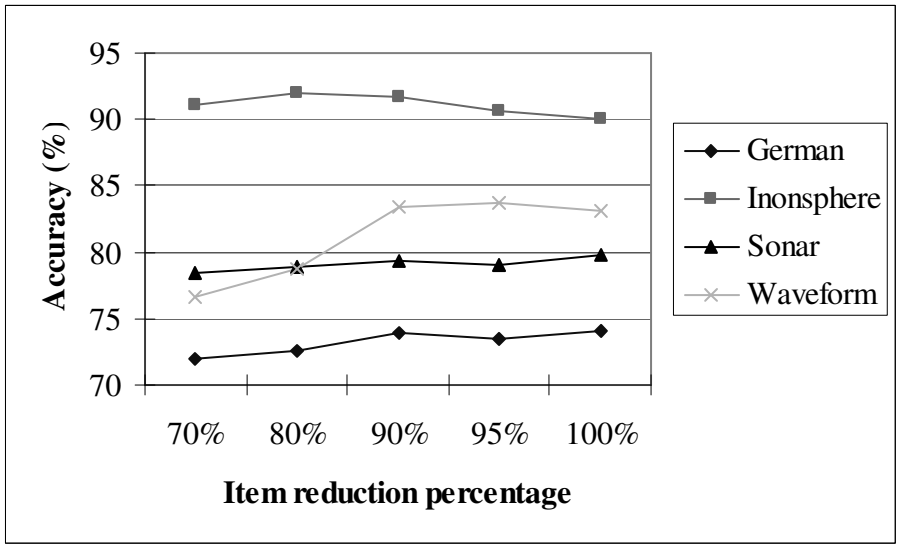


Figure 4.5: EJEP-C: the effect of `item-reduction-percentage` on classification accuracy

to 7,083. The reason behind the above phenomenon is as follows. Our P-tree structure is usually compact by sharing items, and it becomes more compact by applying feature reduction technique. Due to the removal of uninteresting information for EJEP mining, our P-tree based mining algorithm runs faster and generates only important EJEPs.

JEP-Classifier (JEP-C) does not need any parameter. This can be seen as an advantage, which makes the JEP-classifier easier to use. On the other side, having no parameter is a disadvantage, because using the minimum support threshold can give us more control on the quality of patterns, and using `min-ratio` can make us train the classifier faster without sacrificing accuracy by removing irrelevant items (features).

4.6.2 Accuracy Comparison

Table 4.4 summarizes the accuracy results. The first column gives the name of each dataset. (The detailed description of datasets such as the number of instances, attributes and classes, can be found in Appendix.) Columns 2 to 7 give the predictive accuracy of the six classifiers, calculated as percentage of testing instances that are correctly classified. The last column gives the ratio of the accuracy of EJEP-C over JEP-C. The accuracy of NB and C4.5 are obtained using the WEKA implementation (Witten & Frank 1999). The accuracy

of CAEP and JEP-C are obtained using the implementation of (Bailey et al. 2002) and (Dong, Zhang, Wong & Li 1999), respectively. The same dataset partition and exactly the same training data and testing data are used for NB, C4.5, CAEP, JEP-C and our EJEP-C. The accuracy of CBA is quoted from (Liu et al. 1998) as an indirect comparison, because different dataset partition could be used.

We highlight some interesting points observed from Table 4.4 as follows:

- EJEP-C performs perfectly on some datasets: it achieves 98%-99.99% on the UCI Hypo, Mushroom and Tic-tac datasets.
- In comparison with CAEP, EJEP-C wins on 11 datasets while CAEP wins on 5 (they tie once). Sometimes EJEP-C beats CAEP by a comfortable margin on datasets such as Cleve and Tic-tac.
- In comparison with JEP-C, EJEP-C maintains accuracy very close to JEP-C on most datasets, as indicated by the accuracy ratio of EJEP-C over JEP-C, the lowest being 97.85%. Sometimes EJEP-C achieves higher accuracy on datasets such as Cleve, Crx, Heart, Horse, Hypo, Pima and Waveform. On average EJEP-C achieves almost the same accuracy as JEP-C, the absolute accuracy difference being less than 0.3%.
- In comparison with NB, EJEP-C wins on 11 datasets while NB wins on 6. Compared with C4.5, EJEP-C wins 10 on datasets while C4.5 wins on 5 (they tie twice). As an indirect comparison, EJEP-C wins on 11 datasets while CBA wins on 6.
- We also point out that EP-based classifiers (CAEP, JEP-C and EJEP-C) are more accurate than NB, C4.5 and CBA in general.

4.6.3 Other Comparison

We compare the number of JEPs, CARs and EJEPs used in JEP-C, CBA and EJEP-C respectively. The results are shown in Figure 4.6. The runtime (including both training and testing phases) of JEP-C and EJEP-C are also compared, as shown in Figure 4.7 and 4.8.

Table 4.4: Accuracy Comparison

	Accuracy(%)						
Dataset	NB	C4.5	CBA	CAEP	JEP-C	EJEP-C	Ratio
Australian	77.06	84.56	84.10	85.51	85.94	85.79	99.83%
Cleve	82.76	76.55	76.80	79.86	84.19	84.82	100.75%
Crx	77.94	86.03	84.00	86.09	85.94	86.09	100.17%
Diabete	75.66	74.47	74.10	73.83	76.04	75.39	99.15%
German	75.10	71.80	74.80	74.50	74.60	73.90	99.06%
Heart	83.33	74.81	80.00	82.22	82.59	82.96	100.45%
Horse	78.61	85.56	82.40	82.03	83.15	83.70	100.66%
Hypo	97.91	99.30	98.20	96.49	97.31	98.29	101.01%
Iono	82.35	91.74	92.10	89.76	92.31	91.74	99.38%
Labor	91.23	84.21	83.00	91.23	89.47	87.72	98.04%
Mushroom	95.78	100.00	100.00	97.17	100.00	99.99	99.99%
Pima	76.05	75.79	73.10	77.60	73.03	74.22	101.63%
Sonar	67.50	72.20	78.30	78.33	80.29	79.33	98.80%
Tic-tac	69.79	97.68	100.00	89.66	99.16	98.80	99.64%
Vehicle	61.12	70.85	70.50	64.30	67.32	66.08	98.16%
Waveform	80.96	78.10	79.60	83.92	82.89	83.30	100.49%
Zoo	93.07	90.10	93.40	92.08	92.08	90.10	97.85%
Average	80.37	83.16	83.79	83.80	85.08	84.84	99.72%

Note that the detailed description of datasets can be found in Appendix.

The last column "Ratio" refers to the ratio of the accuracy of EJEP-C over JEP-C. A value greater than 100% means that EJEP-C achieves higher accuracy than JEP-C; while a value very close to 100% means that both accuracy are roughly the same.

We draw the following points from these comparisons.

- From Figure 4.6, we can see that EJEP-C always uses much fewer EJEPs than JEPs and CARs. For example, for UCI German dataset, there are 1,137 EJEPs. In contrast, there are 32,510 JEPs and 69,277 CARs. The reduction rates are about 30 and 60, respectively.

- From Figure 4.7 and Figure 4.8, we can see that the running time of EJEP-C is much shorter than JEP-C. The speedup ranges from 2 to 200 times. Because the most time-consuming part for both classifiers is the learning phase, i.e., mining EJEPs or JEPs, it follows that our mining algorithm is much faster than the previous border-based approach.

4.6.4 Summary of Comparison

We summarize what experimental results show as follows:

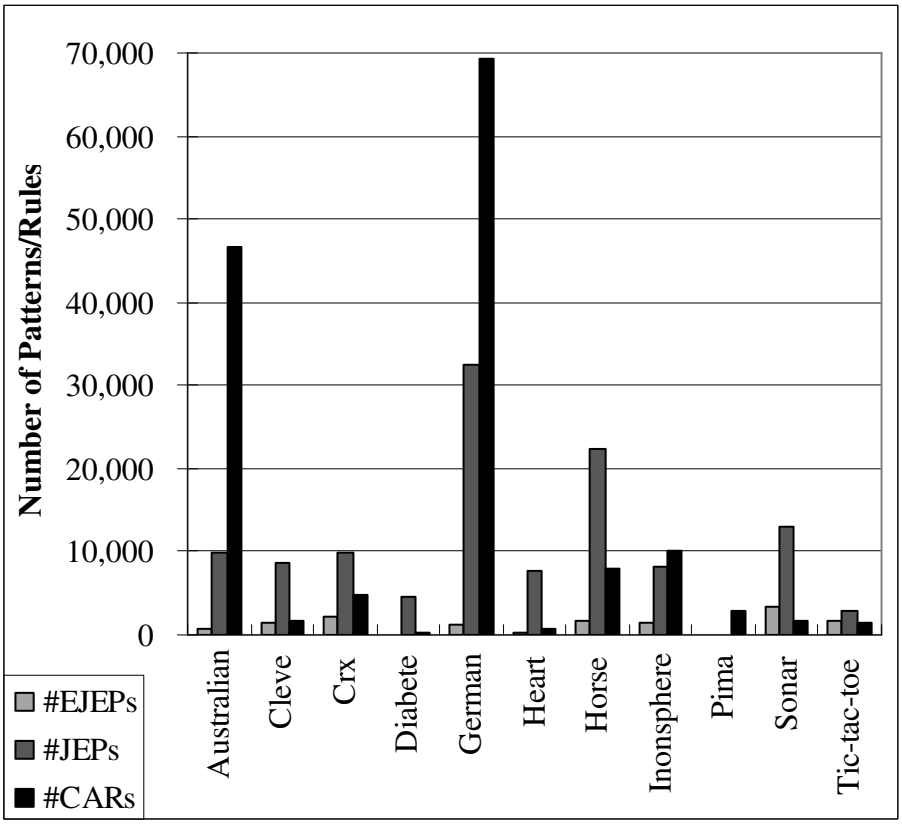
- As a classifier based exclusively on EJEPs, EJEP-C achieves almost the same (sometimes higher) accuracy as JEP-C and it is often also superior to other state-of-the-art classification systems such as NB, C4.5 and CBA.

- EJEP-C is less complex than JEP-C and CBA, because it uses much fewer EJEPs.

- EJEP-C is much faster than JEP-C. Thus EJEP-C can be used effectively for large classification problems.

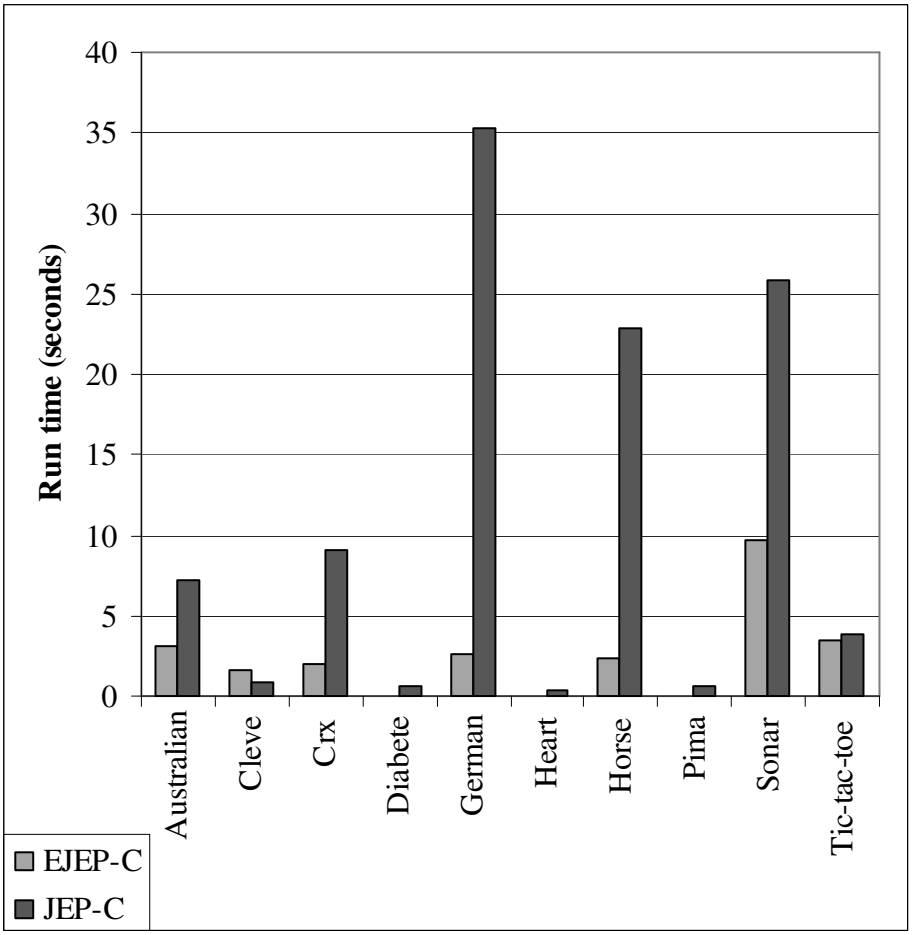
4.7 Related Work

Although JEPs have been proved to have strong discriminating power by JEP-Classifier (Li, Dong & Ramamohanarao 2001), which aggregates the differentiating power of boundary JEPs for classification, some JEPs contain noise. For example, an itemset which "appears"once in one data class but none in another is a JEP. Obviously, such a JEP is not very useful in classification. Mining such JEPs is very time-consuming and using them in classification may even lower accuracy.



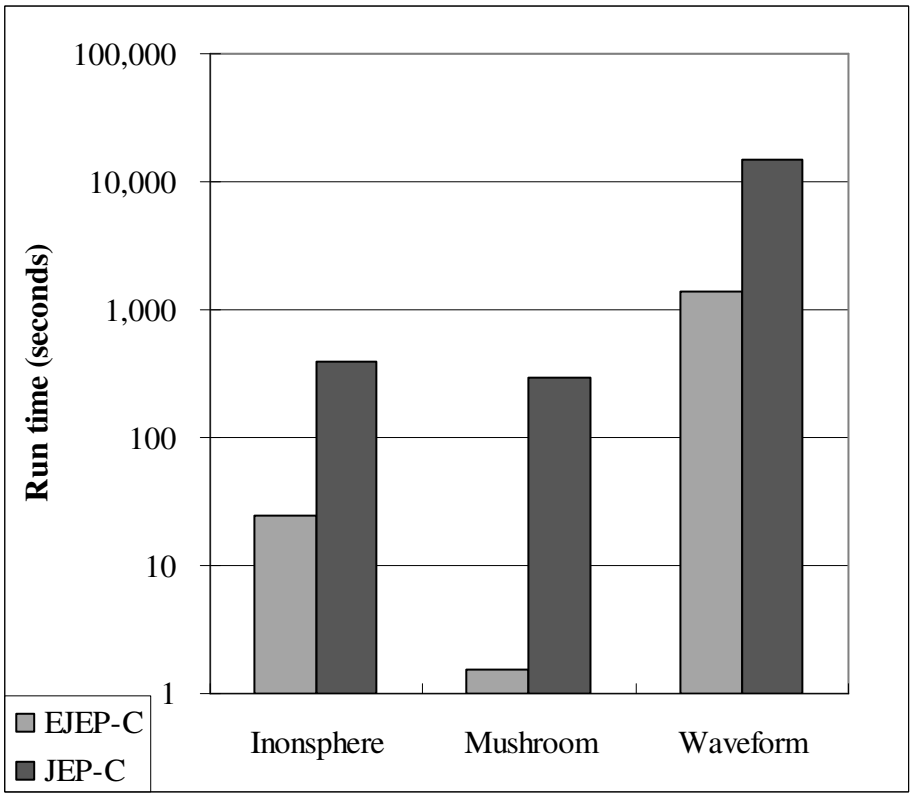
The complexity is measured in terms of number of rules or patterns used in classification. Our EJEP-Classifier (EJEP-C) uses EJEPs, JEP-Classifier (JEP-C) uses JEPs, and CBA uses rules called CARs. Note that for Diabete dataset, both the number of EJEPs and the number of CARs are too small to draw the corresponding columns for EJEP-C and CBA; for Pima dataset, the number of EJEPs and JEPs are too small to draw the corresponding columns for EJEP-C and JEP-C

Figure 4.6: Comparison of classifier complexity: EJEP-C vs. JEP-C vs. CBA



Note that the runtime reported here includes both the time of training classifiers and classification time.

Figure 4.7: Comparison of classifier runtime: EJEP-C vs. JEP-C (part `I`)



Note that the Y axial (Runtime in seconds) is in logarithm scale.

Also note that the runtime reported here includes both the time of training classifiers and the classification time.

Figure 4.8: Comparison of classifier runtime: EJEP-C vs. JEP-C (part `II`)

JEP-Classifier utilizes border-based algorithms (Li et al. 2000) to discover the border description of all JEPs and selects those JEPs in the left bound of a border to use in classification. Because such border representations does not contain actual support, further processing is needed to extract the embodied JEPs and their supports. The process is very slow, especially for databases with a large number of items. It is reported that for the UCI Mushroom, Sonar, German and Ionosphere databases, it took up to two hours to build the classifier (Li, Dong & Ramamohanarao 2001).

Although the number of JEPs used by JEP-Classifier is greatly reduced by selecting the boundary JEPs, in some worst cases, there are still a huge number (it may reach $C^N_{N/2}$, where N is the number of attributes in the dataset) of JEPs. For the UCI German database, it uses 32,510 JEPs. Such a large number of JEPs make the classifier complex and beyond understanding.

4.8 Chapter Summary

In this chapter, we have introduced a special type of Emerging Pattern, called Essential Jumping Emerging Patterns (EJEPs) and shown that EJEPs have the strongest discriminating power and are sufficient to build highly accurate classifiers in many real life applications. We have also presented an efficient "single-scan", pattern-growth algorithm based on the Pattern tree structure to generate EJEPs of both classes of data at the same time. Experiments on databases from the UCI machine learning database repository show that the classifier based on EJEPs is consistent, highly effective at classifying of various kinds of databases and has better average classification accuracy in comparison with CBA and C4.5, and is more efficient and scalable than previous EP-based classifiers.

Chapter 5

Efficiently Mining Interesting Emerging Patterns

An important goal of Knowledge Discovery in Databases (KDD) is to "turn data into knowledge". Emerging Patterns (EPs) are sets of items whose frequency change significantly from one dataset to another. As one type of knowledge pattern, Emerging Patterns can be easily understood and used directly by people because they are expressed in terms of features (conjunctions of attribute values). They are useful as a means of discovering distinctions inherently present amongst a collection of datasets and have been shown to be a powerful method for constructing accurate classifiers (Dong, Zhang, Wong & Li 1999, Li, Dong & Ramamohanarao 2001, Li, Dong, Ramamohanarao & Wong 2004). The idea of Emerging Patterns is also applied in bioinformatics successfully, from the discovery of gene structure features to the classification of gene expression profiles (Li, Liu, Ng & Wong 2003, Li, Liu, Downing & A. Yeoh 2003).

In Chapter 4, we have discussed Essential Jumping Emerging Patterns (EJEPs) and a specific algorithm for mining EJEPs. Although EJEPs are very useful, they belong to a special type of Emerging Patterns and only represent a small fraction of all useful Emerging Patterns.

In this chapter, we first introduce the interestingness measures for Emerging Patterns, including the minimum support, the minimum growth rate, the subset relationship

between EPs and the correlation based on common statistical measures such as a chi-squared value[1]. We then develop an efficient algorithm for mining only those interesting Emerging Patterns (called Chi EPs), where the chi-squared pruning heuristic is used to reduce the search space greatly. The experimental results show that our algorithm maintains efficiency even at low supports on data that is large, dense and has high dimensionality. They also show that the heuristic is admissible, because only unimportant EPs with low supports are ignored. Our work based on EPs for classification confirms that the discovered interesting EPs are excellent candidates for building accurate classifiers.

5.1 Motivation

Previous work mainly concentrated on the problem of mining a special type of Emerging Pattern, called Jumping Emerging Pattern (JEP) (Li, Dong & Ramamohanarao 2001, Bailey et al. 2002, Fan & Ramamohanarao 2002). A JEP is defined as an itemset whose support increases abruptly from zero in one dataset, to non-zero in another dataset – the ratio of support increase being infinity. Minimal JEPs are those boundary JEPs whose proper subsets are no longer JEPs. There are usually a huge number of JEPs contained in a dataset of two classes. For example, it is estimated that the UCI mushroom dataset contains more than 20 million JEPs. It has been shown (Li, Dong & Ramamohanarao 2001, Bailey et al. 2002, Fan & Ramamohanarao 2002) that the set of minimal JEPs is much smaller than all JEPs, e.g., compared with tens of millions of JEPs, there are only 3,461 minimal JEPs in the UCI mushroom dataset. It has also been shown (Li, Dong & Ramamohanarao 2001, Bailey et al. 2002, Fan & Ramamohanarao 2002) that minimal JEPs are the most interesting and the most useful for classification. For instance, if an itemset $\{a, b\}$ is a JEP of positive class with zero-support in negative class, the itemset $\{a, b, c\}$ will remain to be a JEP as long as its positive support is above zero. But $\{a, b, c\}$ is less useful for classification than $\{a, b\}$ since it covers less or the same training instances and has the same infinite growth rate.

Minimal JEPs are "correlated" in the sense that the distribution of its subset is

[1]Chi-squared value is typically used to test independence or correlation/association because of its solid grounding in statistics.

significantly different from that of the JEP itself. Although JEPs represent very strong discriminating knowledge between different classes, EPs with finite growth rates (called general EPs), i.e., patterns which are *frequently* contained in one class of data but *less frequently* (not zero) contained in the other class, are also interesting. For example, in a loan application domain, out of 500 persons who applied for loans in a bank, 280 persons are granted loan (positive class) and 220 are not (negative class); 200 persons have jobs in positive class and 100 in negative class. The itemset {Job=yes}, since its support in positive and negative classes are 200/280=71.43% and 100/220=45.46% respectively, is a general EP from negative to positive class with a grow rate of 1.57. From the view of associations, we can obtain an association rule:

{Job=yes} $\rightarrow$ loan=approved [support=200/500, confidence=200/300].

The rule has strong prediction power since its support is large and confidence high. If only JEPs are used in classification, some useful patterns may not be captured. Furthermore, in real world classification problems, some data classes may contain few or even no JEPs whose supports meet a reasonable threshold (such as 1%). So EPs with moderate growth rates are useful in classification.

However, for general EPs with finite growth rates, minimal ones are not always the most interesting. For instance, if itemset $\{a, b\}$ is an EP with a growth rate of 2, and $\{a, b, c\}$ is an EP with a growth rate of 100, $\{a, b, c\}$ is more interesting because its prediction power, which is measured by its growth rate, is much stronger, although its supports in both classes may be smaller. This makes the problem of mining general EPs even harder than that of mining JEPs: even if a pattern is found to be an EP, unlike mining minimal JEPs, we still have to check its super patterns. Indeed, the number of EPs with moderate growth rates is far more than that of JEPs since the conditions of general EPs are weaker.

Our work is motivated by the goal of generalizing interesting JEPs to interesting EPs. An interesting EPs should also be "correlated" in the sense that the distribution of its subset is *significantly* different from that of itself. Although minimal EPs are not always interesting as discussed earlier, we prefer the shorter one when two EPs have roughly the same growth rates, i.e., any subset of an interesting EP will have smaller growth rate than the EP.

5.2 Introduction

Previous EP mining approaches often produce a large number of EPs, which makes it very difficult for domain experts to choose interesting ones manually. Usually, a post-processing filter step is applied for selecting interesting EPs based on some interestingness measures. There are also difficulties involved in the use of EP for classification. Through our experience of building EP-based classifiers, it has been recognized that of the large number of EPs that EP mining algorithms generate, most are actually of no interest for modelling nor use for classification purpose. Generating a lot of uninteresting EPs not only makes the mining algorithm inefficient, but also makes the classifiers based on these EPs very complex, slow, and hard to interpret. Therefore, our aim is to efficiently mine only those EPs that are interesting and useful for classification. Being able to mine only useful EPs is very important, because it can save mining of many unnecessary EPs and identifying interesting ones from a huge number of candidate EPs.

5.2.1 Interesting Emerging Patterns

What makes Emerging Patterns interesting? The measures of interestingness are divided into objective measures - those that depend only on the structure of a pattern and the underlying data used in the discovery process, and the subjective measures - those that also depend on the class of users who examine the pattern (Silberschatz & Tuzhilin 1996). Deciding what is interesting is an open problem in data mining. In this chapter, we propose to define the interestingness of Emerging Pattern in objective terms. An EP is interesting, if it satisfies all the following conditions:

1. it has a support greater than a minimum threshold ξ in its home class[2];
2. it has a growth rate greater than a minimum growth rate threshold ρ;
3. it has larger growth rate than its subsets;
4. it is highly correlated according to common statistical measures such as chi-square value.

[2]An EP has higher support in its home class than its contrasting class.

Patterns satisfying the above four conditions are called Chi Emerging Patterns (Chi EPs in short). The first condition ensures that a Chi EP should generalize well on unseen instances by imposing a minimum coverage on the training dataset. The second requires that a Chi EP should have sharp discriminating power as indicated by its large growth rate. The third regards those "minimal" EPs as interesting, because if any subset of an EP has larger growth rate, the EP itself is not so useful for classification than the subset. The last condition uses chi-square test (Bethea, Duran & Boullion 1995) for independence, which is based on the differences between the expected number of occurrences for a discovered attribute value combination and the observed number. The primary assumption is that these differences will be less for independent attributes. A chi-square value that has a low probability of occurring strictly by chance leads to a rejection of the hypothesis that the attributes are independent. Attributes that are not independent are considered to be associated. In our EP context, generally speaking, the last condition states that for an EP to be interesting, the distribution (namely, the supports in two contrasting classes) of its immediate subset should be *significantly* different from that of the EP itself, where the difference is measured by the chi-square test. This also means that all the items of which a Chi EP is composed are not independent, because if the items were independent, those distributions would not be so different.

The four objective interestingness measures are used to restrict the resulting EP set and help further prune the search space and save computation. Experiments show that the set of Chi EPs is orders of magnitude smaller than the set of general EPs, thus providing the domain expert with a starting point for further subjective evaluation of these patterns.

In the case that a user wants to use EPs for classification, the subjective measure of EPs can be defined as their utility. To evaluate objective interesting EPs against the subjective measure, we have built classifiers using those EPs. High accuracy on benchmark datasets from the UCI Machine Learning Repository (Blake & Merz 1998) shows that mining EPs using our method can result in high quality EPs with the most differentiating power.

5.2.2 Efficient Discovery of Interesting Emerging Patterns

The task of mining EPs is very difficult for large, dense and high-dimensional datasets, because the number of patterns present in the datasets may be exponential in the worst case. What is worse, the Apriori anti-monotone property, which is very effective for pruning search space in frequent pattern mining, does not apply to EP mining. This is because the fact that a pattern with k items is not an EP, does not necessarily imply that its super-pattern with $(k+1)$ or more items is not an EP. Indeed, the mining of Emerging Patterns is harder than the mining of frequent patterns in this sense. Similar anti-monotone property dose not apply to Chi EPs either, i.e., we can not simply discard those itemsets which has subsets not satisfying the four interestingness measures. Therefore, it is a great challenge to efficiently mine only Chi EPs.

Recently, the merits of a pattern growth method such as FP-growth (Han et al. 2000), have been recognized in the frequent pattern mining. It is possible to use FP-growth to mine EPs: we first find frequent itemsets in one data class for a given support threshold, and then check the support of these itemsets against the other class. Itemsets satisfying the four interestingness measures are Chi EPs. However, there are several obvious difficulties with this approach:

1. a very large number of frequent patterns will be generated when the support is low;
2. a lot of frequent patterns in one class turn out not to be EPs since they are also frequent in the other class;
3. it selects interesting EPs as post-analysis.

For example, for the UCI mushroom dataset, when minimum support threshold is set to 1%, the total number of frequent patterns is 90,751,401; while the number of Chi EPs is only 2,606, which is around 30 thousand times smaller. It is a waste of time to first generate a huge number of patterns and then eliminate most of them.

To overcome these difficulties, we develop a novel pattern fragment growth mining method, called Interesting Emerging Pattern Miner (IEP-Miner), for efficiently extracting only the interesting Emerging Patterns (Chi EPs). IEP-Miner uses a tree structure to store

the raw data, which is similar to the so-called frequent pattern tree (FP-tree). It follows the pattern fragment growth mining paradigm: it recursively partitions the database into sub-database according to the patterns found and search for local patterns to assemble longer global one. However, because Emerging Patterns are different from frequent patterns as discussed above, the way that patterns grow for mining EPs is very different from that for frequent patterns. In fact, IEP-Miner searches the tree in the depth-first, top-down manner, as opposed to the bottom-up order of FP-growth. IEP-Miner operates directly on the data contained in the tree, i.e., no new nodes are inserted into the original tree and no nodes are removed from it during the mining process. This is the novelty of IEP-Miner and the key difference from FP-growth, which constructs conditional FP-trees recursively. The major operations of mining are counting and link adjusting, which are usually less expensive than the construction of conditional FP-trees.

The problem of mining EPs can be seen as to search through the power set of the set of all items for itemsets that are EPs. With low minimum settings on support and growth rate, the candidate EPs embedded in a high-dimensional database are often too numerous to check efficiently. We push the interestingness measures into the pattern growth to reduce the search space. We also use the chi-square test as heuristic to further prune the search space. The heuristic is admissible because (1) it greatly improves the efficiency of mining; (2) only EPs with the lowest supports are lost. Experiments show that IEP-Miner achieves high efficiency on large high-dimensional databases with low support and growth rate threshold, and successfully mines the top 90% Chi EPs.

In summary, IEP-Miner absorbs the advantages of pattern growth. By using objective interestingness measures as heuristic, the search space is greatly reduced and the efficiency of search greatly improved. IEP-Miner effectively controls the explosion of candidates and achieves high efficiency on large high-dimensional databases with low support and growth rate.

Organization: The remaining of this chapter is organized as follows. In Section 5.3, we introduce the four interestingness measures for Emerging Patterns. In Section 5.4, we discuss the chi-square pruning heuristic. Section 5.5 presents our algorithm for mining interesting Emerging Patterns (Chi EPs). In Section 5.6, we provide the performance study

of our algorithm. Section 5.7 discusses related work. Finally, remarks are given in Section 5.8.

5.3 Interestingness Measures of Emerging Patterns

We begin this section with a review of how the chi-square test is used to measure significance in applied statistics. We then formally define the four interestingness measures for Emerging Patterns, using the chi-square statistics. Lastly, we give a comparison of Chi Emerging Patterns (Chi EPs) with other types of Emerging Patterns.

