Investigation on Prototype Learning

Keung Chi-Kin
姜志堅

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Systems Engineering
and Engineering Management

摘要

基於實例的學習算法已被證實爲有效於不同範疇中的模式分類。雖然它們有著良好的歸納精確性，但同時也有高計算成本、高存儲要求及高噪音數據敏感性的缺點。研究人員嘗試用典型例子學習方法解決這些問題，學習典型例子的方法有很多，其中包括實例過濾及實例抽象。以往，這兩種方法是分別地獨立使用的，經過仔細分析兩種方法的優點及缺點後，我們發現它們可以互補不足。在這篇論文中，我們建議了兩種不同的集成方法，用以結合實例過濾及實例抽象。第一種方法名爲增量集成，這方法增量地採用實例過濾及實例抽象。在這種集成方法下，實例過濾及實例抽象能夠以增量的方式互補不足。我們採用了這種集成方法而提議出一個名爲 PGF 的結構。基於不同種類的實例過濾方法及不同的過濾粒度，我們設計了不同的 PGF 算法。第二種集成方法名爲概念集成，這方法首先分別以不同的典型例子學習方法學習出不同的概念，然後把這些概念集成，整合不同概念的優點。我們採用了這種集成方法而提議出一個名爲 ICPL 的算法。我們使用了三十五個現實中的指標數據集來試驗 PGF 及 ICPL 算法，實驗結果顯示它們能夠以少量的典型例子取得高水平的歸納精確性，比現存的算法優勝。

Abstract

Instance-based learning algorithms have been proven to be effective in pattern classification in many domains. Despite their high generalization accuracy, they suffer from three main drawbacks, namely, high classification computational cost, storage requirement and noise sensitivity. Researcher attempt to solve these problems using prototype learning. Some approaches have been proposed to learn representative prototype sets including instance-filtering and instance-abstraction. These two approaches are employed independently in the past. After analyzing the strengths and weaknesses of the two methods, we find that they can be complementary with each other. In this thesis, we propose two approaches integrating the two methods. The first approach is regarded as *incremental integration* in which filtering and abstraction methods are applied incrementally. We develop a framework called PGF which combines filtering and abstraction using this integration approach. Different variants of PGF are designed based on different types of filtering techniques and filtering granularity. The second approach is regarded as *concept integration* in which concepts are learned by filtering and abstraction independently and integrated. It attempts to unify the strengths of the two learned concepts. An algorithm called ICPL is proposed using this integration approach. Empirical results of PGF and ICPL on 35 real-world benchmark data sets suggest that our proposed algorithms achieve a very good data retention rate and noise reduction compared with state-of-the-art algorithms with comparable or even superior generalization accuracy.

Acknowledgments

Thanks to my supervisor Dr. Wai Lam for his unending advice in this work. He taught me how to do quality research and how to solve problems logically. It is really harsh for him to supervise such an amateur researcher like me. Without his patient help, this work can hardly be finished.

All other kind people in the department also helped me a lot. Thanks go to Wai Ip (Egg Big) who gave me valuable advice in many aspects including research and technical support. He really helped me solve a lot of difficult problems. Thank my friends for sharing with me when frustrated by the research including Kun Chung, Silvia (Siu Fei) and Timmy. Thanks also go to the ones spending their valuable to play and talk with me including Aha Luk, Carmen and Tony, Ah So, Sally, Connie (Chu Ching), Brenda and Qiuyue, etc. I would also like to thank the system administrators for tolerating my abuse of the machines.

I am especially grateful to my lover, Ching Ching, my family and my best friends. Their loves and supports have made life wonderful for me. Thanks Ching Ching's physical and mental support during weekends. During these two years, I really spent not enough time with them. After this work, it is the time for me to pay them back. I hope that I can use the knowledge and experience gained through these two years to serve them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We can deal with problems in various domains well using past experience, heuristic rules, trial-and-error, cognition as well as common sense. Compared with human, computer programs are rigid in solving problems. They cannot respond to unseen situations which are not planned by program designers beforehand. However, it is usually impractical and impossible for one to determine all possible situations that the program will be faced in advance.

People try to build robust systems by implanting human expert knowledge in computer programs to handle new situations. This motivates the creation of *expert systems.* However, if the domain knowledge is expensive or not sufficiently developed, expert systems cannot be developed. Besides, it may be difficult to capture in-depth knowledge of the problem domain. Expert systems are also lack of intelligence. Once a system is implemented, expert knowledge is hard-coded and its performance will not be improved to cope with new situations.

In view of this, we need a learning system which can solve problems adapting to new knowledge. This kind of learning method is called *machine* *learning* and has become a rigorous research topic in recent decades. Machine learning attempts to make decisions based on the accumulated experience contained in successfully solved cases [61], that is, learning from examples. With a learning system, compared with an expert system, we can obtain solutions with higher accuracy at a lower cost. To date, machine learning algorithms have been widely used in different data mining tasks, such as clustering, classification, estimation and forecasting [6]. This thesis will focus on *classification.*

1.1 Classification

Learning is the change of a system which allows it to perform better the second time on repetition of the same task or similar task in the same domain [52]. It involves knowledge acquisition from experience. In most problems, the available data is limited and *induction* must be used for a system to acquire knowledge. It is therefore known as *inductive learning. Concept learning*, also known as *classification*, is a typical inductive learning problem. Classification is a supervised machine learning that acquires knowledge from a collection of examples to classify unlabeled cases. Typically, an example is comprised of a feature vector representing the characteristics of the instance and a class label indicating to which category the instance belongs. The collection of examples is regarded as the *training set.* In order to classify new cases, a system has to learn how to map input vectors into correct class labels. When the input vector of a new case comes, it will predict the correct output class based on the knowledge induced. This process is called *generalization.* To evaluate the performance of a classifier, we usually measure its generalization accuracy on unseen input vectors, known as *testing set*.

In recent decades, researchers investigate techniques to learn more, better and faster from examples for concept description. Many machine learning algorithms have been proposed to describe concepts including instance-based learning, decision trees, rule induction, neural networks and genetic algorithms.

Quilan proposed two *decision trees* algorithms which have been proved to be successful in many applications [45, 46]. During the learning phase, trees are built by splitting the training set using attribute values. Information gain is adopted to be the splitting criteria. These algorithms are fast and capable of handling problems with a large number of instances. Tree representation, however, can only represent rectangular-shaped decision regions bounded by lines parallel to the dimension axes [61]. This drawback greatly limits the representation power of the models.

Rule induction algorithms have also been proposed including [14, 40]. Rules induced are usually composed of a boolean combination of attribute tests (selectors) and an output class. Heuristic functions are used to terminate the induction of rules. The induced rules are usually compatible with human cognition, which is an important consideration in many data mining applications. However, these algorithms are quite similar with decision trees in terms of the representation of the acquired knowledge. Therefore, they suffer from the same drawback.

Neural networks are non-linear predictive models that resemble biological nervous system during learning and have been used to perform classification for a long time [32, 39]. They may perform well on applications where intense human sensory processing is required, such as speech and image recognition [61]. However, a large number of iterations may be required in the training phase and the learned knowledge cannot be easily interpreted by human reasoning.

Genetic algorithms [9, 35] learn concepts using heuristic search based on genetic operators and fitness functions. In genetic algorithm, knowledge is represented by a bit string. During the learning phase, the bit string is changed using genetic operators so that the fitness function is optimized. These algorithms usually terminate after a pre-defined number of iterations. Some problems, however, cannot be easily represented using bit strings on which the performance is highly dependent.

Another common learning algorithm is *instance-based learning.* This thesis focuses on this kind of algorithm and a detailed description is given below.

1.2 Instance-Based Learning

When we perform tasks based on memory, we often recall past experiences and map them to the current situation. Relying on his memory of previous cases and recalling patients with similar symptoms, a doctor suggests a treatment for a new patient. Past cases are also used when a lawyer prosecutes a criminal in the court. This kind of problem solving technique lays the foundation for instance-based learning algorithms.

Instance-based learning algorithms fall into the group of *exemplar-based learning* [49]. They simply store past examples or instances in memory without any change in representation. During execution, when an unseen instance comes, it is compared with all the saved instances to perform some tasks. In classification, typically, a similarity function is adopted to measure the similarity between the input instance and the retained ones and the class label of the most similar stored vector will be output. This kind of learning method is sometimes called *lazy learning* where data processing is deferred until queries (i.e., new data to be classified) come.

1.2.1 Three Basic Components

According to [2], instance-based learning algorithms in classification have the following three components:

1. *A representative instance selection function*: It tells the algorithm which of the instances to store in the memory. This function is critical to the performance of the algorithm. If non-representative instances are selected, we can hardly match them with the input instance and correct prediction cannot be made.

2. *A similarity function*: It specifies how similar two instances are in the target domain. Choosing a similarity function is not an easy task, especially when there are attributes with discrete values. For example, it is difficult to measure the difference between two color, says yellow and red.

3. *A classification function*: After determining the similarities between the stored and unseen instances, we have to develop a method to make use of them to perform predictions. For example, this function might return the class label of the instance to which the unseen case is geo-

metrically closest.

Intensive research has been conducted to improve the performance of instance-based learner by developing new algorithms for the above components.

1.2.2 Advantages

Instance-based learning has some advantages over other machine learning algorithms.

1. It learns quickly. As mentioned before, it just simply stores the past examples without preprocessing them so that the time complexity of the learning phase depends on the number of instances retained.

2. It can learn well from a very small data set. Whereas other machine learning algorithms require a reasonable number of instances to learn, instance-based learning can predict correctly using as little as a few instances per each class. It is therefore useful for applications where only a small amount of data is available. Besides, it can also learn incrementally by just adding more instances to the system when extra useful data is available.

3. Instance-based learners store instances without changing their representation. Therefore, unlike other machine learning algorithms, concepts can be represented in a similar way as the instance representation. It allows the learning system to represent complex concepts. Moreover, the induced knowledge is intuitive and easy to understand by human.

1.2.3 Disadvantages

However, instance-based learning algorithms have the following three drawbacks:

1. They require high computational cost during execution. As the algorithms simply store training instances in the learning phase, the burden of data processing is deferred to on-line classification. All the retained training instances must be searched in order to make correct predictions and sometimes the execution time will be unacceptable long.

2. In some applications, a large amount of instances is required to represent the concepts. In order to gain a good generalization accuracy, the storage requirement of instance-based learning methods is usually high. For applications where millions of instances are available, it may be impractical to save all of them.

3. They are sensitive to noisy instances. It is obvious that correct prediction cannot be made using instances with wrong class labels.

1.3 Thesis Contributions

We attempt to overcome the drawbacks of instance-based learning by developing new algorithms for the representative instance selection component in instance-based learning. With a good instance selection component, we can use a few representative instances to substitute the entire training set. Therefore, the high computational cost, storage requirement and noise sensitivity can be improved. The discovery of representative instances is also known as *prototype learning.* In this thesis, we focus on two common prototype learning approaches, namely, *instance-filtering* and *instance-abstraction.* After analyzing the two approaches, we notice that they have their own strengths and weaknesses and can be beneficial to each other. We design two prototype learners which integrate the two approaches and unify their strengths to learn representative prototypes. Our objective is to achieve low data retention rate while maintaining or even improving the generalization accuracy. Empirical results show that our proposed algorithms are applicable to perform accurate classification using small prototype sets on a wide variety of real-world domains.

1.4 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 describes two approaches to learning prototype set including instance-filtering and instance-abstraction. We review some previous work on these two approaches and examine the strengths and weaknesses of them.

In Chapter 3, we investigate how the two prototype learning approaches can be complementary with each other. Two integration approaches, called incremental integration and concept integration, are presented and some issues related to them are discussed.

We propose a framework combining filtering and abstraction based on incremental integration to learn representative prototypes in Chapter 4. Two algorithms are developed under this framework. A thorough empirical analysis on the issues affecting this integration method is also conducted on 35 benchmark data sets.

Chapter 5 proposes another prototype learning algorithm developed by combining filtering and abstraction using concept integration. It provides a detailed analysis on designing and selecting the abstraction and filtering components of the proposed algorithm. Experiments on 35 benchmark data sets are also conducted to compare the performance of the algorithm with existing classification algorithms.

In the last chapter, we provide conclusions on our research. Some future research directions on integration of filtering and abstraction in prototype learning are also described.

Chapter 2

Background

Instance-based learning methods suffer from high computational cost, storage requirement and noise sensitivity. Researchers attempt to solve these problems by scaling-up the nearest neighbor search and data reduction. In this chapter, we first provide some background information on these two methods. Data reduction, regarded as prototype learning in this thesis, is chosen to improve instance-based learning algorithms. Two approaches, namely, instance-filtering and instance-abstraction, are adopted in many existing work. We review the related literature and investigate the strengths and weaknesses of the two approaches in this chapter.

2.1 Improving Instance-Based Learning

Though instance-based learning algorithms are found to be effective in many applications, they suffer from some drawbacks including high on-line computational cost, storage requirement and noise sensitivity. Two main approaches are employed to ameliorate instance-based algorithms in the past.

The first one is to scale-up the nearest neighbor search. This method improves the on-line computational cost by reducing the number of distance calculations in identifying nearest neighbors of unseen cases. The second one is data reduction. This method attempts to find a relatively small amount of representative instances to represent the entire training set and, therefore, both the storage requirement and classification computation of instance-based algorithms can be reduced.

2.1.1 Scaling-up Nearest Neighbor Searching

During on-line classification, most instance-based learning algorithms require the identification of the unseen cases' nearest neighbors. The computational cost of instance-based learning can be improved using some sophisticated nearest neighbor searching methods. Nearest neighbor search has been investigated for many years [5, 31, 43, 55, 71]. Most scaling-up searching methods involve projections of instances to different axis in the dimension of the target domain space in an attempt to decrease the amount of distance calculations when locating the nearest neighbors [31, 43, 71]. These algorithms usually work well under certain assumptions on the data distribution. For example, k-dimensional tree [55] was proposed to find nearest neighbors in $O(logn)$ time in the best case where n is the number of instances stored. However, as the dimensionality of data increases, this algorithm cannot reduce the searching to a large extent. Some algorithms can handle data in high-dimensional space with some data preprocessing [5].

2.1.2 Data Reduction

Improving the nearest neighbor searching can only help the computational cost during on-line classification. However, the remaining two problems for instance-based learning, high storage requirement and noise sensitivity, are still not addressed. One intuitive way to solve these problems is to reduce the data set so that the original training set can be represented using fewer instances. With data reduction, both the number of distance calculations and instances stored are reduced. Moreover, noisy instances can be removed in the reduced data set so that the generalization accuracy can be improved. Therefore, data reduction seems to be a good solution. It is, however, dangerous to eliminate instances from the training set. The performance of instance-based learning algorithms depends highly on the quality of the instances retained for on-line classification. Therefore, if representative instances are carelessly removed, accurate predictions cannot be made. This characteristic explains why reducing the data set usually results in degradation in generalization accuracy if not done properly.

2.2 Prototype Learning

Different communities including machine learning and pattern recognition have proposed different methods for data reduction in instance-based learning in the past few decades. Starting from 1968 [25], intensive research trying to select representative instances has been conducted in pattern recognition. In the pattern recognition community, this problem is also called *reference set selection* and algorithms for performing this task are termed *editing rules* which edit the training set using specifically designed rules. Not until late 1980's, researchers in machine learning showed their interest in this problem [29]. Kibler and Aha pose this problem as an instance selection in instance-based learning triggering numerous research in this direction.

Though the same topic has been investigated in different communities, different terminologies are adopted. In cognitive science, the term *prototype* refers to an ideal example which can be distinguished from each other while in pattern recognition it refers to an instance stored to represent the training set. Some researchers regard prototypes as artificial instances generated by averaging or generalizing instances in the training set [13]. To avoid ambiguity and confusion, we first define some terminologies used in this thesis. The term *prototype* refers to an instance selected or automatically generated by any reduction algorithm to represent the original data set and the set of prototypes is called *prototype set.* According to the definition, data reduction can be regarded as *prototype learning* in which a smaller prototype set is learned from the original training set. In the following sections, we first discuss the objectives of prototype learning. Related work on different prototype learning approaches is then reviewed and the strengths and weaknesses of each of them are studied.

2.2.1 Objectives

The objective of prototype learning is to discover representative prototypes from the training instances. To identify representative prototypes, we must know how to measure the *representativeness* of an instance. Zhang propose a *typicality*[1] function to measure the degree of representativeness of instance [72]. However, this measure is effective in identifying how far an instance is apart from the decision boundary (see Chapter 5) rather than a good indicator to measure representativeness.

As for prototype learning for instance-based algorithms, our objective is to learn as few prototypes as possible to represent the training set so as to reduce the classification computation and storage requirement. We also attempt to eliminate noise which cannot be effective handled by instance-based learning methods. In view of this, we can define *representativeness* of a prototype in this way: the degree of contribution of a prototype to perform accurate classification. Moreover, this concept can be extended to the entire prototype set rather than an individual prototype. A prototypes set containing representative prototypes may not perform well if those prototypes do not have a sufficient coverage of the target domain. However, in order to obtain a sufficient coverage, a large amount of prototypes may be required. Consequently, the most intuitive way to measure the performance of prototype learning methods is to consider both the generalization accuracy as well as the amount of prototype retained. A representative prototype set should achieve a high generalization accuracy and, at the same time, a low data retention rate. This approach actually also takes into account the ability to tolerate noise as noise usually hurts generalization accuracy.

[1]An in-depth analysis of typicality will be provided in Section 5.2.2.

2.2.2 Two Types of Prototype Learning

Prototype learning techniques can be distinguished by the nature of the prototypes it learns. Kibler and Aha classify the techniques into two groups, namely, *instance-filtering* and *instance-averaging* [30]. Instance-filtering represents concepts by filtering in or out instances from the training set to form the target subset. In instance-averaging, prototypes are learned by averaging out some instances in the training set and storing the remaining ones. In this thesis, we define methods generating artificial prototypes by summarizing or generalizing (including averaging) training instances as *instance-generation* or *instance-abstraction.*

2.3 Instance-Filtering Methods

Human often use past experiences to solve problems. However, it is impossible for us to memorize all the past cases and match them with the current situation. Therefore, we usually just memorize some typical and representative examples. Instance-filtering learns concepts in this manner. In instance-filtering methods, representative prototype is learned by filtering rules which are designed to determine whether an instance should be retained as a prototype or not. That is, the learned prototype set is a subset of the original training set. Previous proposed methods differ from search directions and locations of instances retained. According to Wilson and Martinez [69], filtering methods can be distinguished into three groups: 1) *retaining border instances,* 2) *removing border instances* and 3) *retaining center instances,* where border instances refer to instances near decision boundaries while center ones refer to the ones far away.

2.3.1 Retaining Border Instances

Most instance-filtering techniques fall into this category. This method attempts to represent concepts using instances in decision boundaries among different classes while intermediate and center instances are discarded since elimination of them usually do not affect the decision boundaries. This method usually gain high generalization accuracy since class boundaries are preserved and achieves a reasonable data reduction rate. In most filtering methods, border instances are identified using the class labels of their neighbors as their neighbors are usually of other classes. However, noisy instances are also retained as they have similar behavior with border instances under this filtering criteria.

Condensed Nearest Neighbor. The earliest editing rule called *Condensed Nearest Neighbor* (CNN) is introduced by Hart [25]. This bottom-up algorithm randomly selects one instance for each class to form the initial prototype set. An instance is retained if it is misclassified by the current prototype set so that a *consistent subset* is obtained which means all the training instances can be correctly classified by the prototype set. This algorithm tries to retain instances in the class boundaries with a low selection cost. However, it cannot guarantee a minimal subset and is sensitive to the initial state and instance presentation order. Besides, since noisy instances are usually misclassified by their neighbors, they are likely to be retained by the algorithm.

Reduced Nearest Neighbor. Gates proposes an iterative, top-down variant of CNN called *Reduced Nearest Neighbor* (RNN) [23]. Starting with all instances as the prototype set, this algorithm deletes an instance if its deletion does not result in any misclassification of other instances. Instance deletion continues until no more instances are removed. This algorithm also intends to find a consistent subset by retaining instance in the class boundaries. It achieves a greater reduction than CNN. Unlike CNN, RNN removes instances which may be misclassified so that some noise will be removed.

Iterative Condensation Algorithm. Swonger proposes another variant of CNN called *Iterative Condensation Algorithm* (ICA) [57]. ICA allows both deletion and selection of instances within a iteration so that it is not sensitive to the initial state of the algorithm. It discards instances appearing to cause more misclassifications than correct classifications and retains those reducing the number of classification errors. It can handle noise, outliers and allow identical-valued instance with different labels.

Selective Nearest Neighbor. Ritter et al. introduce a top-down *Selective Nearest Neighbor* (SNN) algorithm to determine a possible minimal subset approximating the decision boundaries [48]. It ensures each instance in the original data set is closer to an instance in the selected set of the same class than any other instance of other classes. It leads to a consistent subset and achieves both a greater reduction and higher accuracy than CNN. However, in order to maintain the consistency, noise is also retained.

Mutual Nearest Neighbor. Gowda and Krishna introduce a new concept, called *mutual nearest neighborhood*, which takes into account the two-way nearness of two instances in finding prototype set [24]. They modify CNN retaining instances misclassified by their mutual nearest neighbors instead of nearest neighbors.

Multiedit. Many previous work use all available instances to learn and test the subset so that there is a high dependence between the training and testing samples. Devijver and Kittler mention that this high dependence can lead to a constantly optimistically bias in error estimation of the subset [21]. They try to remove this bias by randomly partitioning the available instances into multiple independent sets. They propose an iterative top-down *MultiEdit* algorithm discarding instances misclassified by its nearest neighbor in other subset. This iterative process continues until no further instance editing. The MultiEdit algorithm initiates the use of sampling in instance-filtering.

Instance-Based Learning Algorithms. Classification accuracy can be improved by removing noisy instances and irrelevant features in instance-based learning. Aha et al. have investigated these two factors in instance-filtering intensively. A noise-tolerant instance filtering called NTGrowth is proposed by Aha and Kibler [3]. NTGrowth retains misclassified instances and keeps track of the classification performance of each of them. Noisy data, usually have low accuracy, will be discarded.

Later, Aha et al. formalize NTGrowth to the well-known IB2 and IB3 algorithms retaining border instances incrementally [2]. IB2 is similar to CNN saving misclassified instances except that instances are normalized by the range of attributes and missing values are handled. However, like CNN, it is sensitive to noise.

IB3 is then proposed to tolerate noise. It accepts instances with classification accuracy significantly greater than the frequency of the observed class. Noise, usually has a low classification accuracy, will probably be eliminated. However, both methods do not retain correctly classified instances which may be useful in classification. Apart from noise tolerance, two incremental instance-based learning algorithms are proposed to determine relative attribute relevances and handle novel attributes which comes up with the IB4 and IB5 algorithms [1]. Later, Cost and Salzberg propose a method called PEBLS using a weighted modified Value Difference Metric (MVDM) to weight symbolic features [16]. Recently, Payne and Edwards a feature selection method also based on Value Difference Metric [44]. A detailed review and evaluation of several major feature weighting methods in lazy learning has been done by Wettschereck et al. [63].

Minimal Consistent Set. Dasarathy presents a new editing rule to achieve a Minimal Consistent Set (MCS) by considering the distance between nearest neighbors of the same class and that between *nearest unlike neighbor* [17]. This method achieves a minimal consistent subset and is insensitive to the initial order of presentation of instances. Instances are also selected in the order of their consistency property. Empirical results show that it outperforms the CNN approach.

Reduction Techniques. Recently, Wilson and Martinez propose three top-down data reduction techniques called RT1-RT3 [69]. RT1 removes an instance if most of its *associate*[2], instances in the current subset having it as one of their k-nearest neighbors, are classified correctly without it. Noisy instances are usually removed as they can hardly categorize their associates accurately while border instances will be retained as their associates tend to be classified correctly with their contribution in KNN classification.

RT2 improves RT1 by considering associates of the original data set instead of the selected subset. It also changes the order of instance removal by sorting the instances in the subset by the distance to their nearest unlike neighbor first. This technique makes RT2 insensitive to the order of data presentation.

RT3 adds a noise-filtering pass before sorting selected instances by ENN and obtains a greater data reduction than RT2. RT3 achieves a high classification accuracy comparable to those of pure KNN with an average 14.32% data retention rate.

Later, they formalize the three algorithms, as well as two extra variants, to form the DROP1-DROP5 algorithms [68]. DROP1-DROP3 refer to RT1-RT3 respectively. DROP4 adopts a careful noise filtering processing instead of simply applying ENN in DROP3. DROP5 is identical to DROP2 except that a decision boundary smoothing pass is performed first. It is achieved by removing instances having nearest neighbor of other classes. DROPs are compared with a number of existing prototype learning algorithms such as CNN, ENN and IB3 and empirical results show that DROPs outperforms all the existing methods in generalization accuracy using relatively small prototype sets. It is also found that DROP2-DROP5 have similar performance in both data retention rate and generalization accuracy while DROP3 performs

[2]An in-depth analysis of associate will be provided in Section 5.3.1.

the best on average.

They also propose an *Integrate Instance-Based Learning Algorithm* which can automatically tunes its parameters, including parameters in distance function, instance pruning techniques and attribute and vote weighting which are essentially the three basic components of instance-based learning, used in the algorithm [67]. A in-depth analysis of the three components can also be found in [66].

2.3.2 Removing Border Instances

These methods remove instances close to decision boundaries and attempt to describe concepts using intermediate and center instances. A smoother decision boundary can be obtained using these methods. Besides, some mislabeled instances can also be removed as they are all like border instances which have neighbors of other classes. However, a relatively low data reduction rate is obtained for this method contrasted with other filtering techniques.

Edited Nearest Neighbor. The *Edited Nearest Neighbor* (ENN) algorithm proposed by Wilson is a top-down algorithm retaining central and intermediate instances [65]. Opposed to CNN, ENN eliminates those instances misclassified by its k nearest neighbors. Since noisy instances are seldom correctly classified, they can be usually removed. However, this algorithm does not achieve a low data retention rate if homogeneous instances are closely packed so that most of them are correctly classified and thus are retained.

Unlimited and All K-NN Editing Nearest Neighbor. Tomek extends ENN to *unlimited editing* and *all K-NN* algorithms [58]. *Unlimited editing* continuously repeats ENN and terminates after the prototype set becomes stable. In *all K-NN*, ENN is also repeated but different number of nearest neighbors are used in determining the acquisition of instances. The two variants of ENN give better performance in terms of both classification accuracy and data reduction, but they require a higher computational cost than pure ENN.

New Edited Nearest Neighbor. A new edited nearest neighbor rule is proposed by Hattori and Takahashi [26]. The new rule retains an instance which k or $(k+l)$ nearest neighbors are of the same class of it where l is the number of instances which tie with the kth nearest neighbor with respect to the distance from it. The authors show that the new algorithm outperforms the classical KNN and ENN algorithms. However, like ENN, the data retention rate of the algorithm is also high.

2.3.3 Retaining Center Instances

In some applications, many instances may be required to describe decision boundaries leading to poor data reduction rate. In view of this, some researchers design filtering methods retaining center instances instead of border ones. It can greatly reduce the storage requirement for instance-based learning algorithms. However, the decision boundaries may be severely distorted with the absence of border instances resulting in poor generalization accuracy. Therefore, center instances must be carefully selected in order to maintain a high accuracy.

Typical Instance-Based Learning. Zhang proposes a measure, called *typicality*, to indicate how typical an instance is by considering the average similarity of instances of the same class and that of instance of other classes. Based on typicality, Zhang presents two instance-filtering techniques, namely Typical Instance-Based Learning (TIBL) storing typical instances and Boundary Instance-Based Learning (BIBL) storing atypical instances [72]. TIBL identifies new typical instances by considering their performance on previously misclassified instances and BIBL retains least typical instances including boundary and exceptional instances. It is found that TIBL attains a better performance in terms of both data retention rate and generalization accuracy.

After reviewing some the filtering techniques, we try to identify the advantages and disadvantages of them.

2.3.4 Advantages

We find that filtering techniques have the following advantages:

1. Filtering methods are usually simpler and faster. Therefore, the behavior such as the convergence of instance-filtering algorithms can be investigated due to their simplicity [30].

2. Apart from their simplicity, most filtering methods can gain high accuracy which is comparable or even superior to instance-based learner storing the entire training set.

3. As filtering techniques select representative instances from the original instance set, they can truly represent the original concept and the

learned concept is easily comprehended.

4. Since filtering rules can be designed to filter in or out different instances such as border [2, 23, 25, 48, 69], central points [57, 58, 65] and even noise and outliers [2, 25, 69], one can easily apply different rules simultaneously or separately to filter different instances. This flexibility can also be extended to other prototype learning methods such as instance-abstraction.

2.3.5 Disadvantages

However, filtering techniques have the following disadvantages:

1. They assume that ideal examples can be found in the training set. If it is not the case, the performance of filtering techniques will not be good.

2. Filtering techniques do not perform abstraction of instances, therefore, the representative power of prototypes may be limited by just selecting from original instances.

3. Some methods enforcing the edited subset to be consistent may lead to overfitting.

4. Noisy instances are usually selected by boundary instance retaining methods as they have similar behaviors under most of the filtering rules.

5. For filtering methods retaining or removing border points, data retention rate is usually high compared with instance-abstraction [58, 65].

6. Some filtering methods are sensitive to the order of presentation of instances.

2.4 Instance-Abstraction Methods

Sometimes, to handle a large amount of past cases, instead of memorizing representative ones, we can conduct generalization on similar cases. Through abstracting past cases, we can generate distinctive cases and use them to make decisions. This is the basic idea of instance-abstraction methods which derive prototype set by abstracting or summarizing training instances. They attempt to find the common characteristics of instances for each class. Unlike instance-filtering methods just selecting prototypes appearing in the training set, they involve generating artificial prototypes. Therefore, the learned prototypes may not be found in the training set. We review some of the major previously proposed algorithms below.

Chang's Averaging Method. An early instance-abstraction algorithm is a top-down approach proposed by Chang [13]. The algorithm merges two nearest instances by weighted averaging them. The merging process stops when the number of incorrect classifications starts to increase. It is found to be very effective to reduce the size of data set.

Disjunctive Spanning. Bradshaw introduces the *Disjunctive Spanning* (DS) algorithm which uses weighted averaging on an selected prototype with instances correctly classified by it [10]. Prototypes with more instances merged have a higher weight so that its influence on the next merge will be larger. However, the author claims that the learned prototypes may be non-prototypical.

Adaptive Threshold. Kibler and Aha improve DS by adding an *adaptive threshold* (AT) to limit the distance between the merged instances [30]. Distance threshold in instance-abstraction is first introduced by Sebestyen to prevent generation of non-prototypical prototypes [51]. Kibler and Aha report that AT improves both classification accuracy and data retention rate of DS. AT is then compared with an filtering technique and it is found that the two prototype learners achieve similar results in both accuracy and retention rate. However, the authors claim that instance-avenging techniques generate misclassified instances in the prototype set and are unable to represent concave concepts.

Symbolic Concept Learning. Maza describes a cognitively based symbolic learning system called PROTO-TO which builds a prototype for each class. Instead of averaging the feature values, PROTO-TO learns a prototype by creating a vector for each attribute. Symbolic attribute vector contains the number of occurrences of all the attribute values while continuous one stores the mean and standard deviation of the attribute. After that, augmented prototypes are created by adding a weight to each attribute. Performance of PROTO-TO is shown to be comparable to C4.5 and superior to NTGrowth.

Nested Generalized Exemplar. An algorithm called Nested Generalized Exemplar (NGE) storing instances as hyperrectangles is proposed by

Salzberg [49]. Instances are generalized to form and resize hyperrectangles after correct predictions. The nested hyperrectangles allow the representation of nested concepts.

However, Wettschereck and Dietterich show that NGE is significantly inferior to KNN on some real-world data sets [64]. NGE is then improved by avoiding the creation of overlapping rectangles and using a feature weighted distance metric. The improved NGE is still inferior to KNN as shown by the author. NGE is very sensitive to the shape of decision boundaries of the learning concept.

Wettschereck also combines the nearest neighbor and hyperrectangle concepts to form a hybrid algorithm that classifies unseen cases by hyperrectangles if they are enclosed by any of them and by KNN otherwise [62]. In order to perform KNN, the hybrid method has to store all the instance. Despite its inability to reduce storage, it achieves a high accuracy comparable to KNN with a lower computational cost as hyperrectangles are used.

Prototype Learner. Datta and Kibler introduce a top-down splitting algorithm called *Prototype Learner* (PL) learning prototypes for a concept by generalization of high quality example partitions [18]. This algorithm works on nominal attributes only and partitions homogeneous instances by values of some features. An example partition is said to be of high quality if the randomness of the feature values in the example partition is low. Different from other abstraction algorithms, prototypes learned by PL contain only a subset of features with specified values as well as a class label.

They then propose a *Symbolic Nearest Mean Classifiers* (SNMC) trying to learn a single symbolic prototype for each concept [19, 20]. Using the

MVDM metric to weigh and define distance of symbolic attributes, SNMC begins by clustering homogeneous instances into different groups using k-means clustering. Cluster means will then become the learned prototypes. Different number of clusters (prototypes) are tried to obtain the best results. A variant of SNMC using sum of squared error to determine the number of clusters is also tested. SNMC achieves a classification accuracy superior to C4.5 using one to two prototypes for each class.

Rule Induction from a Set of Exemplars. Domingos proposes an integrated technique, the RISE algorithm, combining instance-based learning and rule induction [22]. Under this algorithm, instances are treated as rules and data reduction is achieved using specific rules formed by abstraction of instances. During the learning phase, rules (instances) are modified to cover as many instances as possible provided that accuracy is not degraded. RISE obtains a better performance than other rule-based systems, says CN2 and C4.5, and a symbolic nearest neighbor classifier.

Modified Chang's Averaging. Bezdek et al. propose a modified Chang's averaging method (MCA) learning multiple prototypes [7]. MCA averages instances using simple instead of weighted means and restricts the merging of instances with the same class label only. MCA is compared with three sequential competitive learning methods including *learning vector quantization* (LVQ), fuzzy LVQ (GLVQ-F) and *dog-rabbit* (DR) models as well as the original Chang's method and a fuzzy $\hat{c}$-means algorithm using resubstitution[3] error rate. Experiments on the iris data show that MCA gains a zero resubstitution error and DR achieves a minimum number of prototypes. Kuncheva and Bezdek further compare the resubstitution error of above algorithms with that of *genetic algorithm* (GA) and a random search approach [35] finding a subset of instance. They find that generated prototypes usually gain lower resubstitution than subset prototypes because the former may not match with real instance well.

Generalized Instance Set. Lam and Ho propose a Generalized Instance Set (GIS) algorithm which learns generalized instances to solve two-class text categorization problems [36]. Initially, an instance is randomly selected as a GI. Then it is abstracted with its k nearest neighbors using one of the two generalization methods, namely, Rocchio and Widrow-Hoff algorithms. A representative power function is then used to evaluate the GI formed. The generalization of that GI continues until its representative power function starts to decrease. This process ends until all the instances are processed. The two GIS algorithms are found to outperform classical KNN algorithm.

Family-Based Learning. Another abstraction technique, called Family-Based Learning (FAMBL), in language learning tasks is proposed by Van den Bosch [60]. It forms hyperrectangles like NGE [49] but a different instance merging procedure is used. It abstracts an instance with its nearest neighbors until an instance of other classes or an abstracted instance is reached. The abstraction process terminates until all the training instances are processed.

[3]Resubstitution is the classification of the original training set using the learned prototype set.

This algorithm is found to have similar generalization accuracy with IB1 using fewer prototypes.

2.4.1 Advantages

After reviewing a number of instance-abstraction methods, we investigate that they have the following advantages:

1. Artificial prototypes generated by instance-abstraction techniques may be more representative than original instances.

2. Border instances may also be abstracted so that smoother decision boundaries can be obtained.

3. Compared to instance-filtering methods, instance-abstraction techniques have a stronger generalizing power on instances. They usually achieve a lower data retention rate.

4. Through summarizing instances, the common characteristic of a concept can be learned and the learned concept has less risk to overfit the original data.

5. As for noise tolerance, abstraction methods can generalize away mislabeled instances in compact region by merging them with other instances.

2.4.2 Disadvantages

On the other hand, instance-avenging methods suffer from the following drawbacks:

1. They are usually more complex than instance-filtering ones. It limits the study of the behaviors of instance-abstraction techniques such as generality and limitations in terms of the concept descriptions [30].

2. Resubstitution performance is not guaranteed by relabeling merged instances in abstraction techniques. The artificial prototype set may even contain mislabeled instances leading to misclassification for unseen cases [30].

3. Abstraction methods attempt to find the common characteristics of similar instances. However, it is difficult to do so if the concepts are very complicated.

4. Artificial prototypes may be non-prototypical and even not be the concept it intends to represent [10]. Therefore, they are less comprehensive than those selected by instance-filtering methods.

5. Simple abstraction also fails to describe concave concepts [30]. This drawback leads to an inferior classification accuracy in such domains.

6. Though abstraction methods can generalize away mislabeled instances in compact regions, they cannot do so for outliers. Merging instances with outliers may form non-prototypical prototypes.

7. It is not easy to find an interpretation for abstracting discrete features during generalization of instances.

2.5 Other Methods

Some researchers use stochastic approach to find a subset of original instances as prototypes. Instead of filtering instances one by one using editing rules, stochastic techniques search the space of sets of prototypes to determine the best subset.

Random Mutation Hill Climbing. Skalak proposes two stochastic methods using random sampling (RS) and iterative random mutation hill climbing (RMHC) to select a set of prototypes [54]. In RS, random sampling is repeated and samples with maximum classification accuracy become the prototype set. In RMHC, a prototype set is represented by a binary string and a random bit is mutated iteratively. The result prototype set is the binary string scoring maximum fitness. RMHC can also be designed to select prototypes and features simultaneously by changing the bit coding. The author then focuses on combining classifiers built from different samples to form composite nearest neighbor classifiers [53]. Editing rules based on genetic algorithms can also be found in [33, 34].

Vector Quantization. Xie et al. use vector quantization technique to find prototypes which are regarded as quantizers created with minimum average distortion rate of quantization [70]. The performance of the learned prototype set is compared with CNN, RNN, ENN and classical nearest neighbor algorithm and found to have higher accuracy and data reduction.

Minimum Description Length. Minimum Description Length (MDL) principle is first applied in instance-filtering by Cameron–Jones [11]. Like

RMHC, choice of instances indicating the prototype set is encoded in a bit string and a function is designed to calculate the coding cost of a bit string. MDL retains instances which can additionally decrease the coding cost of the current bit string. The classification accuracy of MDL is found to be slightly superior to IB3 with a significant improvement in reduction rate.

Cameron-Jones then adopts the searching heuristic of RMHC [54] to MDL create the Explore algorithm [12]. Explore outperforms the original MDL algorithm in both classification accuracy and data retention rate.

Probabilistic Instance-Based Learning. Tirri et al. introduce a probabilistic instanced-based learning algorithm using predictive distributions of attributes by Bayesian inference to form prototypes instead of considering feature values [59]. Overfitting and sensitivity to choice of distance metric of instance-based learning algorithms can be avoided by this algorithm.

Local Metrics. A local metric for nearest neighbor (LASM) using a prototype set is proposed by Ricci and Avesani [47]. Given a prototype set, which can be randomly selected from the training set, LASM tries to find a system of asymmetric weights for each pair of prototypes. In the learning state, weights of prototypes will be updated by the reinforcement step and the punishment step for correct classification and misclassification of other instances respectively. The learned weights, as well as the prototype set itself, will be used to classify unseen cases. LASM gains a classification accuracy comparable to KNN with optimized k using only 4.5 percent of instances as prototypes on ten real-world data sets.

2.6 Summary

Prototype learning is the technique to learn prototype sets from training instances. With a good prototype learning method, we can substitute the original training set using a relatively smaller prototype set to perform accurate classification. This method is believed to be able to ameliorate traditional instance-based learning algorithms.

There are many approaches to learning distinctive prototype sets including instance-filtering and instance-averaging. Instance-filtering refers to techniques selecting representative prototypes from the original training set. These techniques intend to find a subset of training instances to describe original ones. They can be grouped by the location of instances they retained, namely, 1) retaining border instances, 2) removing border instances and 3) retaining center instances. We have reviewed some previous work for each of the three groups. These methods are found to be fast and simple. Some of them can achieve high generalization accuracy compared with the classical nearest neighbor algorithm and can be flexibly designed and integrated to handle instances in different regions. However, they cannot perform well if ideal instances are not found in original data and are sensitive to noise. Also, their data retention rate is usually high.

Another approach, called instance-abstraction, is to learn artificial prototype set by generalizing or abstracting the training instances. Results from previous work show that this approach can achieve low data retention rate and generate prototypes more representative than original instances. Noise can also be absorbed during the abstraction process. In spite of these strengths, abstraction techniques are usually more complicated and bad prototypes will be formed when merging distant instances, thus leading to poor generalization accuracy.

Considering the strengths and weaknesses of the two prototype learning approaches, we observe that the two approaches are strong in handling different kinds of data distribution and can be advantageous to each other. In the next chapter, we propose two integration methods attempting to unify the strengths of filtering and abstraction to learn representative prototypes for instance-based learning algorithms.

Chapter 3

Integration of Filtering and Abstraction

Different learning methods use different induction techniques and knowledge representation. They behave well in different domains or even different subregions within the same domain space. For this reason, some researchers are interested in building composite learners in which decisions are made by combining different component learners. In instance-based learning, researchers try to improve the generalization accuracy and data retention rate by integrating different prototype learning methods. Wettschereck integrates KNN and hyperrectangle concepts to form a hybrid algorithm [62]. Domingos unifies the strengths of rule induction and instance-based learning developing the well-known RISE system [22]. All these methods are found to be able to outperform any one of the component methods.

After analyzing the strengths and weaknesses of the instance-filtering and instance-abstraction methods, we find that the two methods can be beneficial to each other and can handle different kinds of data distributions.

In this chapter, we propose two approaches integrating the two methods. The first approach is regarded as *incremental integration* in which filtering and abstraction methods are applied incrementally. The second approach is regarded as *concept integration* in which concepts are learned independently and integrated.

3.1 Incremental Integration

In incremental integration, we attempt to integrate the two methods so that they can help each other in an incremental fashion. Through this integration, the two methods can overcome some of their drawbacks when learning prototypes to gain better results in both generalization accuracy and data retention rate. Our early work shows that this integration method can outperform state-of-the-art classification algorithms in generalization accuracy using a few prototypes on a number of benchmark data sets [27, 28, 37, 38].

3.1.1 Motivation

Both the filtering and abstraction methods have their own drawbacks. After investigating the two methods, we find that filtering can help abstraction to overcome some of its drawbacks and vice versa. We describe how they assist each other below.

How Filtering Helps Abstraction

We find that filtering methods can be beneficial to abstraction methods in the following ways:

1. In abstraction, non-prototypical instances will be formed if distant instances, especially for outliers and exceptions, are grouped. To avoid this, we can apply specially designed filtering rules to remove outliers and exceptions first before performing abstraction.

2. Filtering techniques can also be helpful in the middle of or after the abstraction process. We can design a filtering rule to eliminate any non-representative prototypes formed when the abstraction process is in progress.

3. Kibler and Aha observe that abstraction technique may retain misclassified prototypes [30]. These prototypes can also be removed by specifically designed filtering rules during or after the process of abstraction.

4. Some abstraction methods require high computational cost [13]. Filtering rules can be adopted first to eliminate some non-representative instances before abstraction so that computational cost of abstraction can be reduced.

How Abstraction Helps Filtering

Instance-abstraction can assist instance-filtering too.

1. Filtering methods do not conduct generalization on instances and some of them just discard a small portion of instances such as border points [65]. Hence, they usually cannot gain a satisfactory level of data reduction. With the help of instance-abstraction methods, instances in

compact regions can be generalized to single or a few prototype(s) leading to a significant improvement in data retention rate.

2. The representative power of filtering methods will be limited if the good representative instances cannot be found in the original data set. As abstraction approaches summarize the most representative characteristics of similar instances, the generated instances can be more representative than original ones. Therefore, the representation power of filtering approaches can be improved if abstraction technique is suitably integrated.

3. Abstracting instances can also provide a smoother decision boundary for filtering methods and prevent them from overfitting the training set.

4. Some filtering methods are sensitive to mislabeled instances [25]. Abstraction methods can then be used to help filtering methods generalize away those noisy instances before filtering is applied.

We can see that the two methods can assist each other in different ways. Therefore, we attempt to integrate the two methods so that they can complement each other during their learning phases and their performance on both generalization accuracy and data retention rate can be improved. To do so, any effects made by the outcome of one method on the training set should be reflected to the other one. This motivates the development of the incremental integration.

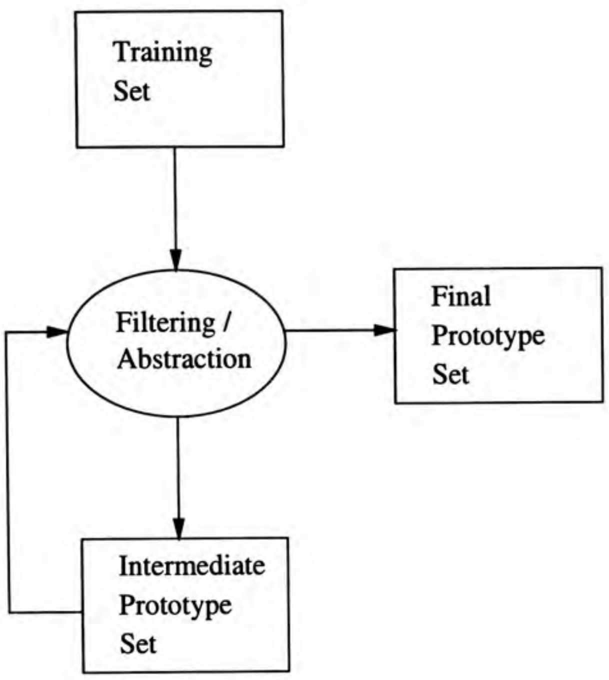

Figure 3.1: The incremental integration method.

3.1.2 The Integration Method

In incremental integration, filtering and abstraction process the training set incrementally. Figure 3.1 shows the overview of our incremental integration method. Given a training set, we first apply filtering or abstraction on it and then the counterpart method is applied on the results of the previous method. For example, we can apply a filtering method first to eliminate all the outliers and then perform abstraction so that the formation of non-prototypical prototypes can be prevented. Alternatively, abstraction can be adopted to reduce the prototype set after performing filtering on the training set. As a result, filtering and abstraction are applied iteratively on the intermediate prototype set. This process continues until the target prototype set is obtained.

3.1.3 Issues

The next challenge is how to design the filtering component and the abstraction component in incremental integration. We observe that there are two main issues needed to be taken into account. The first issue is the type of filtering techniques and the second issue is the filtering granularity. Furthermore, these two factors can interact with each other leading to different behaviors of the integration algorithm.

Type of Filtering Techniques

Abstraction techniques attempt to generalize similar instances in compact regions. Therefore, they work differently on instances in different regions. For example, center and intermediate instances will be generalized to a larger extent compared with border instances. Consequently, the data retention rate of filtering techniques discarding border points will be more significantly improved by abstraction methods than other filtering techniques. To conduct a thorough investigation of this issue, we classify filtering techniques into three types according to the location of instances selected. The first type of filtering methods retains border instances of a cluster. The second type of filtering removes border instances and treats the remaining ones as prototypes. The third type of filtering retains center instances.

Filtering Granularity

The second issue is the filtering granularity. Firstly, filtering can be conducted on the original instances. To do this, one can employ a loose coupling by applying filtering as a preprocessing task and conduct abstraction subsequently. Using this method, filtering techniques can be designed to eliminate instances, such as exceptions, which cannot be effectively handled by abstraction. It helps the generalization of instances in the abstraction process and prevents the formation of non-prototypical instances.

Alternatively, filtering can be conducted on the intermediate prototypes generated when the abstraction process is in progress. We can design a tight coupling technique incorporating filtering into the abstraction process. In the abstraction process, prototypes are generated by abstracting instances. The quality of the generated prototypes differs from each other depending on the locations of instances merged. It is dangerous to merge distant instances where valuable information will be lost or even non-prototypical prototypes will be formed, especially for concave concepts. Filtering on intermediate prototypes can eliminate any of these undesirable prototypes generated during the abstraction process.

In Chapter 4, we propose a prototype learning framework, called *Prototype Generation and Filtering* (PGF), which employs this integration technique. Different variants of PGF, motivated from the considerations of the above two issues, are developed and investigated.

3.2 Concept Integration

The second approach is concept integration. In classification, we attempt to recognize an entity as an instance of a known concept where a concept refers to the intensional representation of a class of instances which are equivalent with respect to the classification goal [4]. To be more concrete, a concept can be regarded as a subset of the instances in a predefined domain [42]. In this way, classification is also known as *concept learning.* In instance-based learning, concepts are represented by the representative prototypes learned by prototype learners. Our proposed concept integration method is designed to combine the concepts separately learned from filtering and abstraction. In other words, it intends to integrate the prototypes derived from the two methods. Through this integration, we attempt to unify the strengths and eliminate the weaknesses of the two learned concepts.

3.2.1 Motivation

Filtering and abstraction methods learn prototypes in different ways leading to different natures of the prototypes learned. They are good at handling instances in different kinds of data distributions or in different regions, but at the same time, weak in processing certain type of instances. After analyzing the strengths and weaknesses of the two methods, we find that they can be complementary to each other in describing different concepts in the following ways.

1. Abstraction methods cannot describe complex decision boundaries effectively leading to low generalization accuracy while some filtering methods are strong in detecting boundary instances so that complex concepts can be represented.

2. Abstraction methods suffer from their inability to represent concave concepts while filtering methods can represent this kind of concept by selecting prototypes from the training instances.