5.3.1 Preliminary: Chi-square Test for Contingency

The chi-square test (χ^2-test) was developed to facilitate testing the significance of data in light of their observed scatter. It is the fundamental principle underlying all tests of significance. One of the most frequent uses of the χ^2-test is in the comparison of observed frequencies and the frequencies we might expect from a given theoretical explanation of the phenomenon under investigation. The χ^2-test statistics used in this case is defined by

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}, \tag{5.1}$$

where O_i is the obversed frequency, E_i the expected frequency, and k the number of classes. The random variable defined by Eq. 5.1 has an approximate χ^2-distribution with $(k-1)$ degrees of freedom.

The χ^2 statistics can be used in contingency testing, where n randomly selected items are classified according to two different criteria. For example, in quality control, it is desired to determine whether all inspectors are equally stringent. For testing the hypothesis that the outcomes are significant from such a series of binomial (yes or no) populations, the χ^2-test is a very popular procedure.

Example 5.1 Consider the case where pressure gauges are being hydraulically tested by three inspectors prior to shipment. It has been noted that their acceptances and rejections for some period of time have been as follows:

	Inspector A	Inspector B	Inspector C	Total
Passed	300	200	100	600
Failed	40	40	20	100
Total	340	240	120	700

To test the hypothesis H_0: all inspectors are equally stringent, the test statistic used is

$$\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{3} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}, \tag{5.2}$$

where O_{ij} are the number observed to be passed by the jth inspector and E_{ij} are the number expected to be passed by the jth inspector. □

In general, a contingency table will have r rows and c columns. The statistic is then

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{ij} - E_{ij})^2}{E_i}, \tag{5.3}$$

which has an approximate χ^2-distribution with $(r-1)(c-1)$ degrees of freedom. The statistic is used to test whether one attribute (corresponding to row) is contingent on the other attribute (corresponding to column) or whether the two attributes are independent. The closer the observed frequencies are to the expected frequencies, the greater is the weight of evidence in favor of independence.

To find the expected number for a given cell, simply multiply the corresponding row and column totals together and divide by the total number of items selected. This is done under the hypothesis that the attributes are independent and the probability p_{ij} that an item selected at random falls in the class corresponding to the ith row and jth column is $p_{i\cdot}p_{\cdot j}$, where $p_{i\cdot}$ is the probability that the item falls in the ith row and $p_{\cdot j}$ is the probability that the item falls in the jth column. A χ^2 value of 0 implies that the attributes are statistically independent. If it is higher than a certain threshold value, we reject the independence assumption. The expected contingency table for our example is as follows:

	Inspector A	Inspector B	Inspector C	Total
Passed	291.428	205.714	102.857	600
Failed	48.571	34.286	17.143	100
Total	340	240	120	700

Thus the computed chi-square value is 3.4314, which is less than $\chi^2_{2,0.95} = 5.9915$ (at the 95% significance level when degrees of freedom is 2). So the hypothesis H_0 is accepted and we may say that no one inspector is more demanding than the other two.

5.3.2 Chi Emerging Patterns (Chi EPs)

We formally define the objective interestingness measures for Emerging Patterns.

Definition 5.1 We say that an itemset, X, is a Chi Emerging Pattern (Chi EP), if all the following conditions about X are true:

1. $supp(X) \geqslant \xi$, where ξ is a minimum support threshold;
2. $GR(X) \geqslant \rho$, where ρ is a minimum growth rate threshold;
3. $\forall Y \subset X, GR(Y) < GR(X)$;
4. $|X| = 1$ or $|X| > 1 \wedge (\forall Y \subset X \wedge |Y| = |X| - 1 \wedge chi(X,Y) \geqslant \eta)$, where $\eta = 3.84$ is a minimum chi-value threshold and $chi(X,Y)$ is computed using the following contingency table (Bethea et al. 1995)

	X	Y	$\sum row$
D_1	$count_{D_1}(X)$	$count_{D_1}(Y)$	$count_{D_1}(X) + count_{D_1}(Y)$
D_2	$count_{D_2}(X)$	$count_{D_2}(Y)$	$count_{D_2}(X) + count_{D_2}(Y)$
$\sum column$	$count_{D_1+D_2}(X)$	$count_{D_1+D_2}(Y)$	$count_{D_1+D_2}(X) + count_{D_1+D_2}(Y)$

Let us explain in detail the above four interestingness measures.

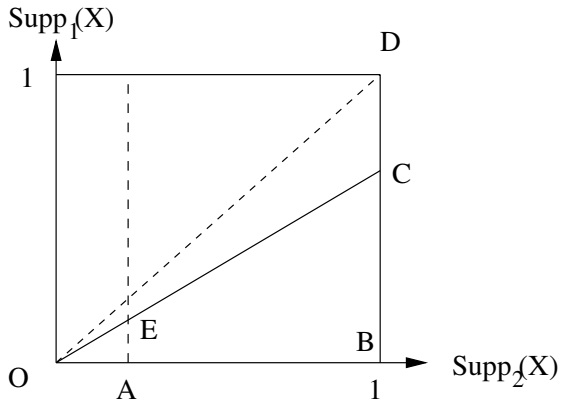
1. The first condition requires that a Chi EP should have minimum support in its home class, i.e., it should have "enough" coverage on the training dataset. Such an EP can generalize well in statistic sense on unseen instances of the same class, that is, if an EP appear frequently in one class, a testing instance belonging to the class will contain the EP with a certain probability, which is measured by the support of the EP in the class.

2. The second condition requires a Chi EP should have minimum growth rate, i.e., it should have "sharp" discriminating power. Such an EP can be regarded as a distinguishing feature of its home class, because it occurs *frequently* in its home class, while *infrequently* in the contrasting class (due to the large growth rate).

3. The third condition explores the subset relationship of EPs. Suppose we have two itemsets, denoted as Y and Z, which satisfy the first two conditions on minimum support and growth rate. If $Y \subset Z \wedge GR(Y) \geqslant GR(Z)$, then we can see that Z is not more useful for classification than Y. The reason behind this is three-fold: (1) Y contain less items (attributes) than Z; (2) The support of Y in its home class is more than or equal to Z; (3) Y has a growth rate no less than Z. Generally, we choose the simplest hypothesis consistent with the data. In our case, a shorter pattern Y is preferred as it also has enough support and large growth rate. If $\forall Y \subset X, GR(Y) < GR(X)$, i.e., we can not find such an itemset that it is a subset of X and its growth rate is higher than X, then we say that X is not "subsumed" by its subsets. X may be more useful for classification than Y, because X has higher growth rate, which means that it is a stronger signal indicating the class membership for unknown instances. Losing such strong signals may adversely affect the classification accuracy.

4. In the last condition, length-1 itemsets that satisfy the above three conditions pass chi-square test directly. For itemset X that satisfies the above three conditions and has length more than 1, a chi-square test is performed on X and any of its immediate subset. Only those itemsets that pass the chi-square test will be regarded as interesting. This condition requires that for any immediate subset of a Chi EP with length more than 1, its support distribution in both classes should be *significantly* different from that of the Chi EP itself. One can use other statistical measures such as the entropy gain, the gini index and the correlation coefficient in place of chi-square value. The bigger the value, the more confident we are to say that their distributions are different. We choose 3.84 as the minimum chi-value threshold, since it gives us 95% confidence when the degree of freedom[3] is 1, which is enough in many real-life applications. If a length-k EP's distribution is *significantly* different from that of any of its length-$(k-1)$ subsets, it shows that adding one item from length-$(k-1)$ subsets makes its behavior on two classes quite different. It also means that those items that

[3]As discussed in Section 5.3.1, the χ^2 statistic has $(r-1)(c-1)$ degrees of freedom, where r and c are number of rows and columns in the corresponding contingency table. In our case, $r = 2$ and $c = 2$. So our degree of freedom is 1.



Note: $\tan \angle BOC = \frac{1}{\rho}$, $\rho > 1$ and $|OA| = \xi$

Figure 5.1: The support plane for Emerging Patterns - Chi EP vs EP

make up of the EP, are highly correlated.

5.3.3 Comparison of Chi Emerging Patterns (Chi EPs) with Other Types of Emerging Patterns

The differences between Chi Emerging Patterns (Chi EPs) and other types of Emerging Patterns such as JEPs and EJEPs can be illustrated by Figure 5.1.

- The whole area of the triangle $\triangle OBC$ represents general Emerging Patterns (ρ-EPs), where only the growth rates of itemsets are concerned.

- JEPs occupy the x-axis from O to B, that is OB, where their support in $\mathcal{D}_1$ is strictly zero.

- EJEPs occupy the x-axis from A to B, that is AB, where their support in $\mathcal{D}_1$ is strictly zero and support in $\mathcal{D}_2$ is above ξ.

- We can only say that any Chi EP from $\mathcal{D}_1$ to $\mathcal{D}_2$ falls into the quadrangle $ABCE$, because $ABCE$ represents the set of EPs that satisfy the first two conditions on minimum support and growth rate. But the reverse is not true: a pattern inside $ABCE$ may not be interesting, because it may not pass conditions 3 and 4.

- Work in (Zhang et al. 2000a) uses a constraint called growth-rate improvement ($GRimp(e)$), which is defined as the minimum difference between the growth rate of an EP e and the growth rates of all its subsets, namely

$$GRimp(e) = \min(\forall e' \subset e, GR(e) - GR(e')).$$

  This is similar to our condition 3. However, we also use chi-square test (condition 4) to determine interesting EPs.

Although the quadrangle $ABCE$ is smaller than the triangle $\triangle OBC$, the space of $ABCE$ is still very large, containing several thousands of millions patterns in many datasets. We define the four objective interestingness measures, in order to characterize those patterns that are useful for classification purpose, or provide insightful knowledge to understand the domain. Experiments show that the set of interesting Emerging Patterns (Chi EPs) is orders of magnitude smaller than the set of general Emerging Patterns (ρ-EPs, only the growth rates are concerned). Moreover, when mining Chi EPs, these four conditions provide us more opportunities to prune the large search space, such as a chi-squared pruning heuristic.

5.4 Chi-squared Pruning Heuristic

In this section, we first illustrate the Emerging Pattern mining problem using a set enumeration tree; we then give a concrete example to show how chi-tests are performed in the process of mining; lastly we discuss the chi-squared pruning heuristic.

5.4.1 Conceptual Framework for Discovering Emerging Patterns

The problem of mining EPs can be seen as a search problem. Given item space $\mathcal{I}$, the mining of EPs is to search through the power set of $\mathcal{I}$ for itemsets that are EPs, i.e., whose growth rates are larger than or equal to a given threshold ρ. Without imposing any conditions on the EPs, given a large item space, the search space for EPs is usually very large. Conditions such as minimums on the support and growth rate of EPs are usually used to reduce the search space and increase the efficiency of search. However, these constraints are not enough. For example, using the UCI Connect4 dataset, when the minimum growth

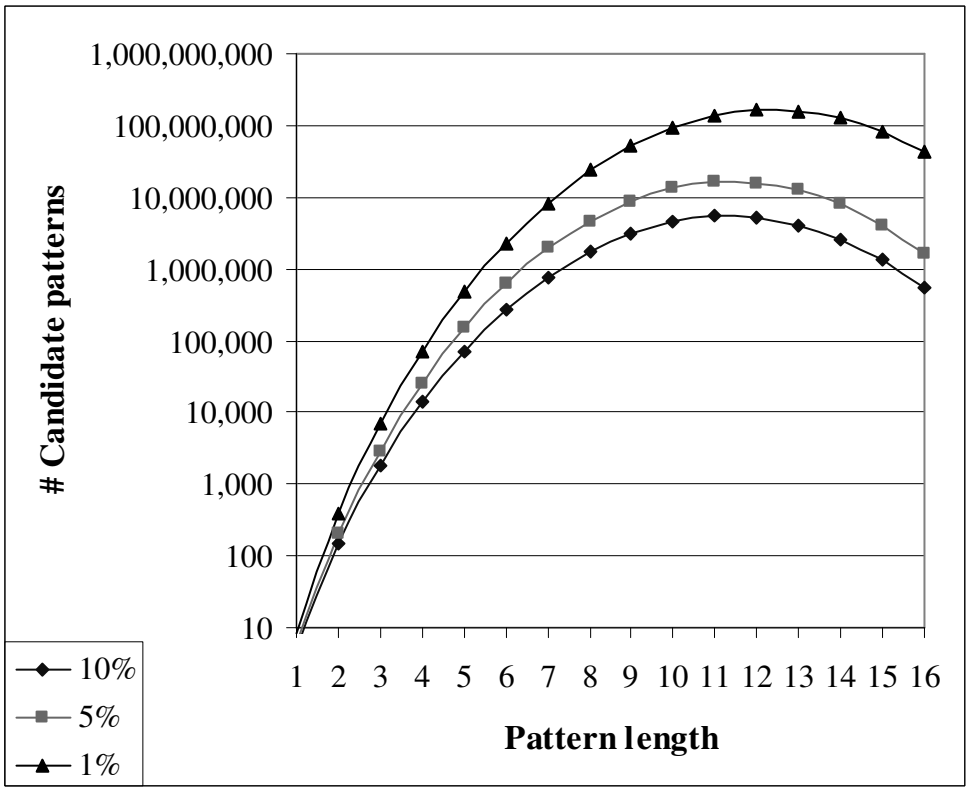


Figure 5.2: The number of candidate patterns with respect to the length - the UCI Connect4 dataset ($\rho = 2$, with varying minimum support thresholds, i.e., 10%, 5%, and 1%)

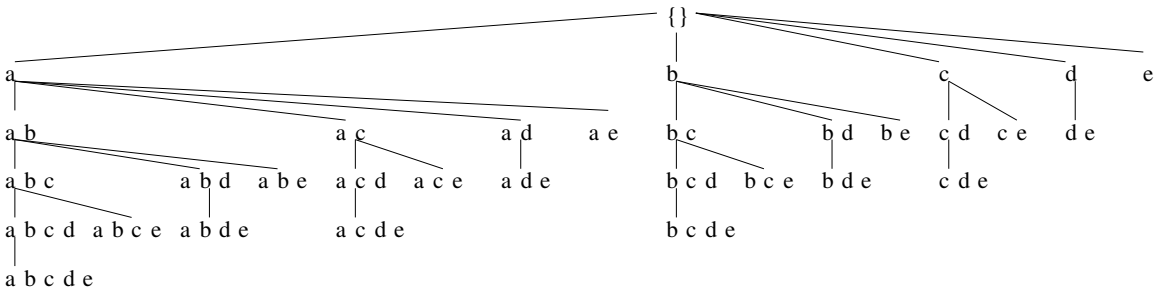


Figure 5.3: A complete set enumeration tree over $\mathcal{I} = \{a, b, c, d, e\}$, with items lexically ordered, i.e., $a \prec b \prec c \prec d \prec e$

rate threshold is set to 2, by varying the minimum support threshold to different values as 10%, 5%, and 1%, we plot three respective curves of the number of candidate patterns with respect to the length, as shown in Figure 5.2. We can see that as the length of candidate patterns increases, the number of candidate patterns grows up exponentially to a certain point and then starts to decrease. The minimum support threshold also plays an important role in the exponential growth of candidates. A lower minimum support threshold (e.g., 1%) makes the number of candidate patterns grow much faster as the length increases, than a higher threshold (e.g., 10%).

A set enumeration tree (Rymon 1993) is usually used to systematically explore the itemset space. For example, when the dataset under consideration contains only five items, i.e., $\mathcal{I} = \{a, b, c, d, e\}$, the corresponding set enumeration tree is shown in Figure 5.3. The set enumeration tree can be traversed depth-first or breadth-first. If breadth-first search is performed, in order to visit a pattern, we have to visit all its subsets. The problem with breadth-first search is two-fold:

- in order to discover a long EP with k items, we will have to test a large number (2^k) of its subsets;
- it is a waste of time to find the support of these subsets, because many of them are usually not EPs.

In our work, we adopt a depth-first strategy. Still, we have several choices to traverse the nodes: pre-order, post-order, in-order and other specific orders. We choose to traverse the

set enumeration tree, depth-first and in the following specific order: first visit the node, then visit the right and left subtree. Namely, the itemsets are considered or "generated" in the following order when $\mathcal{I} = \{a, b, c, d, e\}$:

- $\{e\}$
- $\{d\}$, $\{d, e\}$
- $\{c\}$, $\{c, e\}$, $\{c, d\}$, $\{c, d, e\}$
- $\{b\}$, $\{b, e\}$, $\{b, d\}$, $\{b, d, e\}$, $\{b, c\}$, $\{b, c, e\}$, $\{b, c, d\}$, $\{b, c, d, e\}$
- $\{a\}$, $\{a, e\}$, $\{a, d\}$, $\{a, d, e\}$, $\{a, c\}$, $\{a, c, e\}$, $\{a, c, d\}$, $\{a, c, d, e\}$, $\{a, b\}$, $\{a, b, e\}$, $\{a, b, d\}$, $\{a, b, d, e\}$, $\{a, b, c\}$, $\{a, b, c, e\}$, $\{a, b, c, d\}$, $\{a, b, c, d, e\}$

In Section 5.5.2, we will discuss the reason why we explore the itemset space in the above specific order.

5.4.2 Chi-test for Emerging Patterns: an Example

We give an example to see how contingency tests are performed in the process of mining. Let $X = \{a, b, c\}, Y = \{a, b\}$. Suppose $|\mathcal{D}_1| = |\mathcal{D}_2| = 100$ and $count_{\mathcal{D}_1}(Y) = 80$, $count_{\mathcal{D}_2}(Y) = 60$, $count_{\mathcal{D}_1}(X) = 60$, $count_{\mathcal{D}_2}(X) = 35$. Then we have the following observed contingency table.

count	$Y = \{a, b\}$	$X = \{a, b, c\}$	$\sum row$
$count_{\mathcal{D}_1}$	80	60	140
$count_{\mathcal{D}_2}$	85	35	120
$\sum column$	165	95	260

For each pair $(i, j) \in \{\mathcal{D}_1, \mathcal{D}_2\} \times \{X, Y\}$, we calculate the expectation under the assumption of independence:

$$E_{ij} = \frac{count_{\mathcal{D}_1+\mathcal{D}_2}(j) \times (count_i(X) + count_i(Y))}{count_{\mathcal{D}_1+\mathcal{D}_2}(X) + count_{\mathcal{D}_1+\mathcal{D}_2}(Y)}.$$

For example, the expected value for $count_{\mathcal{D}_1}(Y)$ is computed in the following way:

$$\sum column(Y) \times \frac{\sum row(\mathcal{D}_1)}{\sum row(\mathcal{D}_1) + \sum row(\mathcal{D}_2)} = 165 \times \frac{140}{260} = 88.846 \approx 89.$$

The results are shown in the following expected contingency table.

	$Y = \{a, b\}$	$X = \{a, b, c\}$	$\sum row$
$count_{\mathcal{D}_1}$	89	51	140
$count_{\mathcal{D}_2}$	76	44	120
$\sum column$	165	95	260

The chi-square value is the normalized deviation of observation from expectation; namely, using the above two tables (the observed and expected contingency table), we have

$$chi(X, Y) = \sum_{i \in \{\mathcal{D}_1, \mathcal{D}_2\}} \sum_{j \in \{X, Y\}} \frac{(O_{ij} - E_{ij})^2}{E_{ij}} =$$

$$\frac{(80 - 89)^2}{89} + \frac{(60 - 51)^2}{51} + \frac{(85 - 76)^2}{76} + \frac{(35 - 44)^2}{44} = 5.405.$$

Since the computed χ^2 value is 5.405, which is greater than 5.02 (at 97.5% significance level), we say that the distributions of X and Y are different with a confidence of 97.5%, which is higher than the minimum of 95%. It shows that after adding item c into Y (forming X), the two counts in $\mathcal{D}_1$ and $\mathcal{D}_2$ drop at different rates, i.e., the count in $\mathcal{D}_2$ drops faster. Although neither X nor Y are EPs satisfying a growth rate threshold of 2, conceivably, further growth of X may produce interesting EPs very possibly.

5.4.3 Chi-squared Pruning Heuristic

Our tree based algorithm mines EPs in a pattern growth manner. How do we push the interestingness measures into mining? It is straightforward to push the measure 1 and 2 into the pattern growth (see next section for details). But it is hard to push the measure 3 and 4, because we may not have "seen" all the subsets of the current pattern. A heuristic is proposed to prune as early as possible the search space, i.e., those patterns that are very likely turn out not to satisfy condition 3 or 4. The heuristic is based on the following lemma.

Lemma 5.1 Let X, Y, Z be itemsets. $Y = X \cup \{i\}$, $Z = Y \cup \{j\}$, $S = X \cup \{j\}$, where i and j are items. If $chi(X, Y) < \eta$, $P(\{i\}|X)P(\{j\}|X) = P(\{i, j\}|X)$, and $\eta = 3.84$, then we have $chi(S, Z) < \eta$ with at least 95% confidence.

Proof. Since $\eta = 3.84$, $chi(X, Y) < \eta \Longleftrightarrow chi(X, X \cup \{i\}) < 3.84$. We say i is independent from X with at least 95% confidence. So we have $P(X \cup \{i\}) \approx P(X)P(\{i\})$.

$P(\{i\}|X)P(\{j\}|X) = P(\{i,j\}|X) \Longleftrightarrow$

$$\frac{P(\{i\} \cup X)}{P(X)} * \frac{P(\{j\} \cup X)}{P(X)} = \frac{P(\{i,j\} \cup X)}{P(X)} \Longrightarrow P(\{i\}) * \frac{P(\{j\} \cup X)}{P(X)} \approx \frac{P(\{i,j\} \cup X)}{P(X)}.$$

So $P(X \cup \{i,j\}) \approx P(X \cup \{j\})P(\{i\})$, which means i is independent from $X \cup \{j\}$. Thus, we have $chi(X \cup \{j\}, X \cup \{j,i\}) < 3.84$ with at least 95% confidence.

□

The lemma requires that $P(\{i\}|X)P(\{j\}|X) = P(\{i,j\}|X)$. Although it is not true for all the cases in real datasets, experiments show that for most cases we have $P(\{i\}|X)P(\{j\}|X) \approx P(\{i,j\}|X)$, which is good enough for mining interesting EPs. When $chi(X,Y) < \eta = 3.84$, from the lemma, Z definitely will not be interesting since it does not satisfy condition 4. Our mining method can stop growing Y immediately to avoid searching and generating unnecessary candidate patterns.

The χ^2-test ($chi()$ function) can be used as an effective heuristic for pruning search space. By pruning long patterns as soon as possible, we usually obtain a relatively small set of EPs. One pass over the set of EPs can select Chi EPs according to the four interestingness measures. In contrast, if IEP-Miner does not use the heuristic, it needs to search a huge space, which produces a lot of uninteresting patterns first and discards them later. Experiments show that the χ^2-test heuristic makes IEP-Miner more efficient by an order of magnitude. We also investigate what patterns the heuristic search may lose. Detailed analysis over many datasets from the UCI Machine Learning Repository shows that it loses only a small number of Chi EPs with relatively low support. Moreover, high accuracy of the classifiers based on our discovered Chi EPs confirms that those missing Chi EPs are not important for classification. So the χ^2-test pruning heuristic is admissible, although it is non-monotone.

5.5 Efficient Mining of Chi Emerging Patterns (Chi EPs)

In Chapter 4, we have introduced the Pattern tree (P-tree) structure for mining Essential Jumping Emerging Patterns (EJEPs). The EJEP mining algorithm that operates in the P-tree, is not suitable for mining Chi Emerging Patterns, because the shape of the

tree structure targets the distribution of EJEPs and the `merge` operation during the mining process becomes the performance bottleneck[4]. Therefore, we modify the Pattern tree (P-tree) structure to suit the needs of mining Chi Emerging Patterns.

5.5.1 Data Structure: the Pattern Tree with a Header Table

Without lose of generality, we assume that there is a partial order on the set of all items, denoted as $\prec$. There are many ways to define $\prec$. For example, we can sort items according to their support in ascending or descending order; or simply use lexicographic order.

Definition 5.2 A Pattern Tree (P-tree) is a tree structure defined below.

1. It consists of one root, a set of item prefix subtrees as the children of the root, and a header table.

2. Each node in the item prefix subtrees consists of four fields: item-name, $count_{D_1}$, $count_{D_2}$ and node-link, where item-name registers which item this node represents, $count_{D_1}$ registers the number of transactions in D_1 represented by the portion of the path reaching this node, $count_{D_2}$ registers such number in D_2, and node-link links to the next node in the P-tree carrying the same item or null if there is none.

3. Each entry in the header table consists of four fields: (1) item-name; (2) head of node-link, which points to the first node in the P-tree carrying the item; (3) $total_{D_1}$, the sum of all $count_{D_1}$ in the item's corresponding node-link; (4) $total_{D_2}$, the sum of all $count_{D_2}$ in such node-link.

4. The tree is ordered: if a node M is the parent of a node N, and item i and j appear in M and N respectively, then $i \prec j$.

[4]The `merge` operation is expensive by nature. The EJEP mining algorithm is efficient because the number of times that `merge` is called is relatively small. However, in order to mine Chi Emerging Patterns, `merge` needs to be called much more frequently. This is due to the fact that EJEPs are usually a small subset of all interesting Emerging Patterns and the candidate patterns for Chi EPs are much more than those for EJEPs.

Note that nodes with the same item-name are linked in sequence via node-link, which facilitates tree traversal. Unlike the FP-tree (Han et al. 2000), the P-tree is only traversable top-down (from root to leaves), i.e., there is no pointer from child to parent nodes.

The Pattern Tree (Definition 5.2) is also different from the one discussed in Chapter 4. The major difference is that a header table is now used. As we can see later, the header table plays an important role in mining Chi Emerging Patterns. The other difference is the representation of node, as shown by Figure 5.4.

The construction of the P-tree can be found in Algorithm 5.1 and 5.2. Although the skeletons of these two algorithms are similar to Algorithm 2.2 discussed in Chapter 2 Section 2.2.3, we present them here to stress the differences.

- We can only filter out items whose both support in $\mathcal{D}_1$ and $\mathcal{D}_2$ are below the minimum support threshold. (See Algorithm 5.1 line 3-5)

- Algorithm 5.1 line 6 sorts items by $\prec$. A support-ratio-ascending order is preferred. Looking at Figure 5.3, using the support-ratio-ascending order means that e has the largest support-ratio, while a has the smallest support-ratio. In the way that we explore the itemset space, we will "generate" itemsets that are made up of items with larger support ratios first. Intuitively, itemsets made up of items with large support ratios are more likely to be our defined interesting Emerging Patterns (Chi EPs). Our experiments confirm that the support-ratio-ascending order can make us look at fewer candidate patterns than the lexicographic order and other orders. For example, for the UCI mushroom dataset in Figure 5.9, using the lexicographic order will make IEP-Miner examine 114,044 candidate patterns, about 27% more than using the support-ratio-ascending order (89,624 candidates).

- In Algorithm 5.2, one node in the P-tree contains one item, with two counts in $\mathcal{D}_1$ and $\mathcal{D}_2$.

The P-tree of the example dataset from Table 5.1 is shown in Figure 5.4.

Algorithm 5.1: P-tree construction (for mining Chi EPs)

input : A training dataset $\mathcal{D}$ containing two classes of data (positive $\mathcal{D}_1$ and negative $\mathcal{D}_2$) and a minimum support threshold ξ (in percentage)

/* `Other interestingness measures such as a minimum growth rate threshold` ρ `and a minimum chi-value threshold` η`, are not able to be used here in the P-tree construction` */

/* $|\mathcal{D}_1|$ `= the total number of instances in` $|\mathcal{D}_1|$ */

/* $|\mathcal{D}_2|$ `= the total number of instances in` $|\mathcal{D}_2|$ */

output: The pattern tree of $\mathcal{D}$, P-tree

/* `The first database scan` */

Scan $\mathcal{D}$ once, that is, scan $\mathcal{D}_1$ first and then $\mathcal{D}_2$ (or equivalently, scan $\mathcal{D}_2$ first and then $\mathcal{D}_1$). After the scan, for each item i in $\mathcal{D}$, we have its support in $\mathcal{D}_1$ and $\mathcal{D}_2$. Let $count_1[i]$ and $count_2[i]$ denote item i's support counts in $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively;

Clear the header table, i.e., initialize an empty table with all counts set to zero and all pointers `NULL`;

foreach *item* i *in* $\mathcal{D}$ **do**

  if $(count_1[i] \geqslant |\mathcal{D}_1| \times \xi) \vee (count_2[i] \geqslant |\mathcal{D}_2| \times \xi)$ **then**

    Add item i into the header table, with both counts $count_1[i]$ and $count_2[i]$;

  end

end

/* $\prec$ `is the partial order defined on the set of all items` */

Let J denote the set of all items that appear in the header table. Sort J in the order $\prec$;

/*`The second database scan` */

Create the root of a P-tree, R;

foreach *transaction* t *in* $\mathcal{D}$ **do**

  Select and sort all the J items in t according to the order $\prec$;

  Let the sorted item list in t be $[p|P]$, where p is the first element and P is the remaining list. Call `insert-tree(`$[p|P]$, R`)`;

end

Algorithm 5.2: Function: insert-tree($[p|P], T$) (for mining Chi EPs)

input : a list of sorted items denoted as $[p|P]$, and a subtree of the P-tree denoted as T (T is a pointer to root of the subtree)

output: the modified P-tree after inserting the new itemset $[p|P]$

if *T has a child node N such that N.items-name=p* **then**
  switch *the class label of the instance* $[p|P]$ **do**
    case $[p|P] \in \mathcal{D}_1$ increment $N.count_1$ by 1;
    case $[p|P] \in \mathcal{D}_2$ increment $N.count_2$ by 1;
  end
end
else
  create a new node N with N.items-name = p, $N.count_1 = N.count_2 = 0$, and N.node-link = `NULL`;
  switch *the class label of the instance* $[p|P]$ **do**
    case $[p|P] \in \mathcal{D}_1$ set $N.count_1 = 1$;
    case $[p|P] \in \mathcal{D}_2$ set $N.count_2 = 1$;
  end
  Let N.node-link be pointed to the nodes with the same item-name via the node-link structure;
end
if *P is nonempty* **then** call `insert-tree(`P, N`)`;

Table 5.1: A dataset containing 2 classes

D_1					D_2				
a		c	d	e	a	b			
a							c		e
	b			e	a	b	c	d	
	b	c	d	e				d	e

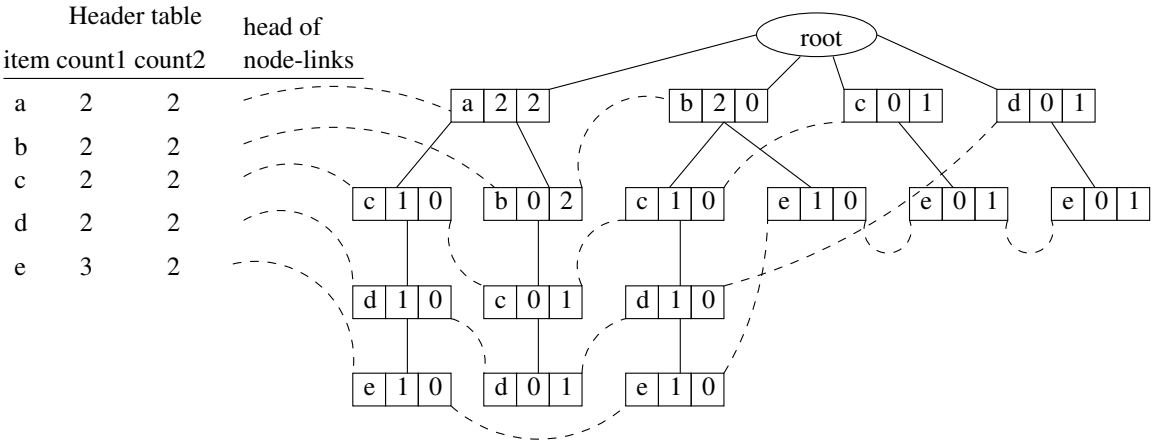


Figure 5.4: The P-tree of the example dataset

5.5.2 Using P-tree to Mine Chi Emerging Patterns (Chi EPs)

One may want to use the ideas of FP-growth (Han et al. 2000) to mine Emerging Patterns, i.e., grow the current pattern by constructing its conditional FP-tree. There are several difficulties.

- Frequent pattern mining has a very nice heuristic to use, the anti-monotone Apriori property: if a pattern with k items is not frequent, any of its super-patterns with $(k+1)$ or more items can never be frequent. Based on this property, an FP-tree is usually much smaller than the raw database since only frequent length-1 items have nodes in the FP-tree, and conditional FP-trees are much smaller than the original FP-tree. However, no such nice heuristics exist for mining Emerging Patterns.

- It is very costly to physically construct separate memory structures of conditional FP-trees, especially when several millions of patterns are to be mined.

Our aim is to avoid constructing extra conditional sub-trees. The solution is to modify the counts and node-links "directly" in the P-tree.

We show the ideas of mining Chi EPs by using the example dataset from Table 5.1 and the tree shown in Figure 5.4. Let $\xi = 1$ is a minimum support threshold (absolute

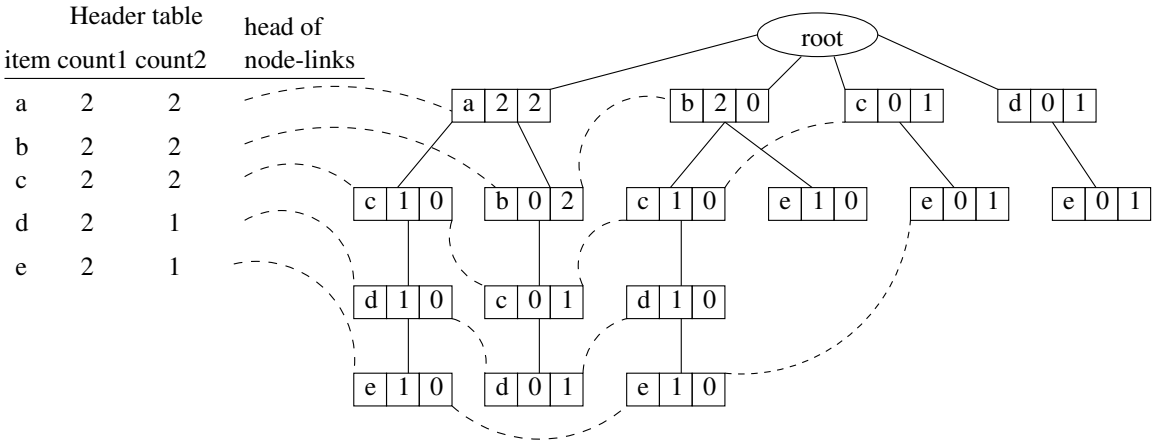


Figure 5.5: The P-tree after adjusting the node-links and counts of d and e under c

occurrence frequency), $\rho = 2$ is a minimum growth rate threshold. Basically, we have to calculate the supports in both D_1 and D_2 for the power set of $I = \{a, b, c, d, e\}$ and then check each itemset against the four interestingness measures. We will explore the itemset space in the order discussed before in Section 5.4.1. The benefit is "in place" modification of P-tree, as demonstrated by the following.

For e, we get its counts in both classes from the head table, denoted as $[e{:}3; 2]$ (the two numbers after ":" indicate the supports in D_1 and D_2, respectively). {e} is not an EP since its growth rate $1.5 < \rho$.

For d, we have $[d{:}2; 2]$. {d} is not an EP. We try to grow {d} via concatenation of e with it. e's counts in both classes change from $[e{:}3; 2]$ to $[e{:}2; 1]$, when only those e co-occurring with d are counted. This can be done by going through d's node-links and visit those d's subtrees. We simply refer the process to recounting e under {d}, which is frequently used in the following. Note that the other two e are not counted since they are not in such subtrees. Then we get $[d{:}2; 2, e{:}2; 1]$, where $\{d, e\}$ is an EP of growth rate 2.

For c, we have $[c{:}2; 2]$. $\{c\}$ is not an EP. Now we have e and d to concatenate with c. The P-tree after the node-links and counts of e and d are adjusted is shown in Figure 5.5. We try e first. After recounting e under $\{c\}$, we obtain $[c{:}2; 2, e{:}2; 1]$, where $\{c, e\}$ is an EP of growth rate 2. We then try d. After recounting d under $\{c\}$, we obtain $[c4{:}2; 2, d{:}2; 1]$, where $\{c, d\}$ is an EP of growth rate 2. Because $\{c, d\}$ has supports in D_1 and D_2 quite

different from $\{c\}$, it may produce interesting patterns to further grow $\{c, d\}$ by adding e. After recounting e under {c,d}[5], we obtain $[c{:}2; 2, d{:}2; 1, e{:}2; 0]$, where $\{c, d, e\}$ is an EP of infinite growth rate. Recall that an EP with infinite growth rate is called a Jumping Emerging Pattern (JEP).

For b, we have $[b{:}2; 2]$. $\{b\}$ is not an EP. Now we have e, d and c to concatenate with b. We try e first. After recounting e under $\{b\}$, we obtain $[b{:}2; 2, e{:}2; 0]$, where $\{b, e\}$ is a JEP. We try d next. After recounting d under $\{b\}$, we obtain $[b{:}2; 2, d{:}1; 1]$. Because the support distributions of $\{b, d\}$ and $\{b\}$ are the same, it is very unlikely that we can get interesting EPs by further growing $\{b, d\}$. In fact, $\{b, d, e\}$ with support counts 1 and 0 in D_1 and D_2, is not interesting since its subset $\{b, e\}$ is also a JEP. It can be seen that our chi-squared heuristic effectively prunes a lot of uninteresting patterns from consideration. We then try c. After recounting c under $\{b\}$, we obtain $[b{:}2; 2, c{:}1; 1]$. For the same reason, we do not further grow $\{b, c\}$.

For a, we have $[a{:}2; 2]$. $\{a\}$ is not an EP. Now we have e, d, c and b to concatenate with a. We try e first. After recounting e under $\{a\}$, we obtain $[a{:}2; 2, e{:}1; 0]$, where $\{a, e\}$ is a JEP. We try d next. After recounting d under $\{a\}$, we obtain $[a{:}2; 2, d{:}1; 1]$. For the above reason, we do not further grow $\{a, d\}$. We then try c. After recounting c under $\{a\}$, we obtain $[a{:}2; 2, c{:}1; 1]$. Again we do not further grow $\{a, c\}$. Lastly, we try b. After recounting b under $\{a\}$, we obtain $[a{:}2; 2, b{:}0; 2]$, where $\{a, b\}$ is a JEP. We do not further grow a JEP, since supersets of a JEP is not interesting.

5.5.3 IEP-Miner Pseudo-code

We are ready to present our algorithm, IEP-Miner. Suppose all items are mapped into a set of consecutive positive integers $[0 \cdots N]$. The high-level description of IEP-Miner is given in Algorithm 5.3. The main procedure of IEP-Miner takes the root of the P-tree as input and performs the mining solely in the P-tree. The function `is-iep()` checks whether an itemset satisfies the four interestingness measures (Definition 5.1). Interesting Emerging Patterns (Chi EPs) are written out (into F) once they are found. `is-iep()` can

[5]Since only those d under c are linked by its node-links, it is easy to go through d's node-links looking for e.

Algorithm 5.3: IEP-Miner (discover Chi Emerging Patterns)

input : The Pattern tree (P-tree) with the root of R, a minimum support threshold ξ, a minimum growth rate threshold ρ, and a minimum chi-value threshold $\eta = 3.84$.

output: the set of Chi Emerging Patterns, F

/* assume $I = \{1, \cdots, N\}$ and $1 \prec \cdots \prec N$ */

/* visit each item i in the header table from bottom to top */

1 **foreach** $i = N$ ***downto*** *1* **do**

2 | $\alpha = \{i\}$;

3 | **if** *is-iep* (α) **then** output α into F;

4 | mine-subtree (α);

end

/*Some patterns in F may not satisfy conditions 3 or 4, because we have not got the support of all their subsets */

5 remove uninteresting Emerging Patterns from F;

Algorithm 5.4: Function: mine-subtree(β), called by IEP-Miner

/* mine-subtree is a recursive function */

/* $\beta = [\alpha|k]$, α is the prefix of β, and k is the last item of β */

/* Note that β is augmented one item by one. So the last item of β (k) is also the most recently added */

/* Go through k's node-links and visit all k's subtrees to adjust node-links and accumulate counts for these sub-nodes */

1 **foreach** *item i which appears in the subtrees of nodes containing k* **do**

2 | adjust the corresponding node-links and accumulate counts of these nodes containing i;

end

3 **foreach** $i = N$ ***downto*** *k+1* **do**

4 | $\gamma = \beta \cup i$;

5 | **if** *is-iep* (γ) **then** output γ into F;

6 | **if** $chi(\gamma, \beta) \geqslant \eta$ **then** mine-subtree (γ);

end

be expensive because of the interestingness measures 3 and 4, which involves the checking of subset relationships. We organize those subsets (candidates) along with their support in a hash tree to enable fast search. Those candidate patterns are hashed first on their length and then their first item. Note that in the way that we explore the itemset space, we will usually have "seen" all its subsets before we look at a pattern (see Figure 5.3). However, some subsets may not be seen because the chi-squared pruning heuristic cuts off branches in the set enumeration tree. Let us look at Figure 5.3. Suppose the heuristic cuts off the branch named "*ac*", leaving itemset *acd*, *ace* and *acde* unexplored. When we visit *abcd*, we do not have support for its subset *acd*. Therefore, some patterns that pass the test of `is-iep()` may not be truly interesting. But experiments show that the number of such false Chi EPs is small. This conforms to intuition, i.e., when testing all subsets ($\forall$ in conditions 3 and 4), one failure of test will lead to the failure of the whole test. The set of the generated candidate interesting EPs is relatively small, and one pass over the set can filter out those that do not satisfy the interestingness measure 3 or 4.

The procedure `mine-subtree()` (Algorithm 5.4) is called recursively. It always tries to grow the current pattern β by adding a new item. The chi-squared pruning heuristic, the test "$chi(\gamma, \beta) \geqslant \eta$", is used to prune a huge number of patterns which are definitely uninteresting. "$chi(\gamma, \beta) \geqslant \eta$" means that γ's count distribution in two classes are *significantly* different from that of β, hence further growing γ may be interesting. However, if the chi-test failed, we stop growing γ. The final set is our defined interesting Emerging Patterns (Chi EPs). There exist some Chi EPs that IEP-Miner cannot discover due to the chi-squared heuristic. However, the detailed analysis on several datasets finds that (1) IEP-Miner loses few Chi EPs (around 5%); (2) those missing Chi EPs have low supports, which are generally not important for classification.

5.6 Performance Study

We now report a performance evaluation of IEP-Miner, including its scalability with respect to support, the effect of the chi-square pruning heuristic, the study of what EPs are lost due to the heuristic and classification by Chi EPs to show their usefulness.

All experiments were conducted on a Dell PowerEdge 2600 (Dual Intel Xeon 2.4GHz with 3G RAM) running Solaris 8/x86, shared by many users of the University of Melbourne. We report experimental results on the following datasets[6] from the UCI Machine Learning Repository.

Dataset	Records	Avg. Record Width	No. of Binary items
adult	45,225	15	154
chess	3,169	36	73
connect-4	61,108	43	128
mushroom	8124	23	121

The interestingness of Emerging Patterns is determined by three parameters, where ξ is a minimum support threshold, ρ is a minimum growth rate threshold, and 3.84 is a minimum chi-square value threshold, both for chi-square test of individual EPs (refer to Definition 5.1 condition 4) and heuristic (used for efficiently mining Chi EPs).

5.6.1 Scalability Study

We have carried out experiments on many datasets from the UCI Machine Learning Repository, i.e., using IEP-Miner to discover useful EPs in the learning phase of our classification systems. All of them exhibit significant improvement in performance. We only present the results of scalability with respect to support, on the following large, high-dimensional datasets, namely, Chess and Connect4, which are relatively difficult for IEP-Miner. Note that Connect4 is a very dense dataset with a lot of long frequent itemsets.

The scalability of IEP-Miner with support threshold is shown in Figure **??** and Figure 5.7. The scalability of ConsEPMiner is also shown in these figures. However, note that ConsEPMiner mines EPs satisfying several constraints, not the same "interesting" Emerging Patterns (Chi EPs) as IEP-Miner. Therefore, we compare ConsEPMiner with our IEP-Miner *indirectly*.

To test the scalability of IEP-Miner against the number of instances, we select 10k, 20k, 30k, 40k, 50k and 60k instances from the UCI Connect4 dataset to form six new datasets. Two minimum support thresholds are used. The result in shown in Figure 5.8,

[6]Among the three classes of the connect-4 dataset, only the Loss (16,635 records) and Win (44,473 records) classes are used since the Draw class consists of relatively few records.

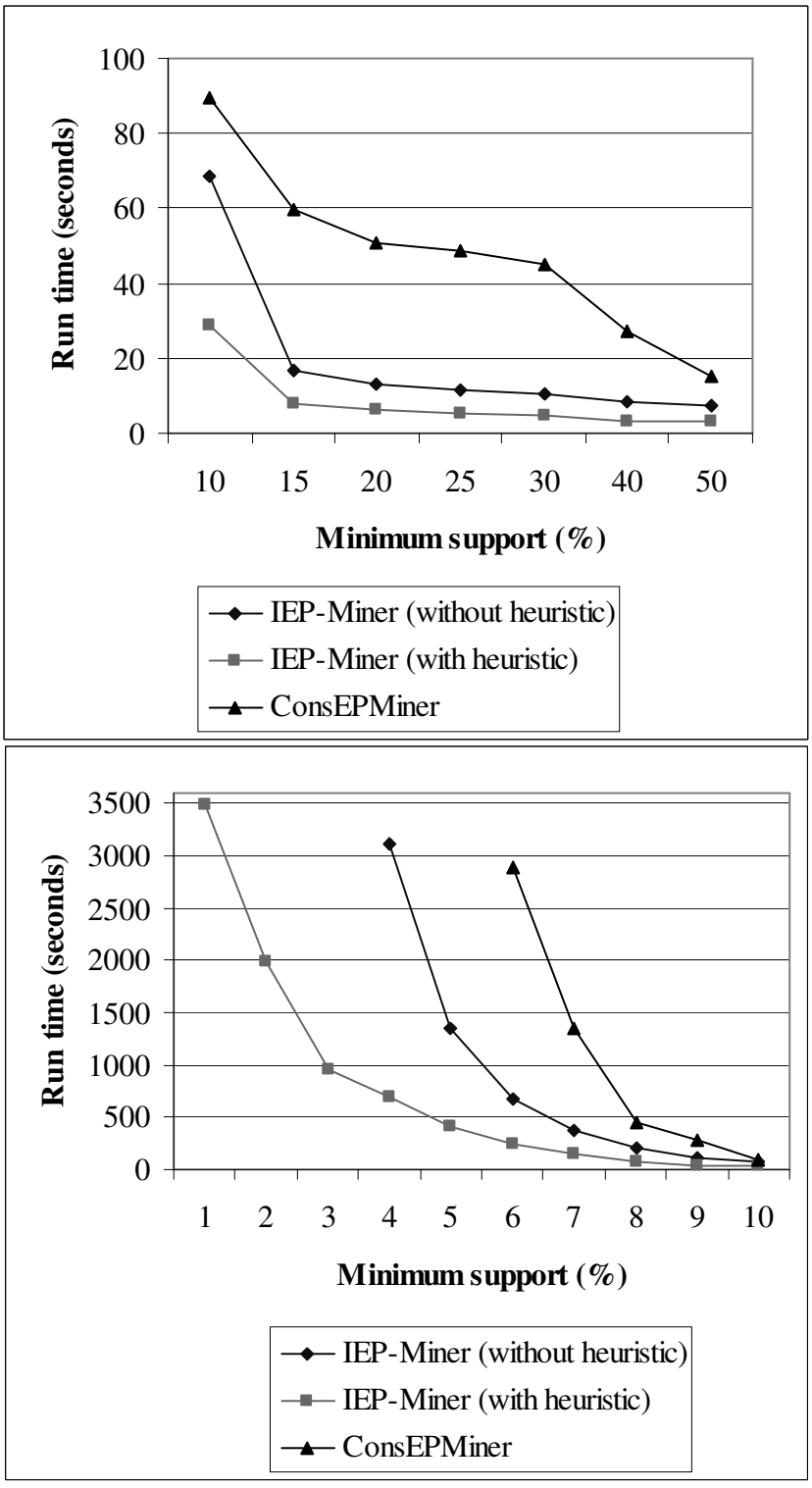


Figure 5.6: Scalability with support threshold: Chess ($\rho = 5$)

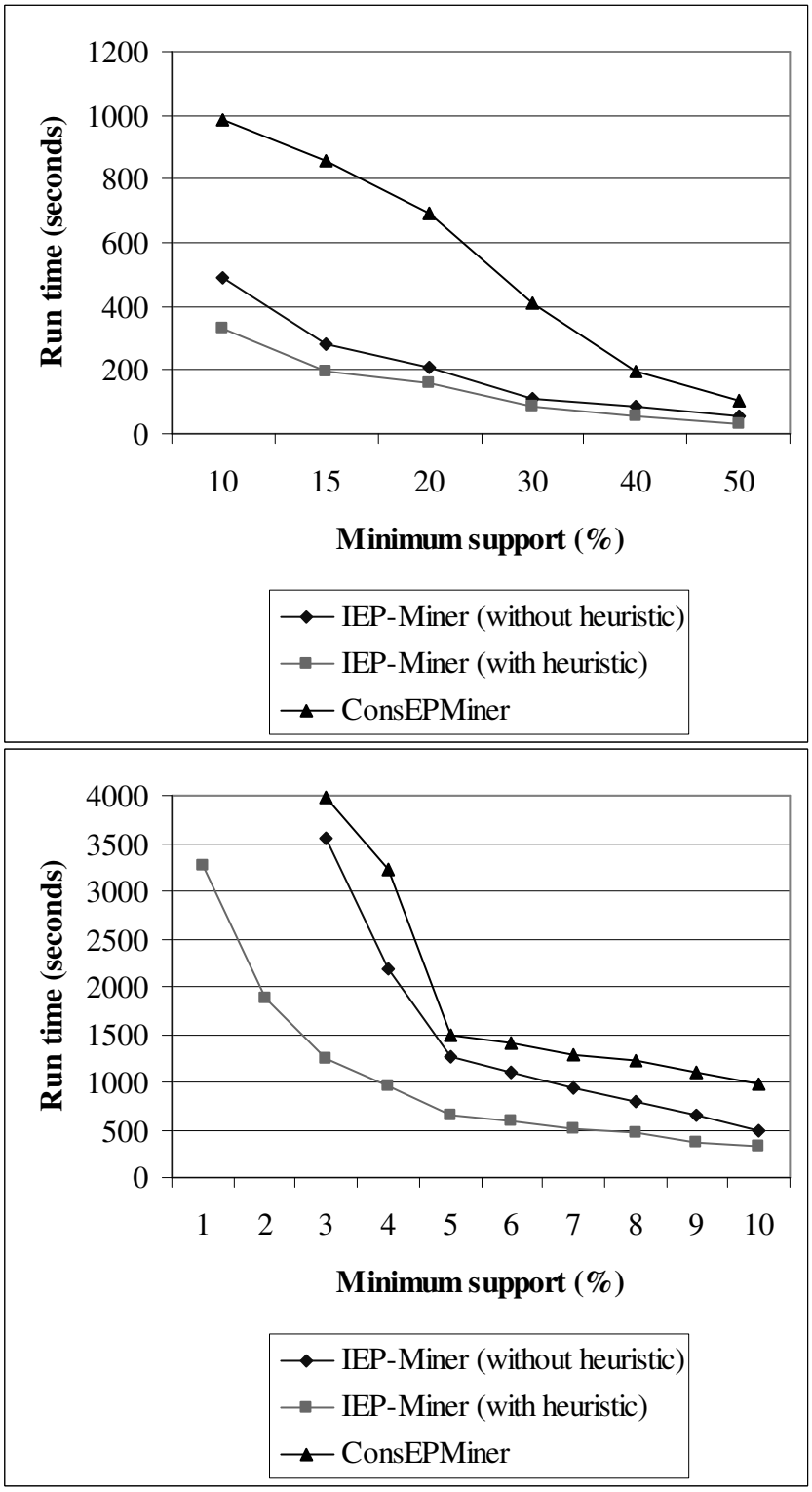


Figure 5.7: Scalability with support threshold: Connect-4 ($\rho = 2$)

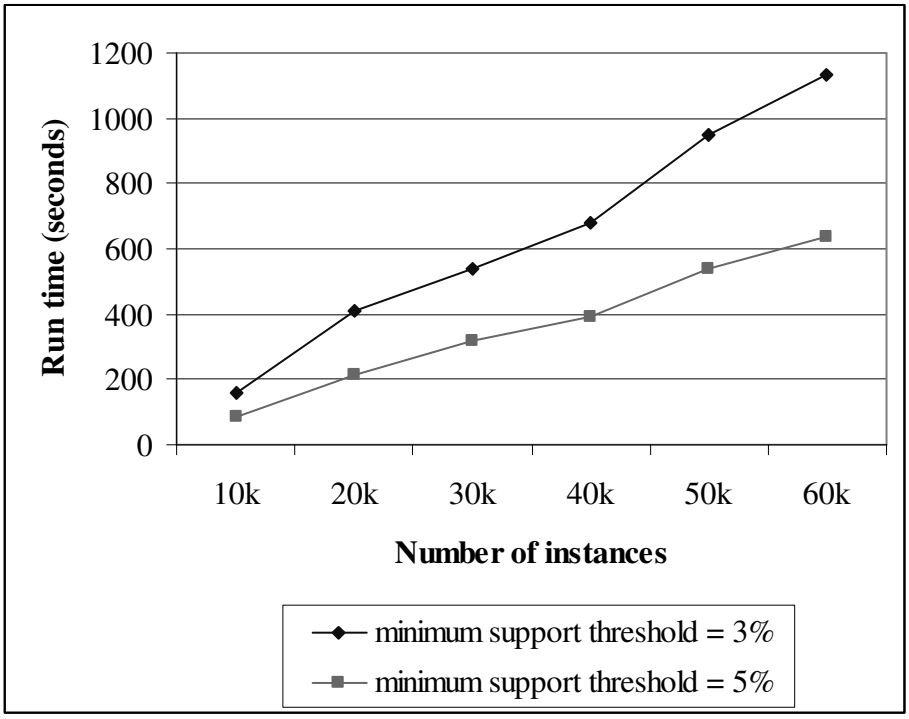


Figure 5.8: Scalability with number of instances: Connect-4 ($\rho = 2$)

which indicates a linear increase of runtime with the number of instances, when the number of attributes is fixed. Note that some work replicate instances in real datasets to test the scalability. We did not replicate instances, because the same instances will share the same branch in our P-tree and the scaling-up of such datasets becomes trivial.

5.6.2 Effect of Chi-Squared Pruning Heuristic

To show the effectiveness of the chi-squared pruning heuristic, we investigate how many candidate patterns we need to "look at" before Chi EPs are generated. The results are shown in Figure 5.9. It can be seen that a huge amount of search space is pruned because our heuristic stops early growing many unpromising branches. On average, using heuristic makes us be able to check 30-40 times fewer candidates than using no heuristic.

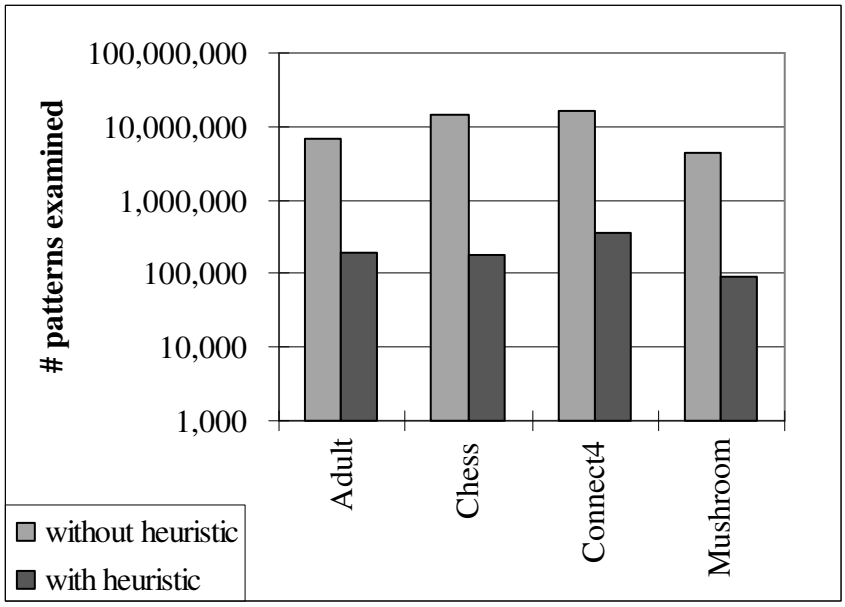


Figure 5.9: The effectiveness of the chi-squared pruning heuristic (IEP-Miner)

5.6.3 Comparison between General EPs and Chi EPs

In order to have some ideas of what proportion of EPs are interesting, we compare the set of "all EPs"(satisfying the support and growth rate threshold only, and their maximum length is no more than the maximum length of all Chi EPs), "all Chi EPs"(satisfying the four interestingness measures) and "mined Chi EPs"(satisfying the four interestingness measures, but not complete due to the heuristic) in terms of their distributions in support intervals. The results are shown in Table 5.2. The ratio is the number of "all EPs" in the specified interval divided by that of "all Chi EPs". The last row gives the percent of missing Chi EPs over "all Chi EPs" due to heuristic searching.

We highlight some interesting points:

- The set of all Chi EPs is 1000-3000 times smaller than the set of all general EPs.
- The ratios decreases from left to right, which means that our interestingness measures eliminate a large number of EPs with low supports, while tend to keep EPs with high supports. This is desirable and reasonable, since EPs with higher supports are

definitely more preferable for classification given the same growth rate. On the other hand, an EPs with high support does not necessarily mean that it is useful, since its subset may have higher support.

- We stress that the set of our discovered Chi EPs is very close to the set of true Chi EPs: they are *exactly the same* at high support; only at very low support, some interesting EPs are ignored. The chi-squared pruning heuristic is very effective since the top 90% Chi EPs are discovered by our algorithm.

5.6.4 The Subjective Measure for Chi Emerging Patterns

As discussed early in this chapter, when a user wants to use Emerging Patterns for classification, the subjective measure of Emerging Patterns is defined as their utility. To evaluate the actual usefulness of Chi EPs that satisfy the four objective interesting measures, we build a classification system exclusively based on Chi EPs, which is called CACEP, i.e., Classification by Aggregating Chi Emerging Patterns.

CACEP: Classification by Aggregating Chi Emerging Patterns

There are no differences between CACEP and CAEP (Classification by Aggregating Emerging Patterns, discussed in Chapter 3 Section 3) in the classification phase: the scoring function is same and scores are normalized in the same way. The differences include different mining algorithms used in the training phase and different kinds of patterns used in classification, namely:

- CAEP uses ConsEPMiner (Zhang et al. 2000a) to mine EPs; while CACEP uses the newly-developed algorithm, IEP-Miner.

- CAEP uses general EPs with infinite (from moderate to large) growth rate; while CACEP uses Chi EPs satisfying the four objective interesting measures.

- CAEP can reduce the number of EPs used in classification. But it first mines as many EPs as possible, then discards some redundant EPs according to certain conditions.

Table 5.2: Comparison between general EPs and Chi EPs

ADULT ($\xi = 0\%, \rho = 10000$, maximum length = 11)

Support range	0-1%	1-2%	2-3%	3-4%	4-5%	5-6%	6-8%	8-100%	0-100%
all EPs	10,490,845	4,366	963	255	126	51	16	0	10,496,622
all Chi EPs	10,072	92	20	9	6	3	4	0	10,206
mined Chi EPs	9,239	83	19	9	6	3	4	0	9,363
Ratio	1,041.6	47.5	48.2	28.3	21	17	4	1	1,028.5
missing Chi EPs	8.3%	9.8%	5%	0	0	0	0	0	8.3%

Chess ($\xi = 0\%, \rho = 10$, maximum length = 10)

Support range	0-3%	3-7%	7-10%	10-30%	30-50%	50-100%	0-100%
all EPs	15,267,071	1,933,223	251,083	123,675	1,436	47	17,576,535
all Chi EPs	7,468	3,379	832	588	24	4	12,295
mined Chi EPs	6,766	3,282	832	588	24	4	11,496
Ratio	2,044.3	572.1	301.8	210.3	59.8	11.8	1,430.0
Missing Chi EPs	9.4%	2.9%	0.0%	0.0%	0.0%	0.0%	6.5%

CONNECT-4 ($\xi = 0\%, \rho = 2$, maximum length = 10)

Support range	1-2%	2-4%	4-6%	6-10%	10-40%	40-100%	0-100%
all EPs	13,837,899	5,938,079	1,372,383	729,788	242,461	0	22,120,610
all Chi EPs	2,064	2,130	747	487	407	0	5,835
mined Chi EPs	1,940	1,993	712	487	407	0	5,539
Ratio	6704.4	2787.8	1837.2	1498.5	595.7	1	3791
missing Chi EPs	6%	6.4%	4.7%	0	0	0	5.1%

MUSHROOM ($\xi = 0\%, \rho = 10000$, maximum length = 6)

Support range	0-2%	2-4%	4-7%	7-10%	10-30%	30-70%	70-100%	0-100%
all EPs	8,113,592	312,120	123,256	18,861	44,480	2,015	0	8,614,333
all Chi EPs	1,032	546	360	175	416	72	0	2,606
mined Chi EPs	1,002	536	360	175	416	72	0	2,526
Ratio	7862	571.6	342.4	107.8	106.9	30	1	3312
missing Chi EPs	2.9%	1.8%	0	0	0	0	0	3.1%

Though it reduces the number of EPs used in the test phrase, it adds more to the training time.