3. Filtering methods do not conduct abstraction of instances so that a

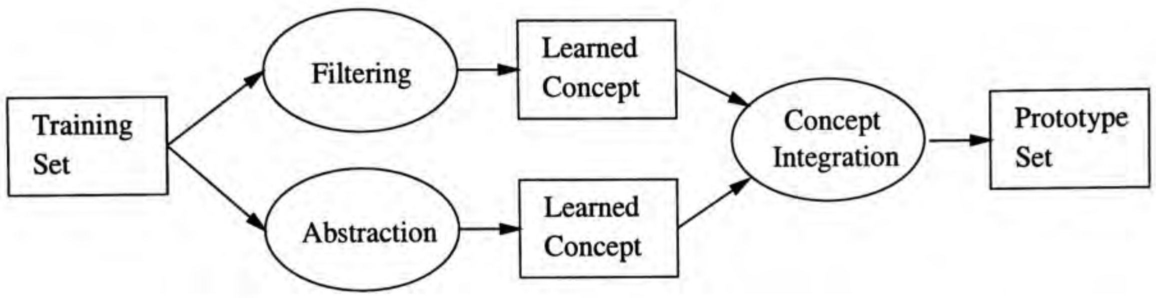

Figure 3.2: The concept integration method.

large amount of instances may be retained to describe even simple concepts. In contrast, abstraction methods are strong in summarizing instances in compact regions to represent simple concepts leading to a low data retention rate.

4. Original instances selected from filtering methods may not be representative enough to describe concepts effectively and efficiently. Using artificial prototypes generated by abstraction methods can, however, summarize the common characteristics of instances for each class and may be more representative than retaining original ones.

If the two concepts learned from each of the two methods are integrated, instances in different regions can be handled effectively and the resulting concept will have a richer representation power for different kinds of data distribution using fewer prototypes. This motivates the development of the concept integration method.

3.2.2 The Integration Method

Figure 3.2 shows the overview of our concept integration method. We first perform filtering and abstraction on training set separately. After that, two concepts, one for each method, will be learned. We develop a method to integrate the two concepts in an attempt to unify the strengths of the filtering and abstraction methods to form the final prototype set.

Unlike incremental integration, the two methods adopted in concept integration learn independently so that they cannot help each other during their learning phases. The concepts learned from the two methods still suffer from their corresponding drawbacks. For example, concept learned from filtering may require a large amount of border prototypes to gain a high generalization accuracy while the one learned from abstraction is not strong in describing complex boundaries resulting in low generalization accuracy. Consequently, the design of the concept integration algorithm becomes important. Before designing the integration algorithm, we have to choose the filtering and abstraction methods first. We discuss the issues for selecting the two methods in concept integration below.

3.2.3 Issues

Concept integration is useful only if the component concepts are learned by methods with different characteristics and strengths. We discuss the considerations of two issues, namely, the prototype characteristic and the concept strength.

Prototype Characteristic

In instance-based learning, concepts are described as prototypes. Different learners represent concepts in different ways, i.e., using different prototypes. In filtering methods, concepts can be described by border, non-border or center instances. In abstraction methods, concepts are learned by abstracting similar instances and so center prototypes are usually generated. As different prototypes are used, the two methods are shown to have good performance, in terms of either generalization accuracy or data retention rate or both, in different situations [27, 38]. This observation suggests that they have expertise in different kinds of data distribution.

In concept integration, we attempt to combine the strengths of the two methods. If the chosen filtering and abstraction methods describe concepts using the same type of prototypes, it will not be useful to integrate the learned concepts. For example, if abstraction is integrated with filtering method retaining center prototypes, two similar concepts described by center prototypes will be learned and there is little use when we combine these two concepts. Therefore, the choice of the two component methods becomes crucial to the performance of the concept integration. As abstraction methods usually learn center prototypes, we should choose filtering methods retaining intermediate or border ones.

Concept Strength

The second issue is the concept strength. Concepts learned from abstraction are strong in summarizing instances in compact regions to gain a low data retention rate. However, they are weak in describing complex boundaries affecting the generalization accuracy. On the contrary, some filtering methods can represent complex boundaries effectively to gain a high generalization accuracy. Nevertheless, too many boundary prototypes will also be retained even in describing a simple concept, such as a linear one, leading to an unnecessarily high data retention rate. Therefore, it seems that the two learned concepts, from abstraction and filtering retaining border instances, can complement to each other. We can use prototypes learned from abstraction to represent simple boundaries so that the data retention rate can be reduced. At the same time, boundary prototypes learned from filtering can be used to describe complex boundaries so that a high generalization accuracy can be obtained.

Summarizing the considerations of these two issues, we observe that the filtering methods suitable for integrated with abstraction should have the following properties:

1. It should retain non-center prototypes so that the prototypes learned will have different characteristics from those learned from the abstraction method.

2. It should be strong in learning complex boundaries as abstraction methods are not good at doing so.

Consequently, a filtering method retaining boundary prototypes should be used in concept learning. We develop a new prototype learner, called *Integrated Concept Prototype Learner* (ICPL), combining abstraction and border-retaining filtering methods based on our concept integration approach. The details of ICPL is described in Chapter 5.

3.3 Difference between Integration Methods and Composite Classifiers

Instead of integrating multiple prototype learners, some researchers develop composite classifiers to combine different learning algorithms [53]. In this section, we intend to point out the differences between the composite classifier approach and our integration methods.

Similar to integration, composite classifiers attempt to combine the strengths of components classifiers as they may have expertise in different regions of instance space. However, the two approaches are different in the following ways.

In our integration methods, one type of concept is the final output which classifies unseen cases. During on-line classification, we only need to process the unseen case with the output concept. However, in composite classifiers, multiple concepts are learned independently and output. In order to classify unseen case, we need to pass it to each learned concepts which may be time consuming.

In incremental integration, component methods are applied on the same training set incrementally so that they can help each other during the learning phase. However, in composite classifers, component methods are independently applied on the training set without interacting with each other. This is similar to our concept integration method. But in our concept integration, the multiple learned concepts are integrated and unified to one final concept while composite classifiers only combine the output of component methods by a combination algorithm such as simple voting [53] and boosting [50, 53].

Chapter 4

The PGF Framework

We develop a prototype learning framework, called Prototype Generation and Filtering (PGF), based on incremental integration of instance-filtering and instance-abstraction methods. We have proposed a simple framework in our previous work and empirical results show that the two methods can improve each other in different ways [27, 28, 37, 38]. In this section, we investigate further and distill out two issues, namely, the type of filtering techniques and the filtering granularity. We propose two PGF algorithms which differ in filtering granularity. Furthermore, different variants of the two algorithms are also designed using different filtering methods.

We first present the first PGF algorithm, called PGF1. Then we describe the two component methods used in it. The second PGF algorithm, called PGF2 employing the same component methods as those in PGF1, is described next. To investigate how the two components help each other in incremental integration, a thorough empirical analysis of different variants of PGF is conducted by comparing them with pure filtering and pure abstraction methods. We also compare them empirically with C4.5 and KNN, as well as the random prototype selection method.

4.1 The PGF1 Algorithm

The first algorithm, called PGF1, conducts filtering on original instances. As shown in Figure 4.1, it first applies an instance-filtering method as specified in Statement 2 as a preprocessing step before prototype abstraction. Statements 3 to 11 in Figure 4.1 is the prototype abstraction component. In

```
1   P = Training Set.
2   FILTER(P).
3   max_score = PROT_SET_SCORE(P).
4   P' = P.
5   while (no. of prototypes in P > no. of class)
6     Find two nearest prototypes, x and y in P.
7     MERGE(P, x, y).
8     If (PROT_SET_SCORE(P) >= max_score)
9       P' = P.
10      max_score = PROT_SET_SCORE(P).
11  Return P'.
```

Figure 4.1: The PGF1 algorithm.

prototype abstraction, grouping of outliers leads to the creation of poor prototypes. These poor prototypes will likely result in degradation in classification accuracy. If outliers or exceptions can be removed before the prototype generation is applied, the result prototypes will have a better quality. Moreover, the computational cost of prototype generation can be significantly reduced as the size of original data set becomes smaller after filtering. Thus, PGF1 essentially conducts filtering on the original instances.

After the algorithm terminates, the output prototype set will be used for classifying unseen cases. Suppose an unseen case needs to be classified, its distance between all the learned prototypes are calculated to find the nearest prototype. The class label of the nearest prototype is then assigned to the unseen case.

4.1.1 Instance-Filtering Component

Different types of filtering methods target at retaining instances in different locations leading to different behaviors when integrated with abstraction techniques [37]. We investigate three filtering techniques in our PGF framework.

Removing Border Instances. The first one is the ENN method introduced by Wilson [65]. This method discards instances misclassified by their k nearest neighbors. As outliers and noise are seldom classified correctly by their neighbors, they will usually be removed. This method also removes border instances as they usually have neighbors of different classes. It retains intermediate and center instances.

Retaining Border Instances. The second filtering rule is called RT3 proposed by [69]. Initially, each instance is considered as a prototype. ENN is applied first to filter out noisy instances. Then the presentation order of instances is sorted in descending order by the distance of an instance to its nearest unlike neighbor. It ensures instances further away from decision borders are processed first. It then removes an instance if most of its *associates*, instances in the training set having it as one of their k nearest neighbors, are classified correctly without it. A detailed analysis on the concept of *associate* is provided in Chapter 5. Noisy instances are usually removed as they can hardly classify their associates correctly while border instances will be retained as their associates tend to be classified correctly with their contribution in KNN classification.

Retaining Center Instances. The third filtering technique, called ACC developed by us, tries to find center instances of compact regions by considering the classification performance of each prototype in the prototype set. Each instance in the training set is classified by its nearest neighbor. If it is correctly classified, classification accuracy of its nearest neighbor will be increased. After classifying all the training instances, ACC discards instances with accuracy lower than a certain threshold Q. As center instances are usually neighbors of other instances with the same class, they usually gain high accuracy and thus being retained by ACC. Noisy and non-representative instances such as outliers and exceptions, will be effectively removed as they usually have lower accuracy.

4.1.2 Instance-Abstraction Component

Our instance-abstraction method is based on an agglomerative clustering technique. A prototype is represented by a set of data instances together with the sufficient statistics, namely, the total number, mean and standard deviation of the instances. In Figure 4.1, from Statement 3 to 11 is the pseudo-code of the instance abstraction component, called ABS. It learns prototypes in a top-down fashion. At the beginning, each instance is considered as a prototype. Let P be the current prototype set. At each iteration, two prototypes with the shortest distance are merged to form a new prototype. The new prototype essentially contains all those instances in the original two prototypes. We calculate the mean and standard deviation of all the instances in the new prototype as the generalized representation of that prototype. After this merging process, the current prototype set P is updated. The majority class of all the instances in the new prototype becomes the class of it. The prototype set is then evaluated by a prototype set score function (PROT_SET_SCORE) to predict the quality of the prototypes. If the prototype set is good, it is stored. The merging process continues until the number of prototypes decreases to the number of classes in the data set. This method can return the prototype set which attains the highest value in the prototype set score function. The class of all the instances in a prototype is also stored.

The Prototype Set Score Function

The choice of prototype set score function is crucial to the abstraction method. We have tried two score functions. The first one is called the Calinski and Harabasz index which was tested by Milligan and Cooper with 29 other different score function to determine the number of clusters in a hierarchical clustering method [41]. It is observed that it performs the best among all of the tested methods. It measures the degree of isolation and internal coherence of clusters. In essence, it attempts to generate high quality clusters.

The second score function is based on classification accuracy. As our objective is to learn prototypes to classify unlabeled instance, classification accuracy on unseen cases is a reasonable indicator to predict the quality of prototypes. We divide the training set into a sub-training set and a tuning set. Prototypes are generated using the sub-training set. The tuning set is used for calculating the prototype set score using classification accuracy. The prototype set with the highest classification accuracy is the output.

Empirical results show that the second method performs better than the first one in terms of both generalization accuracy and data retention rate. It seems that prototypes formed from clusters with high degree of isolation and internal coherence may not be suitable to performance classification. Therefore, it is adopted as the score function in the abstraction component.

Distance Measure

To measure the distance between instances with continuous and nominal feature types, we adopt a heterogeneous distance function similar to the one proposed by [69]. We first normalize all the continuous attributes by their feature ranges. Euclidean distance is employed to calculate distances between continuous feature values whereas a simplified version of Value Difference Metric (vdm) [56] is used to handle nominal features. The distance function vdm_i for feature i is defined as:

$$vdm_i(a,b) = \sum_{c=1}^{C} \Big(\frac{N(i,a,c)}{N(i,a)} - \frac{N(i,b,c)}{N(i,b)} \Big)^2$$

where $N(i,a)$ is the number of occurrences of instances with value a for feature i and $N(i,a,c)$ is the number of occurrences of instances with value a for feature i and class label c. C is the total number of classes in the data set. Our distance measure, $Dist(\boldsymbol{x},\ \boldsymbol{y})$ for two prototypes $\boldsymbol{x} = (x_1, ..., x_n)$ and $\boldsymbol{y} = (y_1, ..., y_n)$, is defined as:

$$Dist(\boldsymbol{x},\ \boldsymbol{y}) = \sqrt{\sum_{i=0}^{n} dist_i^2(x_i, y_i)}$$

where n is the number of attributes, and

$$dist_i(x_i, y_i) = \begin{cases} vdm_i(x_i, y_i) & \text{for nominal features} \\ x - y & \text{for continuous features.} \end{cases}$$

We find that *vdm* and Euclidean distance have different ranges of values leading to different weights for each feature in our distance measure. To ensure an even contribution of each feature, we first calculate the maximum distance of each feature. For continuous feature, the maximum distance is the range of the feature. For discrete feature, the maximum value of *vdm* among all the possible value pairs of that feature becomes its maximum distance. Then we normalize *dist* for each feature by its maximum distance.

Entropy

In abstraction, we attempt to find common characteristics for each class. Therefore, prototypes will be more representative if only homogeneous instances are grouped. To this end, some previous works just split the training set by each class and learn prototypes for each of them separately [20]. These methods guarantee fully homogeneous prototypes but the entire data distribution is distorted. Besides, the advantage of the abstraction method to generalize away mislabeled instances is disabled. In view of this, we introduce a component, called *entropy*, into our distance measure. The entropy, *Ent($\boldsymbol{x}$)*, of a prototype $\boldsymbol{x}$ is related to the class distribution of the instances contained in the prototype. It is defined as:

$$Ent(\boldsymbol{x}) = -\sum_{i=1}^{c} R(\boldsymbol{x}, i) \log R(\boldsymbol{x}, i)$$

where $R(\boldsymbol{x}, i)$ is the relative frequency of the occurrence of the class label i in the prototype $\boldsymbol{x}$. When two prototypes $\boldsymbol{x}$ and $\boldsymbol{y}$ are considered to merge, the entropy distance between $\boldsymbol{x}$ and $\boldsymbol{y}$, *E($\boldsymbol{x}$, $\boldsymbol{y}$)*, is defined as:

$$E(\boldsymbol{x}, \boldsymbol{y}) = Ent(\boldsymbol{z})$$

where $\boldsymbol{z}$ is a hypothetic prototype generated by merging $\boldsymbol{x}$ and $\boldsymbol{y}$. If a small entropy is obtained, most instances in the merged prototypes are of the same class. As the entropy is of range from 0 to 1, we normalize *Dist* by the distance calculated from the maximum distance for each feature. After the two components are calculated, a parameter α $(0 \leq \alpha \leq 1)$ is then used to control the weight of their contributions. The final distance function $FDist$ of PGF is:

$$FDist(\boldsymbol{x}, \boldsymbol{y}) = \alpha Dist(\boldsymbol{x}, \boldsymbol{y}) + (1 - \alpha)E(\boldsymbol{x}, \boldsymbol{y})$$

This distance measure favors the merging of homogeneous instances while preserving the original data distribution.

4.2 The PGF2 Algorithm

The second algorithm, called PGF2, differs from PGF1 in filtering granularity. PGF2 adopts the same filtering and abstraction components as the ones in PGF1. Instead of filtering the training instances, it conducts filtering on the intermediate prototypes in the process of prototype generation. As shown in Figure 4.2, the filtering and the abstraction methods are more tightly coupled in PGF2 compared with PGF1. After two prototypes are merged to form a new intermediate prototype, we conduct filtering on the current prototype set. The procedure " FILTER(*temp*). " conducts the filtering.

Unlike PGF1 which filters on the original instances, PGF2 performs filtering on the prototype set. A prototype set usually contains intermediate prototypes and original instances. The purpose of filtering is to discard less representative prototypes and outliers which can further increase the data

```
P = Training Set.
max_score = PROT_SET_SCORE(P).
P' = P.
while (no. of prototypes in P > no. of class)
  Find two nearest prototypes, x and y in P.
  MERGE(P, x, y).
  temp = P.
  FILTER(temp).
  If (PROT_SET_SCORE(temp) >= max_score)
    P' = temp.
    max_score = PROT_SET_SCORE(temp).
Return P'.
```

Figure 4.2: The PGF2 algorithm.

reduction rate. On top of this, filtering can also remove noisy prototypes or instances, thus improving the classification accuracy.

4.3 Empirical Analysis

4.3.1 Experimental Setup

We have conducted a series of experiments to investigate the performance of our PGF framework. Thirty-five real-world benchmark data sets from the widely used UCI Repository [8] were tested in the experiments. Table 4.1 shows the data sets and their corresponding codes used in this thesis. These data sets are collected from different real-world application in various domains, such as the city-cycle fuel consumption (Am), Wisconsin breast cancer (Bc) and the famous iris plant database (Ir). The detailed information for each data set is provided in Appendix A. We basically use all instances

Data Set	Code	Data Set	Code	Data Set	Code
Automobile	Ab	Letter	Le	Shuttle	Sh
Auto-Mpg	Am	Liver	Li	Sonar	Sn
Audiology	Au	Monk-1	M1	Soyabean	Sb
Balance-Scale	Ba	Monk-2	M2	Tic-Tac-Toe	Tt
Breast-Cancer-W	Bc	Monk-3	M3	Voting	Vo
Car-Evaluation	Ca	Mushroom	Mu	Vowel	Vw
Credit Screening	Cs	New-Thyroid	Ne	Wdbc	Wd
Ecoli	Ec	Nursery	Nu	Wine	Wi
Glass	Gl	Optdigits	Op	Wpbc	Wp
Hepatitis	He	Pendigits	Pe	Yeast	Ye
Ionosphere	Io	Pima-Indians-Diabetes	Pi	Zoo	Zo
Iris	Ir	Segmentation	Se		

Table 4.1: Data sets and their codes.

in every data set in our experiments except seven data sets which we reduce it by randomly selecting 1,500 instances. These seven data sets include Le, Mu, Nu, Op, Pe, Se and Sh.

For each data set, we randomly partitioned the data into ten even portions. Ten trials derived from 10-fold cross-validation were conducted for every set of experiments. The mean the generalization accuracy and data retention rate of 10-fold cross-validation were obtained for each data set, where

$$\textit{generalization accuracy} = \frac{\text{No. of correct classifications on testing instances}}{\text{No. of testing instances}}$$

and

$$\textit{data retention rate} = \frac{\text{No. of prototypes learned}}{\text{No. of training instances}}$$

Note that higher classification accuracy and smaller data retention rate imply better performance.

In the first set of experiments, we investigate the performance of different variants of our PGF framework. Each variant is constructed by integrating a particular PGF method with a filtering algorithm. PGF1-ENN, PGF1-RT3 and PGF1-ACC refer to the integration of abstraction with ENN, RT3 and ACC filtering methods respectively using PGF1 algorithm. PGF2-ENN, PGF2-RT3 and PGF2-ACC have the similar interpretation. We have also conducted some trials on pure filtering and pure abstraction methods using the same data partitions so that comparative analysis can be conducted. In the second set of experiments, we compare our algorithm with existing learning algorithms, namely, C4.5 and KNN, as well as a random prototype learner.

4.3.2 Results of PGF Algorithms

Tables 4.2 shows the average classification accuracy and data retention rate of 10-fold cross-validation across 35 data sets for different variants of the PGF. A range of parameters for these algorithms were tested and the best performance of each algorithm is presented. We observe that the performance of PGF remains quite stable across different parameters.

We also obtained the performance of pure filtering and pure abstraction methods so that comparative analysis can be conducted. Table 4.3 shows the average classification accuracy and data retention rate of pure instance-filtering and instance-abstraction (ABS) methods. The detailed performance of each algorithm for each individual data set can be found in the Appendix B. To investigate the behavior of integrating the two methods, for each variant of PGF, we first compare it with the pure filtering method used in the integration and followed by the pure abstraction method. We first analyze the behavior of PGF1 and followed by PGF2.

	PGF1					
	PGF1-ENN		**PGF1-RT3**		**PGF1-ACC**	
	acc.	size	acc.	size	acc.	size
Average	0.846	0.163	0.834	0.066	0.798	0.055
Better	18-16	27-8	24-11	8-27	33-2	6-29
Wilcoxon	56.60	99.50	99.50	-99.50	99.50	-99.50

	PGF2					
	PGF2-ENN		**PGF2-RT3**		**PGF2-ACC**	
	acc.	size	acc.	size	acc.	size
Average	0.851	0.300	0.837	0.085	0.848	0.103
Better	14-20	35-0	21-14	11-23	0-0	0-0
Wilcoxon	-86.89	99.50	97.89	-99.07	50.00	-50.00

Table 4.2: The average classification accuracy (acc.) and data retention rate (size) of 10-fold cross-validation across 35 real-world data sets for different variants of PGF1 and PGF2.

Pure Filtering						**Pure Abstraction**	
ENN		**RT3**		**ACC**		**ABS**	
acc.	size	acc.	size	acc.	size	acc.	size
0.865	0.871	0.855	0.142	0.800	0.120	0.858	0.216

Table 4.3: The average classification accuracy (acc.) and data retention rate (size) of 10-fold cross-validation across 35 real-world data sets for pure filtering methods and pure abstraction method.

4.3.3 Analysis of PGF1

PGF1-ENN. We investigate ENN and PGF1-ENN to analyze how the abstraction method can help ENN in PGF1. From Tables 4.2 and 4.3, it is found that the data retention rate of ENN is dramatically improved from 87.1% to 16.3% with about 2% degradation in classification accuracy. ENN retains instances which can be correctly classified by their k nearest neighbors. We can imagine that if most of the instances are closely and homogeneously packed, a large portion of data will be retained as they are usually correctly classified. This accounts for the large data retention rate in ENN. On the contrary, our prototype abstraction method is strong in generalizing data sets with this kind of structure. Instances in closely packed regions will be generalized to a few representative prototypes resulting in significant reduction in data retention rate.

When comparing PGF1-ENN with ABS, we find that ENN can assist the abstraction method in PGF1 too. If ENN is performed before abstraction, noise, outliers and exceptions can be removed first. The removal of these instances can avoid the formation of non-representative prototypes in abstraction. Furthermore, a smoother decision boundary can also be obtained by the removal of border instances. It may help the generalization of instances in abstraction. We can see from Tables 4.2 and 4.3 that the data retention rate of ABS is improved from 21.6% to 16.3% while keeping a similar classification accuracy.

PGF1-RT3. When comparing PGF1-RT3 with RT3, we find that the abstraction method reduces the average data retention rate of RT3 from 14.2%

to 6.6% with a 2.5% decrease in classification accuracy. RT3 retains border instances and discards center and intermediate ones. If abstraction technique is applied on those remaining border instances, the structure of the border may be severely distorted resulting in large degradation in classification accuracy. However, as our ABS algorithm applies classification accuracy as the prototype set evaluation function, a prototype set with such kind of distorted boundaries will be eliminated. The above results suggest that our abstraction technique can generalize the remaining border instances without severely reducing the representative power of them.

In PGF1, RT3 is found to be beneficial to ABS by comparing PGF1-RT3 with ABS. The data retention rate of ABS is significantly improved from 21.6% to 6.6%. RT3 retains border instances only. The elimination of center instances, noise and outliers results in the improvement in data retention rate. However, with the absence of center instances, the representative power of generalized prototypes formed in abstraction will be decreased. It accounts for the 2.8% degradation in classification accuracy.

PGF1-ACC. ACC retains instances with classification accuracy higher than a certain threshold. As center instances usually gain high accuracy, they will be retained. When comparing ABS and PGF1-ACC, we find that data retention rate of ABS is improved from 21.6% to 5.5%. Despite the significant improvement in data retention rate, the classification of ABS is degraded from 85.8% to 79.8%. We know that ABS discovers representative instances by generalizing the common characteristics of similar instances. However, in PGF1-ACC, about 90% of instances are discarded by ACC before ABS is applied. Therefore the prototypes generated in abstraction will be less representative leading to the degradation in classification accuracy. We suggest that filtering methods retaining center instances should not be used in PGF1 if classification accuracy is the main concern.

On the contrary, ABS can help ACC in PGF1. When comparing PGF1-ACC with ABS, we can see that the data retention rate of ACC is improved from 12.0% to 5.5% while maintaining similar classification accuracy. It shows that instances selected by ACC are further refined by ABS to form more representative prototypes.

4.3.4 Analysis of PGF2

PGF2-ENN. We investigate how abstraction technique benefits to ENN in PGF2. According to the results of PGF2-ENN and ENN, the data retention rate of ENN is significantly improved by the abstraction technique, from 87.1% to 30.0%, with only little degradation in classification accuracy. ENN removes border instances only so that a low data reduction rate is yielded. However, our abstraction technique can generalize similar instances in compact regions using a few or single abstracted prototypes. Therefore, if instances are generalized using abstraction before, ENN can be performed on a relatively smaller set of generalized prototypes. The outcome is significant improvement in data retention rate of ENN without a large degradation in classification accuracy.