Classification Accuracy

We carry experiments on 20 datasets from the UCI Machine Learning Repository to compare CACEP with state-of-the-art classifiers[7]: Naive Bayes (NB), the widely known decision tree induction C4.5, Classification based on Multiple Association Rules (CMAR), Classification by Aggregating Emerging Patterns (CAEP) and an information-based approach to aggregate Emerging Patterns for classification (iCAEP). We use the Entropy method described in (Fayyad & Irani 1993) taken from the $\mathcal{MLC}$++ machine learning library (Kohavi et al. 1994) to discretize datasets containing continuous attributes.

Table 5.3 summarizes the accuracy results. Column 1 gives the names of the 20 datasets. Columns 2 to 7 give the predictive accuracy of the classifiers. The last column shows the accuracy ratio of CACEP over iCAEP, which achieves generally higher accuracy than its precursor, CAEP. The accuracy of NB, C4.5, CAEP and iCAEP was obtained by using the methodology of *stratified* ten-fold cross-validation (CV-10), on exactly the same CV-10 training and testing data. We use WEKA's Java implementation of NB and C4.5 (Witten & Frank 1999). For iCAEP and CAEP, the default parameter settings are: the minimum support threshold $\xi = 1\%$ or an absolute count of 5; the minimum growth rate $\rho = 5$; and the minimum relative growth-rate improvement equals 1.01. The accuracy of CMAR is according to the results reported in (Li, Han & Pei 2001), where a dash $-$ indicates that the result is not available in literature.

We also compare the number of EPs and Chi EPs used in CAEP and CACEP, respectively. The results are shown in Figure 5.10.

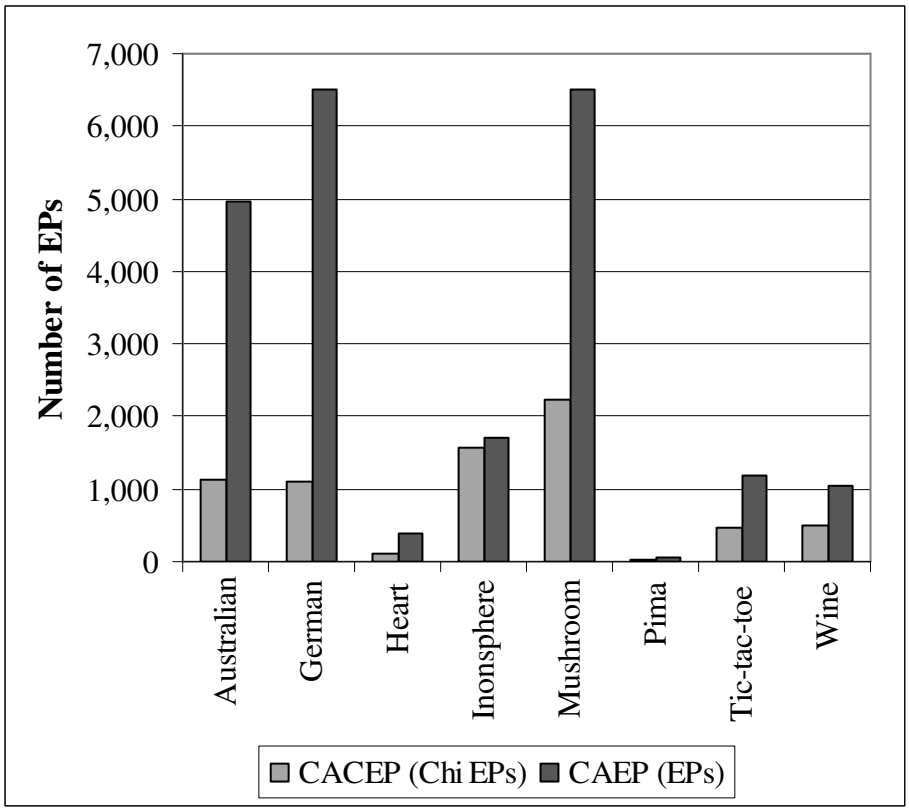
We highlight the following points observed from Table 5.3 and Figure 5.10:

- CAEP, iCAEP and our CACEP perform very well. They all achieve higher accuracy than NB and C4.5. Their accuracy is also competitive with respect to CMAR, which is shown in (Li, Han & Pei 2001) to outperform the early association based classifier,

[7]NB has been discussed in Chapter 2 Section 2.3.2; C4.5 in Chapter 2 Section 2.3.1; CMAR in Chapter 2 Section 2.3.3; CAEP and iCAEP in Chapter 3 Section 3.

Table 5.3: Accuracy comparison

Dataset	NB	C5.0	CMAR	CAEP	iCAEP	CACEP	Ratio
Adult	83.15	85.54	-	83.07	80.86	83.28	1.030
Australian	83.77	84.56	86.1	85.36	85.51	85.44	0.999
Chess	87.15	99.31	-	85.55	89.05	89.34	1.003
Cleve	81.52	74.88	82.2	80.2	80.86	82.07	1.015
Diabete	75.13	72.92	75.8	73.96	74.74	75	1.004
German	74.1	71.3	74.9	74.7	74.3	72	0.969
Heart	82.22	74.07	82.2	82.22	81.85	82.22	1.005
Hypo	97.72	99.21	98.4	96.59	96.52	97.25	1.008
Iono	85.75	90.88	91.5	89.74	90.6	90.6	1.000
Labor	85.96	80.7	89.7	91.23	89.47	87.71	0.980
Lymph	78.33	74.38	83.1	75	80.41	79.38	0.987
Mushroom	95.78	99.87	-	94.21	99.46	96.58	0.971
Pima	75.9	75.39	75.1	77.47	72.27	74.34	1.029
Segment	86.58	90.91	-	86.75	90.91	90.17	0.992
Shuttle-small	90.83	99.43	-	98.09	96.55	99.07	1.026
Sonar	72.1	70.2	79.4	77.4	76.44	78.5	1.027
Tic-tac	70.15	84.74	99.2	86.43	89.56	87.16	0.973
Vehicle	47.17	70.85	68.8	64.3	62.76	64.63	1.030
Waveform-21	78.51	75.62	83.2	83.76	81.88	84.34	1.030
Wine	89.9	92.7	95	96.07	96.63	98.3	1.017
Average	81.09	83.37	84.31	84.11	84.53	84.87	1.004



The complexity is measured in terms of number of EPs or Chi EPs used in classification. Our CACEP uses Chi EPs, while CAEP uses EPs satisfying only the support and growth rate thresholds.

Note that the number of EPs (used in CAEP) for the UCI Waveform and Vehicle dataset is 84,700 and 28,847, which are not shown in the diagram because these two values are much larger than others. In contrast, our CACEP uses 17,385 and 5,339 Chi EPs for the UCI Waveform and Vehicle dataset, respectively. We achieve an average reduction of 5 for these two datasets, i.e., our CACEP uses as around 20% EPs as CAEP. Similar reduction rates can be observed from the above figure.

Figure 5.10: Comparison of classifier complexity: CACEP vs. CAEP

CBA.

- CACEP achieve higher accuracy than both CAEP and iCAEP on average. Compared with CAEP, CACEP wins on 16 datasets, loses on 3, and ties on 1. Compared with iCAEP, CACEP wins on 12 datasets, loses on 7, and ties on 1.

- CACEP uses much fewer Chi EPs than CAEP uses EPs for classification. The number of EPs can be very large. For the UCI Waveform dataset, CAEP uses 84,700 EPs; while our CACEP uses only 17,385 Chi EPs and achieve higher accuracy. For the UCI Vehicle dataset, CAEP uses 28,847 EPs; while CACEP uses only 5,339 Chi EPs and again achieve higher accuracy.

We can see that using only a small number of Chi Emerging Patterns (which are much fewer than general Emerging Patterns) does not degrade classification accuracy at all and very often there is accuracy improvement. Therefore, we conclude that Chi Emerging Patterns are high quality patterns with sharp differentiating power. In other words, Chi Emerging Patterns are very useful for classification. This subjective interestingness measure confirms that our four objective interestingness measures are reasonable and appropriate.

5.7 Related Work

5.7.1 Previous Algorithms for Mining Emerging Patterns

In Chapter 3, we discussed the border-based approach (Li et al. 2000) and ConsEPMiner (Zhang et al. 2000a) for mining Emerging Patterns. Borders are used as means for concisely describing Emerging Patterns. The collection of discovered Emerging Patterns is typically very large; while the borders, which are pairs of the sets of the minimal itemsets and of the maximal ones, are usually much smaller. A suite of algorithms, which manipulates only borders of two collections, were proposed for mining Emerging Patterns. When collections of frequent itemsets (with respect to a *minsup* threshold) for both classes of data, which are represented by borders, are available, the borders of Emerging Patterns can be quickly derived by the border differential procedure. However, they depend on border-finding algorithms such as Max-Miner (Bayardo Jr. 1998). In fact, the task of

mining maximal frequent patterns is very difficult, especially when the minimum support is low (e.g. 5% or even 0.1%). For example, for the UCI Connect-4 dataset, the Max-Miner, one of the most efficient previously known algorithm for finding maximal frequent itemsets, needs more than three hours when minimum support is 10%. Furthermore, the process of extracting the embodied Emerging Patterns with supports and growth rates from the borders and selecting those interesting is very time-consuming. In contrast, our algorithm mines interesting Emerging Patterns directly from the raw data.

ConsEPMiner (Zhang et al. 2000a) mines Emerging Patterns satisfying several constraints including the growth-rate improvement constraint. It follows an Apriori level-wise, candidate generation-and-test approach. It is still not efficient when the minimum support is low. For the UCI Connect-4 dataset, ConsEPMiner needs about 6 hours when the support threshold is 3%. In comparison, our algorithm can finish in less than 10 minutes, with little loss of interesting patterns.

Recent work in (Li & Wong 2002a) proposed to use "shadow patterns" to measure the interestingness of minimal JEPs. Shadow patterns are those immediate subsets of a minimal JEP. If the growth rates of these shadow patterns are on average around small numbers like 1 or 2, compared with the infinite growth rate of the JEP, it is regarded as *adversely interesting*, because the JEP is "unexpected" and the conflict may reveal some new insights into the correlation of the features. Their interestingness measure is a specific case of our correlation measure, since the level of adversity can be detected by χ^2-test. They do post-analysis of mined JEPs, while we push the interestingness measures into the mining process.

5.7.2 Interestingness Measures

The development of interestingness measures is currently an active research area in KDD.

A formalism of rule templates (a form of pattern expressions) is proposed in (Klemettinen, Mannila, Ronkainen, Toivonen & Verkamo 1994) to easily describe the structure of interesting rules. Also users can use the templates to specify which rules are not interesting.

A fuzzy matching techniques is proposed in (Liu & Hsu 1996) to perform the post-analysis of rules. Existing rules from previous knowledge are regarded as fuzzy rules and are represented using fuzzy set theory. The newly discovered rules are then matched against the existing fuzzy rules using the fuzzy matching technique.

A distance metric is used in (Dong & Li 1998) to evaluate the importance of a rule by considering its unexpectedness in terms of other rules in its neighborhood.

In subjective interestingness research in data mining, work in (Silberschatz & Tuzhilin 1995, Silberschatz & Tuzhilin 1996, Padmanabhan & Tuzhilin 1998, Padmanabhan & Tuzhilin 1999) discovers unexpected patterns that takes into consideration prior background knowledge of decision makers. This prior knowledge constitutes a set of expectations or beliefs about the problem domain and is used to seed the search for patterns in data that contradict the belief, by comparing the user's knowledge with the discovered rules.

It shows in (Bayardo Jr. & Agrawal 1999) that the best (optimal, most interesting) rules according to a variety of metrics reside along a support/confidence border.

In (Freitas 1998), a measure is described that determines the interestingness (called surprise there) of discovered knowledge via the explicit detection of Simpson's paradox.

Looking at the problem from another perspective, work in (Sahar 1999) attempts to discover interesting rules via an interactive process that eliminates rules that are not interesting.

Work in (Hilderman & Hamilton 2003) introduces twelve diversity measures (these measures are from various disciplines, such as information theory, statistics, ecology, and economics) used as heuristic measures of interestingness for ranking summaries generated from databases.

5.7.3 Related Work Using Chi-square Test

Work in (Brin, Motwani & Silverstein 1997) analyzes contingency tables to generate dependence rules that identify statistical dependence in both the presence and absence of items in itemsets.

Work in (Liu, Hsu & Ma 1999) first prunes those insignificant associations accord-

ing to chi-square test, and then finds a special subset (called direction setting rules or simply DS rules) of the unpruned associations to form a summary of the discovered associations. The DS rules represent the essential relationships or structure or skeleton of the domain, while non-DS rules give additional details. Their experiments show that the set of DS rules is typically small to be analyzed by a human user. It is an interesting idea to summarize the discovered Emerging Patterns somehow to provide a brief picture of the differences between classes of data.

CMAR, a classifier based on multiple association rules (Li, Han & Pei 2001), calculates chi-square values for each rule and uses a weighted chi-square measure to integrate both information of correlation and popularity, in order to make better classifications based on the combined effect of a group of rules.

Work in (Morishita & Sese 2000) suggests measuring the usefulness of an association rule by the significance of correlation between the assumption and the conclusion, according to chi-squared value or other common statistical measures. It estimates a tight upper bound on the statistical metric associated with any superset of an itemset to prune unproductive supersets while traversing itemset lattices like the Apriori algorithm. Later, (Sese & Morishita 2002) presents a heuristic for the calculation of the N most significant association rules, where N can be specified by the user. It uses a vertical decomposition of a database and the heuristic for pruning unproductive itemsets when traversing a set enumeration tree of itemsets.

It can be seen that both our use of chi-square test and our mining methodology are different from the above work.

5.8 Chapter Summary

In this chapter, we have introduced four objective interestingness measures for Emerging Patterns and developed an efficient algorithm, IEP-Miner, for mining only the interesting Emerging Patterns (Chi EPs). Using the four objective interestingness measures as constraints, IEP-Miner operates directly and solely on a tree data structure. The chi-squared heuristic is also used to prune a huge search space by growing only promising

branches. This achieved considerable performance gains: the heuristic makes IEP-Miner orders of magnitude faster. Although it gives up the completeness of Chi EPs, the heuristic always discovers the top 90% Chi EPs, as shown by detailed analysis over typical datasets from the UCI Machine Learning Repository. Moreover, the Chi EPs discovered by our method are indeed of high quality and excellent candidates for building accurate classifiers, which is confirmed by the high accuracy of the classifiers built upon these Chi EPs.

We state our main contributions in the following:

- Not all EPs are useful for classification purpose or for understanding the domain. Based on previous work on EPs and our experience in using EPs for classification, we define interestingness measures for EPs, in order to characterize the useful EPs. Chi-square test is used to check the relationship between an EP and its subsets. A Chi EP does not behave like its subsets (refer to condition 3 and 4 in Definition 5.1), so it provides new extra information.

  Although chi-test has been used widely in other works, we first apply chi-test on the context of EPs and we perform the chi-test differently.

- First mining all EPs and then selecting interesting ones is not a good idea. We have developed an algorithm for mining only Chi EPs, by pruning uninteresting ones as early as possible. The algorithm is inspired by FP-Growth, but it is fundamentally different, because the properties of EPs are very different from frequent patterns.

- Chi-squared test is used as heuristic to prune the search space. We point out the heuristic is admissible both in theory and by experiments. The heuristic makes our algorithm orders of magnitude faster, with little loss of useful patterns.

Chapter 6

Bayesian Classification by Emerging Patterns

Classification of large datasets is an important data mining problem. As a new type of data mining patterns, Emerging Patterns (EPs) are very useful for constructing accurate classifiers, as demonstrated by the success of the previous EP-based classifiers (Dong, Zhang, Wong & Li 1999, Zhang et al. 2000b, Zhang et al. 2001, Li, Dong & Ramamohanarao 2001, Li, Ramamohanarao & Dong 2001, Li & Wong 2002c, Bailey et al. 2003a, Li, Dong, Ramamohanarao & Wong 2004).

In this chapter, we will discuss a novel approach to use Emerging Patterns as a basic means for classification. It is called Bayesian Classification by Emerging Patterns (BCEP). As a hybrid of the EP-based classifier and Naive Bayes (NB) classifier, it provides several advantages. First, it is based on theoretically well-founded mathematical models to predict an unseen case given a training sample. Second, it extends NB by using essential Emerging Patterns to relax the strong attribute independence assumption. Lastly, it is easy to interpret, as many unnecessary Emerging Patterns are pruned based on data class coverage. An empirical study carried out on a large and varied selection of benchmark datasets from the UCI Machine Learning Repository shows that our method is superior to other state-of-the-art classification methods such as Naive Bayes (NB), the decision tree classifier C5.0, Classification by Aggregating Emerging Patterns (CAEP), Large Bayes (LB),

Classification Based on Association (CBA), and Classification based on Multiple Association Rules (CMAR) in terms of overall predictive accuracy.

Because real-world classification problems almost always contain noise, in Chapter 7, we will address the following question: how can a classifier cope with noisy training data.

Organization: An outline of the remainder of this chapter is as follows. Section 1 reviews related work and motivates our work. Section 2 introduces the basic idea of our Bayesian Classification by Emerging Patterns (BCEP) classifier. Section 3 defines the conditions of Emerging Patterns that will be used for classification. In section 4 the idea of data class coverage is introduced to further prune a lot of unnecessary Emerging Patterns. Section 5 details our Bayesian approach to use Emerging Patterns for classification. In section 6 we compare BCEP with LB. Section 7 presents an extensive experimental evaluation of BCEP on popular benchmark datasets from the UCI Machine Learning Repository and compares its performance with Naive Bayes (NB), the decision tree classifier C5.0, Classification by Aggregating Emerging Patterns (CAEP), Large Bayes (LB), Classification Based on Association (CBA), and Classification based on Multiple Association Rules (CMAR). Finally, in section 8 we provide a summary.

6.1 Background and Motivation

Since our classification method, called Bayesian Classification by Emerging Patterns (BCEP), combines the ideas of EP-based classifiers and the Naive Bayes (NB) classifier, we will discuss the two distinct families of classification methods separately.

6.1.1 The Family of Classifiers Based on Emerging Patterns

Emerging Patterns are defined as multivariate features (i.e., itemsets) whose supports (or frequencies) change *significantly* from one class to another. Because Emerging Patterns represent distinctions inherently present between different classes of data, the concept of Emerging Patterns is well suited to serve as a classification model. Since the conception of Emerging Patterns (Dong & Li 1999), two EP-based classifiers, CAEP (Classification by Aggregating Emerging Patterns) (Dong, Zhang, Wong & Li 1999) and JEP-Classifier (Li,

Dong & Ramamohanarao 2001) were proposed. Both classifiers aggregate each individual EP's sharp differentiating power to compute the aggregate scores for each class. The result is the class with the highest value of such scores.

We take CAEP for example to briefly explain how it works since JEP-Classifier works in a similar way[1]. The main idea of CAEP is: each EP can have very strong power for differentiating the class membership of some instances; such power of an EP is roughly proportional to the growth rate of its supports and its support in the target class. During the training phrase, the training database is first partitioned according to the class label. Then CAEP employs ConsEPMiner (Zhang et al. 2000a) to mine EPs for each class. When classifying a test instance T, it aggregates all contributions of EPs appearing in the test to derive an aggregate score for each class. The score for C_i is then "normalized" by dividing it by some base score (e.g. median) of the training instances of C_i, in order to reduce the negative effect of unbalanced distribution of EPs among the classes. These scores measures the weight which we put on the class label of the test.

Through our experience in using EPs for classification, we have identified the following weakness of the above EP-based classifiers.

1. They almost always depend on a huge number of EPs, which makes the resulting classifiers very complex. Although individual EP is easy to understand, users are overwhelmed by tens of thousands of EPs and do not know what kinds of EPs play important roles in the classification decision. Based on Table 5.2 on page 97 from (Zhang 2001) and Table 5.2 on page 84 from (Li 2000), it turns out that both CAEP and JEP-Classifier use an average of ten thousand EPs or JEPs.

2. The aggregation approach may count repeated contributions. Suppose two EPs which cover exactly the same percentage of the training sample, the contributions of both of them are added. This kind of repeated contributions is undesirable and should be avoided whenever possible.

3. The "normalization" is somewhat intuitive. The normalization is introduced to resolve

[1]Please refer to Chapter 3 Section 3.3 for a general framework of EP-based classifiers and more details (such as scoring functions) of CAEP and JEP-Classifier.

the problem caused by unbalanced distributions of EPs for different classes, i.e., a class which has many more EPs tends to get higher scores. Although it is successful, how to choose the base score remains the art of human.

4. JEP-Classifier uses exclusively *Jumping* Emerging Patterns (JEPs). JEPs are a special type of Emerging Patterns which have infinite growth rates, i.e., itemsets whose support increases abruptly from zero in one data class, to non-zero in another class – the ratio of support-increase being infinite. Only the supports of JEPs are aggregated to compute the scores. JEP-Classifier performs well when there exist many JEPs in each class of a dataset. However, in real world classification problems, some data classes may contain few or even no JEPs whose supports meet a reasonable threshold (such as 1%). In such cases, EPs with large but finite growth rates will play an important role.

To overcome the weaknesses of previous EP-based classifiers, we propose to combine the power of Emerging Patterns and Bayesian theory. Emerging Patterns are excellent discriminators for distinguishing one class of data from another. Bayesian theory provides us a sound base to build a scoring function using EPs. Next, we will discuss the stream of Bayesian classifiers.

6.1.2 The Family of Bayesian Classifiers

Bayes' theorem tells us how to optimally predict the class of a previously unseen example, given a training sample (Duda & Hart 1973, Cheeseman & Stutz 1996). The chosen class should be the one which maximizes

$$P(C_i|T) = \frac{P(T, C_i)}{P(T)} = \frac{P(C_i)P(T|C_i)}{P(T)},$$

where C_i is the class label, $T = \{v_1, v_2, \cdots, v_n\}$ is the test case, $P(Y|X)$ denotes the conditional probability of Y given X, and probabilities are estimated from the training sample. $P(C_i|T) = P(C_i|v_1, v_2, \cdots, v_n)$ is the posterior probability of class C_i given the attribute values, $\{v_1, v_2, \cdots, v_n\}$, observed in the test instance. Since classification focuses on discriminate prediction of a single class, rather than assigning explicit probabilities to each class, the

denominator $P(v_1, v_2, \cdots, v_n)$, which does not affect the relative ordering of the classes, can be omitted. So the class is chosen with the highest probability $P(T, C_i) = P(C_i)P(T|C_i)$. Because in practice it is very hard to calculate the exact probability $P(T, C_i)$, one must use approximations under certain assumptions.

The Naive Bayesian (NB) Classifier

As discussed in Chapter 2 Section 2.3.3, the Naive Bayesian (NB) classifier (Duda & Hart 1973) makes a simple assumption that all attributes are independent given the class C_i. So we have

$$P(T, C_i) = P(a_1 a_2 \cdots a_n C_i) =$$
$$P(C_i)P(a_1|C_i)P(a_2|C_i)\cdots P(a_n|C_i) = \frac{\prod_{j=1}^{n} P(a_j, C_i)}{P(C_i)^{n-1}},$$

which means that the classification solely depends on the values of $P(a_j, C_i)$ and $P(C_i)$. NB has the ability of evidence accumulation from multiple attributes. Despite its simplicity, NB is a surprisingly successful classification method that has demonstrated to outperform much more complicated methods such as decision tree, rule learning and instance-based learning algorithms in many application domains (Langley, Iba & Thompson 1992, Domingos & Pazzani 1996). However, the assumption that the attributes are independent with respect to the class variable is generally false in many real applications.

Previous Work on Improving the Naive Bayesian Classifier

The surprising success of NB has triggered the development of several extensions, most of which aim at relaxing the strong independence assumptions of NB.

A semi-naive Bayesian classifier is proposed in (Kononenko 1991) to detect the dependencies between attributes by performing exhaustive search to iteratively join pairs of attribute values. Its idea is to optimize the tradeoff between the "non-naivety" and the reliability of approximation of probabilities.

NBtree (Kohavi 1996) is hybrid approach combining NB and decision-tree learning, where a decision tree partitions the instance space into regions and a separate NB classifies cases within each region.

Tree Augmented Naive Bayesian classifier (TAN) (Friedman et al. 1997) extends NB by considering additional dependencies among non-class attributes. TAN is based on the theory of learning Bayesian networks, which are factored representations of probability distributions that generalize the naive Bayesian classifier and explicitly represent statements about independence. TAN outperforms naive Bayes, yet at the same time maintains the computational simplicity (no search involved) and robustness that are characteristic of naive Bayes.

Adjusted Probability NB (Webb & Pazzani 1998) infers a weight for each class, which is applied to derive an adjusted probability estimation used for the classification.

Lazy Bayesian rule learning algorithm (LBR) (Zheng & Webb 2000) applies lazy learning techniques to Bayesian tree induction which supports a weaker conditional attribute independence assumption. For each example, it builds a most appropriate rule with a local naive Bayesian classifier as its consequent.

Approximate Probabilities Using Itemsets

The probability $P(T, C_i)$ can be estimated using different product approximations, where each product approximation assumes different independence of the attributes (Lewis 1959). For instance, $P(a_1a_2a_3C_i)P(a_4a_5|a_1C_i)$ and $P(a_1a_2a_3C_i)P(a_5|a_2C_i)P(a_4|a_1a_5C_i)$ are both product approximations of $P(a_1a_2a_3a_4a_5C_i)$, where the first product approximation uses $\{a_1a_2a_3\}$ first and then $\{a_1a_4a_5\}$; the second uses $\{a_1a_2a_3\}$, $\{a_2a_5\}$ and $\{a_1a_4a_5\}$ sequentially.

We use the following example to illustrate the advantage of using itemsets rather than individual items to approximate probabilities.

a	b	c
a	b	
a		c
a	b	c

Obviously, we have $p(a) = 1$, $p(b) = 3/4$, $p(c) = 3/4$, $p(ab) = 3/4$, $p(ac) = 3/4$, $p(bc) = 1/2$, $p(abc) = 1/2$. Because $p(ab) = p(a)p(b)$, and $p(ac) = p(a)p(c)$, we can see that a is independent from b and that a is independent from c. However, since $p(bc) \neq p(b)p(c)$, it can be seen that b is dependent on c, and vice versa.

Suppose we want to approximate $p(abc)$. NB assumes that a, b, c are all independent from each other. So using the NB approach, we will have

$$p(abc) = p(a)p(b)p(c) = 1 \times 3/4 \times 3/4 = 9/16.$$

If we use itemsets $\{a\}$ and $\{b, c\}$, we can relax the strong assumption of NB, i.e., we only require that a should be independent from b and c, but we do not require that b and c are independent from each other. Therefore, using the two itemsets, we will have

$$p(abc) = p(a)p(bc) = 1 \times 1/2 = 1/2.$$

From this example, we can see that a more accurate probability estimation is made by using itemsets, because the strong independence assumption is weakened. (Note that items are attributes actually.)

Large Bayes (LB) (Meretakis & Wuthrich 1999) is a recently proposed extension of NB using long itemsets to relax the strong independence assumptions implied by NB. It is called Large Bayes because it happens to reduce to NB when all itemsets selected are of size one only. LB uses the supports of interesting long, frequent (large) itemsets to approximate probabilities. It selects a certain product approximation by selecting certain large itemsets. In order to make the approximation reliable, only "interesting" large itemsets are used. LB's accuracy was claimed to be consistently better than NB, and generally better than that of the widely known and used decision tree classifier C4.5 (Quinlan 1993) and TAN (a Bayesian network extension of NB) (Friedman et al. 1997). LB is discussed further and compared with our BCEP in section 6.

6.2 Combining the Two Streams: Bayesian Classification by Emerging Patterns (BCEP)

Having discussed the two different streams of classifiers, we now put them together. To overcome the weaknesses of previous EP-based classifiers, we propose to combine the power of Emerging Patterns and Bayesian theory, which leads to Bayesian Classification by Emerging Patterns (BCEP). BCEP can be thought of as extending NB from a new

direction. Instead of looking at "raw" training instances, it focuses on the regular patterns (i.e., Emerging Patterns) found from the training data.

An EP can be regarded as a distinguishing feature of its home class. Its support in its home class estimates the probability that the pattern occurs given the home class. And its support in the contrasting class estimates the probability that the pattern occurs given the contrasting class. EPs are itemsets which maximize the former probability and minimize the latter. EPs express the relationships between items (attributes), indicating that those items (attributes) contained in a single EP are not actually independent. Recall that if X and Y are independent given Z then $P(X|Z,Y) = P(X|Z)$. As an extreme case, an item a_1 appears in half of class C_1 and an item a_2 appears in the other half of class C_1 (therefore, a_1 and a_2 can not appear at the same time in C_1); while both a_1 and a_2 appear in the same half of class C_2. Then $\{a_1, a_2\}$ is an EP with support 50% in C_2 and 0% in C_1. Obviously, $P(a_1|C_1, a_2) = 0$, $P(a_1|C_1) = 0.5$, $P(a_1|C_2, a_2) = 1$, $P(a_1|C_2) = 0.5$. So $P(a_1|C_1, a_2) \neq P(a_1|C_1)$ and $P(a_1|C_2, a_2) \neq P(a_1|C_2)$. We can see that both a_1 and a_2 are not independent given either C_1 or C_2. If we use EPs in the product approximation, we can relax the strong independence assumption implied by NB.

There can be a very large number (e.g., 10^9) of general EPs in the dense and high-dimensional dataset of a typical classification problem. We have shown that EPs that satisfy our four interestingness measures (Chapter 5 Definition 5.1) are the essential discriminating knowledge for classification. The first two interestingness measures (conditions 3 and 4 in Definition 5.1) need two thresholds, namely, the minimum support threshold ξ and the minimum growth rate threshold ρ. Based on our previous experience in using EPs for classification, we use a typical threshold of 1% as an EP's minimum support in its home class; we use a typical threshold of 100 as an EP's minimum growth rate. Intuitively, $\xi = 1\%$ requires these EPs should cover enough training instances to resist noise and avoid the problem of small disjuncts. Large growth rates ($\rho = 100$) ensure that these EPs should have sharp discriminating power. These EPs must also satisfy the remaining two interestingness measures (conditions 3 and 4 in Definition 5.1). Condition 3 requires that these EPs should have larger strength than their subsets. In other words, we prefer a shorter EP as long as its super sets do not have larger strength. These EPs can be regarded as the

"minimal" EPs with the largest strength. The last condition ensures that all the items that make up of an EP contribute *significantly* to the EP's discriminating power, i.e., support, growth rate and strength. That is, these EPs are "correlated". By using EPs satisfying the above four conditions instead of general EPs (where only conditions 1 and 2 are used), the number of pattern involved in classification is much smaller. To efficiently mine the complete set of those EPs, we use a variant of the IEP-miner (no chi-square pruning heuristic is used) discussed in Chapter 5.

Pruning rules based on data coverage is a commonly employed method to reduce the number of rules in rule based classification systems. Applying this kind of pruning may reduce the effect of repeated contributions which is mentioned above and further reduce the number of EPs to be used in classification. Suppose an EP e_1 covers[2] a set of training instances and another EP e_2 covers the same set or a subset of those training instances. We can see that e_2 does not provide any more useful information, as both of their growth rates are very large and the support of e_1 is larger than that of e_2. So e_2 can be safely removed without losing much classification information. After a set of interesting EPs are mined, as the last step in the training phrase, we select only a small set of high quality EPs based on data coverage. The final set of EPs which is used in classification is even smaller, which is around a few hundreds and no more than one tenth of the EPs used by CAEP and JEP-Classifier. Experimental results show that such pruning does not lose any useful patterns for classification and actually removes noisy or unnecessary patterns. We also notice fewer EPs lead to faster classification.

Reducing the number of EPs alone will not work unless we have a better, more reliable approach to use the EPs for discriminating classification learning. We choose Bayesian theory as the underlying classification basis. BCEP uses EPs of arbitrary size for estimating $P(T, C_i)$. Under different independence assumptions about the attributes, the probability $P(T, C_i)$ can be obtained using different product approximations. EPs are combined using the chain rule of probability when all necessary independence assumptions are assumed true. We stress that it is due to the mathematical soundness of our method, we are able to use only a few EPs to build high accuracy classifiers.

[2]Here we say an EP "covers" an instance if the instance contains the EP.

Let F denote the final high quality EPs ready for classification. When a new case $T = \{a_1, a_2 \cdots a_n\}$ arrives to be classified, BCEP combines the evidence provided by the subsets of T that are present in F to approximate $P(T, C_i)$, which determines the conditional probability $P(C_i|T)$, i.e., the probability that the case belongs to class C_i given the evidence. The evidence which is selected from F is denoted as B. An EP from F, with its high support in its home class and low support in the contrasting class, can be seen as a strong signal which implies the items contained in the EP are dependent. The strength of such a signal is expressed by its growth rate and its support in the both classes. We select EPs from B sequentially in the strength-descending-order to construct the product approximation of $P(T, C_i)$ and use their supports in both classes to calculate the probabilities. The result is the class with the highest value of $P(T, C_i)$. The strength-descending-order is important because different combinations of itemsets lead to different product approximation, and even when the same itemsets are used in different order, the product approximations are still different. Experiments show that our approach to select and aggregate a small number of essential EPs is an effective strategy to compute the class probabilities.

We experimentally test our Bayesian approach to use Emerging Patterns for classification, using benchmark problems from the University of California at Irvine repository (Blake & Merz 1998), and compare BCEP to Naive Bayes (NB), the decision tree classifier C5.0, Classification by Aggregating Emerging Patterns (CAEP), Large Bayes (LB), Classification Based on Association (CBA), and Classification based on Multiple Association Rules (CMAR). Our performance study shows that BCEP is superior to other state-of-the-art classification methods in terms of overall predictive accuracy, and that it achieves the best or very close (within 1%) to the best accuracy on 22 out of 32 datasets and also performs very well on the remaining ten.

We highlight our main contributions in this chapter as follows:

1. We have proposed a hybrid system of the EP-based classifier family and the Bayesian classifier family, called BCEP. As a new extension of NB, it relaxes NB's strong attribute independence assumption by using Emerging Patterns of arbitrary size for estimating probabilities; it also retains NB's strength.

2. We have greatly improved the scoring functions, which is shown both in theory and experiments. BCEP is based on theoretically well-founded mathematical model to compute the probability that the case belongs to some class given the evidence. The scoring functions of previous EP-based classifiers are coarse approximations of such probability.

3. We have successfully applied the idea of pruning EPs by coverage to solve the problem of interpretability of a classifier, which is an important issue in data mining. Not only the number of EPs used to make a decision is greatly reduced, but also higher accuracy is achieved.

6.3 Emerging Patterns Used in BCEP

In our Bayesian classification approach, we use the EPs that satisfy the following conditions:

Condition 1 They have enough supports in its home class (usually the support threshold is 1%);

Condition 2 They have large growth rates (typically more than 100);

Condition 3 They have large strength than any of their subsets.

Condition 4 They pass chi-square test, as specified in Definition 5.1.

Basically, the above four conditions conform to the four interestingness measures in Definition 5.1. We stress that EPs satisfying the above four conditions are the most expressive patterns for classification.

1. The minimum support threshold makes every EP cover at least a certain number of training instances. Therefore, these EPs are reliable for product approximations, because itemsets with too low supports are regarded as noise.

2. Very large or even infinite growth rates ensure that each EP should have significant level of discrimination. However, we do not want to use infinite growth rate threshold.

An EP of infinite growth rate means that it must have zero support in the contrasting class. This is a very strict constraint. If it had a very small support in that class (e.g. 0.1%), it would not be generated. But since this EP has large (while not infinite) growth rate, it also has very sharp discriminating power. A threshold of infinite may lose many such good EPs. Our experiments indicate that 100 is a good choice.

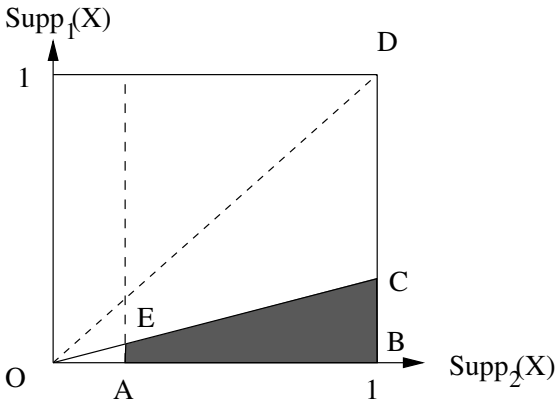
3. The third condition means that they are minimal or the most general, i.e., any proper subset will not have such a large growth rate. In our Bayesian approach, as many EPs as possible should be used, i.e., the product approximations should contain as many factors as possible. So short EPs are also desirable.

   Supersets are not useful for classification because of the following reason. Suppose $e_2 \supset e_1$, where e_1 satisfy conditions 1 and 2. e_1 covers more (at least equal) training instances than e_2 because $supp(e_1) \geqslant supp(e_2)$. And by condition 3, e_2 does not have higher strength. So e_2 does not provide any more information for classification than e_1.

4. The last condition ensures that all the items contained in these EP should be "correlated", i.e., they appear together not because of chance. Experiments show that these "correlated" EPs are high quality pattern or itemsets for product approximations in our Bayesian approach.

We show the kind of EPs that will be used for Bayesian classification in Figure 6.1. These EPs are the minimal patterns that occupy the area $ABCE$, which is very close to x-axis due to the large growth rate threshold. They do not need to have zero support in the contrasting class, which relaxes the strict requirement of EJEPs (Definition 4.1) discussed in Chapter 4.

The learning phase of BCEP employs IEP-miner discussed in Chapter 5 to efficiently mine EPs from the training data, namely itemsets passing the user-specified support and growth rate thresholds, the "minimal" requirement and chi-square test (**Condition 1, 2, 3** and **4**). Note that we do not use chi-square pruning heuristic during the process of mining to ensure that the complete set of EPs satisfying the four conditions are generated.



Note: $\tan \angle BOC = \frac{1}{\rho}, (\rho \in [100, 1000])$ and $|OA| = \xi$

Figure 6.1: The support plane for Emerging Patterns used in BCEP

6.4 Pruning Emergin Patterns Based on Data Coverage

The number of the EPs generated in the learning phase can be huge. To make the classification effective and efficient, we need to further prune unnecessary EPs to delete redundant and noisy information.

The pruning is based on a global order defined on the EPs.

Definition 6.1 Given two EPs, e_i and e_j $(i \neq j)$, e_i is said to have higher rank than e_j (also called e_i precedes e_j), denoted as $e_i \prec e_j$, if and only if

1. the support of e_i is greater than that of e_j, or

2. their supports are the same, but the length of e_i is smaller than that of e_j.

The basic idea is to choose a set of high precedence EPs from those discovered to cover the training data. This method is related to the traditional covering method. However, the major difference is in the rules or patterns that they use. We discover all patterns from the entire training data using exhaustive search, while each rule in the traditional covering method is learned using a heuristic method from the remaining data after the examples covered by previous rules are deleted, where high quality rules may lose due to greedy heuristic.

Algorithm 6.1: Prune Emerging Patterns based on data class coverage

input : a set E_i of EPs for class C_i, the training dataset D_i of class C_i and a coverage threshold θ

output: a final set of EPs F_i for classification

```
1  Sort E_i by rank in descending order;
   /* S refers to the set instances that have not been ''covered'' */
2  S ← D_i;
3  F_i ← ∅;
   /* s.covered refers to the number of EPs that cover the instance s */
   /* cover-more is a boolean variable.  cover-more is true when an
     EP ''covers'' more new instances that have not been covered;
     otherwise, cover-more is false */
4  foreach instance s ∈ S do s.covered =0;
5  while S ≠ ∅ ∧ E_i ≠ ∅ do
6      foreach EP e ∈ E_i in the rank descending order do
7          cover-more ← false ;
8          foreach instance s ∈ S do
9              if e ⊆ s then
10                 s.covered ++;
                   /*Only if s has been covered by a certain number of EPs,
                     then remove s from S */
11                 if s.covered ⩾ θ then S ← S − {s};
12                 cover-more ← true ;
               end
           end
           /*If e covers at least one more instance of the remaining
             dataset, e is useful for classification.  Only in that case,
             we add e to F_i. */
13         if cover-more == true then F_i = F_i + {e};
       end
   end
```

We use Algorithm 6.1 to prune the set E_i of discovered Emerging Patterns based on data class coverage with respect to class C_i. The pruning is based on the assumption that if each instance in the training dataset is covered by a given set of high quality EPs, a new test should also be covered by the same set of EPs, because the training and testing dataset supposedly have the same distribution. We understand that such assumption is not always true in the real world classification problems, so we choose more EPs by using the coverage threshold θ ($\theta \geqslant 2$). Instead of removing one instance from the training dataset immediately after it is covered by an EP, we let the instance stay there until it is covered by at least θ EPs. EPs that can cover some "new" instances on the remaining dataset are selected; while EPs that fail to do so are discarded.

There are two reasons why we need more EPs.

- First, if too few EPs are chosen after pruning, some effective EPs may be lost.
- Second, more EPs are desirable in our Bayesian approach, because generally the more itemsets we use in the product approximation, the more reliable the product approximation is for classifying new data objects.

There are also reasons why we prefer fewer EPs (that is why we prune EPs), which have been discussed in the beginning of the section. This raises the following question, "How to determine θ?" Our empirical study on some datasets from the UCI Machine Learning Repository suggests that a value of 2 is a good choice for θ.

As a byproduct of the pruning process, the percentage of D_i covered by the set E_i of EPs is available. If we find the percentage is high (typically more than 90%), we are much confident that we have mined enough high quality EPs for classification. However, in rare cases when such percentage is low (typically below 80%), where there are few or even no EPs with very large (more than 1000) growth rates, it means the thresholds for EPs are too aggressive, hence we lose some high quality EPs. We have to adjust the growth rate threshold (reduce it to 10-100) and repeat the EP mining process to obtain a larger set of EPs. This can be done *automatically*: Repeat mining a set E_i of EPs until E_i covers enough percent of D_i.

To obtain the final set F of high quality EPs for classification, the pruning is done

on all classes in the training dataset D. Let the number of classes in D is c,

$$F = F_1 \cup \cdots \cup F_c = \bigcup_{i=1}^{c} F_i.$$

Compared with the thousands of EPs used by previous EP-based classifiers, the set F is quite small due to pruning by coverage. We emphasis that previous approaches need such a huge number of EPs to achieve good accuracy because of lacking sound theory underneath; and that the pruning of EPs may decrease their accuracy.

6.5 The Bayesian Approach to Use Emerging Patterns for Classification

6.5.1 Basic Ideas

When a new case $T = \{a_1, a_2, \cdots a_n\}$ is to be classified, BCEP combines the evidence provided by the subsets of T that are present in F, where F denotes the final set of high quality EPs for classification. The evidence is denoted as B, $B = \{s \in F | s \subset T\}$. BCEP uses the EPs of B to derive product approximations of $P(T, C_i)$ for all classes. The product approximation of the probability of an n-itemset T for a class C_i contains a sequence of at most n subsets of T such that each itemset contains at least one item not covered[3] in the previous itemsets. Recall the general chain rule is

$$P(X_1, X_2, \cdots, X_n) = P(X_1)P(X_2|X_1)\cdots P(X_n|X_1, ..., X_{n-1}).$$

To obtain the product approximation of $P(T, C_i)$, the itemsets are combined using the chain rule of probability while assuming that all necessary attribute independence assumptions are true.

Suppose a test instance $T = \{a_1, a_2, a_3, a_4, a_5\}$ arrives. After consulting F, we find its corresponding $B = \{\{a_2, a_5\}, \{a_3, a_4\}, \{a_1, a_2, a_3\}, \{a_1, a_4, a_5\}\}$. We can use some itemsets of B to make several different product approximations of $P(T, C_i)$ as follows:

I $\{a_1, a_2, a_3\}$, $\{a_1, a_4, a_5\} \Longrightarrow P(C_i)P(a_1a_2a_3|C_i)P(a_4, a_5|a_1C_i)$

[3] An item which is already included in the product approximation is said to be covered.

II $\{a_1, a_4, a_5\}$, $\{a_2, a_5\}$, $\{a_3, a_4\} \Longrightarrow P(C_i)P(a_1a_4a_5|C_i)P(a_2|a_5C_i)P(a_3|a_4C_i)$

III $\{a_2, a_5\}$, $\{a_3, a_4\}$, $\{a_1, a_4, a_5\} \Longrightarrow P(C_i)P(a_2a_5|C_i)P(a_3a_4|C_i)P(a_1|a_4a_5C_i)$

IV $\{a_1, a_2, a_3\}$,$\{a_2, a_5\}$,$\{a_3, a_4\} \Longrightarrow P(C_i)P(a_1a_2a_3|C_i)P(a_5|a_2C_i)P(a_4|a_3C_i)$

Note that $\{\{a_1, a_2, a_3\}, \{a_1, a_4, a_5\}, \{a_2, a_5\}\}$ is not a valid product approximation since all items of $\{a_2, a_5\}$ are already covered by the first two itemsets. We also point out that although II and III use the same itemsets $\{a_2, a_5\}$, $\{a_3, a_4\}$ and $\{a_1, a_4, a_5\}$, the final product approximation is different because these itemsets are used in different order.

We take I for example to illustrate the underlying assumptions.

$$P(T, C_i) = P(a_1, a_2, a_3, a_4, a_5, C_i) = P(C_i)P(a_1, a_2, a_3, a_4, a_5|C_i)$$

$$= P(C_i)P(a_1, a_2, a_3|C_i)P(a_4, a_5|a_1, a_2, a_3, C_i)$$

Because $\{a_1, a_2, a_3\}$, $\{a_1, a_4, a_5\}$ are EPs, we make the assumption that the different items of the two EPs are independent. In this case, $\{a_2, a_3\}$ is independent from $\{a_4, a_5\}$. Also note that the above assumption is weaker than the naive Bayes assumption. So,

$$P(T, C_i) = P(C_i)P(a_1, a_2, a_3|C_i)P(a_4, a_5|a_1, C_i)$$

is a better approximation.

Clearly, different combinations of itemsets of B lead to different product approximation, and the product approximations are different even when the same itemsets are used in different order. The product approximation of $P(T, C_i)$ is created incrementally adding one EP at a time until no more EPs can be added, that is, either all the items of the remaining EPs from B are already covered or no more EPs are available in B.

High support and high growth rate itemsets, that is, itemsets with large strength (see Definition 3.3) are obviously preferred for classification as they can be regarded as strong signals indicating the class label of a test instance containing it. The more strength an EP has, the earlier it should be used in the product approximation as long as it provides items which have not been covered. We are also concerned about EPs' length. When two EPs have the same strength, the shorter EPs should be used first in order to use more EPs in the product approximation.

EPs of B are first sorted in the strength-descending-order, and the final list is denoted as O. EPs are extracted from the list O from the beginning to incrementally construct the the product approximation. The set of covered items is denoted as cov. An EP p inserted in the product approximation should satisfy the following rules:

Rule 1: $|p - cov| >= 1$;

Rule 1 means p contains new items which have not been covered. It guarantees that the product solution satisfies the chain rule and hence it is a valid product approximation.

Selection among alternatives is done as follows. EP p is selected instead of another EP q if the following rules are satisfied in the given order of importance:

Rule 2: $strength(p) > strength(q)$;

Rule 3: $length(p) < length(q)$;

Rule 4: $|p - cov| \leqslant |q - cov|$.

Rule 2 ensures EPs with larger strength (stronger signals) be used first if they do not break Rule 1. Rule 3 assures shorter EPs be considered first if there are alternatives with the same strength. This is equivalent to maximizing the number of EPs used in the product approximation. Rule 4 gives priority to those EPs among the remaining alternatives which contain the smallest number of not covered items. This is last attempt to make as many EPs as possible stay in the sequence.

6.5.2 The Algorithm to Compute the Product Approximation

After the final set F of high quality EPs is available, a new unlabelled test $T = \{a_1, a_2, \cdots a_n\}$ is classified by the Algorithm 6.2. The supports in both classes and the strength of each EP can be calculated in the training phrase.

The algorithm incrementally builds the product approximation of $P(T, C_i)$ by adding one itemset (EP) at a time until no more can be added. It first finds the evidence B provided by the subsets of T that are present in F. *covered* is the subset of T already covered; *numerator* and *denominator* are the sets of itemsets in numerator and denominator, respectively. Procedure $Next(covered, B)$ (Algorithm 6.3) then repeatedly pick up

next itemsets from B. The algorithm stops once all items in T have been covered. In the for loop, the numerator and denominator itemsets, B_i and $B_i \cap covered$ respectively, are stored in two separate set *numerator* and *denominator*. Finally, they are used to derive the value of the product approximation of $P(T, C_i)$ for each class C_i. The most likely class is then returned.

Procedure $Next(covered, B)$ selects from B the next itemset to be used in the product approximation. This is uniquely determined by the Rules 1-4. To facilitate the selection, all the discovered EPs are sorted by the strength-descending order. Because strength is represented as real values, when we say two strength are equal, we do not require them *exactly* equal, instead, we require the absolute difference between the two values is less than 0.01. EPs with the same strength are further sorted by their length in the length-ascending order.

6.5.3 Zero Counts and Smoothing

The support (or observed frequency) of an itemset can be unreliable substitution for its probability, especially when the dataset is small or when the training dataset contains noise. A typical example is a Jumping Emerging Pattern (JEP) with a growth rate of ∞ ($\frac{\geq 0}{0}$), where the zero-support in the background class is very unreliable, and it is very undesirable to use zero in the product approximation.

A standard statistical technique is to incorporate a small-sample correction into the observed probabilities. This is called smoothing and helps eliminate unreliable estimates and zero-counts (Friedman et al. 1997). In our experiment, we use M-estimate with m=2 to estimate $P(X|C_i)$ and Laplace-estimate to estimate $P(C_i)$.

M-estimate:

$$\frac{\#(X, C_i) + n_0 \frac{\#(X)}{|D|}}{|D_i| + n_0}$$

where $\#(X, C_i)$ denotes the number of training examples belonging to class C_i and containing itemset X; $\#(X)$ denotes the number of training examples containing itemset X; n_0 is a small number which is set to 5.

Algorithm 6.2: Bayesian Classification by Emerging Patterns (BCEP)

input : a set of EPs F and a test instance T
output: the classification of T

/* Find all the EPs contained in the test */

1 $B = \{s \in F \mid s \subset T\}$;

/* *covered* refers to the set of items in T already covered */

2 $covered = \emptyset$;

/* *numerator* refers to the set of itemsets in numerator */

3 $numerator = \emptyset$;

/* *denominator* refers the set of itemsets in denominator */

4 $denominator = \emptyset$;

5 **while** *there exist items from T that have not been covered* ***AND*** *there exist EPs from B that have not been used* **do**

  /* Select the next ''best'' EP to use in the product approximation */

6 $e =$ next(*covered,B*);

  /* All the items contained in the EP will appear in the numerator */

7 $numerator = numerator \cup e$;

  /* Only the items of the EP that have been covered before will appear in the denominator */

8 $denominator = denominator \cup \{e \cap covered\}$;

  /* All the items contained in the EP are covered up to now */

9 $covered = covered \cup e$;

end

/* Approximate probabilities according to Bayesian rule and our assumptions (these assumptions are weaker than NB) */

10 **for** *each class C_i* **do**

11 $P(T, C_i) = P(C_i) \frac{\prod_{u \in numerator} P(u, C_i)}{\prod_{v \in denominator} P(v, C_i)}$;

end

12 output the class with maximal $P(T, C_i)$;

Algorithm 6.3: Function: $Next(covered, B)$, called by BCEP

input : a set of items that have been covered before, *covered*, and all the EPs contained in the test, B

output: the next "best" EP to use in the product approximation

1 $Z = \{s | s \in B \ \wedge \ |s - covered| \geqslant 1\}$;

2 return an itemset $B_i \in Z$ such that for all other itemsets $B_j \in Z$:

- $strength(B_i) > strength(B_j)$;
- $strength(B_i) = strength(B_j) \ \wedge \ length(B_i) < length(B_j)$;
- $strength(B_i) = strength(B_j) \ \wedge \ length(B_i) = length(B_j)$ $\wedge \ |B_i - covered| \leqslant |B_j - covered|$.

Laplace-estimate:

$$\frac{|D_i| + k}{|D| + c * k}$$

where $|D_i|$ is the number of training objects belonging to C_i; $|D|$ is the total number of training objects; c is the number of classes; and k is normally 1.

Let $P(X, Y, C_i)$ and $P(Y, C_i)$ be the observed frequencies in D of $\{X, Y, C_i\}$ and $\{Y, C_i\}$ respectively (X and Y are itemsets). Instead of $P(X|Y, C_i) = P(X, Y, C_i)/P(Y, C_i)$, we use the smoothed conditional probability:

$$P(X|Y, C_i) = \frac{|D| * P(X, Y, C_i) + 5 * P(X)}{|D| * P(Y, C_i) + 5}.$$

6.6 A Comparison between BCEP and LB

Both BCEP and LB are powerful extensions of NB that uses itemsets of arbitrary size when estimating $P(T, C_i)$. They are very different from probabilistic inference methods in Bayesian Network literature (Friedman et al. 1997), where conditional independence testing methodology is investigated. Because most real datasets contain only a small fraction of the possible attribute-value pairs (also called items), both classifier focus on estimating the probabilities of sets of items which satisfy certain conditions in the training dataset. When classifying a test instance, they use the same framework to incrementally build the

product approximation by selecting one itemset at a time according to some rules until no more itemsets can be added.

The underlying theory has been discussed in (Lewis 1959). For example, the probability $P(a_1a_2a_3a_4a_5C_i)$ can be estimated using different product approximations, where each product approximation assumes different independence of the attributes. For instance, $P(a_1a_2a_3C_i)P(a_4a_5|a_1C_i)$ and $P(a_1a_2a_3C_i)P(a_5|a_2C_i)P(a_4|a_1a_5C_i)$ are both product approximations of $P(a_1a_2a_3a_4a_5C_i)$. The first product approximation uses $\{a_1a_2a_3\}$ first and then $\{a_1a_4a_5\}$ and it assumes that the items a_4 and a_5 are independent from a_2 and a_3 given C_i. The second uses $\{a_1a_2a_3\}$, $\{a_2a_5\}$ and $\{a_1a_4a_5\}$ sequentially, and makes similar assumptions.

Our idea of using EPs instead of frequent itemsets to compute those product approximations is motivated by the following example. Given two classes of data C_1 and C_2 as follows.

	C_1	C_2
1	$\{a_1a_2a_3a_5\}$	$\{a_1a_3a_6\}$
2	$\{a_1a_2a_3a_4\}$	$\{a_3a_5\}$
3	$\{a_3a_4a_6\}$	$\{a_1a_5\}$
4	$\{a_4a_5\}$	$\{a_2a_3a_5\}$

To estimate the probability $P(a_1a_2a_3a_4|C_1)$, both BCEP and LB select some itemsets which are subsets of $\{a_1a_2a_3a_4\}$. The "boundary" frequent itemsets selected by LB with respect to $\{a_1a_2a_3a_4\}$ are $\{a_1a_2a_3\}$ and $\{a_3a_4\}$. The essential EPs "contained" in $\{a_1a_2a_3a_4\}$ are $\{a_1a_2\}$ and $\{a_3a_4\}$. Based on LB strategy which uses frequent itemsets, we have

$$P(a_1a_2a_3a_4|C_1) \approx P(a_3a_4|C_1)P(a_1a_2|a_3C_1) = 0.5 * 1 = 0.5$$

Our BCEP uses EPs, therefore,

$$P(a_1a_2a_3a_4|C_1) \approx P(a_1a_2|C_1)P(a_3a_4|C_1) = 0.5 * 0.5 = 0.25.$$

Clearly, from data class C_1, we have $P(a_1a_2a_3a_4|C_1) = 0.25$. Thus BCEP provides a better product approximation by using EPs. From this example, we make the following argument: it is more likely in the real datasets that the different items of two EPs are independent than the different items of two frequent itemsets are independent. It is also supported by the experiments.

Although influenced by LB, BCEP is different from LB.

- First, BCEP uses EPs, which are itemsets frequent in one data class while very infrequent in another. In contrast, LB uses frequent itemsets. EPs are better discriminatory between data classes than frequent itemsets, and contain more useful information, i.e., the growth rate from one class to another.

- Second, LB uses Apriori (Agrawal & Srikant 1994) algorithm to discover frequent itemsets while BCEP uses tree based algorithms to mine EPs. It is well-known that on dataset that is large, dense and has high dimensionality, Apriori and its variants are unable to mine frequent itemsets at low supports (Bayardo et al. 2000). Our tree based algorithms can mine EPs with large growth rate at low support level, thus take into account all aspects of the underlying model, i.e., no important patterns are missed.

- Third, LB selects frequent itemsets by their interestingness and the interestingness of an itemset l is defined in terms of the error when estimating $P(l, C_i)$ using subsets of l. While BCEP picks EPs sequentially in the strength-descending-order to construct the product approximation of $P(T, C_i)$, where the strength of an EP is based on its support and growth rate. EPs with large strength are good indicator of class membership, but frequent itemsets, even with big interestingness values, have no such predictive power.

- Finally, experimental results show that in most cases BCEP achieves higher accuracy than or the same accuracy as LB, which confirms the advantages of BCEP over LB.

6.7 Experimental Evaluation

In this section, we present the performance study and experimental resluts for our BCEP classifier. All the experiments were performed on a 500Mhz Pentium III PC with 512Mb of memory. We use the Entropy method described in (Fayyad & Irani 1993) taken from the $\mathcal{MLC}$++ machine learning library (Kohavi et al. 1994) to discretize datasets containing continuous attributes.

6.7.1 Accuracy Study

In order to investigate BCEP's performance compared to that of other classifiers, we carry experiments on 34 datasets from the UCI Machine Learning Repository. We compare BCEP with six other state-of-the-art classifiers: Naive Bayes (NB), the widely known decision tree induction C5.0, CAEP, LB (a recently proposed classifier extending NB using long itemsets), CBA (Classification Based on Association) and CMAR (Classification based on Multiple Association Rules). Note that both CBA and CMAR have been discussed in Chapter 2 Section 3.3.

Table 6.1 summarizes the accuracy results. Columns 1 gives the name of each dataset. Columns 2 to 8 give the predictive accuracy of the classifiers. The accuracy of NB, C5.0 and CAEP was obtained by using the methodology of *stratified* ten-fold cross-validation (CV-10), on exactly the same CV-10 training and testing data. We use WEKA's Java implementation of NB (Witten & Frank 1999). We use C5.0 [Release 1.2], a commercial version of C4.5 (Quinlan 1993). For CAEP, the default parameter settings are: the minimum support threshold $\xi = 1\%$ or an absolute count of 5; the minimum growth rate $\rho = 5$; and the minimum relative growth-rate improvement equals 1.01. The accuracy of LB (Meretakis & Wuthrich 1999), CBA (Liu et al. 1998) and CMAR (Li, Han & Pei 2001) are according to the results reported in literature, where a dash $-$ indicates that the result is not available in literature.

Keeping in mind that no classification method can outperform all others in all possible domains, we can draw some interesting points from Table 6.1 as follows:

- Our BCEP performed perfectly (98% to 100% accuracy) on some datasets, such as chess, Hypo, mushroom, shuttle, shuttle-small, tic-tac.

- In comparison with NB, BCEP consistently achieves higher accuracies. For the 34 datasets, BCEP wins on 30; they tie on the UCI Heart and Pima datasets; while NB wins only on the UCI Hepatitis and Splice dataset (please note BCEP's accuracy is very close to that of NB). This confirms our belief that BCEP really relaxes the strong attribute independence assumption implied by NB.