We now compare PGF2-ENN with ABS. As ENN discards noise and exceptions, any non-representative and mislabeled prototypes formed in abstraction will be removed. However, after abstraction, clusters of similar instances of the same class will be grouped to form generalized prototypes and neighbors of these prototypes may probably be abstracted prototypes of different classes. Then these representative prototypes will be discarded by ENN as they are not correctly classified by their k nearest neighbors leading to degradation in classification accuracy. However, this undesirable effect is eliminated in our PGF framework. As classification accuracy is used as the prototype set score function in PGF, prototype sets with low accuracy will not be returned as output. From the above tables, we can see that ABS gains almost the same level of classification accuracy when integrated with ENN in PGF2. It is interesting to see that ABS retains more prototypes, from 21.6% to 30.0%, when integrated with ENN. Formation of isolated and representative prototypes is usually done at later stages in the abstraction process. If ENN is applied during these stages, useful prototypes will be discarded. To avoid degradation in classification accuracy, PGF will select prototype set formed in earlier abstraction stages. Therefore, the number of prototypes formed is even larger than pure prototype abstraction method. These results suggest that filtering techniques removing border instances cannot improve the performance of the abstraction technique in PGF2.

PGF2-RT3. In PGF2-RT3, RT3 is applied in the abstraction process. During abstraction process, similar instances, including border instances, are merged to form artificial prototypes which are as representative as the original instances. Therefore, RT3 can retain fewer prototypes to represent the decision boundaries. Compared with RT3, PGF2-RT3 stores 5.7% fewer of the total instances with a 2.1% degradation in classification accuracy.

When comparing PGF2-RT3 with ABS, we find that the data retention rate of ABS is improved from 21.6% to 8.5% without large degradation in classification accuracy. It is because RT3 can eliminate non-representative prototypes formed by ABS effectively in PGF2. Besides, RT3 also further reduces the data retention rate of ABS by removing center prototypes which usually do not affect the decision boundaries. These reasons account for the fact that the removal of these kinds of prototypes do not result in a large decrease in classification accuracy in PGF2.

PGF2-ACC. We first investigate how filtering technique assists the abstraction component. From the results of PGF2-ACC and ABS, we can see that the data retention rate of ABS is improved from 21.6% to 10.3% with about 1% decrease in classification accuracy when it is integrated with ACC using PGF2. ACC retains instances with accuracy higher than a certain threshold. Therefore, highly representative instances will be retained and noise and exceptions can be discarded. If we apply ACC in the process of abstraction, representative generalized prototypes will be selected and less representative and mislabeled ones will be discarded. These reasons account for the improvement in data reduction rate in PGF2-ACC with only a little degradation in classification accuracy.

For the filtering component ACC in PGF2, ABS can also help. The results of PGF2-ACC and ACC show that ACC improves its classification accuracy from 80.0% to 84.8% using even 14.2% fewer prototypes when integrated with ABS. In abstraction, the most common characteristics of similar instances are found by generalization of those instances. Therefore, the representative power of those generalized prototypes is often higher than original instances in the data set. When these highly representative prototypes are selected, the classification accuracy of filtering technique can be improved as shown from the experiment results.

4.3.5 Overall Behavior of PGF

In conclusion, we find that filtering techniques and abstraction techniques are beneficial to each other in our PGF framework. In PGF1, filtering techniques can remove noisy instances and outliers. It avoids the formation of non-representative prototypes in abstraction techniques. Also, as different filtering techniques remove instances in different regions, we can find different improvements in data retention rate when comparing different variants of PGF1 with pure abstraction method. Empirical results show that the filtering technique discarding border instances (ENN) seems to be most beneficial when integrated with the abstraction technique as it significantly reduces the data retention rate of abstraction method while maintaining similar classification accuracy. Though we find that the filtering technique retaining border instances (RT3) obtains similar benefits from abstraction technique in PGF1, it may not work equally well if other abstraction techniques are used. It is because abstraction of border instances often leads to severe destruction of class boundaries and such prototype sets may be returned as output if classification accuracy is not used in the prototype set evaluation. The filtering technique retaining center instances (ACC) is found not suitable in PGF1 as it reduces the representative power of generated prototypes in the abstraction method. On the other hand, the abstraction method also helps filtering techniques to improve their data reduction rates effectively in PGF1. The three filtering techniques achieve significant improvements in data reduction when comparing with their PGF1 variants.

In PGF2, we find that both filtering techniques removing border instances (ENN) and retaining border instances (RT3) perform better by reducing their data retention rate while maintaining similar classification accuracy when integrated with ABS in PGF2. For the filtering technique retaining center instances (ACC), in addition to the data retention rate, the classification accuracy is also significantly improved in PGF2. It seems to be the most suitable filtering technique to integrate with ABS in PGF2. On the other hand, ABS, cannot be beneficial from all the filtering techniques. The data retention rate of ABS is significantly reduced by filtering techniques retaining border (RT3) and center (ACC) instances without severely sacrificing the classification accuracy. However, for the filtering technique removing border instances (ENN), we find that both the data retention rate and classification accuracy of ABS are degraded in PGF.

PGF2-ACC vs. Other PGF Variants

Among the six variants of PGF algorithms, we find that PGF2-ACC achieves the best mix of generalization accuracy and data retention rate. In order to make rigorous comparisons between PGF2-ACC and other variants, we conducted comparative analysis on their performance. These analyses are shown in the last two rows of Table 4.2.

The first analysis counts the number of data sets in which PGF2-ACC performs better and worse than each of the PGF variants on generalization accuracy and data retention rate. It is shown in the row labeled as "Better". For example, in the accuracy columns for PGF1-ENN, "18-16" is reported in the "Better" row. The result indicates that PGF2-ACC achieves a better performance for generalization accuracy on 18 data sets while gaining worse performance on the remaining 16. Note that there can be a tie between PGF2-ACC and other variants. Thus, the sum of the numbers of "better" and "worse" data sets may not be equal to the number of data sets used.

In the second analysis, we adopt a one-tailed Wilcoxon Signed Ranks test [15] to compare PGF2-ACC with other variants by a more formal statistical analysis. This test indicates whether the differences in accuracy and data retention rate on the entire set of classification tasks are statistically significant or not. The row "Wilcoxon" presents the confidence level of a difference on the performance. A positive confidence indicates that the performance of PGF2-ACC is better than that of other methods while a negative one means a worse performance for PGF2-ACC. If the magnitude of a confidence level is larger than 90%, the performance of the two variants can be regarded as significant different. For example, under the variant PGF1-ENN, the confidence value of accuracy is 56.60% showing that PGF2-ACC is believed to gain a higher generalization accuracy at a low confidence level.

From Table 4.2, we find that PGF2-ACC gains similar generalization accuracy than PGF1-ENN does. However, the data retention rate of PGF2-ACC is significantly lower than that of PGF1-ENN. For PGF1-RT3 and PGF1-ACC, PGF2-ACC is believed to retain a significantly larger amount of prototypes. However, PGF2-ACC significantly outperforms these two variants on generalization accuracy. These results suggest that PGF2-ACC is better than PGF1 variants.

As for PGF2-ENN, PGF2-ACC attains a lower generalization accuracy at a confidence level of 87%. It suggests that PGF2-ENN achieves a significantly higher generalization accuracy. However, PGF2-ACC retains much fewer prototypes to gain the comparable generalization accuracy. Comparative results shows that PGF2-ACC learns fewer prototypes than PGF2-ENN does in the 35 data sets and Wilcoxon test also suggests that PGF2-ACC achieves a significantly lower data retention rate. Comparing with PGF2-RT3, PGF2-ACC is found to obtain a significantly higher data retention rate. However, it gains a significantly higher generalization accuracy according to Wilcoxon test.

These comparative analyses suggest that PGF2-ACC is the best PGF variant. In the next set of experiment, PGF2-ACC is chosen to compare with other classification algorithms.

4.3.6 Comparisons with Other Approaches

In the second set of experiments, we compare PGF (i.e. PGF2-ACC) with existing classification algorithms, namely, C4.5 and KNN. Table 4.4 shows the average classification accuracy and data retention rate of 10-fold cross-validation of these algorithms on the same 35 data sets. The two comparative analyses used in comparing PGF variants are also used in the set of experiments and the results are shown in the last two rows of the table. In KNN, a range of k (k = 1,3,5,7,9,11,13,15,20) is tested and the best results are reported.

PGF2-ACC vs. C4.5

PGF2-ACC performs slightly better than C4.5 in the average classification accuracy across all the data sets. Based on Wilcoxon test, C4.5 is observed to

	PGF2-ACC		C4.5	KNN
Data	accuracy	size	accuracy	accuracy
Ab	0.586 (0.163)	0.202	0.794 (0.156)	0.766 (0.076)
Am	0.786 (0.076)	0.101	0.776 (0.056)	0.771 (0.082)
Au	0.672 (0.135)	0.130	0.756 (0.064)	0.761 (0.102)
Ba	0.853 (0.041)	0.012	0.792 (0.066)	0.775 (0.066)
Bc	0.963 (0.037)	0.026	0.939 (0.041)	0.960 (0.014)
Ca	0.935 (0.019)	0.176	0.928 (0.012)	0.956 (0.016)
Cs	0.842 (0.041)	0.019	0.832 (0.054)	0.807 (0.047)
Ec	0.833 (0.074)	0.117	0.822 (0.060)	0.822 (0.095)
Gl	0.649 (0.213)	0.051	0.666 (0.083)	0.681 (0.300)
He	0.818 (0.097)	0.031	0.773 (0.182)	0.805 (0.186)
Io	0.874 (0.074)	0.035	0.900 (0.032)	0.866 (0.058)
Ir	0.933 (0.104)	0.073	0.953 (0.063)	0.947 (0.043)
Le	0.701 (0.059)	0.206	0.692 (0.043)	0.810 (0.034)
Li	0.585 (0.119)	0.072	0.642 (0.054)	0.632 (0.089)
M1	0.939 (0.082)	0.250	0.960 (0.084)	0.969 (0.039)
M2	0.951 (0.062)	0.120	0.625 (0.079)	0.993 (0.016)
M3	0.950 (0.081)	0.093	0.988 (0.033)	0.955 (0.045)
Mu	0.995 (0.008)	0.010	0.997 (0.006)	0.999 (0.002)
Ne	0.925 (0.075)	0.059	0.921 (0.081)	0.972 (0.031)
Nu	0.853 (0.035)	0.144	0.909 (0.018)	0.863 (0.024)
Op	0.946 (0.032)	0.114	0.824 (0.029)	0.962 (0.045)
Pe	0.972 (0.028)	0.104	0.914 (0.015)	0.987 (0.009)
Pi	0.715 (0.078)	0.046	0.694 (0.085)	0.706 (0.114)
Se	0.952 (0.015)	0.143	0.951 (0.015)	0.967 (0.016)
Sh	0.985 (0.042)	0.142	0.989 (0.045)	0.987 (0.050)
Sn	0.789 (0.090)	0.131	0.706 (0.094)	0.876 (0.152)
Sb	0.861 (0.068)	0.156	0.930 (0.034)	0.908 (0.053)
Tt	0.865 (0.061)	0.197	0.862 (0.036)	0.914 (0.027)
Vo	0.926 (0.047)	0.061	0.960 (0.021)	0.935 (0.031)
Vw	0.944 (0.039)	0.210	0.779 (0.046)	0.992 (0.016)
Wd	0.942 (0.053)	0.092	0.944 (0.031)	0.945 (0.028)
Wi	0.949 (0.050)	0.086	0.888 (0.081)	0.954 (0.054)
Wp	0.748 (0.120)	0.015	0.676 (0.168)	0.701 (0.108)
Ye	0.523 (0.056)	0.103	0.545 (0.049)	0.524 (0.054)
Zo	0.920 (0.101)	0.085	0.926 (0.101)	0.970 (0.034)
Average	0.848	0.103	0.836	0.870
Better	0-0	0-0	19-16	9-26
Wilcoxon	50.00	-50.00	73.13	-99.50

Table 4.4: The average classification accuracy (accuracy) and data retention rate (size) of 10-fold cross-validation for PGF2-ACC, C4.5 and KNN. The standard deviation of classification accuracy is given inside the bracket.

obtain a better generalization accuracy than C4.5 does, however, without a significantly high confidence. Among the 35 data sets, PGF2-ACC performs better on generalization accuracy on 19 of them while C4.5 gains better results on the remaining 16. On some data sets, such as M2, Op and Vw, PGF2-ACC achieves a significantly higher accuracy. However, C4.5 performs significantly better than PGF2-ACC on some other data sets such as Ab and Sb. These results suggests that the two methods are strong in processing different kinds of data distribution.

PGF2-ACC vs. KNN

When compared with KNN, PGF2-ACC gains a 2.5% lower generalization accuracy. KNN gains better accuracy on 26 of the 35 data sets. Wilcoxon test also suggests that KNN significantly outperforms PGF2-ACC on generalization accuracy. However, contrasted with KNN, PGF2-ACC only retains 10% of the total instances to obtain a high accuracy. It can compensate the inferior generalization accuracy of PGF2-ACC.

PGF2-ACC vs. Random Prototype Selection

We also compare our PGF algorithm with the random prototype selection method in which prototypes are selected randomly from the training set. A wide range of percentage storage of the random method was tested. Figure 4.3 shows the generalization accuracy versus the data retention rate for different PGF variants and the random prototype selection method. We find that all the PGF variants outperform the random method showing the learning ability of PGFs. PGF2-ACC is found to perform the best since it achieves about 13% higher generalization accuracy than the random method. This further confirms the effectiveness of the integration method in classification.

4.4 Time Complexity

The time complexity of prototype learners is highly dependently on the number of distance computation in the algorithms. So we focus on the amount of distance calculation in this section. We first discuss the computational complexity of the three filtering methods and then the abstraction component. The demand on computational time in the two PGF algorithms will then be investigated. Let n be the number of training instances and m be the time required to calculate a distance between two instances.

4.4.1 Filtering Components

ENN. In ENN, each instance is examined using their nearest neighbors. To find the nearest neighbors of a training instance, we have to calculate the distance between that instance and all other training instances. The time required to compute all the distance of an instance is $O(m(n-1))$. For n instances, ENN requires $n * (O(m(n-1)) = O(n^2m)$ time to process all of them.

RT3. ENN is first applied in RT3. Therefore, RT3 also takes $O(n^2m)$ time in the initial state to calculate all pair-wise distance. Each instance is then processed once by the algorithm to examine whether it should be removed. This process does not involve the calculation of distance between instances

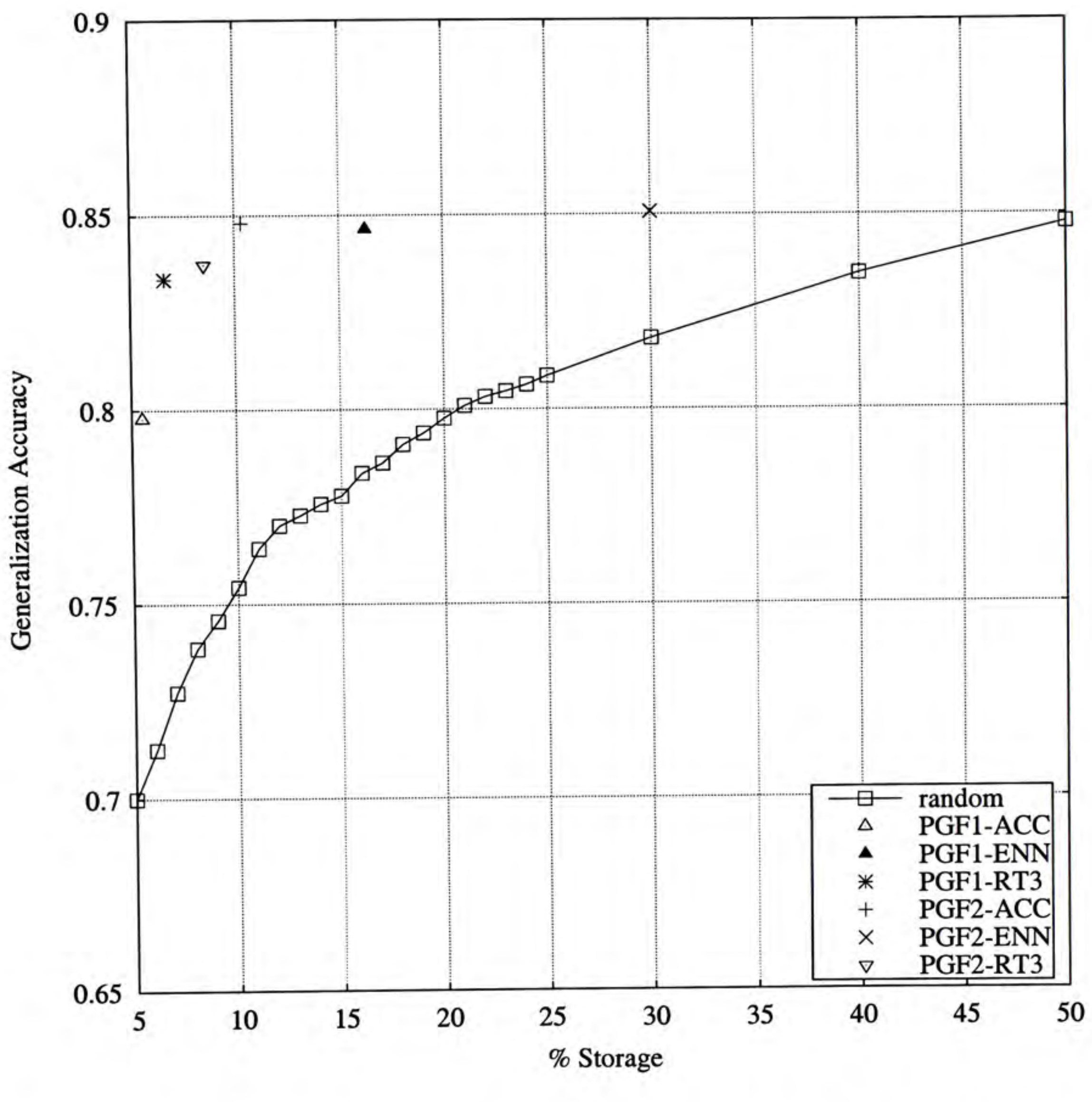

Figure 4.3: Generalization accuracy for PGF variants vs. the random prototype selection method.

as all the pair-wise distance is computed in ENN. Thus, the time complexity of RT3 is also $O(n^2m)$.

ACC. ACC stores instances with accuracy higher than a certain threshold. Each instance in the training set is classified by it nearest neighbors. This process is identical to the core step in ENN and also takes $O(n^2m)$ time to complete.

4.4.2 Abstraction Component

Our ABS algorithm adopts agglomerative clustering technique to perform abstraction. This technique merges two nearest prototypes hierarchically until the number of prototypes is reduced to the number of class in the target domain. It requires the computation of all the pair-wise distance after merging two prototypes. Therefore, the time required in ABS is $\sum_{i=c}^{n} O(i^2m)$ in distance computation, where c is the number of class. The time complexity of ABS is $O(n^3m)$.

4.4.3 PGF Algorithms

PGF1. In PGF1, filtering is applied before abstraction and the computation time of filtering technique is $O(n^2m)$. As for abstraction, the time complexity depends on the number instances retained after filtering. Considering the worst case, we can regard all the instances are selected and it takes $O(n^3m)$ time for ABS to finish. Consequently, PGF1 requires $O(n^2m)+O(n^3m) = O(n^3m)$ time.

PGF2. In PGF2, filtering is performed in middle of the process of abstraction. After each prototype merging in abstraction, filtering is employed and the time needed to conduct all the filtering processes is $\sum_{i=c}^{n} O(i^2m) = O(n^3m)$, where c is the number of class. Totally, the time complexity of PGF2 is $O(n^3m) + O(n^3m) = O(n^3m)$.

Our PGF2 algorithm takes $O(n^3m)$ time to learn prototypes. As n grows large, a significant amount of computation resources is required. Though the computational complexity can be reduced by scaling up the nearest neighbor search to reduce the number of distance calculation, we need a faster algorithm to handle large data set. The most complicated step in PGF is the abstraction process and we intend to design a faster abstraction algorithm in Chapter 5.

4.5 Summary

We have presented a new prototype learning method, called *Prototype Generation and Filtering* (PGF), which integrates instance-filtering and instance-abstraction techniques based on incremental integration method. There are two issues affecting the performance of the integration method, namely, the type of filtering methods and the filtering granularity. We develop two PGF algorithms taking these two issues into account. We analyze the generalization accuracy and the data retention rate of different variants of PGF on 35 real-world benchmark data sets. Through comparative analysis, we find that all the variants do not perform equally well. We find that most of the integrated algorithms gain a significant improvement on data retention rate with equal or slight degradation in generalization accuracy when comparing with their corresponding component methods. It shows that the two methods can be beneficial to each other in prototype learning using incremental integration. Empirical results also suggest our PGF algorithm achieves comparable results to state-of-the-art algorithms, namely, C4.5 and KNN. We also investigate that under what circumstances the incremental integration can be beneficial to the two methods. We find that PGF2-ACC performs the best on average. PGF2-ACC is comprised of a filtering method retaining center instances integrated with the abstraction method with a high filtering granularity.

In the next chapter, our second algorithm, called *Integrated Concept Prototype Learning* (ICPL), is proposed. Like PGF, this method also attempts to integrate instance-filtering and instance-abstraction techniques but it adopts concept integration instead of incremental integration in PGF. We conduct a detailed investigation in designing the two components in ICPL. Empirical performance of ICPL will also be provided.

Chapter 5

Integrated Concept Prototype Learner

In this chapter, we present the *Integrated Concept Prototype Learner* (ICPL) algorithm which is the second approach for integrating the strengths of abstraction and filtering methods. ICPL is based on the concept integration approach presented in Chapter 3. We first describe the motivation of ICPL followed by in-depth descriptions of the two components of ICPL, namely, the abstraction and filtering methods. Then we present the concept integration of ICPL attempting to combine the two independently learned concepts. Empirical results from 35 real-world benchmark data sets show that ICPL achieves a slightly higher generalization accuracy and a significant improvement in data retention rate compared with existing prototype learners. Recall that improvement in data retention rate implies a higher efficiency in applying the learned prototypes to conduct prediction for unseen cases.

5.1 Motivation

Though PGF is found to be effective in representing concepts with a few prototypes, there are still some drawbacks.

1. The training set is processed by the two methods incrementally in PGF. After the first method is applied, the entire data distribution will be changed. This prohibits the learning ability of the second method. For example, if abstraction is applied first, instances in compact regions will be abstracted to a relatively small amount of prototypes. After that, a filtering method is performed on these prototypes in attempt to retain representative ones, says border ones, by considering the class label of every prototypes' nearest neighbors. However, without original training instances, the filtering method can hardly distinguish border prototypes from others.

2. Concepts learned from the first method may be lost. If the filtering method is applied first to select border prototypes and followed by abstraction method, the boundary learned by filtering may then be severely distorted.

3. PGF suffers from high computational cost in the abstraction component leading to inefficiency in handling large data sets.

4. Though PGF achieves a lower data retention rate than most existing methods, it cannot outperform some of them, like RT3, in generalization accuracy.

We attempt to design a better prototype learner by eliminating the drawbacks of PGF. To do this, we develop a prototype learner, called *Integrated*

Concept Prototype Learner (ICPL), which integrates filtering and abstraction methods using the concept integration approach. Concept integration is designed to solve the problems of incremental integration. In concept integration, the two component methods learn concepts independently so that it does not suffer from the undesirable effects caused by incremental processing of the two components.

In concept integration, the two component concepts must be of different strengths. In the design of the two component methods, we have to investigate the characteristics and the strengths of their learned concepts. Chapter 3 has discussed the considerations in designing the two components. In the following, we summarize some important criteria for the two components.

Filtering Component. For the filtering component, it must be able to represent complex concepts effectively so that the representative power of the resulting concept can be increased and thus improving the generalization accuracy. An effective way to do this is to represent concepts using border instances. Thus, a filtering method retaining border instances should be chosen in concept integration.

Abstraction Component. For the abstraction component, it should have a high generalizing power for intermediate and center instances so that representative prototypes can be learned to describe the concept effectively. As filtering methods retaining border instances are usually sensitive to noise, the abstraction component must be able to handle noisy data. Also, one drawback of PGF is the high computational cost in abstraction. We attempt also design an abstraction method which has lower complexity in ICPL. In the next two sections, we will describe the abstraction component and the filtering component adopted in ICPL.

5.2 Abstraction Component

In this section, we propose a new abstraction method, called *Typical Prototype Abstraction* (TPA), adopted in ICPL. TPA abstracts typical instances to form prototype sets. It makes use of the *typicality* measure proposed by Zhang [72] in the abstraction process. We first show how typicality can help abstraction and then describe the TPA algorithm. Investigation of TPA on artificial data shows that it is able to generalize instances efficiently and tolerate the presence of noise.

5.2.1 Issues for Abstraction

In abstraction, we have to decide *when* to merge *which* instances.

Which Instances to Be Merged?

Most abstraction methods involve merging of similar instances [10, 13, 20, 36, 60]. Merging instances is an effective way to generalize different characteristics of different instances. However, the performance of abstraction depends highly on the kind of instances merged. For example, non-prototypical prototypes will be formed if dissimilar instances are merged. Also, merging of border instances may lead to distortion of class boundaries. Therefore, researchers try different ways to design the merging process. Chang's averaging method stops merging instances when generalization accuracy starts to degrade [13]. Kibler and Aha use an *adaptive threshold* to prevent the merging of distant instances [30]. GIS [36] conducts generalization on instances only if a more representative prototype is formed.

In concept integration, the role of abstraction is to generalize non-border instances to a few prototypes to reduce the data retention rate. As for border instances, abstraction can discard all of them since they are handled by the filtering component in the integration. Therefore, we have to distinguish non-border instances from border ones to conduct abstraction.

When to Merge?

Different instance merging orders result in different prototype sets learned and thus affecting the performance of abstraction. Chang's averaging method [13] and our abstraction component in PGF (ABS) attempt to find the best outcome by iteratively merging instances with shortest distance in the current prototype set. These methods ensure distant instances are not merged during early abstraction stages but they require a high computational cost to identify nearest instances in each iteration. To reduce computation, GIS [36] and FAMBL [60] randomly select an instance from the training set and perform abstraction on its nearest neighbors until the merging criteria is not met. This kind of method has less computational cost but pays no attention to the instance merging order.

In the abstraction component of concept integration, center and intermediate instances should be abstracted. We believe that prototypes formed from merging center instances are more stable and representative than instances formed from merging intermediate ones. Therefore, we intend to merge center instances first followed by intermediate ones until class boundaries are reached. In attempt to do this, we have to know which instances are far apart from or close to the decision boundary. Our proposed TPA algorithm employs typicality measure to determine which instances are merged as well as the merging order.

5.2.2 Investigation on Typicality

Typicality was first proposed by Zhang to indicate how typical an instance is [72]. It makes use of the intra-concept similarity and the inter-concept similarity to calculate the typicality of an instance. In our design, the typicality of an instance i, $Typ(i)$, is defined as:

$$Typ(i) = \frac{\text{average similarity of } i \text{ to instances of the same class}}{\text{average similarity of } i \text{ to instances of other classes}}$$

The similarity for two instances $\boldsymbol{x}$ and $\boldsymbol{y}$, $Sim(\boldsymbol{x},\ \boldsymbol{y})$, is defined as:

$$Sim(\boldsymbol{x},\ \boldsymbol{y}) = 1 - Dist(\boldsymbol{x},\ \boldsymbol{y})$$

where $Dist(\boldsymbol{x},\ \boldsymbol{y})$ is the distance measure used in our PGF algorithm (see Chapter 4).

According to [72], typicality has the following properties:

1. Typical instances usually have typicality much larger than 1.

2. Boundary instances usually have typicality close to 1.

3. Noisy instances usually have typicality less than 1.

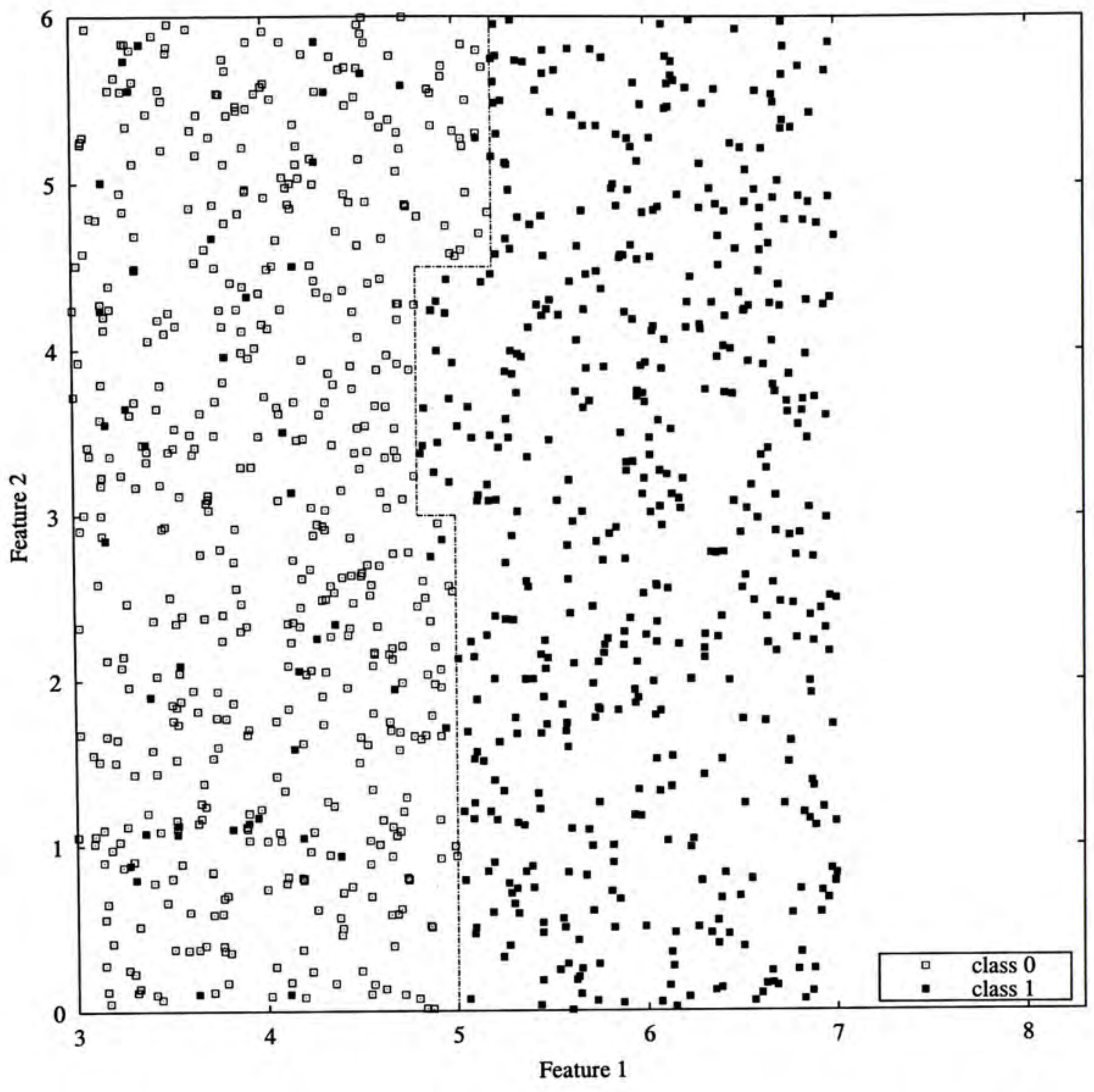

Figure 5.1: A two-class data set with noise.

It seems that it is a good indicator to locate border and non-border instances, which is useful in our abstraction process.

To aid the design of using typicality in ICPL more effectively, we conduct an in-depth analysis on typicality. Figure 5.1 shows an artificial data set consisting of two classes. A decision boundary, indicated by the dotted line, is used to generate the data set. It separates instances into two classes. Some noise is also introduced to class 0 to investigate its effects on typical-

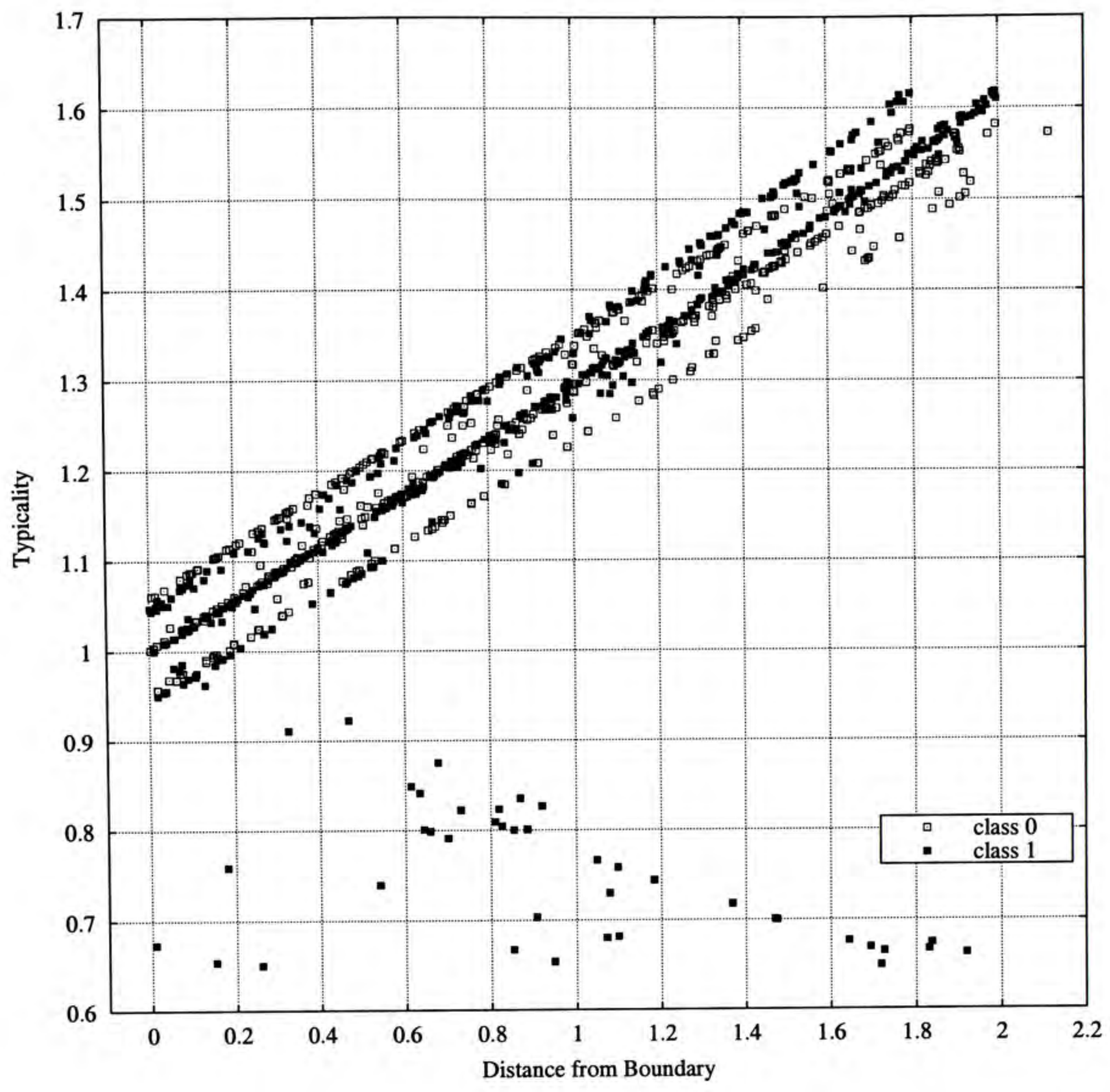

Figure 5.2: Typicality vs. distance from boundary.

ity. We intend to find the relationship between typicality and the location of instance. To do this, we calculate the typicality of each instance and the perpendicular distance between the instance and the class boundary. We plot a graph depicting the relationship between typicality and distance from boundary as shown in Figure 5.2. We can see clearly from Figure 5.2 that for ordinary instances, i.e., typicality greater than 1, there is a trend showing that typicality is linearly proportional to distance from the decision boundary. Another observation is that noisy instances have typicality less than 1, i.e., those class 1 instances scattered in the bottom of the plot. These observations agree with the properties of typicality claimed by Zhang [72]. This information is useful in the design of our abstraction component in ICPL. Instances with large distance from the boundary are center instances which should be generalized first during the abstraction process to generate more stable and representative prototypes. Essentially, we process instances with high typicality first as they are observed to be center instances. The abstraction process can take place towards the class boundary by merging instances in the descending order of their typicality until border instances are reached.

Also, it is found that instances of different classes have different range of values of typicality. We can see from Figure 5.2 that instances of class 1 have an average higher typicality than that of class 0. It suggests that instances should be merged in the order of their typicality within the range of typicality value of the same class. Besides, we also find that typicality of noise is inversely proportional to the distance from the boundary.

5.2.3 Typicality in Abstraction

According to above observations, we design a function, called *IdentifyBorder*, to identify border instances for each class based on typicality. Figure 5.3 shows the pseudo code of the function. It first calculates the typicality of all the training instances. Then training instances for each class are sorted in descending order of their typicality. After sorting, center instances will be processed first and border or noisy ones will be handled at appropriate time during abstraction. Using the typicality calculated, the function can then focus on identifying non-border instances which are the target elements for abstraction. As discussed above, instances of each class may have a different range of typicality values. It is not effective to use a single threshold to differentiate border and non-border points of all classes. To tackle this problem, we design different thresholds on typicality for each class to distinguish border instances. To do this, the mean and standard deviation of typicality of instances for each class are calculated. An instance is said to be a border one if its typicality is less than the mean typicality of its class subtracting the standard deviation. An array of border flag *Bp* stores a boolean value for each instance indicating whether the instance is in a border instance or not. The outputs of *IdentifyBorder* are *Bp* and the training instances sorted by typicality.

After processing the training set with the above function, we can then perform abstraction on non-border instances in descending order of their typicality in the TPA algorithm presented below.

5.2.4 The TPA algorithm

We now describe the core of our abstraction component in ICPL, namely, the *Typical Prototype Abstraction* (TPA) algorithm. In concept integration, the abstraction method should have high instance generalizing power and noise tolerance. Besides, in order to handle large data sets, it should have low computational complexity. After presenting the TPA algorithm, we conduct some analysis of TPA demonstrating these three strengths.

Figure 5.4 shows the TPA algorithm. The algorithm learns prototypes in a bottom-up fashion. It begins with an empty prototype set, S. The

```
Procedure **IdentifyBorder**(Training Set $T$).

For each instance $I$ in $T$ :
    Find $Typ(I)$, typicality of $I$.
For each class $C$ :
    Find $T_{mean}(C)$, the mean of typicality of class $C$ instances.
    Find $T_{sd}(C)$, the standard deviation of typicality of class $C$ instances.
    Sort class C instances in descending order of typicality.
For each class $C$ :
    For each instance $I$ of class $C$ in $T$ :
        If ( $Typ(I) < ( T_{mean}(C) - T_{sd}(C) )$ ) :
            Set $Bp(I)$ = TRUE, indicating $I$ is a border point.
        Else :
            Set $Bp(I)$ = FALSE, indicating $I$ is not a border point.
Return $Bp$.
```

Figure 5.3: The IdentifyBorder function.

```
Procedure **TPA**(Training Set $T$).

Set $S$ = empty.
Set $Bp$ = **IdentifyBorder**($T$).
For each class $C$ :
    Set $I$ = first instance of class $C$ in $T$.
    For each non-border instance $I$ ( i.e. $Bp(I)$ = FALSE ) :
        If ( $I$ is not processed ) :
            Set $P$ = **Merge**($T$, $S$, $Bp$, $I$).
            If ( no. of instances in $P > 1$ ) :
                Add $P$ to $S$.
            End If.
        End If.
        Set $I$ = next instance of class $C$ in $T$.
    End For.
End For.
Return $S$.
```

Figure 5.4: The TPA algorithm.

IdentifyBorder function is first called to sort instances for each class by their typicality and identify all the border instances before doing abstraction. After that, instances with high typicality will be processed first. It learns prototypes for each class by merging non-border instances. A prototype is represented in a similar way as in PGF. Specifically, it is represented by the mean of all the merged instances. At the beginning, the first non-border instance, says I, is selected. A prototype, says P, will then be formed by invoking the *Merge* function. TPA only retains abstracted prototypes rather than original instances since these prototypes are usually more representative. The algorithm terminates until all the non-border instances in each class are processed.

The *Merge* function is described in Figure 5.5. It accepts a training set T, a prototype set S, a border flag Bp indicating which instances in T are border ones and an instance I. It continuously merges I with its nearest neighbors in T until an instance of another class or a border instance is encountered. Let P be an abstracted prototype which is equal to I initially and N be the nearest neighbor of I. If N is not a border instance and is of the same class as I, it will then be merged with P and the next nearest neighbor of I is examined. If N is of different class from I, then it may be a noise or a border instance of other classes. TPA distinguishes these two cases by considering the class label of the next neighbor of I. If N is really a noise, the next neighbor of I should be in the same class of I. In this case, we should discard N and continue the merging process. This achieves the noise tolerance of the TPA algorithm. The merging process continues until one of the following two situations is encountered.

```
Procedure **Merge**(Training Set $T$, Prototype Set $S$, Border Flag $Bp$, Instance $I$).

Set $P = I$.
Set $N$ = nearest neighbor of $I$ in $T$.
While ( (class of $N$ not equal $C$) Or (Not $Bp(N)$) ) :
    If ( class of $N$ not equal $C$ ) :
        Set $N$ = next nearest neighbor of $I$ in $T$.
        If ( class of $N$ not equal $C$ ) :
            Return $P$.
        End If.
    End If.
    If ( $Bp(N)$ ) :
        Return $P$.
    Else :
        If ( $N$ is not merged before ) :
            Merge $P$ and $N$.
            Set $N$ = next nearest neighbor of $I$ in $T$.
        Else :
            Find $M$, the prototype containing $N$ from $S$.
            Merge $P$ with $M$.
            Remove $M$ from $S$.
            Return $P$.
        End If.
    End If.
End While.
Return $P$.
```

Figure 5.5: The Merge function.

1. N is a border instance:

   Merging of border instance, either of the same or different class, may lead to distortion of the class boundary.

2. N is an instance merged before:

   In this case, the prototype containing N, says M, in S will be found first. It will then be merged with P and discarded from S.

5.2.5 Analysis of TPA

We conduct analysis of TPA on computational complexity, instance generalizing power and noise tolerance.

Computational Complexity

In the *IdentifyBorder* function, to find the typicality of each instance, we have to calculate the similarity for each pair of instances which is of $O(n^2m)$ where n is the number of instances and m is the time required to calculate a similarity between two instances. After that, each instance is only processed once in the TPA algorithm and can be decided to be grouped into a prototype or discarded. It dramatically reduces the time complexity of the abstraction process compared with PGF.

Instance Generalizing Power

The instance generalizing power is the most important feature for the abstraction component in concept integration. An algorithm having high instance generalizing power should be able to generalize instance in compact regions into a few or single prototypes. We conduct an analysis on this characteristic

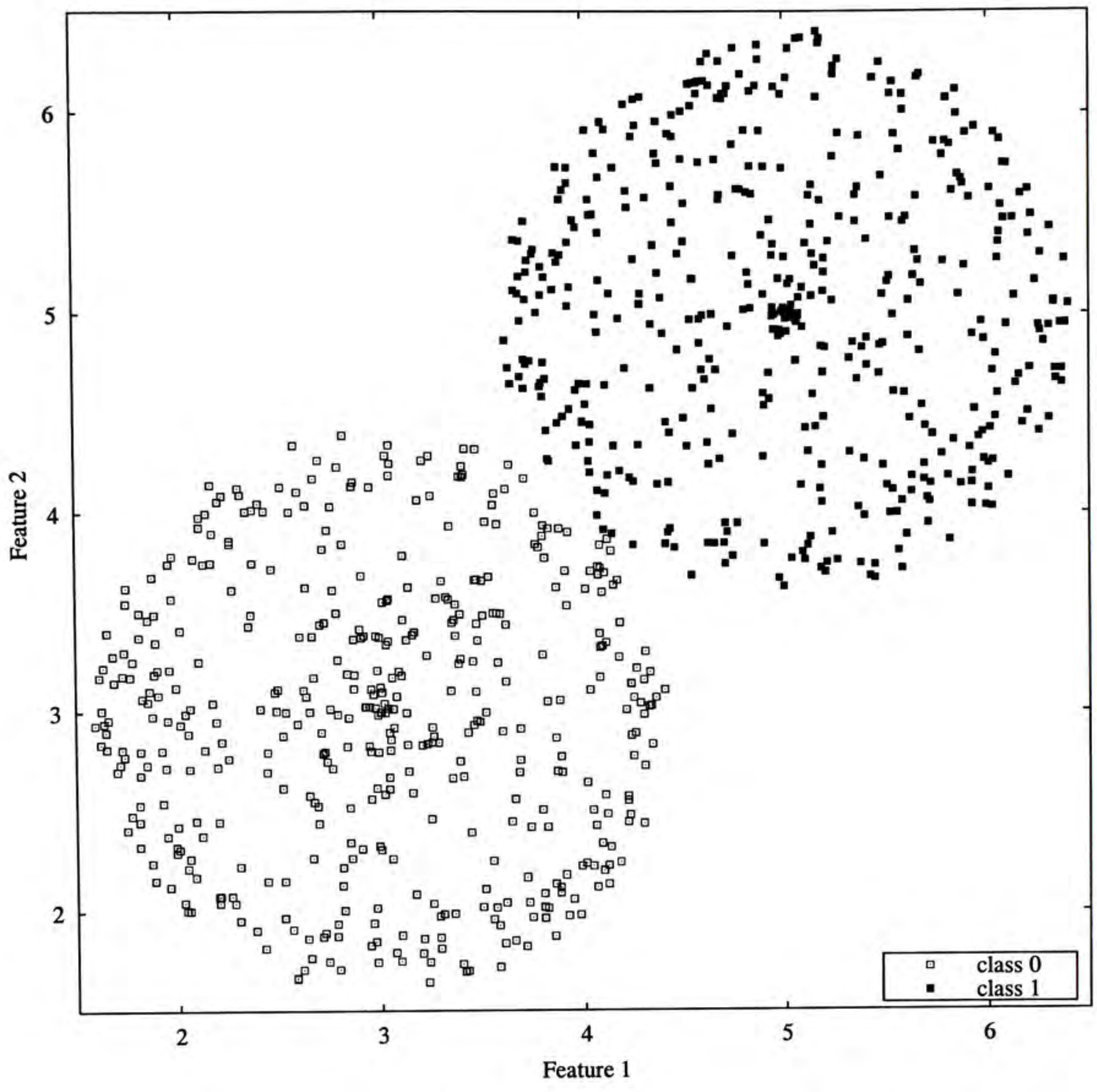

Figure 5.6: Two clouds of instances with different classes.

of the TPA algorithm. Figure 5.6 shows an artificial data set consisting of instances of two classes distributed in two clouds of regions. There are totally 900 instances in the data set, 450 for each class. After applying TPA to this data set, we find that only three prototypes are generated to represent the two concepts as shown in Figure 5.7. The prototypes generated are of high quality as they achieve 92.7% generalization accuracy on 300 unseen cases.

For the data set as shown in Figure 5.1, the result of applying TPA is

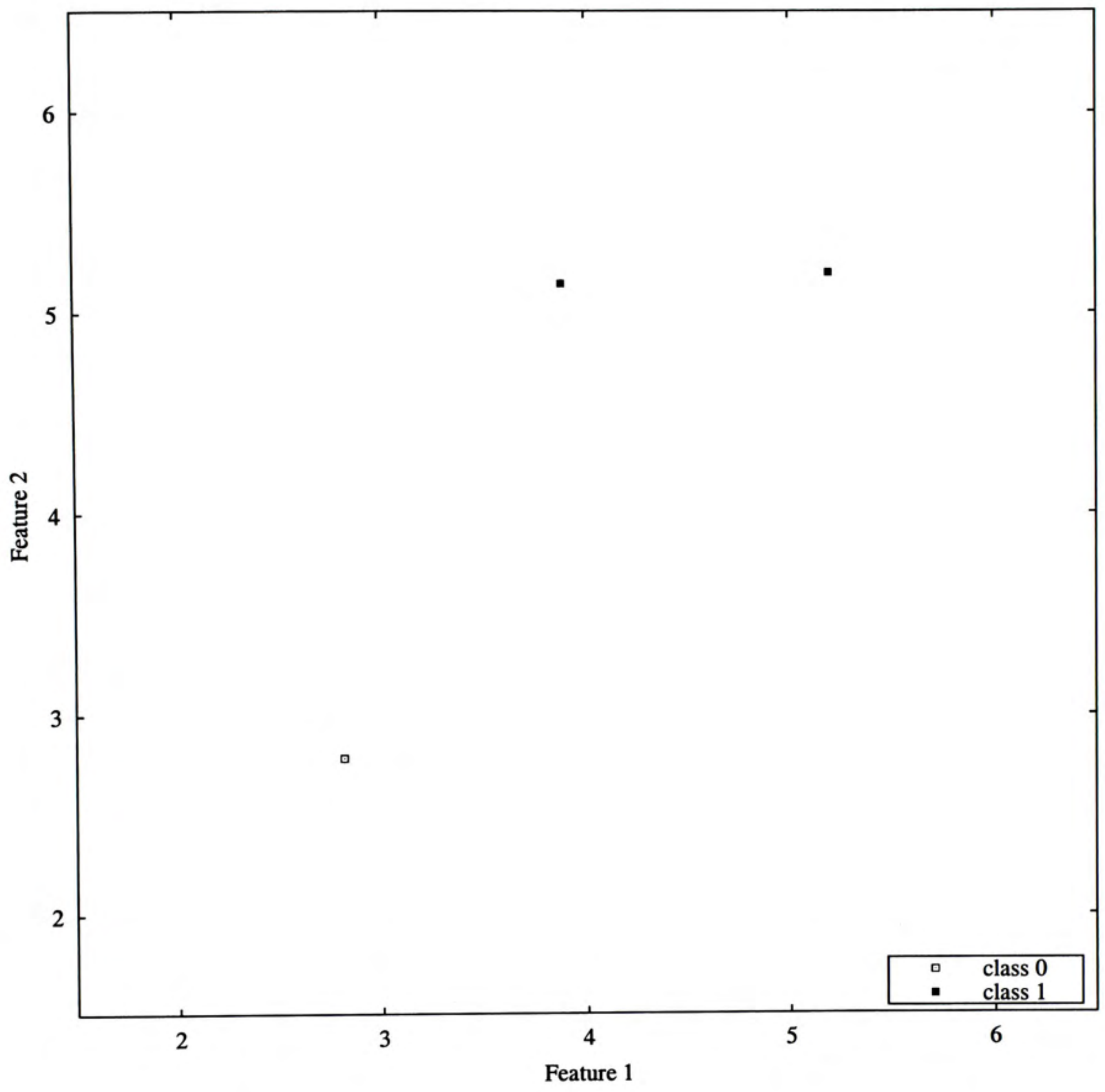

Figure 5.7: Prototypes generated from TPA on two clouds of data.

depicted in Figure 5.8. The boundary used to generate the original data set is also shown using a dotted line. It shows that TPA only retains 9 prototypes to represent the original 1,000 training instances with generalization accuracy higher than 80% on 340 unseen cases. The above two results demonstrate the high instance generalizing power of TPA.