Table 6.1: Accuracy comparison

Dataset	NB	C5.0	CAEP	LB	CBA	CMAR	BCEP
Adult	83.15	85.9	83.12	85.11	83.3	-	85
Australian	84.49	84.56	85	85.65	85.4	86.1	86.4
Breast	96.09	93.77	96.96	96.86	95.28	96.4	97.25
Chess	87.89	98.9	85.45	90.24	98	-	98.9
Cleve	82.77	76.55	81.38	82.19	82.9	82.2	82.41
Crx	77.94	84.9	85	87.1	85.4	84.9	86.76
Diabete	75.66	74.47	75.26	76.69	74.5	75.8	76.8
Flare	80.47	81.16	81.32	81.52	83.11	-	80.57
German	74.1	71.8	74.5	74.8	73.5	74.9	74.7
Glass	65.79	68.7	65.79	69.2	73.9	70.1	73.68
Heart	81.85	77.41	82.96	82.22	81.9	82.2	81.85
Hepatitis	83.92	78.67	81.33	84.5	81.1	80.5	83.33
Horse	78.61	82.6	83.06	79.3	82.4	82.6	83.06
Hypo	97.91	99.27	97.28	98.4	99	98.4	98.92
Iono	89.45	90	90.59	91.2	92.3	91.5	93.24
Labor	86.33	76	88	87.7	86.3	89.7	90
Letter	74.94	86.3	78.39	76.4	70	-	84.28
Lymph	78.75	73.5	74.38	84.57	77.9	83.1	83.13
Mushroom	95.78	100	95.91	-	-	-	100
Pima	75.66	75.79	77.6	75.77	72.9	75.1	75.66
Satimage	81.8	85.2	85.83	83.9	85.4	-	87.42
Segment	91.82	93.4	90.04	94.16	94	-	95.15
Shuttle	99.32	99.97	98.54	-	-	-	99.89
Shuttle-small	98.7	99.52	99.08	99.38	99.48	-	99.65
Sick	84.34	97.78	91.61	97	97.2	97.5	97.34
Sonar	75.4	70.2	76	76	77.5	79.4	78.4
Splice	94.64	93.3	91.48	94.64	70.03	-	94.1
Tic-tac	70.14	85.91	85.91	67.9	99.6	99.2	99.37
Vehicle	61.12	69.82	63.9	68.8	69	68.8	68.05
Vote	87.86	89.52	88.81	94.72	93.54	-	89.76
Waveform-21	80.96	78.94	83.31	79.43	75.7	83.2	82.73
Wine	96.88	92.7	95.63	98.3	95	95	97.5
Yeast	57.4	55.73	53.22	58.16	55.1	-	58.22
Zoo	92.73	92.73	93.64	94.2	96.8	97.1	94.55
Average	83.08	84.26	84.13	84.25	83.98	85.17	87

- Compared with C5.0, BCEP achieves higher accuracies. For the 34 datasets, BCEP wins on 24; C5.0 wins 8; they achieve the same accuracy on the UCI Chess and Mushroom datasets.

- In comparison with CAEP, BCEP also achieves higher accuracies. For the 34 datasets, BCEP wins on 29; CAEP wins 4; they achieve the same accuracy on the UCI Horse dataset. This shows our Bayesian approach to approximate the class probabilities is better than the scoring functions of CAEP.

- We compare BCEP with LB *indirectly*, a NB extension using large itemsets. BCEP produces better results. For the 32 datasets where results of LB are available, BCEP wins on 20; LB wins on 12. Note that out of the 12 datasets LB wins on, the accuracy of BCEP are very close (within 1%) to LB on 9 datasets. We believe it is because EPs contain more useful information than large itemsets when used in the product approximations.

- We compare BCEP with CBA and CMAR *indirectly*. BCEP produces better results than both of them.

  For the 32 datasets where results of CBA are available, BCEP wins on 23; CBA wins on 9. Note that out of the 9 datasets CBA wins on, the accuracy of BCEP are very close (within 1%) to CBA on 6 datasets.

  For the 22 datasets where results of CMAR are available, BCEP wins on 15; CMAR wins on 7. Note that out of the 7 datasets CMAR wins on, the accuracy of BCEP are very close (within 1%) to CMAR on 6 datasets.

6.7.2 Study on Pruning EPs Based on Data Class Coverage

Table 6.2 shows the effect of applying data class coverage to prune EPs. It compares the accuracy, the number of EPs used in classification, and the classification time (average one fold time over the 10 CV-folds) when pruning with those when not pruning (w/o pruning). It can be seen that the number of EPs has been dramatically reduced after the pruning process, which leads to much shorter classification time, while in almost all

Table 6.2: Effect of pruning EPs based on data class coverage

	BCEP Accuracy		#EPs		classification time(sec's)	
Dataset	w/o prune	prune	w/o prune	prune	w/o prune	prune
Adult	83.94	85	3316	1115	774.123	340.613
Australian	85.59	86.4	1009	126	0.343	0.052
Chess	96.98	98.56	1078	56	9.341	1.378
Cleve	80.69	82.41	461	72	0.055	0.011
Diabete	75.4	76.8	62	19	0.011	0.010
German	74.8	74.5	807	77	0.316	0.024
Heart	81.48	81.85	103	36	0.012	0.006
Iono	90	93.24	2732	48	0.775	0.026
Letter	82.47	84.28	71714	3628	1145.97	118.083
Lymph	81.25	83.13	102	26	0.005	0.002
Mushroom	100	100	1958	36	240.846	13.634
Pima	74.2	75.66	54	21	0.008	0.006
Satimage	86.23	87.42	77684	1247	1560.57	57.819
Segment	93.68	94.76	11885	211	11.399	0.803
Sonar	76.5	78.4	699	41	0.227	0.016
Shuttle-small	99.65	99.65	378	45	11.094	3.253
Splice	91.14	94.1	10289	360	107.218	5.185
Tic-tac	86.63	99.37	704	27	0.177	0.008
Vehicle	66.22	68.05	1377	81	0.609	0.034
Waveform-21	82.71	82.25	5500	1069	29.35	6.575
Yeast	58.08	58.22	55	16	0.016	0.015

cases (except the German and waveform-21 datasets) there is an increase in the accuracy. For example, before pruning the splice dataset contains 10289 EPs and it takes BCEP 107 seconds to finish one classification fold with a accuracy of 91.14%. However, the pruning reduces the number of EPs down to 360 (around 3.5% of 10289 before reduction) and BCEP only needs 5 seconds (about 20 times faster) finishing one fold with a higher accuracy of 94.1%. We highlight some interesting points from Table 6.2.

- Many EPs are unnecessary or redundant for classification, because by pruning 50%-95% EPs, there is no loss in predictive accuracy, and often there is an increase in accuracy. Furthermore, some removed EPs are noisy, since using those EPs in classification leads to a decrease in accuracy, although the decrease is slight.

- Classification based on as few EPs as possible is desirable, as long as the small set of EPs provide sufficient information for prediction, because it is ten or even a hundred times faster to use fewer EPs to classify a test instance.

6.7.3 Parameter Tuning Study

In our Bayesian classification approach, we use EPs with large growth rates, i.e., more than or equal to 100. To show the effect of growth rate threshold on accuracy, we perform experiments using the UCI Chess, German, Mushroom and Waveform datasets. From Figure 6.2, we can see that the accuracy always increases as the growth rate threshold goes up and that the accuracy becomes stable after the threshold reaches 100. Note that a threshold of infinite may lead to decreased accuracy for some datasets, for example, a decrease of 0.3% for Chess and 0.62% for Waveform. The reason behind this is that the strict requirement of infinite growth rate (i.e., the support in one class must be zero) leads to lose some important EPs with large growth rates. Our investigation finds that if we use only EPs with infinite growth rates, the coverage becomes a little lower for the UCI Chess and Waveform dataset. This justifies our choice of 100 as the growth rate threshold.

We set the minimum support threshold as 1%. This conforms to the minimum support threshold used in CAEP, CBA and CMAR.

Another threshold is the data class coverage threshold θ. A value of 1 means that

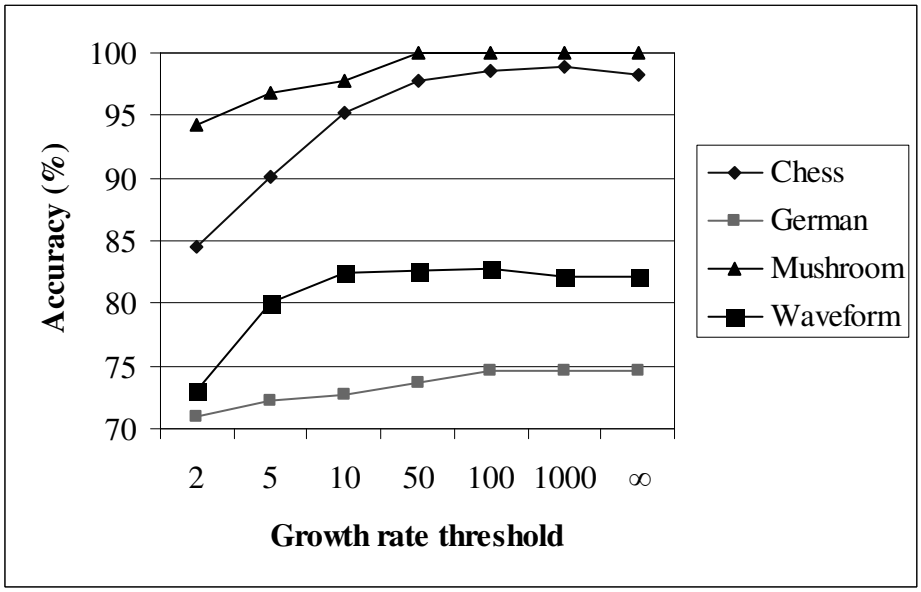


Figure 6.2: The effect of growth rate threshold on accuracy

an instance is removed immediately after it is covered by one EP. In this way, less EPs will be selected. Large values mean that we can remove an instance only after the instance has been covered by several EPs. In this way, more EPs can be chosen. When $\theta = \infty$, it means no pruning EPs by coverage at all.

We conduct experiments on the UCI Australian, Chess and German datasets to see the effect of coverage threshold on accuracy and number of selected EPs. The results are shown in Figure 6.3 and Figure 6.4. From Figure 6.3, we find that classification accuracy remains almost the same when θ goes from 1 to 2 (a little more EPs are selected when $\theta = 2$ than $\theta = 1$); but the accuracy runs down slowly when θ goes from 2 to ∞. We also give the number of EPs selected using different θ in Figure 6.4. Because for some datasets, there are few EPs discovered, $\theta = 1$ may prune EPs too aggressively, leaving fewer EPs left. As a trade-off, we let θ be 2, in order to prune many redundant EPs while keeping a little more EPs than necessary to deal with the real world cases.

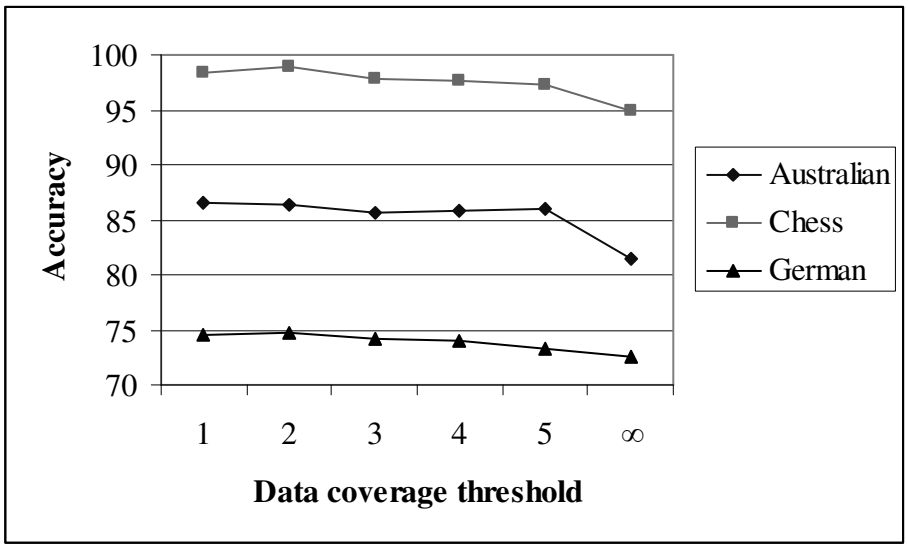


Figure 6.3: The effect of coverage threshold on accuracy

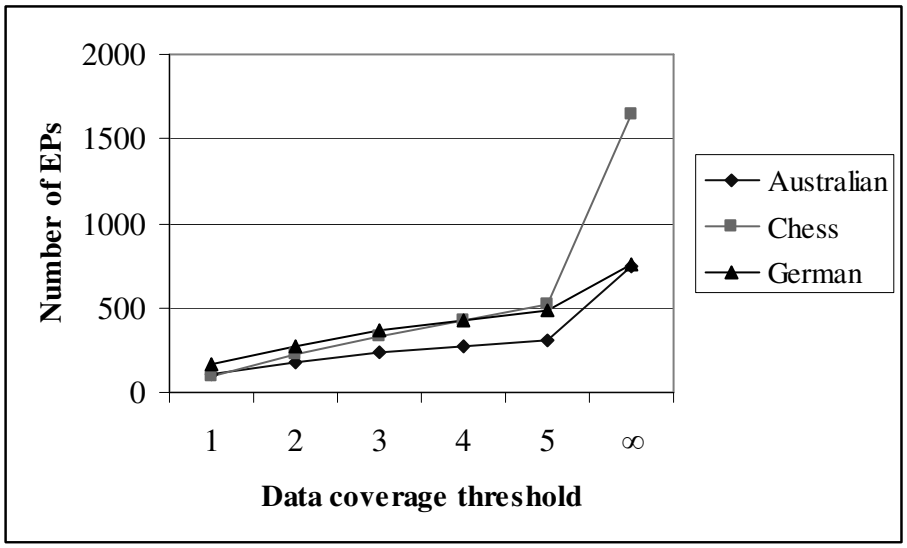


Figure 6.4: The effect of coverage threshold on the number of EPs selected

6.8 Chapter Summary

In this chapter, we have described a novel classification method, BCEP, i.e., Bayesian Classification by Emerging Patterns. BCEP is based on Bayesian theory to predict an unseen case given a training sample, which greatly improves the scoring function of the EP-based classifiers. It relaxes the strong attribute independence assumption implied by NB by using essential Emerging Patterns to compute probabilities. The method also utilizes the data class coverage to prune a lot of unnecessary EPs. BCEP uses only a small set of high quality EPs for classification, which is much less complex and much more understandable than previous EP-based classifiers. Our extensive experiments on a large number of benchmark datasets from the UCI Machine Learning Repository show that BCEP has good overall predictive accuracy, and in many cases it is also superior to other state-of-the-art classification methods such as Naive Bayes (NB), the decision tree classifier C5.0, Classification by Aggregating Emerging Patterns (CAEP), Large Bayes (LB), Classification Based on Association (CBA), and Classification based on Multiple Association Rules (CMAR).

Chapter 7

A Study of Noise Tolerance of the BCEP Classifier

Real-world classification problems always contain noise. A desirable property of a classifier is noise tolerance, that is, a reliable classifier should be tolerant to a reasonable level of noise. In this chapter, we investigate the noise tolerance of the BCEP classifier, Bayesian Classification by Emerging Patterns, discussed in Chapter 6. We systematically compare the noise resistance of our BCEP classifier with other major classifiers, such as Naive Bayes (NB), the decision tree classifier C4.5, Support Vector Machines (SVM) classifier, and the JEP-C classifier, using benchmark datasets from the UCI Machine Learning Repository that have been affected by three different kinds of noise. The empirical study shows that BCEP is superior to other well-known classification methods in terms of overall predictive accuracy. Out of the 116 cases, BCEP wins on 70 cases, NB wins on 30, C4.5 wins on 33, SVM wins on 32 and JEP-C wins on 21. We also study how classification accuracy decreases with the increase of noise. Our results indicate that BCEP decreases its accuracy slower than its competitors. We believe that BCEP can handle noise very well due to the inherent noise tolerance of the Bayesian approach and high quality patterns (Chi Emerging Patterns discussed in Chapter 5) used in the probability estimation.

7.1 Introduction

Data mining is typically concerned with observational retrospective data, i.e., data that has already been collected for some other purpose. For many reasons such as encoding errors, measurement errors, unrecorded causes of recorded features, the information in a database is almost always noisy. The problem of especially acute in business or government databases. For example, it is reported that there are error rates as high as 20% in some fields of the U.S. census data (Fayyad et al. 1996). The problem of dealing with noisy data is one of the most important research and application challenges in the field of Knowledge Discovery in Databases (KDD).

As an important data mining problem, the task of classification is to build a concise model from the training dataset such that it can be used to predict class labels of unknown instances. Given the prevalence of noise in data, real-world classification tasks will inevitably entail inaccuracies in the description of instances. The noise in the training data can mislead a learning algorithm (a classifier) to fit it into the classification model. As a result, the classifier finds many meaningless "regularities" in the data. The phenomenon is often referred to as overfitting.

7.1.1 Motivation

Recall that Emerging Patterns (EPs) are defined as multivariate features (i.e., itemsets) whose supports (or frequencies) change *significantly* from one class to another; and that Jumping Emerging Patterns (JEPs) are Emerging Patterns with infinite growth rates. Previous studies have shown that by aggregating the differentiating power of EPs/JEPs, the constructed classification systems (Dong, Zhang, Wong & Li 1999, Li, Dong & Ramamohanarao 2001) are usually more accurate than other existing state-of-the-art classifiers on generally "clean" or "noise-free" datasets. Motivated by the success of EP-based classifiers, we further investigate how they cope with a certain mount of noise in the training data.

Recently, the noise tolerance of EP-based classifiers such as CAEP classifier (Dong, Zhang, Wong & Li 1999) and JEPC classifier (Li, Dong & Ramamohanarao 2001) has been studied using a number of datasets from the UCI Machine Learning Repository where noise

is purposely introduced to the original datasets (Sun, Zhang & Ramamohanarao 2003). The results shows that both CAEP and JEPC do not experience overfitting due to the aggregating approach used in the classification and they are generally noise tolerant. Their comparison of the learning curves of a number of classifiers shows that JEPC and NB are the most noise tolerant, followed by C5.0, CAEP and kNN. In the training phase of JEPC, usually the latest tree-based mining algorithm (Bailey et al. 2002) is used to discover JEPs, because it is generally much faster than the previous border-based approach. However, there are difficulties to apply JEPC on noisy datasets.

- On one hand, by definition, an itemset which occurs once (or very few times) in one data class while zero times in another class is a JEP. Such JEPs are usually regarded as noise information and are not useful for classification. The number of those useless JEPs can be very large due to the injection of noise, which not only cause lots of difficulties to the mining of JEPs, but also makes JEPC very inefficient or even unusable. Although the number of the most expressive JEPs[1] is usually small, we have observed that such number becomes exponential for some noisy datasets. For example, for the UCI Australian dataset, there are 5,630 minimal JEPs in the original dataset; after injecting 40% mix noise, there are 64,415 minimal JEPs, about 10 times more. We take the UCI mushroom dataset as another example. There are 3,083 minimal JEPs in the original dataset; after injecting 40% attribute noise, there are 292,741 minimal JEPs, about 100 times more. Moreover, although JEPC is fast (only a few seconds) on the original Mushroom dataset, it becomes very difficult to discover JEPs from the noisy Mushroom dataset, i.e., JEPC needs about an hour to finish the training phase for the Mushroom dataset with 40% attribute noise.

- On the other hand, by definition, an itemset with large but finite support growth rate is not a JEP. The number of JEPs for some noisy datasets can be very small or even zero, because of the strict requirement that JEPs must have zero support in one class. Large-growth-rate EPs are also good discriminators to distinguish two classes. The exclusive of using JEPs makes JEPC very unreliable when it depends on only a few

[1]The most expressive JEPs are also known as the minimal JEPs. They are Emerging Patters that have infinite growth rates and reside in the left border of the border description.

JEPs to make decisions for unknown instances.

To summarize, difficulties occur in both the training and classification phase of the JEPC classifier. The mining of JEPs at the present of noise is very difficult. Too many JEPs will make the JEPC classifier very inefficient or even unusable; while too few JEPs will make it very unreliable.

To address the problems of JEP-C classifier, we propose to use Emerging Patterns that satisfy the following interestingness measures (see Chapter 5 Definition 5.1):

1. they must have minimum coverage (usually $\xi = 1\%$) in the home class. This is to ensure that they have a certain generalization capability, i.e., not too specific to the training data. JEPs have no such support constraint, so they may be very specific to some training examples. (e.g. every positive instance is a JEP given that an instances can not be both positive and negative at the same time)

2. they have moderate minimum growth rate (usually $\rho \in [5, 10]$). EPs with higher growth rates have sharper discriminating power, but those EPs are rare when noise is present. For example, the UCI mushroom datasets have many JEPs with minimum coverage of 1% when it is "clean". However, there are few EPs with large growth rates (e.g., 100) and minimum coverage of 1%. EPs with too low growth rates will lose the discriminating power. We choose 5-10 as the tradeoff to generate a certain number of EPs with moderate growth rates.

3. the third condition explores the subset relationship of EPs. Shorter EPs usually have larger supports, but lower growth rates; Longer EPs usually have higher growth rates, but smaller support. Recall that the strength of EP is defined by both support and growth rate (see Chapter 3 Definition 3.3). Our aim is two-fold: one is to find EPs as short as possible, as long as the supersets will not have more strength; the other is to find EPs as strong as possible by augmenting their length, as long as the subsets have less strength.

4. they must pass chi-test, which ensures that all the items contained in a Chi EP should be "correlated", i.e., they appear together not because of chance, nor due to noise.

It also means that all the items contribute *significantly* to EP's discriminating power, i.e., support, growth rate and strength.

Experiments show that our IEP-miner (discussed in Chapter 5) can efficiently discover EPs satisfying the above four conditions, even when there are much noise in the datasets. Therefore, IEP-miner can solve the problem of training the JEP-C classifier.

Another question is "using the Chi EPs that discovered by IEP-miner, can our BCEP classifier do a better job than the JEP-C classifier at the present of noise?" We believe the answer is yes, because of the following reasons. Recall that BCEP stands for Bayesian Classification by Emerging Patterns and that it is a hybrid of the EP-based classifier and Naive Bayes (NB) classifier. BCEP can handle noisy datasets very well due to its hybrid nature.

- The NB classifier has been shown inherently noise tolerant due to its collection of class and conditional probabilities (Domingos & Pazzani 1996). BCEP retains the noise tolerance of the Bayesian approach by manipulating probabilities.

- The main weakness of NB is the assumption that all attributes are independent given the class. By using high quality patterns (Chi EPs) in the probability approximation, BCEP extends NB and relaxes the strong attribute independence assumption. BCEP also provides a better scoring function than JEPC, i.e., BCEP is more reliable because there are usually enough EPs involved in the classification decisions.

We carried out experiments on a large number of benchmark datasets from the UCI Machine Learning Repository (Blake & Merz 1998) to study the noise tolerance of our BCEP and other state-of-the-art classification methods such as NB, C4.5, SVM and JEP-C. In all cases, 29 clean datasets, 29 attribute noise datasets, 29 label noise datasets and 29 mix noise datasets, the average number of times BCEP performs best is 17, which is the highest compared with NB (seven times), C4.5 (eight times), SVM (eight times) and JEP-C (five times). We also conduct experiments to see classifier's behavior under increasing noise. We find that all classifiers decrease accuracy when more noise is added, and that importantly, our BCEP method decrease slower than others. It follows that BCEP is superior in dealing

with noisy datasets. We attribute the good noise-tolerance of BCEP to its hybrid nature of using robust Chi Emerging Patterns and using better estimation of probabilities.

7.2 Noise Generation in the Training Data

Noise is error in data. Several potential sources of random errors have been identified (Quinlan 1986), including faulty measurement, data-entry error, ill-defined thresholds (e.g., when is a person "tall"?), and subjective interpretation of a multitude of inputs (e.g., what criteria are used when describing a person as "athletic"?).

There are three kinds of noise in the training data.

- **Attribute noise**. An attribute noise occurs when an attribute-value data take on the wrong value.
- **Label noise**, also called **classification noise**. A label noise happens when a training instance is not assigned to the correct class label.
- **Mix noise**. In the training data, both classification noise and attribute noise are present.

For example, a continuous attribute may take values from a temperature sensor, which is accurate to within 10% of its operating range. The reading may be too high on one occasion and too low on another. A discrete attribute representing temperature may take values like "hot", "mild" and "cool". Some people may argue a temperature of 30 degree is "hot", while others think it is "mild". Both of the above two cases lead to attribute noise. In some domains, experts disagree on the label of an instance, which leads to subjective classification errors. In other domains, a frequent type of classification error is mistakes made during data-entry.

To test the noise tolerance ability of a classifier, we purposely introduce these three kinds of noise into the training data. Note that no noise is added to the test data. In this way, we can also control the noise level in the training data. We choose 40% as the noise level to conduct a reasonable study, because the influence of too low noise level may not be clear and too much noise may completely disable the classifiers.

To apply 40% attribute noise to the training data, we replace 40% of all attribute values with random values in the corresponding domain. To inject 40% classification noise, we assign 40% of all training instances to a random class. To generate 40% mix noise, both attribute noise and classification noise are injected *independently*, i.e., for a particular instance, the probability of having wrong attribute values is independent from the probability of having wrong class labels. See Algorithm 7.1, Algorithm 7.2 and Algorithm 7.3 for details.

Algorithm 7.1: Add Attribute Noise

input : a dataset D and the attribute noise level np (np $\in [0, 1]$)
output: the dataset with np attribute noise
/* A is the set of all attributes in D */
/* Function random() returns a random number in $[0, 1]$ */
/* Function max() returns the maximum value for a continuous attribute */
/* Function min() returns the minimum value for a continuous attribute */
1 **foreach** *instance inst* $\in D$ **do**
2 **foreach** *attribute atr* $\in A$ **do**
3 **if** *random()* $<$ *np* **then**
4 **if** *atr is a continuous attribute* **then**
5 v = random() $\times$ (max(atr) $-$ min(atr)) + min(atr) ;
6 replace the old value of atr with the new value v ;
 end
 /* In the case below, atr is a discrete attribute. Note that the set of all the possible values of a discrete attribute is finite. */
7 **else** replace the old value of atr with a random element from all the possible values of atr
 end
 end
end

Algorithm 7.2: Add Label Noise

input : a dataset D and the label noise level np (np $\in [0, 1]$)

output: the dataset with np label noise

/* `Function random() returns a random number in` $[0, 1]$ */

1 **foreach** *instance inst* $\in D$ **do**

2 | **if** *random()* $<$ *np* **then** replace the old label of inst with a random element from the set of all class labels;

end

Please note that the above noise generation procedure affects the training data differently with respect to the number of attribute values and the number of classes. Let us discuss 40% attribute noise first. For a continuous attribute, the chance of taking a different value is almost 0.4, because there are infinite real values and we almost always have a new value in the corresponding domain by $v = random() \times (\max(atr) - \min(atr)) + \min(atr)$. However, for a discrete attribute with a finite domain (suppose there are n possible values), the possibility of taking a different value is $0.4 \times (n-1)/n$. When there is only one discrete value for an attribute, whatever the noise level is, the attribute is always "noise-free". When $n = 2$, the "real" noise level for this attribute is reduced to half of the original one. When $n = 10$, which is a common case, the "real" noise level becomes 0.36. As $n \to \infty$, which is like continuous attributes, it becomes 0.4 approximately. We then discuss 40% classification noise. For a dataset with c class labels, the possibility for an instance taking a different label is $0.4 \times (c-1)/c$. When $c = 2$, which is a common two-class problem, the classification noise level is actually 0.2. When there are many class labels, such as the UCI letter dataset ($c = 26$), the "real" classification noise level is almost 0.4.

Because training datasets with different number of attribute values and classes are injected different levels of "real" noise by the above noise generation procedure, we perform experiments to investigate the behavior of classifiers on these datasets.

Algorithm 7.3: Add Mix Noise

input : a dataset D and the attribute noise level np (np $\in [0, 1]$)

output: the dataset with np attribute noise

/* A is the set of all attributes in D */

/* Function random() returns a random number in $[0, 1]$ */

/* Function max() returns the maximum value for a continuous attribute */

/* Function min() returns the minimum value for a continuous attribute */

1 **foreach** *instance inst* $\in D$ **do**

  /* Add attribute noise */

2 **foreach** *attribute atr* $\in A$ **do**

3 **if** *random()* $<$ *np* **then**

4 **if** *atr is a continuous attribute* **then**

5 $v = \texttt{random()} \times (\max(\text{atr}) - \min(\text{atr})) + \min(\text{atr})$;

6 replace the old value of atr with the new value v ;

  end

  /* In the case below, atr is a discrete attribute. Note that the set of all the possible values of a discrete attribute is finite. */

7 **else** replace the old value of atr with a random element from all the possible values of atr

  end

  end

  /* Add classification noise */

8 **if** *random()* $<$ *np* **then** replace the old label of inst with a random element from the set of all class labels;

end

7.3 Experimental Evaluation

In order to investigate BCEP's performance compared to that of other classifiers, we carry experiments on 29 datasets from the UCI Machine Learning Repository. We regard the original datasets downloaded from UCI as "clean" datasets, although they are not guaranteed to be free from noise. For each dataset, we inject three kinds of noise at the level of 40% to generate three noisy datasets, namely "attribute noise" dataset, "label noise" dataset and "mix noise" dataset. Note that when we evaluate the performance under noise, the test datasets do not contain injected noise; only the training datasets are affected by noise.

We compare BCEP against Naive Bayes, the widely known decision tree induction C4.5, Support Vector Machines (SVM), and JEP-C (Li, Dong & Ramamohanarao 2001). We include SVM for comparison because it is well-known resistant to overfitting. We use WEKA's Java implementation of NB, C4.5 and SVM (Witten & Frank 1999). All experiments were conducted on a Dell PowerEdge 2500 (Dual P3 1GHz CPU, 2G RAM) running Solaris 8/x86, shared by many users of the University of Melbourne. The accuracy was obtained by using the methodology of *stratified* ten-fold cross-validation (CV-10), on exactly the same CV-10 training and testing data. We use the Entropy method from the $\mathcal{MLC}$++ machine learning library (Kohavi et al. 1994) to discretize datasets containing continuous attributes. Note that the discretization is performed on the UCI datasets into which we have injected three kinds of 40% noise. This has the following impact on a continuous attribute: although a value changes at the probability of almost 0.4, it may still fall into the same interval as discretized using the "clean" dataset.

7.3.1 Classification Accuracy

Table 7.1, 7.2 and 7.3 summarize the accuracy results on attribute noise, label noise and mix noise datasets. A dash for SVM means that it is terminated due to memory problems; a dash for JEP-C means that JEP-C aborts because the number of JEPs is more than 10,000,000 or the mining time exceeds 24 hours. In the last two rows, the average accuracy and the number of times being top classifiers are given. We regard the following

classifiers as top classifiers: (1) the classifier with the highest accuracy; and (2) those classifiers whose accuracy difference from the highest is within 1%.

We draw some interesting points as follows:

- The average accuracy of NB on clean, attribute noise and label noise datasets are lower than other classifiers. But NB deals with mix noise surprisingly well, when other classifiers are confused by the two-fold noise.

- C4.5 is fast in all cases, clean, attribute noise, label noise or mix noise. There is a big drop of accuracy on mix noise datasets.