Noise Tolerance

We conduct analysis on noise tolerance. We randomly introduce 10% mislabeled instances in class 0 for the data set in Figure 5.6 to see the noise tolerance of the TPA algorithm. The abstraction result is shown in Figure 5.9. We find that noisy instances can be generalized away from the original data set. Besides, the resulting prototype set achieves even a higher generalization accuracy compared with that abstracted from data set without noise. When TPA is applied on the data set in Figure 5.1, the abstraction result is illustrated in Figure 5.8. The prototype set shown in the figure is learned from a noisy data set in which 9.1% of mislabeled instances is introduced in class 0. We can see that no mislabeled instance is retained in the prototype set. These results support that TPA can handle noise effectively.

5.3 Filtering Component

In this section, we describe the filtering component in ICPL. It is derived from RT2 which retains border instances to form prototype sets. RT2, based on a concept called *associate*, was first proposed by Wilson and Martinez [69]. We conduct some analysis of associate and briefly discuss the RT2 algorithm. We also investigate its performance on prototype learning and find that it

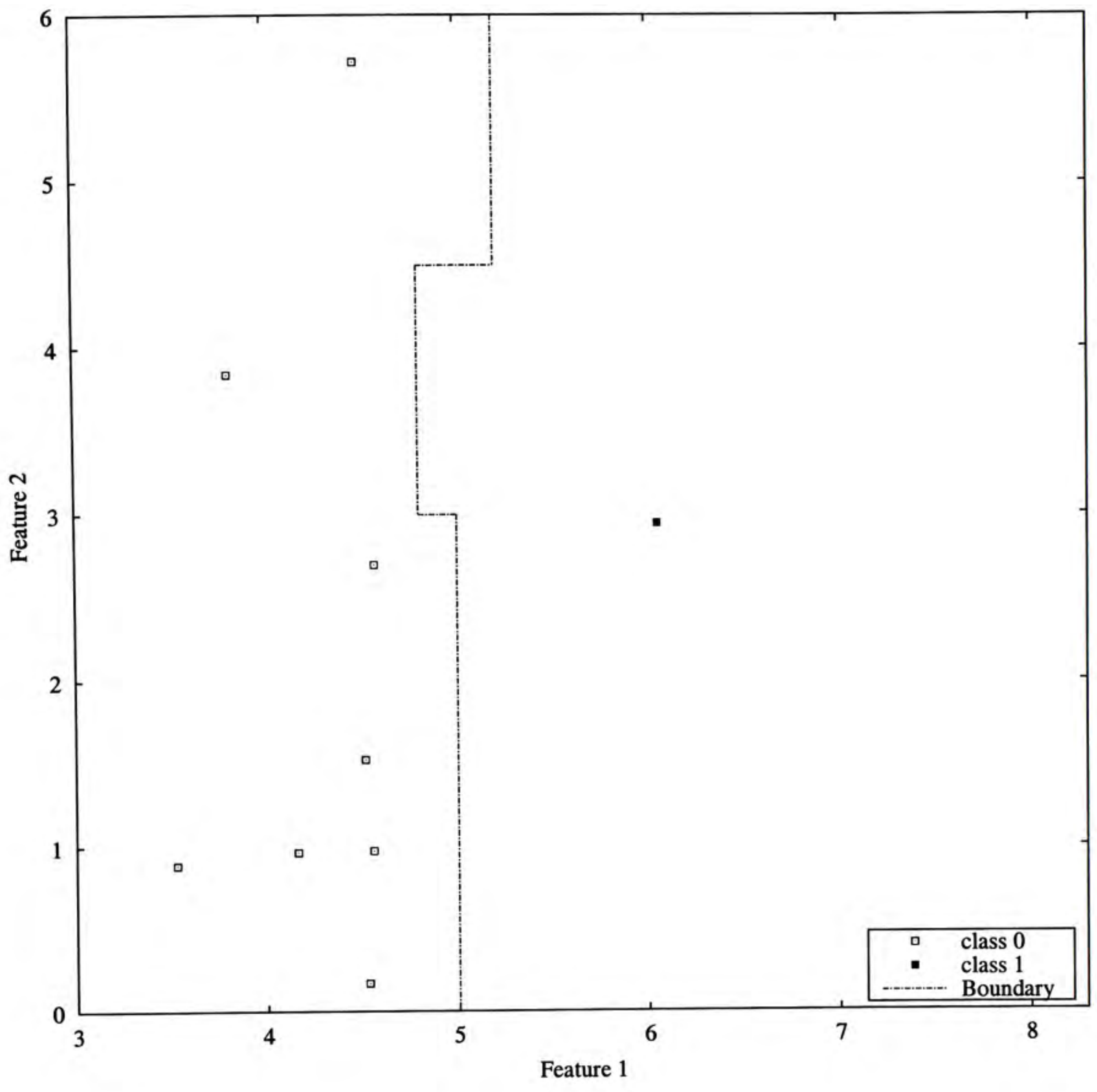

Figure 5.8: Prototypes generated from TPA on data set as shown in Figure 5.1.

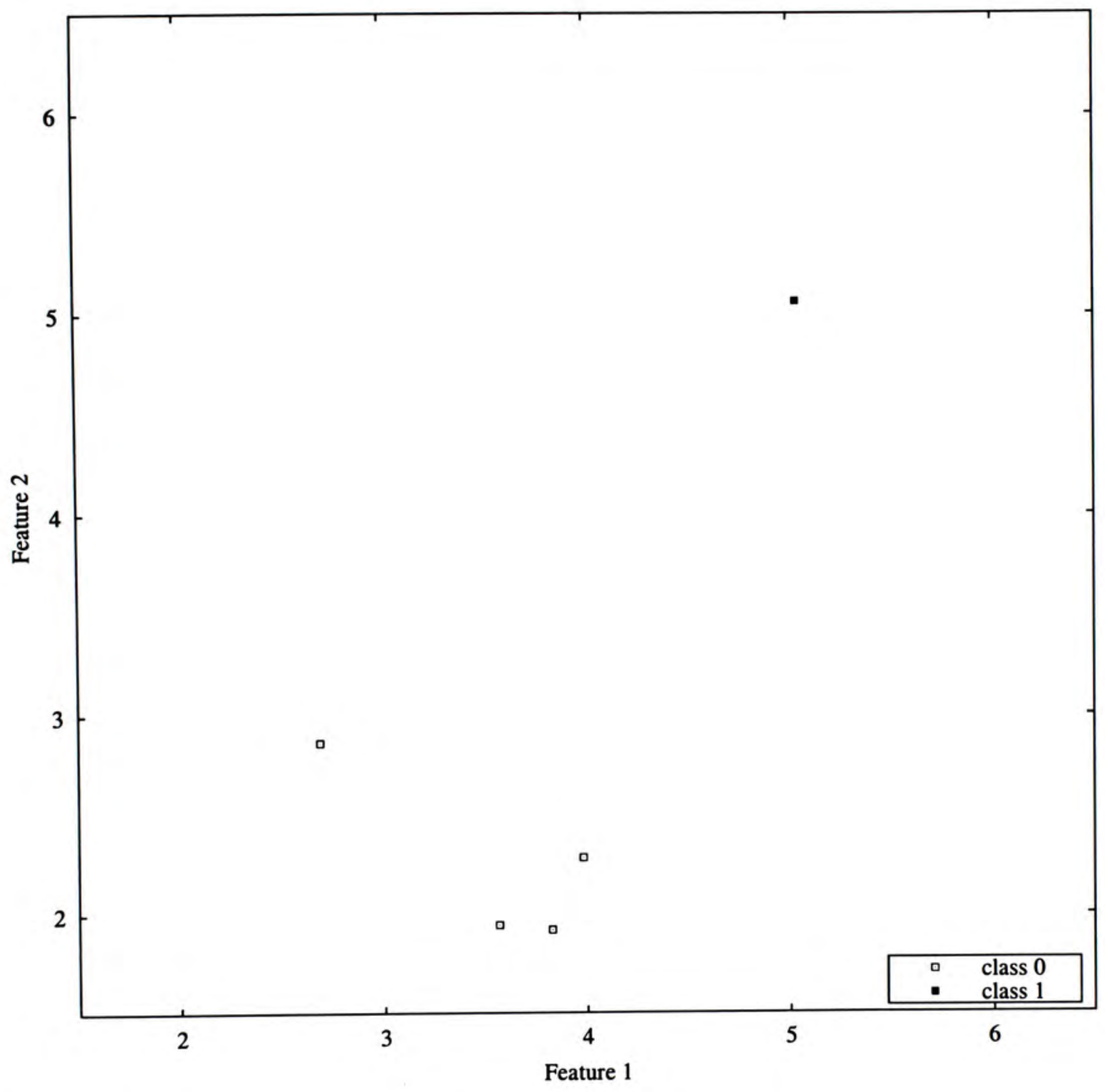

Figure 5.9: Prototypes generated from TPA on two clouds of data with noise.

is effective in detecting boundary instance to represent complex concepts. This nice property is the most important feature triggering our design of the filtering method in ICPL.

5.3.1 Investigation on Associate

Associates of an instance is the list of instances in the training set having it as one of their k-nearest neighbors. As the filtering component in ICPL should be able to describe complex boundaries using border instances, we attempt to investigate the ability of associate to locate border instances. Figure 5.10 shows a two-class data set with instances uniquely labeled according to their classes and locations counting from left to right. The class boundary used to generate the data set is also shown by a dashed line. Let k equal to 3. We first find out the k-nearest neighbors of each instance as shown in Table 5.1. Here, the $(k+1)$th nearest neighbor is also shown for later use. The nearest neighbor lists are then used to determine the associate list for each instance as shown in Table 5.2. For example, considering the nearest neighbors of *A1*, we know that *A3*, *A8* and *A10* should contain *A1* in their associate lists. After investigating the location and associate of all the instances, we have the following findings.

1. Instances with short or even empty associate list, such as *A1*, *A2* and *B21*, are usually outer instances. They seldom contribute to the class boundary.

2. Instances having most associates of the same class, such as *A10*, *A21*, *B26* and *B27*, are usually center or intermediate instances. They also do not affect the class boundary.

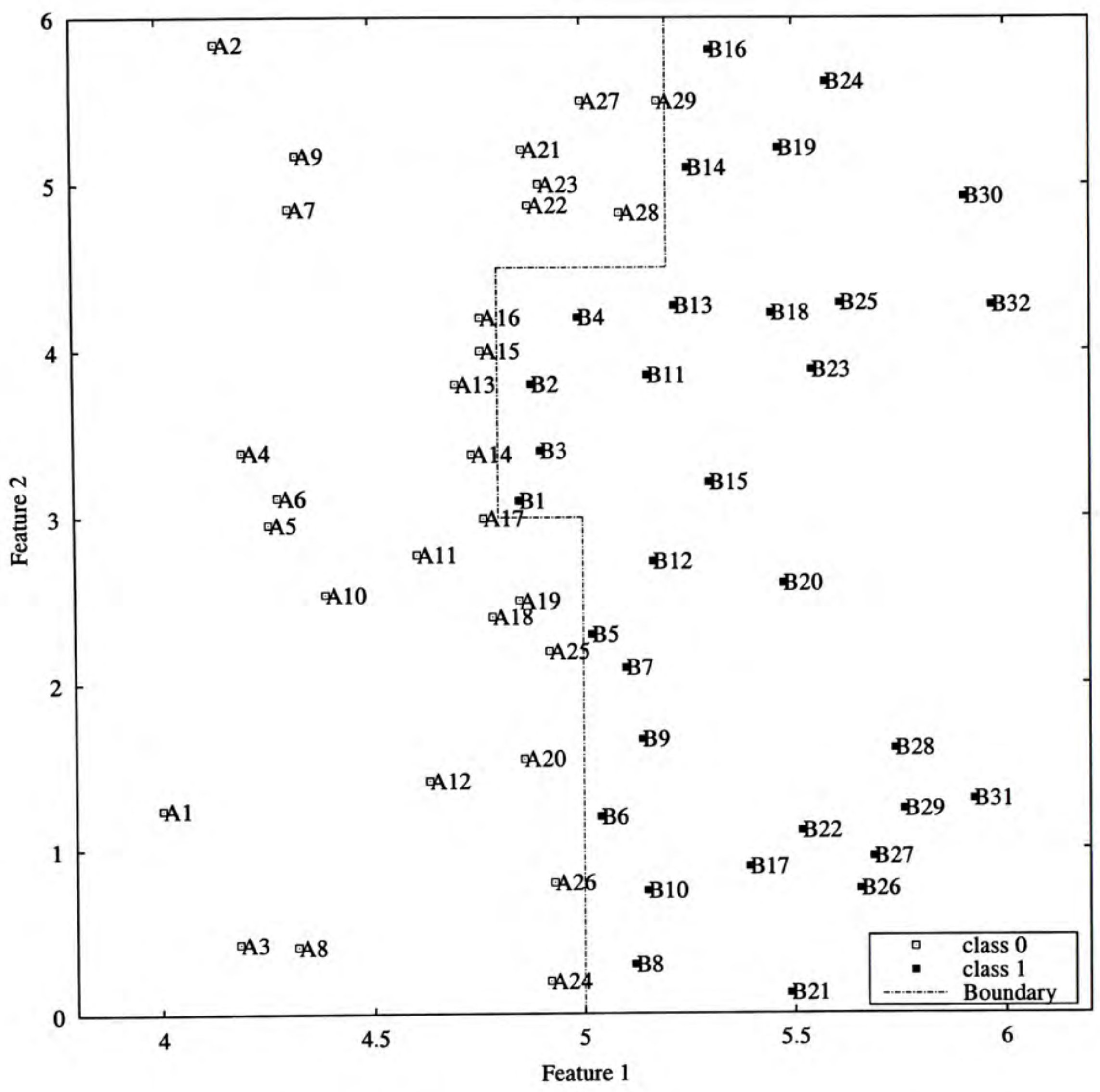

Figure 5.10: A two-class data set with labeled instances.

Data	**(*k*+*1*)-nearest neighbors**				**Data**	**(*k*+*1*)-nearest neighbors**			
A1	A3	A8	A10	A12	B1	A17	B3	A14	A19
A2	A9	A7	A21	A22	B2	A15	B3	B4	A13
A3	A8	A1	A12	A24	B3	B1	B2	A14	A17
A4	A6	A5	A10	A11	B4	B2	B11	B13	A16
A5	A6	A4	A10	A11	B5	B7	A25	A19	B12
A6	A5	A4	A10	A11	B6	A26	B9	B10	A20
A7	A9	A2	A16	A4	B7	B5	B9	A25	B12
A8	A3	A1	A12	A24	B8	B10	A24	A26	B6
A9	A7	A2	A21	A16	B9	B7	B6	B5	A20
A10	A5	A6	A11	A4	B10	B8	B6	A26	B17
A11	A17	A18	A10	A14	B11	B13	B4	B15	B2
A12	A20	A26	A18	A25	B12	B5	B15	B7	A25
A13	A15	A14	A16	B2	B13	B11	B4	B18	A28
A14	A17	B1	A13	B3	B14	A29	A28	B19	B16
A15	A16	A13	B2	A14	B15	B12	B11	B20	B23
A16	A15	A13	B2	B4	B16	A29	B14	B19	B24
A17	B1	A14	A11	A19	B17	B22	B10	B26	B21
A18	A19	A25	A17	A11	B18	B23	B25	B13	B11
A19	A18	A25	B5	A17	B19	B24	B14	B16	A29
A20	B6	A12	A25	A26	B20	B15	B12	B7	B28
A21	A23	A22	A27	A28	B21	B26	B17	B22	B27
A22	A23	A21	A28	A27	B22	B17	B27	B26	B29
A23	A22	A21	A27	A28	B23	B18	B25	B15	B13
A24	B8	A26	B10	B6	B24	B19	B16	B14	A29
A25	B5	A19	A18	B7	B25	B23	B18	B19	B32
A26	B6	A24	B10	B8	B26	B27	B22	B29	B17
A27	A21	A29	A23	A28	B27	B26	B29	B22	B28
A28	B14	A23	A22	B13	B28	B29	B31	B27	B22
A29	B14	B16	A27	A28	B29	B27	B28	B31	B26
					B30	B32	B25	B24	B19
					B31	B29	B28	B27	B26
					B32	B30	B25	B23	B18

Table 5.1: The (*k*+*1*)-nearest neighbors list of instances in Figure 5.10.

Data	Associate					Data	Associate				
A1	A3	A8				B1	A14	A17	B3		
A2	A7	A9				B2	A15	A16	B3	B4	
A3	A1	A8				B3	B1	B2			
A4	A5	A6				B4	B2	B11	B13		
A5	A4	A6	A10			B5	A19	A25	B7	B9	B12
A6	A4	A5	A10			B6	A20	A26	B9	B10	
A7	A2	A9				B7	B5	B9	B12	B20	
A8	A1	A3				B8	A24	B10			
A9	A2	A7				B9	B6	B7			
A10	A1	A4	A5	A6	A11	B10	A24	A26	B6	B8	B17
A11	A10	A17				B11	B4	B13	B15		
A12	A3	A8	A20			B12	B15	B20			
A13	A14	A15	A16			B13	B4	B11	B18		
A14	A13	A17	B1	B3		B14	A28	A29	B16	B19	B24
A15	A13	A16	B2			B15	B11	B12	B20	B23	
A16	A7	A13	A15			B16	A29	B19	B24		
A17	A11	A14	A18	B1		B17	B21	B22			
A18	A11	A12	A19	A25		B18	B13	B23	B25		
A19	A18	A25	B5			B19	B14	B16	B24	B25	
A20	A12					B20	B15				
A21	A2	A9	A22	A23	A27	B21					
A22	A21	A23	A28			B22	B17	B21	B26	B27	
A23	A21	A22	A27	A28		B23	B18	B25	B32		
A24	A26	B8				B24	B19	B30			
A25	A18	A19	A20	B5	B7	B25	B18	B23	B30	B32	
A26	A12	A24	B6	B8	B10	B26	B17	B21	B22	B27	
A27	A21	A23	A29			B27	B22	B26	B28	B29	B31
A28	A22	B14				B28	B29	B31			
A29	A27	B14	B16			B29	B26	B27	B28	B31	
						B30	B32				
						B31	B28	B29			
						B32	B30				

Table 5.2: The associate list of instances built from k-nearest neighbors listed in Table 5.1.

3. Instances with associates of multiple classes, such as *A14*, *A25*, *B2* and *B5*, are usually border instances. They form the class boundary. Besides, when focusing on *A14*, we can also find that most of its associates, such as *A13*, *A15* and *B1* are border instances too.

According to the above findings, we can make use of associate to identify instances not contributing to the class boundaries. Filtering rules can then be applied to eliminate those instances leaving border instances. We will present how associate is used in RT2.

5.3.2 The RT2 Algorithm

RT2 [69] learns prototypes in top-down fashion. Initially, each training instance is regarded as a prototype. Given a k, RT2 first finds a nearest neighbor list and an associate list for each instance. Each instance is then examined to see whether it should be removed or not. RT2 removes an instance if the removal does not hurt the classification accuracy of the prototype set on the training set. It calculates the classification accuracy by considering the associate list rather than the entire training set. When an instance is processed, its associates are identified first. The associates are then classified using the k-nearest neighbor algorithm with and without the presence of the instance. If more associates are correctly classified without the presence of the instance, it should be discarded. Supposing that *A1* is removed from the prototype set, it should then be eliminated from its associates' nearest neighbor lists, i.e., nearest neighbor lists of *A3* and *A8* in Table 5.1 should become "*A8, A12, A24*" and "*A3, A12, A24*" respectively. Also, *A3* and *A8* should be added to the associate list of *A24*. Note that the associate lists of *A1*'s neighbors, i.e., "*A3, A8* and *A10*" remain unchanged after the removal. The order of removal affects the performance of the filtering rule. RT2 solves this problem by sorting the prototypes by the distance to their nearest neighbor with other classes in attempt to remove instances furthest from the class boundary first.

Similar to TPA, RT2 also adopts $Dist(\boldsymbol{x}, \boldsymbol{y})$, the distance measure used in our PGF algorithm described in Section 4.1.2, to measure the distance between instance. It is identical to the heterogeneous distance function [69] used in the original RT2 algorithm except that the distance calculated from each feature is normalized by the maximum distance of that feature.

5.3.3 Analysis of RT2

We can see the effects of this filtering rule from Table 5.1 and 5.2. Considering the instance *A2*, we can see that it has 2 associates, *A7* and *A9*. As *A7* and *A9* can be classified correctly without *A2*, *A2* will then be discarded. This eliminates instances which neighbors are surrounded by class instances of the same class, i.e., center instances. On the contrary, for border instances, removal of them may cause others to be misclassified. Near the boundary, neighbors of instances, says *A13* and *A14*, are usually of different classes. If *A13* is removed, the majority of *A14*'s neighbors may become instances of other class leading to misclassification of *A14*. Therefore, RT2 tends to retain border instances instead of center one.

Figure 5.11 shows the performance of RT2 on describing the class boundary. In the figure, an artificial data set consisting of 1,000 instances of two different classes is shown as well as the boundary generating the instances.

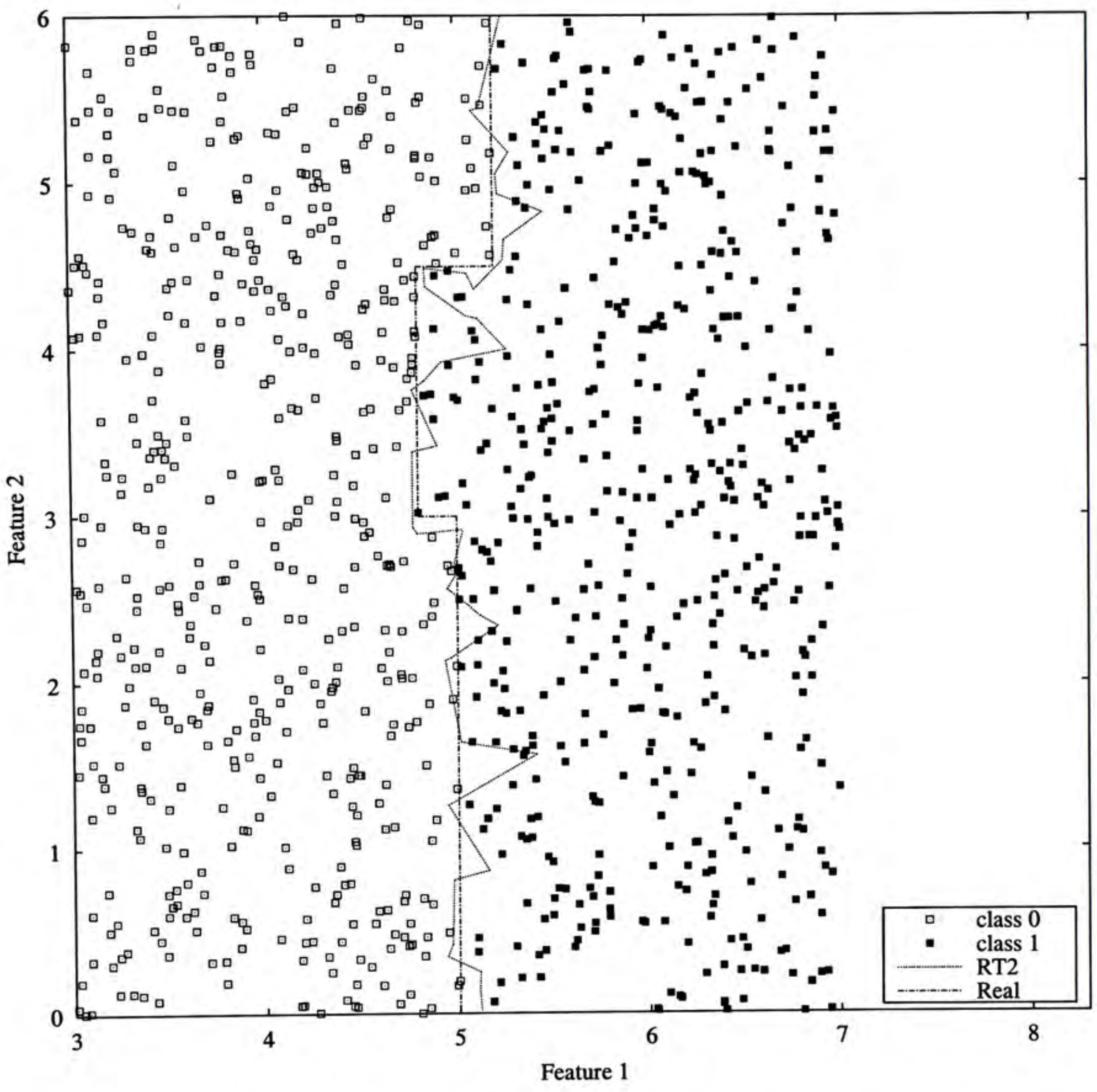

Figure 5.11: The class boundary learned by RT2 in a two-class data set.