- SVM is much slower than C4.5, especially when datasets are large. It deal with noise fairly well: its average accuracy is very close to BCEP on label noise datasets. But unfortunately we were unable to produce results for some large datasets such as Adult, Shuttle and Splice using the SVM implementation from WEKA (Witten & Frank 1999).

- JEP-C performs well on clean datasets. When attribute noise, label noise and mix noise is introduced, it is harder and harder to mine JEPs, generating either too many JEPs or too few JEPs. Either case leads to decreased accuracy.

- Our BCEP deals with all three kinds of noise very well, as evidenced both by the highest accuracy and the number of times being top classifiers. Out of 29 datasets, BCEP performs best on 19 clean datasets where the second best is SVM on 11 datasets; BCEP performs best on 21 attribute noise datasets, where the second best is JEP-C on 11 datasets; BCEP performs best on 13 label noise datasets, where the second best is C4.5 on 11 datasets; BCEP performs best on 17 mix noise datasets, where the second best is NB on 12 datasets. It can be seen that different classifiers are good at handling different kinds of noise. We believe that the success of BCEP on all three kinds of noise is mainly due to its hybrid nature, combining Bayesian and EP-based classification.

- The accuracy of BCEP is almost always higher than its ancestor NB. The tree-based EP mining algorithm (IEP-miner) used in BCEP mines a relatively small number of

useful EPs very fast on datasets where finding JEPs is very hard, i.e., JEPC uses up 1GB memory and gives up.

7.3.2 The Study of Classifier's Tolerance to Increasing Noise

To understand more on how different classifiers deal with increasing noise, we perform experiments on the following UCI datasets: Letter (a large-size dataset with many classes), Segment (a mid-size dataset with several classes), Sonar (a small dataset with high dimensionality) and Waveform21 (a mid-size datasets with medium dimensionality). The results are shown in Figure 7.1, 7.2, 7.3 and 7.4.

We can see that as the mix noise level goes up, usually the accuracy of all the classifiers goes down; however, the accuracy of our BCEP drops slower than others.

7.4 Related Work

Researchers in the machine learning community have recognized that it is harder for a leaning algorithm to perform generalization task on noisy data than on noise-free data (Angluin & Laird 1987). A formal model of noise process, called classification noise process, is introduced in (Angluin & Laird 1987) to study how to compensate for randomly introduced errors, or noise, in classifying the example data. It is also shown theoretically in (Angluin & Laird 1987) that algorithms that produce an "approximately correct" identification with "high probability" for reliable data can be adapted to handle noisy data. A few years Later, efficient learning algorithms in the presence of classification noise were proposed in (Sakakibara 1993). It is shown in (Sakakibara 1993) that using their noise-tolerant Occam algorithms, one can construct a polynomial-time algorithm for learning a class of Boolean functions in the presence of classification noise. Specifically, they focus on the applications of the noise-tolerant Occam algorithms to learning decision trees. It is interesting to see how their Occam algorithms can be applied to our Emerging Patterns based classifiers.

Work in (Brunk & Pazzani 1991) describes two approaches to addressing noise in a relational concept learning algorithm: an encoding length metric and reduced error

pruning. Their experiments are based on the artificially generated data for a king-rook-king board classification problem. Work in (Ali & Pazzani 1993) proposes HYDRA, a noise-tolerant relational concept learning algorithm. On noisy data, DNF concept learner typically learn a few reliable disjuncts and many unreliable disjuncts (called small disjuncts) each of which covers a small number of training instances. To address the issue, HYDRA estimates the reliable of each clause, i.e., it gives clauses that cover few instances (usually unreliable clauses) smaller weights. Their experiments show that attaching weights increases HYDRA's tolerance to noise.

In (Sebban & Janodet 2003), a statistical approach is proposed for dealing with noisy data during the inference of automata, by the state merging algorithm `RPNI`.

There is also a significant interest in noise identification and data cleaning. In (Kubica & Moore 2003), the authors present an iterative and probabilistic approach to the problem of identifying, modelling, and cleaning data corruption or noise and show that this approach leads to benefits in classification on both real world and artificial data. In (Yi & Liu 2003), a noise elimination technique is proposed to clean noisy regions/items within a Web page. Their experimental results based on popular data mining tasks such as Web page classification and clustering, show that the technique is able to boost the mining result dramatically.

7.5 Chapter Summary

In this chapter, we have performed a systematic study of the noise tolerance for a number of classifiers including Naive Bayes (NB), the decision tree classifier C4.5, Support Vector Machines (SVM) classifier, the JEP-C classifier and our BCEP classifier (introduced in Chapter 6), using benchmark datasets from the UCI Machine Learning Repository which have been deliberately added three kinds of noise into. Our extensive experiments on 29 benchmark datasets show that BCEP has good overall predictive accuracy on all three kinds of noisy datasets; it is superior to other state-of-the-art classification methods such as NB, C4.5, SVM and JEP-C: out of 116 cases (note that there are 29 datasets, each has four versions, namely, clean, attribute noise, label noise and mix noise datasets), BCEP wins on

70 datasets, which is much higher than any other classifiers. (as comparison, NB wins on 30, C4.5 wins on 33, SVM wins on 32 and JEP-C wins on 21)

Based on our experiments, we have the following conclusions:

- our Chi Emerging Patterns (Chi EPs), discussed in Chapter 5, resist noise better than other kinds of Emerging Patterns (e.g., JEPs);
- our classification method discussed in Chapter 6, Bayesian Classification by Emerging Patterns (BCEP), deals with noise well because it combines the advantages of the Bayesian approach (inherent noise tolerant) and the EP-based approach (high quality patterns with sharp discriminating power).

Table 7.1: Classification accuracy on 40% Attribute-Noise datasets

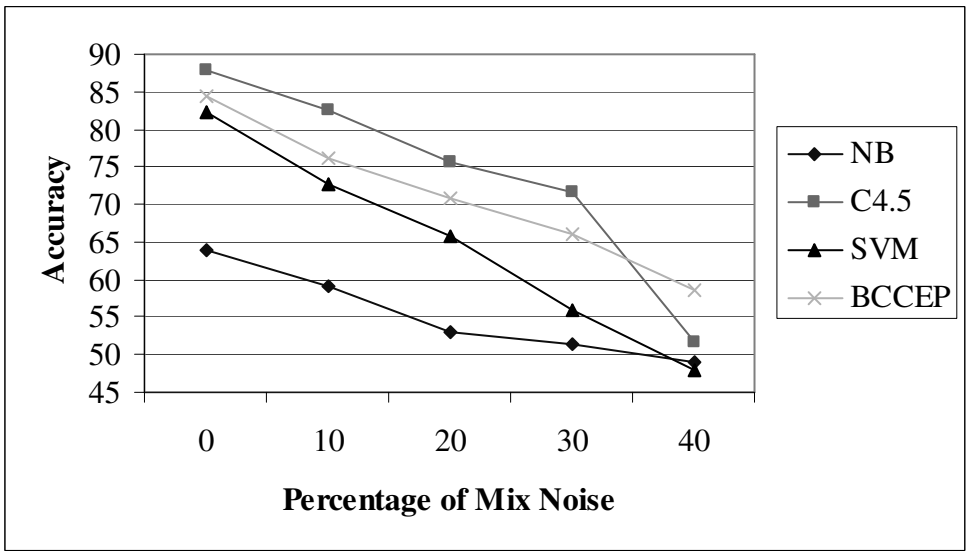
DB	NB	C4.5	SVM	JEP-C	BCCEP
Adult	80.73	82.72	-	83.88	84.52
Australian	86.62	86.18	85.59	83.24	84.85
Chess	79.5	83.77	85.41	86.86	89.34
Cleve	84.48	78.28	82.76	81.72	82.41
Diabete	69.47	71.18	65.79	70.53	72.76
German	74.1	68.2	70	73	73.6
Glass	48.95	57.37	51.58	54.74	53.68
Heart	83.33	72.59	81.11	83.7	82.22
Hypo	95.25	98.58	95.25	97.59	98.2
Ionosphere	80.29	86.18	81.76	71.76	89.71
Letter	52.64	66.51		-	68.21
Labor	88	64	88	82	86
Lymph	76.88	70.63	73.13	76.25	75.63
Mushroom	91.62	99.52	-	99.65	99.77
Pima	69.74	71.71	65.79	67.24	72.76
Satimage	78.59	82.09	82.05	79.17	85.47
Segment	83.38	88.48	79.87	89.09	90.35
Shuttle	78.64	99.72	-	99.82	99.66
Shuttle-Small	80.47	99.36	82.14	99	99.64
Sick	90.79	95.44	90.82	93.64	94.72
Sonar	64	65.5	62.5	-	73.5
Splice	89.15	79.75	88.39	-	87
TTT	72.53	71.47	68.74	75.68	81.26
Vehicle	40.61	64.88	47.44	65.12	61.59
Vote	87.62	85.71	90.71	-	90.48
Wave-form	77.05	72.97	82.13	-	81.75
Wine	94.38	90.63	94.38	93.75	94.38
Yeast	41.64	41.71	33.42	22.12	43.08
Zoo	90.91	79.09	89.09	90	90
Average	76.94	78.42	76.71	79.98	82.29
#Top Classifier	9	7	5	11	21

Table 7.2: Classification accuracy on 40% Label-Noise datasets

DB	NB	C4.5	SVM	JEP-C	BCCEP
Adult	83.36	85.67	-	80.61	84.81
Australian	78.97	83.97	85.59	81.76	85.15
Chess	84.62	97.26	90.6	89.21	94.09
Cleve	81.03	70.34	83.45	76.21	84.14
Diabete	75.13	73.95	76.45	69.21	75.79
German	73	69.1	71.6	72.69	71.8
Glass	42.63	51.05	49.47	35.79	42.63
Heart	83.33	73.33	79.26	75.93	83.33
Hypo	96.61	98.67	95.85	93.67	95.7
Ionosphere	81.47	80	84.12	85.88	85.59
Letter	58.7	79.11	75.25	-	76.1
Labor	86	70	86	82	84
Lymph	81.25	62.5	75	63.13	75
Mushroom	91.62	99.68	99.86	98.16	99.25
Pima	74.21	72.89	76.58	68.42	72.24
Satimage	78.53	59.59	82.28	-	84.58
Segment	64.29	79.96	90.04	85.11	91.26
Shuttle	88.4	99.86	92.09	86.12	97.08
Shuttle-Small	79.03	99.03	91.29	82.8	96.22
Sick	71.36	97.41	90.79	88.1	93.23
Sonar	69	62	72.5	-	73
Splice	90.5	85.9	88.68	-	81.36
TTT	70	78.53	88.32	84.84	83.16
Vehicle	42.32	57.44	57.44	56.22	56.22
Vote	87.38	89.05	87.14	-	87.86
Wave-form	77.05	65.52	84.36	-	82.25
Wine	95	74.38	96.25	85.6	97.5
Yeast	56.51	42.05	52.67	27.53	58.01
Zoo	85.45	80	83.64	54.55	94.55
Average	76.78	77.18	81.66	74.94	82.27
#Top Classifier	5	11	10	2	13

Table 7.3: Classification accuracy on 40% Mix-Noise datasets

DB	NB	C4.5	SVM	JEP-C	BCCEP
Adult	79.13	82.29	-	-	83.62
Australian	85.15	81.47	85.44	81.32	83.24
Chess	76.73	76.04	80.5	-	76.98
Cleve	79.31	68.97	76.9	66.9	79.31
Diabete	70.13	72.76	65.79	65.79	73.03
German	70	65.5	70	66.6	70.1
Glass	49.47	43.16	48.42	38.95	40
Heart	81.85	71.48	80.37	72.96	67.41
Hypo	95.25	96.8	95.25	92.91	96.04
Ionosphere	83.82	81.18	85.59	35.3	81.47
Letter	48.93	51.7	48.03	-	58.71
Labor	84	64	78	58	64
Lymph	75	54.38	60	57.5	68.75
Mushroom	91.27	98.59	95.84	-	98.16
Pima	70.53	73.82	65.79	68.68	74.87
Satimage	74.22	72.47	75.56	-	83.52
Segment	72.73	70.56	78.66	82.47	84.5
Shuttle	78.64	98.6	-	83.14	96.58
Shuttle-Small	84.47	90.53	78.62	83.62	98.62
Sick	90.82	94.43	90.82	-	94.78
Sonar	63	64	60	-	74.5
Splice	84.42	63	-	-	84.04
TTT	68.74	68.21	65.26	65.89	68.21
Vehicle	39.02	46.95	46.95	57.32	50.49
Vote	86.19	77.14	86.43	82.62	86.9
Wave-form	76.93	60.64	81.69	-	79.56
Wine	91.25	73.13	86.25	82.5	83.13
Yeast	35.48	30.89	31.51	22.12	40.48
Zoo	84.55	62.73	80.91	66.36	80.91
Average	74.86	70.88	73.02	66.55	76.62
#Top Classifier	12	5	6	1	17



The performance of JEP-C on Letter is not plotted because the number of JEPs for each class is already large. Consider here are 26 classes in Letter dataset, the number of all JEPs exceeds the limit of 10,000,000. Classification by such a huge number of JEPs is very inefficient.

Figure 7.1: The effect of increasing noise on accuracy - Letter

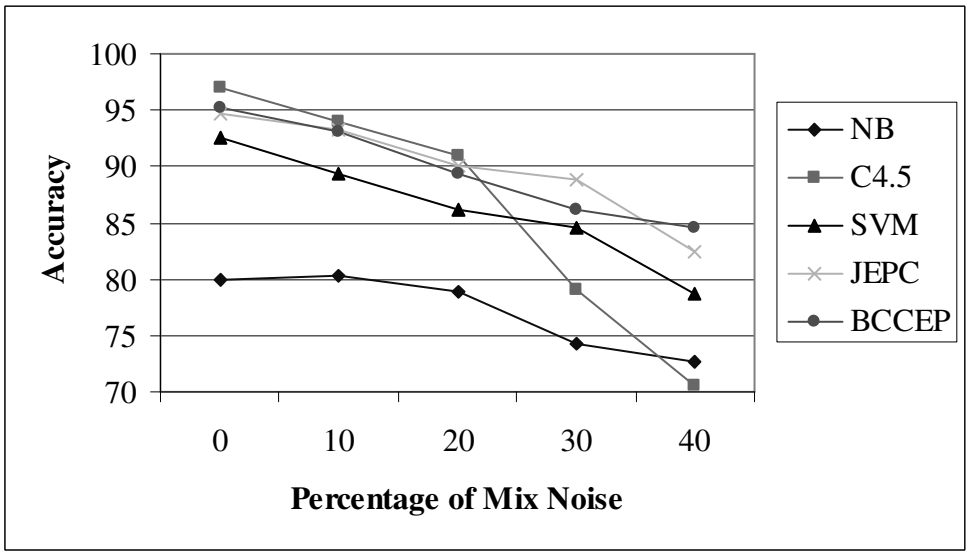
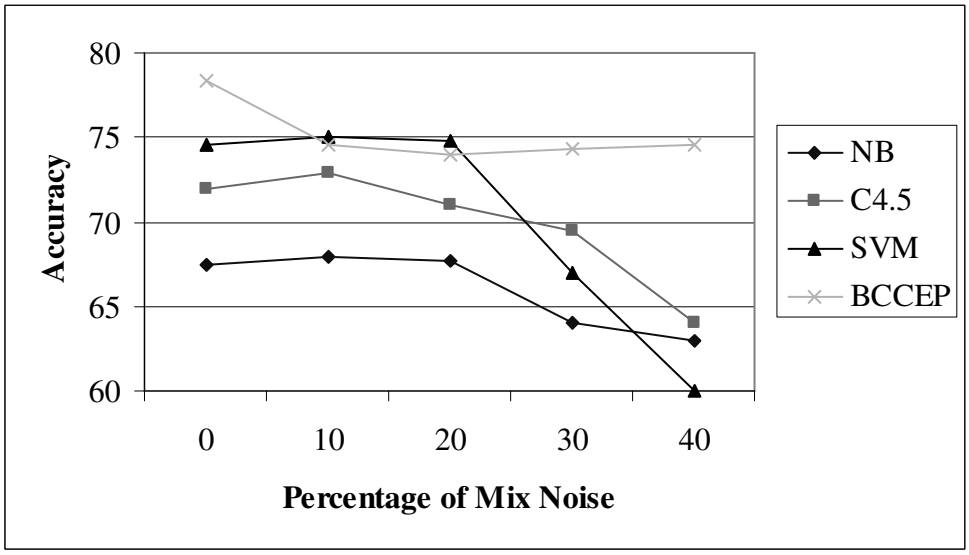


Figure 7.2: The effect of increasing noise on accuracy - Segment



The performance of JEP-C on Sonar is not plotted because Sonar's high dimensionality keeps JEP-C mining JEPs more than one day.

Figure 7.3: The effect of increasing noise on accuracy - Sonar

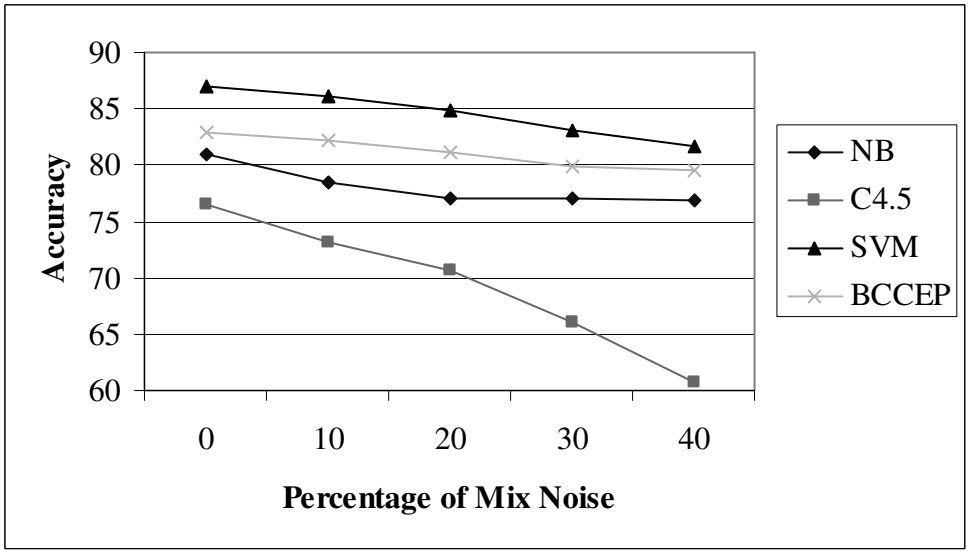


Figure 7.4: The effect of increasing noise on accuracy - Waveform

Chapter 8

Conclusions

In this thesis, we have studied three major research problems related to Emerging Patterns in the field of data mining and Knowledge Discovery in Databases (KDD). Specifically, we have investigated the following problems: (1) how to define various kinds of Emerging Patterns that provide insightful knowledge and are useful for classification; (2) how to efficiently mine those useful Emerging Patterns; (3) how to use those Emerging Patterns to build accurate classifiers. We have also investigated other problems such as interestingness measures for Emerging Patterns and how to use Emerging Patterns to build noise-tolerate classifiers.

In this last Chapter, we first summarize the research findings that my past few years' PhD work leads to. In Section 9.2, we discuss some future research problems involved in the discovery of Emerging Patterns and applications of Emerging Patterns.

8.1 Summary of Results

We state our main contributions in this thesis as the following:

- We have proposed Essential Jumping Emerging Patterns (EJEPs), and shown that they are high quality patterns with the most differentiating power and thus are sufficient for building accurate classifiers. We have also developed a new "single-scan" algorithm for two-class problem to efficiently mine EJEPs of both classes of data (both

directions) at the same time, unlike previous algorithms that cannot directly mine EJEPs. Experimental results show that the classifiers based exclusively on EJEPs, can be built faster than JEP-classifier, and use much fewer patterns to achieve almost the same accuracy as JEP-classifier.

- We have proposed four objective interestingness measures for Emerging Patterns, including the minimum support, the minimum growth rate, the subset relationship between EPs and the correlation based on common statistical measures such as chi-squared value. We have shown that these "interesting" Emerging Patterns (called Chi EPs) not only provide insightful knowledge of differences between two classes of data, but also are excellent candidates for building accurate classifiers. We have also developed an efficient algorithm, IEP-miner, for mining only Chi EPs. An admissible chi-squared pruning heuristic is used to reduce the search space greatly, while retaining nearly 100% high support patterns and 90% low support patterns. Experimental results show that IEP-miner maintains efficiency even at low support (implying much more patterns than at moderate or high support) on data that is large, dense and has high dimensionality.

- We have proposed a novel approach to use Emerging Patterns as a basic means for classification, i.e., Bayesian Classification by Emerging Patterns (BCEP). As a hybrid of the EP-based classifier and Naive Bayes (NB) classifier, BCEP provides several advantages. First, it is based on theoretically well-founded mathematical models to predict an unseen case given a training dataset. Second, it extends NB by using essential Emerging Patterns to relax the strong attribute independence assumption. Lastly, it is easy to interpret, as many unnecessary EPs are pruned based on data class coverage. An empirical study carried out on a large number of benchmark datasets from the UCI Machine Learning Repository shows that our method is superior to other state-of-the-art classification methods such as Naive Bayes (NB), the decision tree classifier C5.0, Classification by Aggregating Emerging Patterns (CAEP), Large Bayes (LB), Classification Based on Association (CBA), and Classification based on Multiple Association Rules (CMAR).

- A reliable classifier should be tolerant to a reasonable level of noise, which is almost always present in real-world classification problems. To address this issue, we have shown that our Chi Emerging Patterns are more resilient to noise than other kinds of Emerging Patterns. We have also shown that our BCEP classification method can handle noise very well due to the inherent noise tolerance of the Bayesian approach and high quality patterns used in the probability approximation. The empirical study confirms that our method is superior to other well-known classification methods such as NB, C4.5, SVM and JEP-C in terms of overall predictive accuracy, on noisy benchmark datasets from the UCI Machine Learning Repository.

8.2 Towards the Future

We list the following problems as future research issues.

A Vertical Bitmap Representation of Data for Mining Emerging Patterns

The database representation is an important factor in the efficiency of generating and counting itemsets. Most previous algorithms use a horizontal row layout, with the database organized as a set of rows and each row representing a transaction or an instance. The alternative *vertical* column layout associates each item with a set of transaction identifiers (tids) that contain the item. In a vertical bitmap representation of a database, there is one bit for each transaction in the database. If item **i** appears in transaction **j**, then the **j**-th bit of the bitmap for item **i** is set to one; otherwise, the bit is set to zero. The advantage of the vertical bitmap representation is that it allows simple and efficient support counting. MAFIA, an algorithm for generating frequent itemsets using the vertical representation of databases, is claimed to outperform previous work by a factor of three to five (Burdick et al. 2001).

It would be interesting to investigate whether we can use the vertical bitmap representation to improve the efficiency of mining Emerging Patterns. Obviously, new search strategy and pruning techniques need to be found, because of the different properties between frequent patterns and Emerging Patterns.

Parallel Mining of Emerging Patterns

Parallelism is expected by many researcher to be able to relieve current data mining methods from the sequential bottleneck, providing scalability to massive data sets and improving response time. A parallel algorithm called MLFPT (Multiple Local Frequent Pattern Tree) is proposed in (Zaiane, El-Hajj & Lu 2001) for parallel mining of frequent patterns, based on FP-growth algorithm. Their partition strategies at different stages of the mining process achieve near optimal balancing between processors, as shown by their experiments using high dimensionality data that are of a factor of hundreds of thousands of items, and transactional sizes that range in tens of gigabytes.

Our EP mining algorithms operate in P-tree, a similar structure to Frequent Patter tree. Parallelization of EP mining algorithm is an interesting problem. To achieve good performance on multiprocessor systems, we have to address the difficulties such as synchronization and communication minimization, workload balancing, finding good data layout and data decomposition, and disk I/O minimization. Other issues to be considered include the hardware platforms (distributed or shared memory systems[1]), the types of parallelism (data or task parallelism), and the load-balancing strategies (static or dynamic load balancing).

Round Robin Classification by Emerging Patterns

An important issue of classification is how to handle multi-class decision problems, which are common in real world. Our learning algorithm, Bayesian Classification by Emerging Patterns (BCEP) discussed in Chapter 6, is inherently binary, that is, it is only able to discriminate between two classes, because EPs are defined upon the background and target class. Therefore, BCEP needs to turn multi-class problems into a set of binary problems. For a c-class problem, BCEP transforms it into c two-class problems, which are constructed by regarding class i as the background class and class j ($j = 1 \cdots c, j \neq i$) as the target class. This class binarization technique is usually referred to as one-against-all

[1]Distributed memory (where each processor has a private memory) and shared memory (where all processors access common memory) are two dominant approaches for using multiple processors. Parallel programs are easy to implement on a shared-memory architecture. SMP stands for Shared-memory MultiProcessor.

class binarization. A set of EPs is discovered for each class, and one score is derived for each class, which is based the set of EPs for the corresponding class.

Recently, a *round robin* binarization technique has been shown as a general ensemble technique to improve classification accuracy (Furnkranz 2002). The round robin technique transforms a c-class problem into $c(c-1)/2$ two-class problems, one for each pair of classes, i.e., it learns one classifier for each pair of classes, using only training examples for these two classes and ignoring all others. It is also of interest to investigate the performance of applying the round robin binarization to BCEP in future research. A major issue may be how to combine or weight the decisions of the $c(c-1)/2$ classifiers.

Mining Emerging Patterns from Data Streams

Recently, a data stream model has been proposed for those data-intensive applications, such as network monitoring, telecommunications data management, sensor networks and others (Babcock, Babu, Datar, Motwani & Widom 2002). Data streams are potentially unbounded in size and arrive online at a very high speed, which makes it impossible to store the entire data in disk. The system has no control over the order in which data elements arrive to be processed. Moreover, because of limited computing power (e.g., CPU and memory), once an element from a data stream has been processed, it is discarded or archived. This means that the whole data streams can only be scanned in one pass and that only a small fraction of the input data can be stored in memory for random access at one time. These characteristics of data streams present new difficulties and challenges for previous data mining algorithms operating data stored in conventional relations.

It has been recognized that both *approximation* and *adaptivity* are key ingredients in data mining over rapid data streams. Our EP mining algorithms discussed in this thesis focus largely on the opposite goal of precise answers. Modifications on existing algorithms or new EP mining methods are needed to address the challenge of data streams. Some potentially important issues include the sliding window technique, batch processing, sampling, and synopsis structures.

Applying Emerging Patterns to Other areas

Emerging Patterns have been proved to be useful by their success in bioinformatics[2]. Emerging Patterns based classifiers provide new alternatives for pattern classification. There are many other areas in which Emerging Patterns have the potential to succeed, such as network intrusion detection, anti-spam Email, image data, semi-structured texts (e.g., XML documents), unstructured texts (e.g., World Wide Web documents).

World Wide Web (WWW) is particular interesting. WWW is a vast repository of useful knowledge. It is interesting to apply Emerging Patterns to web content mining, web usage mining, search engines and intelligent agents. It is projected that with advances in intelligent agents, widespread adoption of XML, and web services, web mining will reach the semantic level in the next ten years (Fayyad, Piatetsky-Shapiro & Uthurusamy 2003). It is also an interesting problem to investigate how to use Emerging Patterns to catch "semantic differences" between two classes of data.

[2]A list of relevant publications can be found at http://sdmc.i2r.a-star.edu.sg/jinyan/

Bibliography

Agarwa, R. C., Aggarwal, C. C. & Prasad, V. V. V. (2001). A tree projection algorithm for generation of frequent item sets, *Journal of Parallel and Distributed Computing* **61**(3).

Agrawal, R., Imielinski, T. & Swami, A. (1993a). Database mining: A performance perspective, *IEEE Transactions on Knowledge and Data Engineering* **5**(6): 914–925.

Agrawal, R., Imielinski, T. & Swami, A. (1993b). Mining association rules between sets of items in large databases, *Proc. 1993 ACM-SIGMOD Int'l Conf. Management of Data (SIGMOD'93)*, Washington, DC, USA, pp. 207–216.

Agrawal, R. & Srikant, R. (1994). Fast algorithms for mining association rules, *in* J. B. Bocca, M. Jarke & C. Zaniolo (eds), *Proc. 1994 Int'l Conf. Very Large Data Bases(VLDB'94)*, Santiago, Chile, pp. 487–499.

Agrawal, R. & Srikant, R. (1995). Mining sequential patterns, *Proc. of the 11th Int'l Conf. on Data Engineering*, Taipei, Taiwan.

Aha, D., Kibler, D. & Albert, M. (1991). Instance-based learning algorithms, *Machine Learning* **6**: 37–66.

Ali, K. & Pazzani, M. (1993). Hydra: A noise-tolerant relational concept learning algorithm, *Proc Int'l Joint Conf. on Artificial Intelligence*, Chambery, France.

Alon, U., Barkai, N., Notterman, D. A., Gish, K., Ybarra, S., Mack, D. & Levine, A. J. (1999). Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays, *Proc. National Academy of Science*, Vol. 96 of *Cell Biology*, pp. 6745–6750.

Alsabti, K., Ranka, S. & Singh, V. (1998). Clouds: A decision tree classifier for large datasets, *Proc. the Fourth Int'l Conf. on Knowledge Discovery and Data Mining (KDD-98)*, New York, USA, pp. 2–8.

Angluin, D. & Laird, P. (1987). Learning from noisy examples, *Machine Learning* **2**(4): 343 – 370.

Babcock, B., Babu, S., Datar, M., Motwani, R. & Widom, J. (2002). Models and issues in data stream systems, *Proc. 21st ACM Symposium on Principles of Database Systems (PODS 2002)*.

Bailey, J., Manoukian, T. & Ramamohanarao, K. (2002). Fast algorithms for mining emerging patterns, *Proc. 6th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'02)*, Helsinki, Finland.

Bailey, J., Manoukian, T. & Ramamohanarao, K. (2003a). Classification using constrained emerging patterns, *Proc. 4th Int'l. Conf. on Web-Age Information Management (WAIM2003)*, Chengdu, China, pp. 226–237.

Bailey, J., Manoukian, T. & Ramamohanarao, K. (2003b). A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns, *Proc. 3rd IEEE Int'l Conf. on Data Mining (ICDM2003)*, Melbourne, Florida, USA, pp. 485–488.

Bay, S. D. & Pazzani, M. J. (2001). Detecting group differences: Mining contrast sets, *Data Mining and Knowledge Discovery* **5**(3): 213–246.

Bayardo Jr., R. J. (1998). Efficiently mining long patterns from databases, *Proc. 1998 ACM-SIGMOD Int'l Conf. Management of Data (SIGMOD'98)*, ACM Press, Seattle, WA, USA, pp. 85–93.