The dotted line is used to depict the decision boundary learned by RT2. We can see that prototypes learned by RT2 can represent the class boundary quite well. Besides, to describe this concept, RT2 retains only 40 border instances. This shows that RT2 is effective in retaining representative border instances.

RT2 versus RT3

Indeed, RT3 proposed by Wilson and Martinez [69] has similar strengths as RT2. RT3 is identical to RT2 except that it adopts a noise-filtering pass before sorting the instances. Experimental results show that RT3 outperforms RT2 on data retention rate [69]. However, RT3 is not adopted in our ICPL for two reasons. Firstly, the noise-filtering pass of RT3 intends to remove noise. However, noise can be handled by the abstraction component in ICPL. Not adopting the noise-filtering pass helps reduce the computational cost. Besides, the noise-filtering pass is usually able to smooth the class boundary. However, during the smoothing process, some information about the class boundary, which is essential to our concept integration, may be lost. Therefore, we choose RT2 instead of RT3 as our filtering component in ICPL.

5.4 Concept Integration

This section presents the concept integration method attempting to combine the two concepts learned by the two components. We present the ICPL algorithm and conduct some analysis on it using artificial data sets. We find that ICPL is capable of learning a small set of good prototypes to describe concepts in domains. Next, we present the results of some experiments on 35 real-world benchmark data sets.

5.4.1 The ICPL Algorithm

As the main objective of a classifier is to correctly classify unseen cases based on the knowledge learned from the training set, we use generalization accuracy of the prototype set on the original training set as the main criteria to combine the concepts. Besides, we also attempt to reduce the data retention rate of the resulting prototype set. We design an integration method to achieve these two goals.

Figure 5.12 shows the ICPL algorithm. Let the prototype sets learned from abstraction and filtering be A and F respectively and S be the resulting prototype set. In the integration, we first retain all prototypes in A. Each prototype learned from F will then be examined whether it should be added to the integrated prototype set. A temporary prototype set Tmp containing both the current prototype set S and the examined prototype P is created and is evaluated using the training set T. If the addition of P increases the number of correct classifications on the training set, P will be accepted. It ensures that concepts unsuccessfully learned by abstraction will be complemented by prototypes learned from filtering. After all the prototypes in F are processed, we start to refine the integrated prototype set by examining the abstracted prototypes. Similar to filtered prototypes, abstracted ones are also evaluated by the training set. When P is examined in the evaluation process, a temporary prototype set Tmp is also created with P removed from the current prototype set S. The abstracted prototype P is discarded if its

```
Procedure ICPL(Training Set $T$, Integer $k$).

1   Set $A$ = TPA($T$).
2   Set $F$ = RT2($T$, $k$).
3   Set $S = A$.
4   For each prototype $P$ in $F$ :
5       Set $Tmp = S \cap P$.
6       Set $Good$ = No. of instances in $T$ correctly classified by $P$ in $Tmp$.
7       Set $Bad$ = No. of instances in $T$ incorrectly classified by $P$ in $Tmp$.
8       If ( $Good > Bad$ ) :
9           Set $S = S \cap P$.
10  For each prototype $P$ in $A$ :
11      Set $Tmp = S \setminus P$.
12      Set $With$ = No. of instances in $T$ correctly classified by $S$.
13      Set $Without$ = No. of instances in $T$ correctly classified by $Tmp$.
14      If ( $Without >= With$ ) :
15          Set $S = S \setminus P$.
16  Return $S$.
```

Figure 5.12: The ICPL algorithm.

removal does not hurt the generalization accuracy of the current prototype set on training set. During abstraction, not all the prototypes learned are essential to the formation of the class boundary. This pruning step helps eliminate those prototypes.

5.4.2 Analysis of ICPL

We try to investigate the performance of ICPL for describing concepts. A two-class artificial data set, also used in the investigation of RT2 as shown in Figure 5.11, containing 1,000 instances is generated to conduct the investigation. We compare the boundary learned by ICPL on the data set with the true boundary used to generate the data set. Figure 5.13 shows the results. We can see that our ICPL algorithm is able to approximate the true boundary. Most importantly, ICPL uses only 7 instances to describe the concept while RT2 uses 40.

We also investigate the noise tolerance of ICPL. To do this, we introduce about 10% mislabeled instance in class 0. Figure 5.14 shows the results. We find that the integrated prototype set contains no noise and can approximate the true boundary using only 14 instances. This shows that our ICPL is capable of handling noise or mislabeled instances.

5.5 Empirical Analysis

5.5.1 Experimental Setup

We have conducted a series of experiments which is similar to that in PGF. Our ICPL was tested on 35 real-world benchmark data sets from UCI Repos-

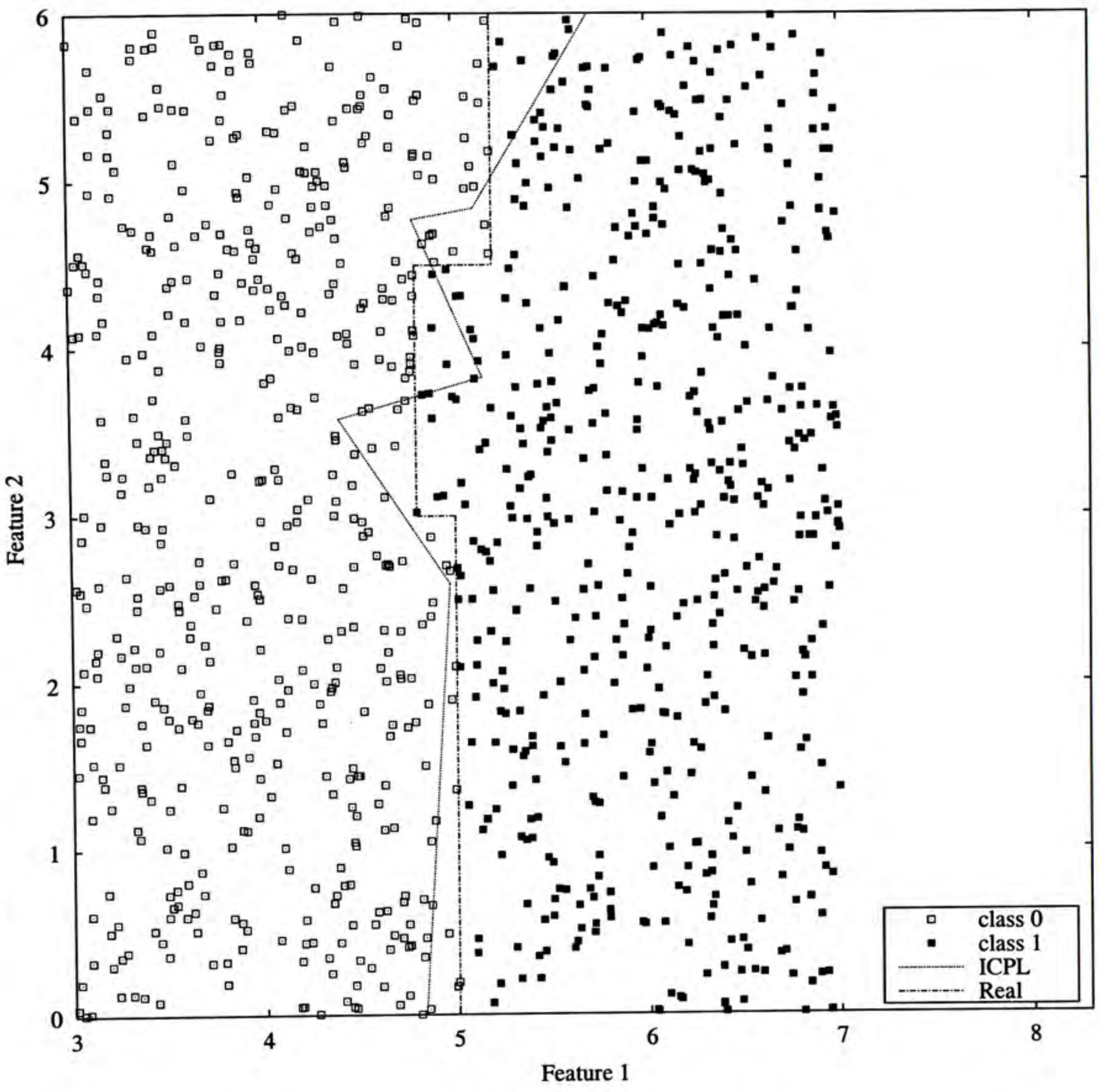

Figure 5.13: The class boundary learned by ICPL in a two-class data set.

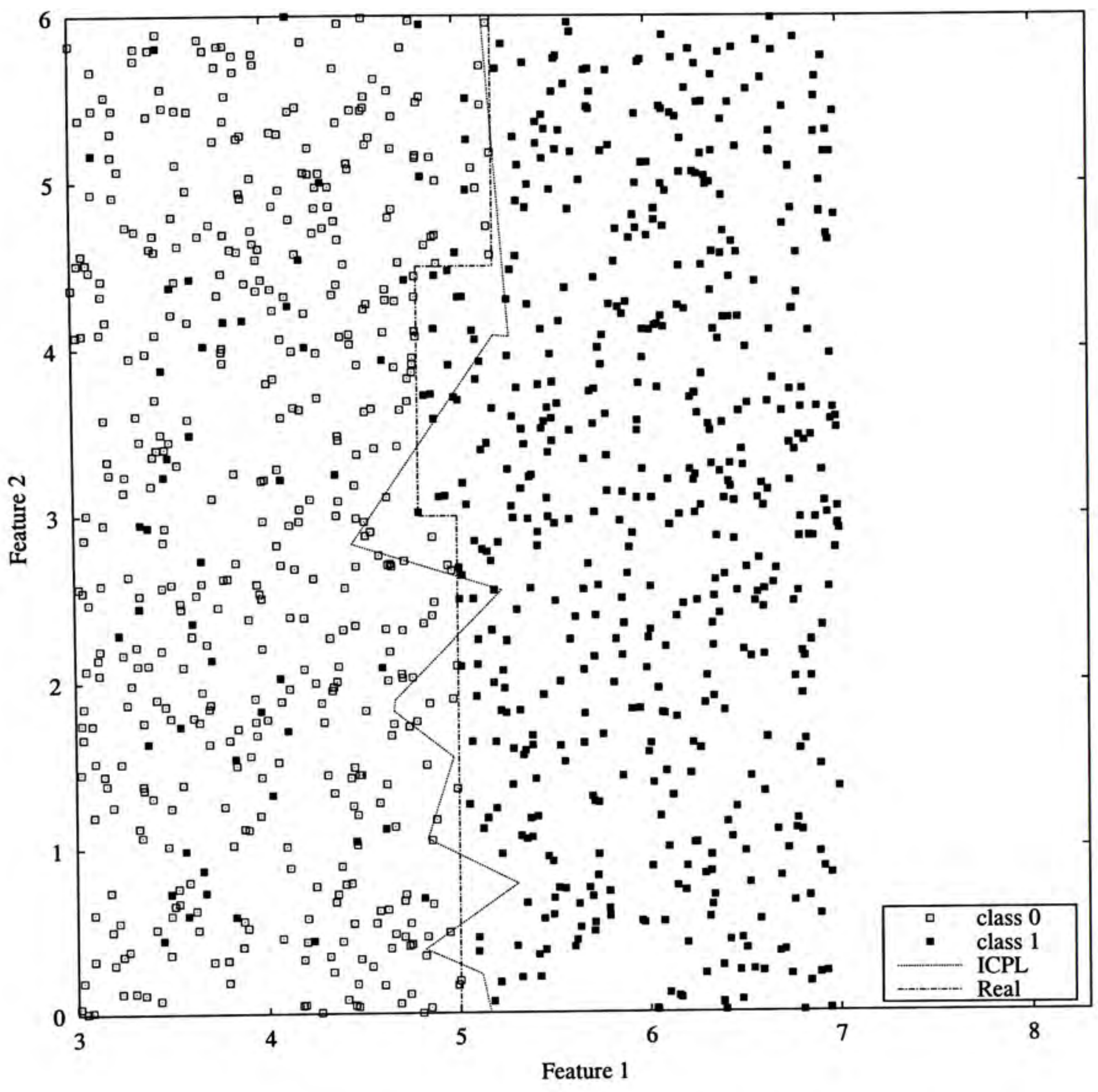

Figure 5.14: The class boundary learned by ICPL in a two-class data set with noise.

itory [8]. The mean generalization accuracy and data retention rate of 10-fold cross-validation were obtained for each data set. Recall that some of the large data sets are randomly selected to form smaller subsets in experiments on PGF. As the complexity of ICPL is reduced, we can apply it on large data sets so that we do not need to reduce the size of those data sets. In other words, the full amount of data is used in the experiments.

We compare ICPL with its two component methods to see the effects of concept integration in the first set of experiments. In the second set of experiments, we compare our algorithm with existing learning algorithms, namely, RT3, C4.5 and KNN, as well as a random prototype learner.

5.5.2 Results of ICPL Algorithm

We have obtained the performance in terms of the generalization accuracy, together with its standard deviation, and data retention rate of our ICPL algorithm. Note that higher classification accuracy and smaller data retention rate imply better performance. A range of parameters k for ICPL, RT2 and KNN were tested and their best results were presented. Table 5.3 reports the performance for ICPL and its component methods, namely, TPA and RT2, on 35 real-world benchmark data sets. The average generalization accuracy and data retention rate across 35 data sets for each algorithm are also shown. In Section 4.3.5, we investigate the performance of different PGF variants using two comparative analyses. The first one counts the number of data sets in which PGF performs better or worse than each of the algorithm on generalization accuracy and the second one employs a one-tail Wilcoxon Signed Ranks test [15] to compare the significance of the improvement of

PGF by formal statistical analysis. These two analyses are also adopted here to make rigorous comparison between ICPL and other algorithms and the results are shown in the last two rows of the table.

The performance for the state-of-the-art prototype learner, namely, RT3, and classification algorithms, namely, C4.5 and KNN are also obtained for each of the data set. Table 5.4 presents the average generalization accuracy and data retention rate of the 10-fold cross-validation results for each data set and the average performance across the data sets.

5.5.3 Comparisons with Pure Abstraction and Pure Filtering

In this set of experiment, we analyze ICPL by comparing its performance with pure abstraction and pure filtering. The pure abstraction method tested is TPA which is the abstraction component of ICPL. The pure filtering method tested is RT2 which is the filtering component of ICPL.

ICPL vs. TPA

TPA has a good instance generalization power which can abstract compact instances into a few prototypes but it is ineffective to describe complex concepts. It leads to its good average data retention rate but poor generalization accuracy as shown in Table 5.3. When comparing ICPL with TPA, we observe that ICPL achieves a 11.8% higher generalization accuracy with only about 3% more in prototype set size. ICPL is found to outperform TPA in generalization accuracy on all the 35 data sets and the Wilcoxon test shows a 99.5% confidence that ICPL gains a significant higher accuracy

			Pure abstraction		**Pure filtering**	
	ICPL		**TPA**		**RT2**	
Data	accuracy	size	accuracy	size	accuracy	size
Ab	0.624 (0.058)	0.196	0.566 (0.064)	0.141	0.620 (0.060)	0.372
Am	0.812 (0.084)	0.062	0.744 (0.106)	0.036	0.804 (0.035)	0.200
Au	0.685 (0.107)	0.131	0.601 (0.096)	0.109	0.716 (0.112)	0.271
Ba	0.859 (0.043)	0.016	0.849 (0.107)	0.008	0.839 (0.034)	0.170
Bc	0.968 (0.055)	0.009	0.951 (0.035)	0.005	0.960 (0.030)	0.060
Ca	0.946 (0.020)	0.053	0.822 (0.063)	0.039	0.949 (0.041)	0.136
Cs	0.851 (0.045)	0.035	0.762 (0.050)	0.015	0.831 (0.049)	0.177
Ec	0.846 (0.102)	0.053	0.827 (0.087)	0.028	0.872 (0.101)	0.171
Gl	0.677 (0.279)	0.154	0.592 (0.280)	0.097	0.692 (0.214)	0.259
He	0.864 (0.076)	0.052	0.800 (0.111)	0.020	0.855 (0.231)	0.166
Io	0.878 (0.086)	0.048	0.783 (0.148)	0.017	0.906 (0.036)	0.092
Ir	0.947 (0.071)	0.043	0.887 (0.092)	0.030	0.960 (0.044)	0.107
Le	0.691 (0.047)	0.179	0.598 (0.048)	0.093	0.716 (0.044)	0.305
Li	0.644 (0.076)	0.151	0.577 (0.077)	0.082	0.597 (0.051)	0.278
M1	0.914 (0.047)	0.116	0.698 (0.106)	0.078	0.953 (0.056)	0.291
M2	0.968 (0.026)	0.082	0.753 (0.258)	0.055	0.987 (0.021)	0.184
M3	0.946 (0.062)	0.037	0.701 (0.110)	0.017	0.966 (0.051)	0.127
Mu	0.987 (0.006)	0.001	0.933 (0.012)	0.002	1.000 (0.000)	0.003
Ne	0.935 (0.076)	0.031	0.903 (0.065)	0.020	0.944 (0.043)	0.127
Nu	0.941 (0.006)	0.063	0.828 (0.008)	0.055	0.947 (0.011)	0.174
Op	0.958 (0.021)	0.023	0.924 (0.019)	0.009	0.965 (0.017)	0.060
Pe	0.976 (0.006)	0.015	0.869 (0.025)	0.005	0.986 (0.004)	0.042
Pi	0.716 (0.111)	0.087	0.699 (0.111)	0.039	0.733 (0.077)	0.230
Se	0.937 (0.032)	0.040	0.821 (0.045)	0.020	0.949 (0.009)	0.102
Sh	0.998 (0.002)	0.003	0.784 (0.058)	0.003	0.998 (0.002)	0.008
Sn	0.880 (0.095)	0.135	0.687 (0.194)	0.072	0.861 (0.108)	0.236
Sb	0.903 (0.053)	0.085	0.822 (0.049)	0.061	0.899 (0.073)	0.176
Tt	0.851 (0.031)	0.089	0.745 (0.026)	0.043	0.861 (0.032)	0.209
Vo	0.947 (0.050)	0.029	0.869 (0.041)	0.009	0.924 (0.030)	0.092
Vw	0.923 (0.045)	0.151	0.722 (0.104)	0.089	0.957 (0.013)	0.292
Wd	0.946 (0.019)	0.025	0.912 (0.021)	0.010	0.943 (0.048)	0.086
Wi	0.960 (0.105)	0.040	0.944 (0.039)	0.031	0.943 (0.087)	0.126
Wp	0.763 (0.182)	0.119	0.631 (0.128)	0.058	0.692 (0.137)	0.258
Ye	0.566 (0.042)	0.141	0.538 (0.028)	0.087	0.549 (0.041)	0.270
Zo	0.950 (0.069)	0.083	0.920 (0.101)	0.085	0.950 (0.126)	0.149
Average	0.864	0.074	0.773	0.045	0.866	0.172
Better	0-0	0-0	35-0	2-33	14-21	35-0
Wilcoxon	50.00	-50.00	99.50	-99.50	-85.01	99.5

Table 5.3: The average classification accuracy and data retention rate (size) of 10-fold cross-validation for ICPL, TPA and RT2. The standard deviation of classification accuracy is given inside the bracket.

	ICPL		RT3		C4.5	KNN
Data	accuracy	size	accuracy	size	accuracy	accuracy
Ab	0.624 (0.058)	0.196	0.621 (0.152)	0.319	0.792 (0.066)	0.766 (0.076)
Am	0.812 (0.084)	0.062	0.794 (0.067)	0.136	0.939 (0.041)	0.771 (0.082)
Au	0.685 (0.107)	0.131	0.667 (0.116)	0.247	0.666 (0.083)	0.761 (0.102)
Ba	0.859 (0.043)	0.016	0.837 (0.065)	0.103	0.900 (0.032)	0.775 (0.066)
Bc	0.968 (0.055)	0.009	0.957 (0.035)	0.033	0.953 (0.063)	0.960 (0.014)
Ca	0.946 (0.020)	0.053	0.952 (0.018)	0.112	0.692 (0.043)	0.956 (0.016)
Cs	0.851 (0.045)	0.035	0.826 (0.035)	0.085	0.642 (0.054)	0.807 (0.047)
Ec	0.846 (0.102)	0.053	0.878 (0.059)	0.108	0.921 (0.081)	0.822 (0.095)
Gl	0.677 (0.279)	0.154	0.672 (0.341)	0.211	0.894 (0.031)	0.681 (0.300)
He	0.864 (0.076)	0.052	0.856 (0.208)	0.093	0.964 (0.007)	0.805 (0.186)
Io	0.878 (0.086)	0.048	0.869 (0.047)	0.068	0.694 (0.085)	0.866 (0.058)
Ir	0.947 (0.071)	0.043	0.947 (0.114)	0.080	0.962 (0.014)	0.947 (0.043)
Le	0.691 (0.047)	0.179	0.696 (0.050)	0.282	0.999 (0.001)	0.810 (0.034)
Li	0.644 (0.076)	0.151	0.566 (0.110)	0.218	0.706 (0.094)	0.632 (0.089)
M1	0.914 (0.047)	0.116	0.971 (0.071)	0.296	0.779 (0.046)	0.969 (0.039)
M2	0.968 (0.026)	0.082	0.984 (0.014)	0.182	0.944 (0.031)	0.993 (0.016)
M3	0.946 (0.062)	0.037	0.955 (0.069)	0.086	0.888 (0.081)	0.955 (0.045)
Mu	0.987 (0.006)	0.001	1.000 (0.000)	0.003	0.676 (0.168)	1.000 (0.000)
Ne	0.935 (0.076)	0.031	0.948 (0.073)	0.111	0.545 (0.049)	0.972 (0.031)
Nu	0.941 (0.006)	0.063	0.968 (0.008)	0.167	0.756 (0.064)	0.991 (0.004)
Op	0.958 (0.021)	0.023	0.964 (0.029)	0.057	0.928 (0.012)	0.985 (0.010)
Pe	0.976 (0.006)	0.015	0.986 (0.003)	0.040	0.960 (0.084)	0.994 (0.005)
Pi	0.716 (0.111)	0.087	0.719 (0.097)	0.141	0.625 (0.079)	0.706 (0.114)
Se	0.937 (0.032)	0.040	0.949 (0.008)	0.091	0.988 (0.033)	0.969 (0.015)
Sh	0.998 (0.002)	0.003	0.998 (0.001)	0.007	1.000 (0.000)	0.999 (0.001)
Sn	0.880 (0.095)	0.135	0.812 (0.046)	0.226	0.979 (0.004)	0.876 (0.152)
Sb	0.903 (0.053)	0.085	0.889 (0.051)	0.155	0.930 (0.034)	0.908 (0.053)
Tt	0.851 (0.031)	0.089	0.876 (0.025)	0.181	0.862 (0.036)	0.914 (0.027)
Vo	0.947 (0.050)	0.029	0.922 (0.098)	0.062	0.960 (0.021)	0.935 (0.031)
Vw	0.923 (0.045)	0.151	0.956 (0.017)	0.293	0.926 (0.101)	0.992 (0.016)
Wd	0.946 (0.019)	0.025	0.952 (0.041)	0.056	0.776 (0.056)	0.945 (0.028)
Wi	0.960 (0.105)	0.040	0.938 (0.125)	0.114	0.794 (0.156)	0.954 (0.054)
Wp	0.763 (0.182)	0.119	0.723 (0.153)	0.138	0.832 (0.054)	0.701 (0.108)
Ye	0.566 (0.042)	0.141	0.544 (0.040)	0.173	0.822 (0.060)	0.524 (0.054)
Zo	0.950 (0.069)	0.083	0.931 (0.059)	0.169	0.773 (0.182)	0.970 (0.034)
Average	0.864	0.074	0.861	0.138	0.842	0.875
Better Wilcoxon	0-0 50.00	0-0 -50.00	17-17 74.01	35-0 99.50	17-18 73.66	15-19 -84.48

Table 5.4: The average classification accuracy and data retention rate (size) of 10-fold cross-validation for ICPL, RT3, C4.5 and KNN. The standard deviation of classification accuracy is given inside the bracket.

than TPA. This result suggests that with the help of the filtering component RT2, TPA can retain border prototypes, learned from RT2, to describe more complicated concepts resulting in significant improvement in generalization accuracy.