Bayardo Jr., R. J. & Agrawal, R. (1999). Mining the most interesting rules, *Proc. 5th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pp. 145–154.

Bayardo, R., Agrawal, R. & Gunopulos, D. (2000). Constraint-based rule mining in large, dense databases, *Data Mining and Knowledge Discovery* **4**(2/3): 217–240.

Bethea, R. M., Duran, B. S. & Boullion, T. L. (1995). *Statistical methods for engineers and scientists*, New York : M. Dekker.

Bishop, C. M. & Bishop, C. (1995). *Neural Networks for Pattern Recognition*, Oxford University Press.

Blake, C. L. & Merz, C. J. (1998). UCI repository of machine learning databases. *http://www.ics.uci.edu/∼mlearn/MLRepository.html

Brachman, R., Khabaza, T., Kloesgen, W., Piatetsky-Shapiro, G. & Simoudis, E. (1996). Mining business databases, *Communications of the ACM* **39**(11): 42–48.

Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (1984). *Classification and Regression Trees*, Chapman & Hall, New York.

Breslow, L. A. & Aha, D. W. (1997). Simplifying decision trees: a survey, *Knowledge Engineering Review* **12**(1): 1–40.

Brin, S., Motwani, R. & Silverstein, C. (1997). Beyond market baskets: Generalizing association rules to correlations, *in* J. Peckham (ed.), *Proc. 1997 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD'97)*, ACM Press, Tucson, Arizona, USA, pp. 265–276.

Brin, S., Motwani, R., Ullman, J. D. & Tsur, S. (1997). Dynamic itemset counting and implication rules for market basket data, *in* J. Peckham (ed.), *Proc. 1997 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD'97)*, Tucson, Arizona, USA, pp. 255–264.

Brunk, C. & Pazzani, M. (1991). An investigation of noise tolerant relational learning algorithms, *Proc. Eighth Int'l Workshop on Machine Learning*, Morgan Kaufmann, pp. 389–391.

Burdick, D., Calimlim, M. & Gehrke, J. (2001). MAFIA: A maximal frequent itemset algorithm for transactional databases, *Proc. 17th Int'l Conf. on Data Engineering*, Heidelberg, Germany, pp. 443–452.

Chaudhuri, S. & Dayal, U. (1997). An overview of data warehousing and olap technology, *SIGMOD Record* **26**(1): 65–74.

Cheeseman, P. & Stutz, J. (1996). Bayesian classification (autoclass): Theory and results, *Proc. 2nd Int'l Conf. on Knowledge Discovery and Data Mining (KDD-96)*, AAAI/MIT Press, pp. 153–180.

Chen, M.-S., Han, J. & Yu, P. S. (1996). Data mining: An overview from a database perspective, *IEEE Trans. Knowledge and Data Engineering* **8**: 866–883.

Christensen, R. (1997). *Log-Linear Models and Logistic Regression*, Springer.

Dasarathy, B. (1991). *Nearest neighbor norms: NN pattern classification techniques*, IEEE Computer Society Press, Los Alamitos, CA.

Dietterich, T. G. (1998). Machine-learning research: Four current directions, *The AI Magazine* **18**(4): 97–136.

Domingos, P. & Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss, *Machine Learning* **29**(2-3): 103–130.

Domingos, P. & Pazzani, M. J. (1996). Beyond independence: Conditions for the optimality of the simple bayesian classifier, *Proc. 13th Int'l Conf. on Machine Learning (ICML'96)*, pp. 105–112.

Dong, G. & Li, J. (1998). Interestingness of discovered association rules in terms of neighborhood-based unexpectedness, *Proc. 2th Pacific Asia Conf. on Knowledge Discovery in Databases (PAKDD'98)*, Melbourne, Australia, pp. 72–86.

Dong, G. & Li, J. (1999). Efficient mining of emerging patterns: Discovering trends and differences, *Proc. 5th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD'99)*, San Diego, CA, USA, pp. 43–52.

Dong, G., Li, J. & Wong, L. (2004). The use of emerging patterns in the analysis of gene expression profiles for the diagnosis and understanding of diseases, *in* J. Zurada & M. Kantardzic (eds), *New Generation of Data Mining Applications*, IEEE Press/Wiley.

Dong, G., Li, J. & Zhang, X. (1999). Discovering jumping emerging patterns and experiments on real datasets, *Proc. 9th Int'l Database Conf. on Heterogeneous and Internet Databases (IDC'99)*, Hong Kong, China, pp. 155–168.

Dong, G., Zhang, X., Wong, L. & Li, J. (1999). Classification by aggregating emerging patterns, *Proc. 2nd Int'l Conf. on Discovery Science (DS'99)*, Tokyo, Japan, pp. 30–42.

Dougherty, J., Kohavi, R. & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features, *Proc. 12th Int'l Conf. on Machine Learning*, pp. 194–202.

Duda, R. O. & Hart, P. E. (1973). *Pattern classification and scene analysis*, John Wiley & Sons, New York.

Fan, H. & Ramamohanarao, K. (2002). An efficient single-scan algorithm for mining essential jumping emerging patterns for classification, *Proc. 6th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD2002)*, Taipei, Taiwan, China, pp. 456–462.

Fan, H. & Ramamohanarao, K. (2003a). A bayesian approach to use emerging patterns for classification, *Proc. 14th Australasian Database Conference (ADC2003)*, Adelaide, Australia, pp. 39–48.

Fan, H. & Ramamohanarao, K. (2003b). Efficiently mining interesting emerging patterns, *Proc. 4th Int'l. Conf. on Web-Age Information Management (WAIM2003)*, Chengdu, China, pp. 189–201.

Fan, H. & Ramamohanarao, K. (2004a). Noise tolerant classification by chi emerging patterns, *Proc. 8th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD2004)*, Sydney, Australia.

Fan, H. & Ramamohanarao, K. (2004b). Noise tolerant classification by chi emerging patterns, *Technical report*, Department of Computer Science and Software Engineering, University of Melbourne.

Fayyad, U. M. & Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning, *Proc. 13th International Joint Conference on Artificial Intelligence (IJCAI)*, San Francisco, CA, pp. 1022–1029.

Fayyad, U. M., Piatetsky-Shapiro, G. & Smyth, P. (1996). From data mining to knowledge discovery in databases, *AI Magazine* **17**: 37–54.

Fayyad, U. M., Piatetsky-Shapiro, G. & Uthurusamy, R. (2003). Summary from the kdd-03 panel – data mining: The next 10 years, *SIGKDD Explorations* **5**(2): 191–196.

Freitas, A. A. (1998). On objective measures of rule surprisingness, *Proc. 2nd European Symp. on Principles of Data Mining and Knowledge Discovery, PKDD'98*, Nantes, France, pp. 1–9.

Freitas, A. A. (1999). On rule interestingness measures, *Knowledge-Based Systems Journal* **12**(5-6): 309–315.

Freitas, A. A. (2002). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*, Spinger-Verlag, Berlin.

Friedman, J. H. (1997). Data mining and statistics: what's the connection? Available at: http://www-stat.stanford.edu/ jhf/.

Friedman, N., Geiger, D. & Goldszmidt, M. (1997). Bayesian network classifiers, *Machine Learning* **29**(2-3): 131–163.

Furnkranz, J. (2002). Round robin classification, *Journal of Machine Learning Research* **2**: 721–747.

Garofalakis, M., Hyun, D., Rastogi, R. & Shim, K. (2000). Efficient algorithms for constructing decision trees with constraints, *Proc. sixth ACM SIGKDD Int'l Conf. on Knowledge discovery and data mining*, ACM Press, pp. 335–339.

Gehrke, J., Ganti, V., Ramakrishnan, R. & Loh, W.-Y. (1999). Boat-optimistic decision tree construction, *Proc. 1999 ACM-SIGMOD Int'l Conf. Management of Data (SIGMOD'99)*, Philadephia, Pennsylvania, USA, pp. 169–180.

Gehrke, J., Ramakrishnan, R. & Ganti, V. (2000). Rainforest - a framework for fast decision tree construction of large datasets, *Data Mining and Knowledge Discovery* **4**(2): 127–162.

Glymour, C., Madigan, D., Pregibon, D. & Smyth, P. (1997). Statistical themes and lessons for data mining, *Data Mining and Knowledge Discovery* **1**(1): 11–28.

Goldman, S. A. & Scott, S. D. (1999). A theoretical and empirical study of a noise-tolerant algorithm to learn geometric patterns, *Machine Learning* **37**(1): 5 – 49.

Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J. R., Caligiuri, M. A., Bloomfield, C. D. & Lander, E. S. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring, *Science* **286**(5439): 531–537.

Grahne, G., Lakshmanan, L. V. S. & Wang, X. (2000). Efficient mining of constrained correlated sets, *Proc. 2000 Int'l Conf. Data Engineering (ICDE'00)*, San Diego, CA, pp. 512–521.

Gunopulos, D., Mannila, H., Khardon, R. & Toivonen, H. (1997). Data mining, hypergraph transversals, and machine learning, *Proc. 16th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (PODS'97)*, pp. 209–216.

Hall, M. A. (1999). *Correlation-based Feature Subset Selection for Machine Learning*, PhD thesis, Department of Computer Science, University of Waikato.

Han, J., Altman, R. B., Kumar, V., Mannila, H. & Pregibon, D. (2002). Emerging scientific applications in data mining, *Communications of the ACM* **45**(8): 54–58.

Han, J., Cai, Y. & Cercone, N. (1992). Knowledge discovery in databases: An attribute-oriented approach, *in* L.-Y. Yuan (ed.), *Proc. 18th Int'l Conf. on Very Large Databases*, Morgan Kaufmann Publishers, San Francisco, U.S.A., pp. 547–559.

Han, J., Cai, Y. & Cercone, N. (1993). Data-driven discovery of quantitative rules in relational databases, *IEEE Trans. Knowledge and Data Engineering* **5**(1): 29–40.

Han, J. & Kamber, M. (2000). *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers.

Han, J., Pei, J. & Yin, Y. (2000). Mining frequent patterns without candidate generation, *Proc. 2000 ACM-SIGMOD Int'l Conf. Management of Data (SIGMOD'00)*, Dallas, TX, USA, pp. 1–12.

Han, J., Wang, J., Lu, Y. & Tzvetkov, P. (2002). Mining top-k frequent closed patterns without minimum support, *Proc. 2002 Int'l Conf. on Data Mining (ICDM'02)*, Maebashi, Japan.

Hand, D. J., Mannila, H. & Smyth, P. (2001). *Principles of Data Mining*, The MIT Press.

Hilderman, R. & Hamilton, H. (2003). Measuring the interestingness of discovered knowledge: A principled approach, *Intelligent Data Analysis* **7**(4): 347–382.

Imielinski, T. & Mannila, H. (1996). A database perspective on knowledge discovery, *Communications of the ACM* **39**(11): 58–64.

Jain, A., Murty, M. & Flynn, P. (1999). Data clustering: A review, *ACM Computing Surveys* **31**(3): 264–323.

John, G. H. (1997). *Enhancements to the Data Mining Process*, PhD thesis, Department of Computer Science, Stanford University.

Klemettinen, M., Mannila, H., Ronkainen, P., Toivonen, H. & Verkamo, A. (1994). Finding interesting rules from large sets of discovered association rules, *in* N. Adam, B. Bhargava & Y. Yesha (eds), *Proc. the Third Int'l Conf. on Information and Knowledge Management (CIKM'94)*, ACM Press, Gaithersburg, Maryland, pp. 401–407.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection, *Proc. 14th Int'l Joint Conf. on Artificial Intelligence IJCAI'95*, Morgan Kaufmann, San Mateo, CA, pp. 1137–1145.

Kohavi, R. (1996). Scaling up the accuracy of Naive-Bayes classifiers: A decision-tree hybrid, *Proc. 2nd Int'l Conf. on Knowledge Discovery and Data Mining (KDD-96)*, Portland, Oregon, USA, pp. 202–207.

Kohavi, R., Becker, B. & Sommerfield, D. (1997). Improving simple bayes, *Proc. European Conf. on Machine Learning (ECML'97)*.

Kohavi, R., John, G., Long, R., Manley, D. & Pfleger, K. (1994). MLC++: a machine learning library in C++, *Tools with artificial intelligence* pp. 740–743.

Kohavi, R., Sommerfield, D. & Dougherty, J. (1997). Data mining using MLC++: A machine learning library in C++, *International Journal on Artificial Intelligence Tools* **6**(4): 537–566.

Kononenko, I. (1991). Semi-naive bayesian classifier, *Proc of 6th European Working Session on Learning*, Springer-Verlag, pp. 206–219.

Kubica, J. & Moore, A. (2003). Probabilistic noise identification and data cleaning, *in* X. Wu, A. Tuzhilin & J. Shavlik (eds), *Proc. 3rd IEEE Int'l Conf. on Data Mining (ICDM2003)*, IEEE Computer Society, pp. 131–138.

Langley, P. (2000). Crafting papers on machine learning, *Proc. 17th Int'l Conf. on Machine Learning*, Morgan Kaufmann, San Francisco, CA, pp. 1207–1212.

Langley, P., Iba, W. & Thompson, K. (1992). An analysis of bayesian classifiers, *Proc. Tenth National Conference on Artificial Intelligence*, AAAI Press, San Jose, CA, pp. 223–228.

Last, M., Klein, Y. & Kandel, A. (2001). Knowledge discovery in time series databases, *IEEE Transactions on Systems, Man, and Cybernetics* **31B**(1): 160–169.

Last, M., Maimon, O. & Minkov, E. (2002). Improving stability of decision trees, *International Journal of Pattern Recognition and Artificial Intelligence* **16**(2): 145–159.

Lewis, P. (1959). Approximating probability distributions to reduce storage requirements, *Information and Control* **2**(3): 214–225.

Li, J. (2000). *Mining emerging patterns to construct accurate and efficient classifiers*, PhD thesis, Melbourne University.

Li, J., Dong, G. & Ramamohanarao, K. (2001). Making use of the most expressive jumping emerging patterns for classification, *Knowledge and Information Systems* **3**(2): 131–145.

Li, J., Dong, G., Ramamohanarao, K. & Wong, L. (2004). DeEPs: A new instance-based discovery and classification system, *Machine Learning* **54**(2): 99–124.

Li, J., Liu, H., Downing, J. R. & A. Yeoh, L. W. (2003). Simple rules underlying gene expression profiles of more than six subtypes of acute lymphoblastic leukemia (ALL) patients, *Bioinformatics* **19**: 71–78.

Li, J., Liu, H., Ng, S.-K. & Wong, L. (2003). Discovery of significant rules for classifying cancer diagnosis data, *Bioinformatics* **19**(Suppl. 2): ii93–ii102.

Li, J., Manoukian, T., Dong, G. & Ramamohanarao, K. (2004). Incremental maintenance on the border of the space of emerging patterns, *Data Mining and Knowledge Discovery* p. to appear.

Li, J., Ramamohanarao, K. & Dong, G. (2000). The space of jumping emerging patterns and its incremental maintenance algorithms, *Proc. 17th Int'l Conf. on Machine Learning*, Morgan Kaufmann, San Francisco, CA, pp. 551–558.

Li, J., Ramamohanarao, K. & Dong, G. (2001). Combining the strength of pattern frequency and distance for classification, *Proc. 5th Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD'01)*, Hong Kong, China, pp. 455–466.

Li, J. & Wong, L. (2002a). Geography of differences between two classes of data, *Proc. 6th European Conf. on Principles of Data Mining and Knowledge Discovery (PKDD'02)*, Helsinki, Finland, pp. 325–337.

Li, J. & Wong, L. (2002b). Identifying good diagnostic genes or genes groups from gene expression data by using the concept of emerging patterns, *Bioinformatics* **18**(5): 725–734.

Li, J. & Wong, L. (2002c). Solving the fragmentation problem of decision trees by discovering boundary emerging patterns, *Proc. IEEE Int'l Conf. on Data Mining (ICDM2002)*, Maebashi City, Japan, pp. 653–656.

Li, J., Zhang, X., Dong, G., Ramamohanarao, K. & Sun, Q. (1999). Efficient mining of high confidience association rules without support thresholds, *Proc. 3th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'99)*, Prague, Czech Republic, pp. 406–411.

Li, W., Han, J. & Pei, J. (2001). CMAR: Accurate and efficient classification based on multiple class-association rules, *Proc. IEEE Int'l Conf. on Data Mining, 2001 (ICDM'01)*, pp. 369–376.

Lim, T.-S., Loh, W.-Y. & Shih, Y.-S. (2000). A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms, *Machine Learning* **40**: 203–228.

Liu, B. & Hsu, W. (1996). Post-analysis of learned rules, *Proc. Thirteenth National Conf. on Artificial Intelligence (AAAI-96)*, Portland, Oregon, USA, pp. 828–834.

Liu, B., Hsu, W. & Ma, Y. (1998). Integrating classification and association rule mining, *Proc. the Fourth Int'l Conf. on Knowledge Discovery and Data Mining (KDD-98)*, New York, USA, pp. 80–86.

Liu, B., Hsu, W. & Ma, Y. (1999). Pruning and summarizing the discovered associations, *Proc. 5th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD-99)*, ACM Press, San Diego, CA, USA, pp. 125–134.

Liu, H., Hussain, F., Tan, C. L. & Dash, M. (2002). Discretization: An enabling technique, *Data Mining and Knowledge Discovery* **6**(4): 393–423.

Liu, J., Pan, Y., Wang, K. & Han, J. (2002). Mining frequent item sets by opportunistic projection, *Proc. 2002 Int'l Conf. on Knowledge Discovery in Databases (KDD'02)*, Edmonton, Canada.

Mannila, H., Toivonen, H. & Verkamo, A. I. (1997). Discovery of frequent episodes in event sequences, *Data Mining and Knowledge Discovery* **1**(3): 259–289.

Mehta, M., Agrawal, R. & Rissanen, J. (1996). SLIQ: A fast scalable classifier for data mining, *Proc. 5th Int'l Conf. on Extending Database Technology*, pp. 18–32.

Mehta, M., Rissanen, J. & Agrawal, R. (1995). MDL-based decision tree pruning, *Proc. the first Int'l Conf. on Knowledge Discovery in Databases and Data Mining (KDD'95)*, Montreal, Canada, pp. 216–221.

Meretakis, D. & Wuthrich, B. (1999). Extending naive bayes classifiers using long itemsets, *Proc. the 5th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining, (KDD-99)*, pp. 165–174.

Mitchell, T. (1997). *Machine Learning*, McGraw Hill.

Morishita, S. & Sese, J. (2000). Traversing itemset lattice with statistical metric pruning, *Proc. 19th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pp. 226–236.

Ng, R. T., Lakshmanan, L. V. S., Han, J. & Pang, A. (1998). Exploratory mining and pruning optimizations of constrained associations rules, *Proc. 1998 ACM SIGMOD Int'l Conf Management of Data*, Seattle, WA, USA, pp. 13–24.

Ozden, B., Ramaswamy, S. & Silberschatz, A. (1998). Cyclic association rules, *Proc. 1998 Int'l Conf. Data Engineering (ICDE'98)*, Orlando, FL, pp. 412–421.

Padmanabhan, B. & Tuzhilin, A. (1998). A belief-driven method for discovering unexpected patterns, *Proc. 4th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD-98)*, pp. 94–100.

Padmanabhan, B. & Tuzhilin, A. (1999). Unexpectedness as a measure of interestingness in knowledge discovery, *Decision Support Systems* **27**(3): 303 – 318.

Park, J. S., Chen, M.-S. & Yu, P. S. (1995). An effective hash based algorithm for mining association rules, *in* M. J. Carey & D. A. Schneider (eds), *Proc. 1995 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD'95)*, San Jose, California, pp. 175–186.

Pei, J., Han, J. & Lakshmanan, L. V. S. (2001). Mining frequent item sets with convertible constraints, *Proc 2001 Int'l Conf on Data Engineering (ICDE'01)*, pp. 433–442.

Pei, J., Han, J., Lu, H., Nishio, S., Tang, S. & Yang, D. (2001). H-mine: Hyper-structure mining of frequent patterns in large databases, *Proc. 2001 Int'l Conf. on Data Mining (ICDM'01)*, San Jose, CA.

Piatetsky-Shapiro, G. & Frawley, W. (1991). *Knowledge Discovery in Databases*, AAAI/MIT Press, Cambridge, MA.

Piatetsky-Shapiro, G. & Tamayo, P. (2003). Microarray data mining: Facing the challenges, *SIGKDD Explorations* **5**(2): 1–5.

Quinlan, J. R. (1986). Induction of decision trees, *Machine Learning* **1**: 81–106.

Quinlan, J. R. (1987). Simplifying decision trees, *Interational Journal of Man-Machine Studies* **27**: 221–324.

Quinlan, J. R. (1991). Improved estimates for the accuracy of small disjuncts, *Machine Learning* **6**: 93–98.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA.

Quinlan, J. R. & Rivest, R. L. (1989). Inferring decision trees using the minimum description length principle, *Information and Computation* **80**: 227–248.

Ramaswamy, S., Mahajan, S. & Silberschatz, A. (1998). On the discovery of interesting patterns in association rules, *Proc. 1998 Int'l Conf. Very Large Data Bases*, Morgan Kaufman, New York, NY, pp. 368–379.

Rastogi, R. & Shim, K. (2000). PUBLIC: A decision tree classifier that integrates building and pruning, *Data Mining and Knowledge Discovery* **4**(4): 315–344.

Ripley, B. (1996). *Pattern Recognition and Neural Networks*, Cambridge University Press.

RuleQuest (2000). See5/c5.0. RULEQUEST RESEARCH data mining tools [http://www.rulequest.com/].

Russell, S. J. & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*, Prentice Hall.

Rymon, R. (1993). An SE-tree based characterization of the induction problem, *Proc. 10th Int'l Conf. on Machine Learning (ICML'93)*, pp. 268–275.

Sahar, S. (1999). Interestingness via what is not interesting, *Proc. 5th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD-99)*, ACM Press, San Diego, CA, USA, pp. 332–336.

Sakakibara, Y. (1993). Noise-tolerant occam algorithms and their applications to learning decision trees, *Machine Learning* **11**(1): 37 – 62.

Salzberg, S. L. (1997). On comparing classifiers: Pitfalls to avoid and a recommended approach, *Data Mining and Knowledge Discovery* **1**(3): 317 – 328.

Savasere, A., Omiecinski, E. & Navathe, S. B. (1995). An efficient algorithm for mining association rules in large databases, *Proc. 21st Int'l Conf. on Very Large Databases*, Zurich, Switzerland, pp. 432–444.

Sebban, M. & Janodet, J.-C. (2003). On state merging in grammatical inference: A statistical approach for dealing with noisy data, *Proc. 20th Int'l Conf. on Machine Learning (ICML2003)*, Washington, DC, USA, pp. 688–695.

Sese, J. & Morishita, S. (2002). Answering the most correlated n association rules efficiently, *Proc. 6th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'02),*, Helsinki, Finland, pp. 410–422.

Shafer, J. C., Agrawal, R. & Mehta, M. (1996). SPRINT: A scalable parallel classifier for data mining, *in* T. M. Vijayaraman, A. P. Buchmann, C. Mohan & N. L. Sarda (eds), *Proc. 22nd Int'l Conf. Very Large Databases (VLDB'96)*, Morgan Kaufmann, pp. 544–555.

Silberschatz, A. & Tuzhilin, A. (1995). On subjective measures of interestingness in knowledge discovery, *Proc. First Int'l Conf. on Knowledge Discovery and Data Mining*, Montreal, Canada, pp. 275–281.

Silberschatz, A. & Tuzhilin, A. (1996). What makes patterns interesting in knowledge discovery systems, *IEEE Transactions on Knowledge and Data Engineering* **8**(6): 970–974.

Silverstein, C., Brin, S., Motwani, R. & Ullman, J. D. (2000). Scalable techniques for mining causal structures, *Data Mining and Knowledge Discovery* **4**(2/3): 163–192.

Srikant, R., Vu, Q. & Agrawal, R. (1997). Mining association rules with item constraints, *in* D. Heckerman, H. Mannila, D. Pregibon & R. Uthurusamy (eds), *Proc. 3rd Int'l Conf. Knowledge Discovery and Data Mining, KDD'97*, AAAI Press, pp. 67–73.

Sun, Q., Zhang, X. & Ramamohanarao, K. (2003). The noise tolerance of ep-based classifiers, *Proc. 16th Australian Conf. on Artificial Intelligence*, Perth, Australia, pp. 796–806.

Tham, S.-S., Doucet, A. & Ramamohanarao, K. (2002). Sparse bayesian learning for regression and classification using markov chain monte carlo, *Proc. 19th Int'l Conf. on Machine Learning*, Sydney, Australia.

Toivonen, H. (1996). Sampling large databases for association rules, *in* T. M. Vijayaraman, A. P. Buchmann, C. Mohan & N. L. Sarda (eds), *Proc. 1996 Int'l Conf. Very Large Data Bases*, Morgan Kaufman, Mumbay, India, pp. 134–145.

Wang, K., He, Y. & Han, J. (2000). Mining frequent itemsets using support constraints, *Proc. Int'l Conf. on on Very Large Data Bases (VLDB'00)*, Cairo, Egypt, pp. 43–52.

Wang, K., Xu, C. & Liu, B. (1999). Clustering transactions using large items, *Proceedings of the eighth international conference on Information and knowledge management*, ACM Press, pp. 483–490.

Wang, K., Zhou, S. & He, Y. (2000). Growing decision trees on support-less association rules, *in* R. Ramakrishnan, S. Stolfo, R. Bayardo & I. Parsa (eds), *Proc. 6th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD-00)*, ACM Press, N. Y., pp. 265–269.

Wang, Z. (2003). *Classification based on maximal contrast patterns*, Master's thesis, University of Melbourne.

Webb, G. & Pazzani, M. (1998). Adjusted probability naive bayesian induction, *Proc. 10th Australian Joint Conf. on Artificial Intelligence*, Brisbane, Australia, pp. 285–295.

Witten, I. H. & Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, San Francisco, CA.

Wong, L. (2002). Datamining: Discovering information from bio-data, *in* T. Jiang, Y. Xu & M. Zhang (eds), *Current Topics in Computational Biology*, MIT Press, Cambridge, MA, pp. 317–342.

Yeoh, E.-J., Ross, M. E., Shurtleff, S. A., William, W. K., Patel, D., Mahfouz, R., Behm, F. G., Raimondi, S. C., Reilling, M. V., Patel, A., Cheng, C., Campana, D., Wilkins, D., Zhou, X., Li, J., Liu, H., Pui, C.-H., Evans, W. E., Naeve, C., Wong, L. & Downing, J. R. (2002). Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling, *Cancer Cell* **1**: 133–143.

Yi, L. & Liu, B. (2003). Eliminating noisy information in web pages for data mining, *Proc. ACM SIGKDD Int'l Conf on Knowledge Discovery and Data Mining (KDD2003)*, Washington, DC, USA.

Yin, X. & Han, J. (2003). CPAR: Classification based on predictive association rules, *Proc. 2003 SIAM Int'l Conf. on Data Mining (SDM'03)*, San Fransisco, CA.

Zaiane, O. R., El-Hajj, M. & Lu, P. (2001). Fast parallel association rule mining without candidacy generation, *Proc. IEEE 2001 Int'l Conf. on Data Mining (ICDM'2001)*, San Jose, CA, USA, pp. 665–668.

Zhang, X. (2001). *Emerging Patterns: Efficient Constraint-based Mining and the Aggregating Approach for Classification*, PhD thesis, Melbourne University.

Zhang, X., Dong, G. & Ramamohanarao, K. (2000a). Exploring constraints to efficiently mine emerging patterns from large high-dimensional datasets, *Proc. 6th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD'00)*, Boston, USA, pp. 310–314.

Zhang, X., Dong, G. & Ramamohanarao, K. (2000b). Information-based classification by aggregating emerging patterns, *Proc. 2nd Int'l Conf. on Intelligent Data Engineering and Automated Learning (IDEAL'00)*.

Zhang, X., Dong, G. & Ramamohanarao, K. (2001). Building behaviour knowledge space to make classification decision, *Proc. 2001 Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD'01)*.

Zheng, Z. & Webb, G. (2000). Lazy learning of bayesian rules, *Machine Learning* **41**(1): 53–84.

Zhu, X., Wu, X. & Chen, Q. (2003). Eliminating class noise in large datasets, *Proc. 20th Int'l Conf. on Machine Learning (ICML2003)*, Washington D.C., USA, pp. 920–927.

Zou, Q., Chu, W., Johnson, D. & Chiu, H. (2002). A pattern decomposition algorithm for data mining of frequent patterns, *Knowledge and Information Systems* **4**(4): 466–482.

Appendix A

Summary of Datasets

In this Appendix, we provide a brief description of the datasets used in our experiments, as shown in Table A.1. The characteristic of these domains is given, including dataset size, the number of classes, the number of continuous attributes, the number of discrete attributes and the total number of all attributes (both continuous and discrete attributes).

Table A.1: Description of classification problems

Domain (Dataset Name)	Size	No. of	No. of Attributes		
		Classes	Continous	Discrete	Total
Adult census income	48,842	2	6	8	14
Australian credit approval	690	2	6	8	14
Breast cancer	699	2	9	0	9
Chess End-Game	3,196	2	0	36	36
Cleve	303	2	6	7	13
Connect-4 Opening	67,557	3	0	42	42
Crx	690	2	6	9	15
Diabetes	768	2	8	0	8
Flare (solar flare)	1,389	2	0	10	10
German Credit	1,000	2	7	13	20
Glass identification	214	6	9	0	9
Heart disease	270	2	13	0	13
Hepatitis prognosis	155	2	6	13	19
Horse colic	368	2	7	15	22
Hypothyroid diagnosis	3,163	2	7	18	25
Ionosphere	351	2	34	0	34
labor negotitions	57	2	8	8	16
Letter	20,000	26	16	0	16
Lymphography	148	4	0	18	18
Mushroom	8,124	2	0	22	22
Pima	768	2	8	0	8
Satimage (satellite image)	6,435	6	36	0	36
Segment (Image Segmentation)	2,310	7	19	0	19
Shuttle	58,000	7	9	0	9
Shuttle-small	5,800	7	9	0	9
Sick	2,800	2	7	22	29
Sonar	208	2	60	0	60
Splice	3,190	3	0	60	60
Tic-Tac-Toe Endgame	958	2	0	9	9
Vehicle silhouette	846	4	18	0	18
Vote	435	2	0	16	16
Waveform-21	5,000	3	21	0	21
Wine Recognition	178	3	13	0	13
Yeast	1,484	10	8	0	8
Zoo	101	7	2	16	18