ICPL vs. RT2

RT2 retains border instances to describe concepts while center and intermediate instances, which do not affect the decision boundary, are discarded. Therefore, RT2 can gain a very high generalization accuracy and, at the same time, achieve a rather low level of data retention rate. However, we find that RT2 retains unnecessarily large amount of instances to describe even a very simple concept. From Figure 5.11, we can see that a rather complex boundary is learned by RT2 to represent a linear boundary in the lower half of the plot. With the integration of abstraction component TPA, we attempt to eliminate those extra instances to reduce the data retention rate while maintaining the high generalization accuracy. From Table 5.3, we observe that when integrated with TPA, RT2 achieves a significant reduction in data retention rate, from 17.2% to 7.4% with only a 0.23% decrease in generalization accuracy. When further investigating the performance of the two methods, we notice that RT2 does not gain a high confidence level in accuracy on Wilcoxon test which indicates that it has a similar accuracy to ICPL. As for data retention rate, ICPL is found to be able to achieve a significantly better performance than RT2 with over 99% confidence level. These results show that our ICPL is capable of discarding border prototypes which do not contribute much to represent the class boundary.

In conclusion, after testing our ICPL algorithm on real-world benchmark data sets, we find that the two component methods, TPA and RT2, are strong in summarizing instances and detecting decision boundaries respectively. Empirical results also suggest that ICPL can unify the strengths of filtering and abstraction technique in prototype learning to achieve a high generalization accuracy and low data retention rate.

5.5.4 Comparisons with Other Approaches

In the second set of experiments, we compare with other algorithms, namely RT3, C4.5 and KNN. We then contrast ICPL with a random prototype learner to show its learning ability. Our ICPL algorithm is also compared with the PGF algorithm proposed in Chapter 4. We first compare ICPL with RT3 and then C4.5 and KNN. Table 5.4 shows the average generalization accuracy and data retention rate of ICPL and the three algorithms on 35 real-world data sets.

ICPL vs. RT3

RT3 and its derivatives, renamed as DROP1-DROP5, were intensively investigated in [68]. Among the five algorithms, DROP3 (actually the RT3 algorithm) is found to have the best mix of generalization accuracy and data retention rate. DROP3 is also compared with 10 other prototype learners, namely, CNN [25], SNN [48], IB2, IB3 [2], ENN [65], Unlimited and All K-NN [58] and Explore [12]. Experiments on 31 real-world benchmark data sets show that DROP3 performs the best among these data reduction techniques considering both the average generalization accuracy and data retention rate.

Therefore, we choose RT3 in our comparative study.

It is found from Table 5.4 that ICPL achieves a slightly higher generalization accuracy than RT3. ICPL outperforms RT3 in generalization accuracy on 17 data sets while RT3 gains better performance on the other 17. Wilcoxon test on accuracy also suggests that the two methods obtain similar performance over the 35 data sets. However, we can see that ICPL is better than RT3 in terms of the data retention rate. Compared with RT3, ICPL only retains half amount of prototypes to gain such a high generalization accuracy. From the table, we observe that ICPL retains only 7.4% in size of the original training set while RT3 stores 13.4%. Using the Wilcoxon test, we can see that ICPL has a significant lower data retention rate than RT3 does. All these results support that our ICPL algorithm outperforms the state-of-the-art prototype learner, RT3, in learning representative prototypes.

ICPL vs. C4.5 and KNN

Contrasting ICPL with C4.5, we notice that ICPL outperforms C4.5 in average generalization accuracy across all the data sets by 2.5%. The two algorithms achieve high generalization accuracy on nearly the same number of data sets. Based on Wilcoxon test, ICPL is observed to obtain a better generalization accuracy than C4.5 does, however, without a high confidence level. On some data sets, such as Ab and Am, C4.5 gains a significantly higher generalization accuracy while on some other data sets, such as Ca and Cs, ICPL performs significantly better. These results suggest that the two methods are strong in handling different kinds of data distribution.

When comparing ICPL with KNN, we observe that KNN has a 1.27%

higher average generalization accuracy. Among the 35 data sets, KNN performs better on generalization accuracy on 19 of them while ICPL gains better results on other 15. A value of −84.48 confidence level is obtained indicating that KNN is believed to be able to achieve a better generalization accuracy, however, with only a low confidence level. The above results suggest that our ICPL achieves comparable, or even higher, generalization accuracy with state-of-the-art learning algorithms such as C4.5 and KNN.

We now consider the data retention rate of ICPL and KNN. In KNN, all the training instances are retained to perform classification. It leads to high data retention rate and classification computation. In contrast, as for our ICPL algorithm, only 7.4% of the total instances are stored. For some data sets, such as Bc, Mu and Sh, ICPL obtains similar or even better generalization accuracy than KNN with only less than 1% of the total instances. With significantly fewer prototypes, ICPL can classify unseen cases with lower computational cost and storage requirement than that of KNN.

ICPL vs. Random Prototype Selection

Our ICPL algorithm and its two component methods are also compared with a random prototype selection method in which prototype set is formed by randomly selecting a subset of the training instances. We present the generalization accuracy of the random prototype selection method with different size of the selected subset. Figure 5.15 indicates the generalization accuracy versus the data retention rate for ICPL, TPA, RT2 and the random prototype selection method. The three methods are found to more effective than the random method to describe concepts. Specifically, TPA and RT2 gain 10%

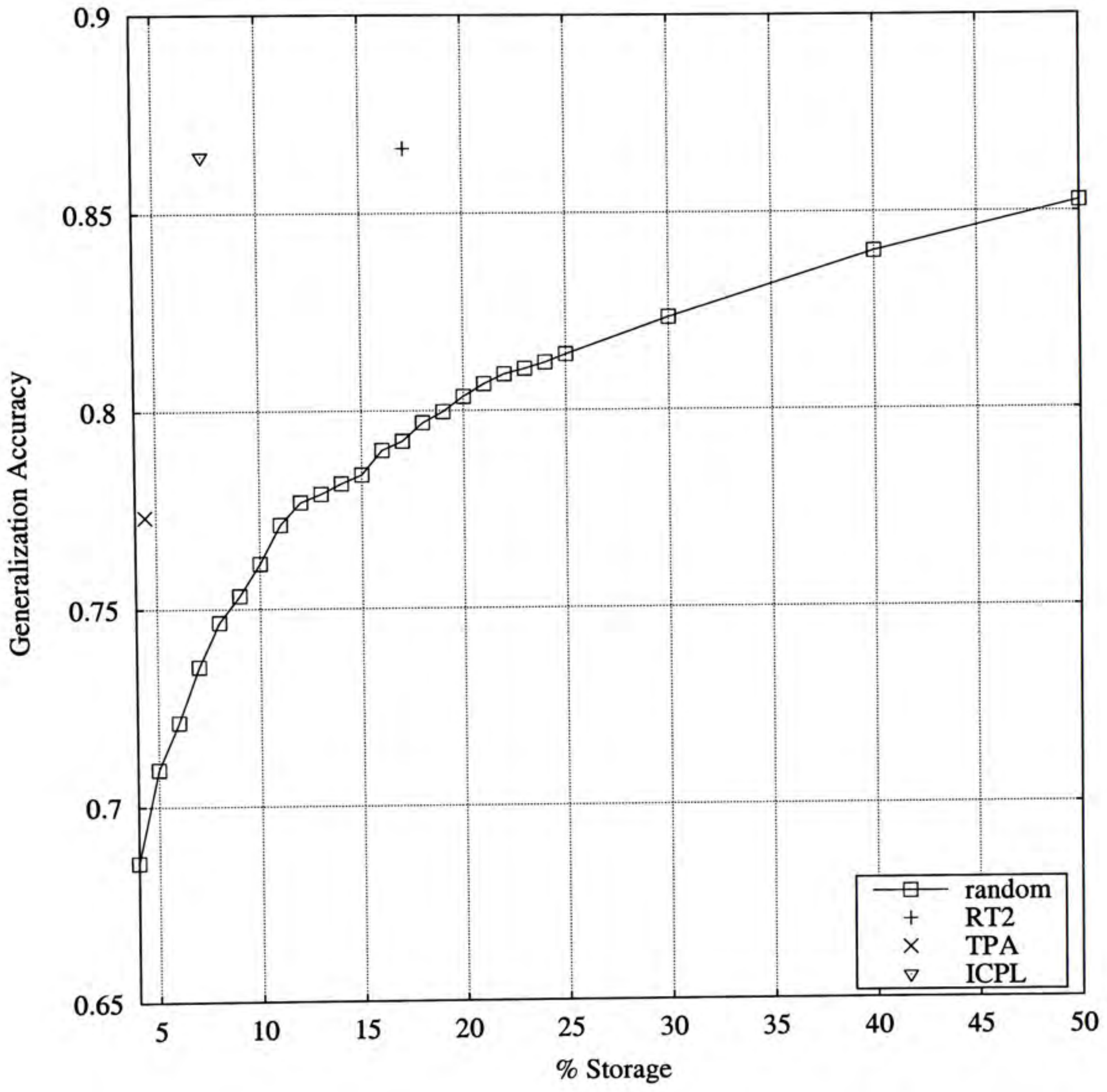

Figure 5.15: Generalization accuracy for ICPL, TPA and RT2 vs. the random prototype selection method.

and 9% higher generalization accuracy respectively compared with the random method at the same level of data retention rate. The results show that our component methods can learn from the training set to get better performances. As for ICPL, a significantly higher generalization accuracy, i.e., 17%, is obtained. It suggests that concept integration is useful for improving component methods in prototype learning.

ICPL vs. PGF

Although both ICPL and PGF are tested on the same 35 data sets, the size of seven data sets tested in PGF is reduced due to the high complexity of the PGF algorithms. Therefore, we cannot directly compare the results of the two algorithms on the reduced data sets. Instead, we analyze the results on the remaining 28 data sets to compare the performance of PGF and ICPL. PGF2-ACC is chosen to contrast with ICPL as it performs the best among the six PGF variants. Table 5.5 shows the average generalization accuracy and data retention rate of ICPL and PGF2-ACC across the 28 data sets. The results of two previous used comparative analyses are given in the last two rows of the table.

	ICPL		**PGF2-ACC**	
	accuracy	size	accuracy	size
Average	0.85	0.08	0.83	0.10
Better	0-0	0-0	24-4	17-11
Wilcoxon	50.00	-50.00	99.50	95.25

Table 5.5: The average classification accuracy (accuracy) and data retention rate (size) of 10-fold cross-validation across 28 real-world data sets for ICPL and PGF2-ACC.

According to Table 5.5, we find that ICPL gains a higher average generalization accuracy and lower average data retention rate than PGF2-ACC does. Also, ICPL outperforms PGF2-ACC in both accuracy and data retention rate on most of the data sets. Wilcoxon test also suggests that ICPL gains significantly better results. These results show that ICPL learns fewer prototypes to gain higher accuracy on the 28 data sets compared with PGF2-ACC.

5.6 Time Complexity

As shown in Section 5.2.5, the abstraction component in ICPL, TPA, takes $O(n^2m)$ time to learn prototypes where n is the number of training instances and m is the time to calculate a distance between two instances. As for the filtering component, RT2, its time complexity is similar to that of RT3 except that ENN is not employed. However, to find the associate lists of all the training instances, $O(n^2m)$ time is still required. In order to integrate the two concepts, we need to calculate the distance between prototypes of the two concepts learned by the two components. In the worst case, both the two concepts contain the whole training set and the number of distance computation is $n * n$ and the time to calculate all the required distance is $O(n^2m)$. Therefore, the time complexity of our ICPL algorithm is $O(n^2m) + O(n^2m) + O(n^2m)$ which is $O(n^2m)$.

5.7 Summary

We have proposed a second prototype learning methods, called *Integrated Concept Prototype Learner* (ICPL), which integrates the strengths of instance-filtering and instance-abstraction techniques using the concept integration approach. ICPL attempts to unify the high instance generalizing power of abstraction and high boundary description power of filtering to learn prototype sets with high generalization accuracy and low data retention rate. To do this, we propose an abstraction method, called TPA, which adopts typicality in deciding the order and choice of instances to be merged. As for filtering component, we employ RT2 to retain boundary instances. A thorough investigation on the behavior of RT2 and on artificial data sets confirm the ability of RT2 to represent the class boundary. In ICPL, two concepts are independently learned by the two component methods. After that, an integration method, based on the classification accuracy on the training set, is used to combine the strengths of the two learned concepts to form the target prototype set.

We conduct comparative analysis of our ICPL and the pure filtering and pure abstraction methods on both generalization accuracy and data retention rate. Empirical results on 35 real-world benchmark data sets show that ICPL can gain a high generalization accuracy using few prototypes. ICPL is found to be able to unify the strengths of the two component methods. ICPL is also contrasted with the state-of-the-art filtering algorithm (RT3) and classification algorithms (C4.5 and KNN). Through comparative analysis, we observe that our ICPL algorithm obtains a significant improvement in data retention rate with comparable generalization accuracy. In particular, ICPL gains comparable or even superior generalization accuracy to C4.5 and KNN using a prototype set with only 7.4% of the training set in size.

Chapter 6

Conclusions and Future Work

This chapter summarizes this thesis and discusses its main contributions. Future research directions for our work are also suggested.

6.1 Conclusions

Traditional instance-based learning algorithms suffer from high on-line computational cost, storage requirement and noise sensitivity. Our goal in this thesis is to overcome these drawbacks using prototype learning methods. In prototype learning, a relatively small prototype set is learned to represent the original training set. This method can obviously reduce the computational cost and storage requirement of instance-based learning algorithms. Moreover, a good prototype set removing noise can solve the problem of high noise sensitivity.

Instance-filtering and Instance-abstraction

There are many approaches to learning representative prototype sets including instance-filtering and instance-abstraction. Instance-filtering refers to techniques selecting representative prototypes from the original training set. They intend to find a subset of training instances to describe original ones. Filtering techniques can be grouped according to the location of instances they retained, namely, 1) retaining border instances, 2) removing border instances and 3) retaining center instances. Another approach, called instance-abstraction, is to learn artificial prototype set by generalizing or abstracting training instances.

We have reviewed major previous work on these two prototype learning approaches and investigate the strengths and weaknesses of them. Filtering methods are found to be fast and simple. Some of them can achieve high generalization accuracy compared with the classical nearest neighbor algorithm and can be flexibly designed and integrated to handle instances in different regions. However, they cannot perform well if prototype instances are not found in original data and are sensitive to noise. Also, their data retention rate is usually high.

As for instance-abstraction, results from previous work show that this method can achieve low data retention rate and generate prototypes more representative than original instances. Noise can also be absorbed during the abstraction process. In spite of these strengths, abstraction techniques are usually more complicated and bad prototypes will be formed when merging distant instances, thus leading to poor generalization accuracy.

Incremental Integration and Concept Integration

After analyzing the strengths and weaknesses of these two methods, we find that they can be beneficial to each other and are strong in handling different kinds of data distributions. We propose two approaches integrating the two methods. The first approach is regarded as *incremental integration* in which filtering and abstraction methods are applied incrementally. Under this approach, filtering and abstraction can be complementary with each other in an incremental fashion. Some drawbacks of one component can be overcome by the counterpart component in incremental integration. The second approach is regarded as *concept integration* in which concepts are learned independently and integrated. It attempts to unify the strengths of the two learned concepts.

We develop two new integration algorithms, namely, PGF and ICPL, one for each of the two approaches.

PGF

The first prototype learning method, called *Prototype Generation and Filtering* (PGF), which integrates instance-filtering and instance-abstraction techniques based on incremental integration approach. We propose an abstraction method, called ABS, based on an agglomerative clustering technique to merge nearest prototypes forming the prototype set. Three filtering methods, i.e., retaining prototypes in border, non-border and center regions, are investigated to integrate with the abstraction method. There are two issues affecting the performance of this integration method, namely, the type of filtering methods and the filtering granularity. We propose two PGF algorithms which differ in filtering granularity. This design leads to different variants of the two algorithms using different filtering methods as well as state-of-the-art algorithms, such as C4.5 and KNN.

ICPL

The second prototype learning method, called *Integrated Concept Prototype Learner* (ICPL), which integrates the strengths of instance-filtering and instance-abstraction techniques based on concept integration approach is also proposed. ICPL attempts to unify the high instance generalizing power of abstraction and high boundary description power of filtering to learn prototype sets with high generalization accuracy and low data retention rate. To do this, we propose an abstraction method, called TPA, which adopts typicality in deciding the order and choice of instances to be merged. It is found that TPA achieves high instance generalizing power using artificial data sets. As for filtering component, we employ an existing algorithm, called RT2, to retain boundary instances. A thorough investigation of the behavior of RT2 and investigation on artificial data sets confirm the ability of RT2 to represent the class boundary. In ICPL, two concepts are independently learned by the two component methods. After that, an integration method, based on classification accuracy on the training set, is used to combine the strengths of the two learned concepts to form the target prototype set. Through comparative analysis, we suggest that our ICPL algorithm obtains a significant improvement in data retention rate for existing prototype learners. In particular, ICPL gains comparable or even superior generalization accuracy to C4.5 and KNN using a prototype set with only 7.4% of the training set in size.

6.2 Future Work

Instance-based learning algorithms consists of three main components, namely, representative instance selection function, similarity function and classification function. The performance of instance-based learning algorithms depends highly on these three components. Our thesis focuses on the first component. One future work can investigate the remaining two components. Instance-based learning involves learning from the similarity between instances. In many applications, instances may contain both continuous and discrete features. A good similarity function is therefore essential to instance-based learning. In our proposed algorithms, the class label of prototype with shortest distance to the unseen case is the output. Although this simple classification may perform well on many applications, the generalization accuracy may be improved if a more sophisticated classification function is employed.

Another future work is to focus on feature selection. In instance-based learning, instances are retained without any change in data representation. However, this method stores irrelevant features as well which will normally hurt the classification performance. With the removal of irrelevant features, both the generalization accuracy and storage requirement of instance-based learning algorithms can be improved.

In our prototype learning algorithms, a lot of computation is required on identifying nearest neighbors of instances. Therefore, the learning computational cost of our integrated systems can be further reduced if appropriate scaling-up nearest neighbor search method is adopted. This method can also be applied on on-line classification so that unseen cases can be classified faster.

In abstraction, merging of outliers may lead to formation of non-representative prototypes. Therefore, we design filtering rules to discard outliers. However, some outliers may be informative in classifying unseen cases. The performance of prototype learner may be improved by designing filtering rules to retain the useful outliers.

Also, in ICPL, we proposed a new abstraction method (TPA) based on typicality to merge non-border instances. It will be interesting to consider the use of TPA in the PGF framework so that its computational complexity can be reduced.

Bibliography

[1] D.W. Aha. Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies*, 36:267–287, 1992.

[2] D.W. Aha, D. Kibler, and M.K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[3] D.W. Aha, D. Kibler, and M.K. Albert. Noise-tolerant instance-based learning algorithms. In *International Joint Conference on Artificial Intelligence*, pages 794–799, 1991.

[4] R. Bareiss. *Exemplar-Based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification and Learning.* Academic Press, Inc., 1989.

[5] S. Berchtold, B. Ertl, D. A. Keim, H. P. Kriegel, and T. Seidl. Fast nearest neighbor search in high-dimensional space. In *Proceedings of Fourteenth International Conference on Data Engineering*, pages 209–218, 1998.

[6] Michael (Michael J. A.) Berry. *Data mining techniques : for marketing, sales, and customer support / Michael J.A. Berry, Gordon Linoff.* New York : Wiley Computer, 1997.

[7] J.C. Bezdek, T.R. Reichherzer, G.S. Lim, and Y. Attikiouzel. Multiple-prototype classifier design. *IEEE Transactions on Systems, Man, and Cyberneics*, 28(1):67–79, 1998.

[8] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[9] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40:235–282, 1989.

[10] G. Bradshaw. Learning about speech sounds: The nexus project. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 1–11, 1987.

[11] R.M. Cameron-Jones. Minimum description length instance-based learning. In *Proceedings of the Fifth Australian Joint Conference on Artificial Intelligence*, pages 368–373, 1992.

[12] R.M. Cameron-Jones. Instance selection by encoding length heuristic with random mutation hill climbing. In *Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence*, pages 293–301, 1995.

[13] C.L. Chang. Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers*, 23(11):1179–1184, 1974.

[14] P. Clark and T. Niblett. The cn2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.

[15] W.J. Conover. *Practical nonparametric statistics.* New York: Jogn Wiley, 1971.

[16] S Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic feature. *Machine Learning*, 10:57–78, 1993.

[17] B.V. Dasarathy. Minimal consistent set (MCS) identification for optimal nearest neighbor decision systems design. *IEEE Transactions on Systems, Man, and Cyberneics*, 24(3):511–517, 1994.

[18] P. Datta and D. Kibler. Learning prototypical concept description. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 158–166, 1995.

[19] P. Datta and D. Kibler. Learning symbolic prototypes. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 75–82, 1997.

[20] P. Datta and D. Kibler. Symbolic nearest mean classifier. In *Proceedings of the Fourteenth National Conference of Artificial Intelligence*, pages 82–87, 1997.

[21] P.A. Devijver and J. Kittler. On the edited nearest neighbor rule. In *Proceedings of the IEEE Fifth International Conference on Pattern Recognition*, pages 72–80, 1980.

[22] P. Domingos. Unifying instance-based and rule-based induction. *Machine Learning*, 24:141–168, 1996.

[23] G.W. Gates. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, 18(3):431–433, 1972.

[24] K.C. Gowda and G. Krisha. The condensed nearest neighbor rule using the concept of mutual nearest neighborhood. *IEEE Transactions on Information Theory*, 25(4):488–490, 1979.

[25] P.E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(3):515–516, 1968.

[26] K. Hattori and M. Takahashi. A new edited k-nearest neighbor rule in the pattern classification problem. *Pattern Recognition*, 33:521–528, 2000.

[27] C. K. Keung and W. Lam. Prototype generation based on instance filtering and averaging. In *Processings of the Fourth Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 142–152, 2000.

[28] C.K. Keung and W. Lam. Discovering representative instances using clustering and pruning. In *Proceedings of International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000)*, 2000.

[29] D. Kibler and A.W. Aha. Learning representative exemplars of concepts: An initial case study. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 24–30, 1987.

[30] D. Kibler and D.W. Aha. Comparing instance-averaging with instance-filtering learning algorithms. In *Proceedings of the Third European Working Session on Learning*, pages 63–80, 1988.

[31] B. S. Kim and S. B. Park. A fast k nearest neighbor finding algorithm based on the ordered partition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):367–372, 1986.

[32] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[33] L.I. Kuncheva. Editing for the k-nearest neighbors rule by a genetic algorithm. *Pattern Recognition Letters*, 16:809–814, 1995.

[34] L.I. Kuncheva. Fitness functions in editing k-nn reference set by genetic algorithms. *Pattern Recognition*, 30(6):1041–1049, 1997.

[35] L.I. Kuncheva and J.C. Bezdek. Nearest prototype classification: Clustering, genetic algorithms, or random search? *IEEE Transactions on Systems, Man, and Cyberneics*, 28(1):160–164, 1998.

[36] W. Lam and C. Y. Ho. Using a generalized instance set for automatic text categorization. In *Proceedings of the Twenty-First Annual lnternational ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 81–89, 1998.

[37] W. Lam, C.K. Keung, and C.X. Ling. *Learning Via Prototype Generation and Filtering*, chapter 13. To appear in *Instance Selection and Construction - A Data Mining Perspective*, Kluwer Academic, 2000.

[38] W. Lam, C.K. Keung, and C.X. Ling. Learning good prototypes for classification using filtering and abstraction of instances. Submitted to *Pattern Recognition*.

[39] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 3(4):4–22, 1987.

[40] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–161, 1983.

[41] G.W. Milligan and M.C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, 1985.

[42] B. K. Natarajan. *Machine Learning: A Theoretical Approach.* San Mateo, Calif. : M. Kaufmann Publishers, 1991.

[43] Papadimitriou, H. Christos, and Jon Louis Bentley. A worst-case analysis of nearest neighbor searching by projection. *Lecture Notes in Computer Science*, 85:470–482, 1980.

[44] T.R. Payne and P. Edwards. Implicit feature selection with the value difference metric. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence*, pages 450–454, 1998.

[45] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[46] J. R. Quinlan. *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann, 1993.

[47] F. Ricci and P. Avesani. Date compression and local metrics for nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):380–384, 1999.

[48] G.L. Ritter, H.B. Woodruff, and S.R. Lowry. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, 21(6):665–669, 1975.

[49] S. Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 6:251–276, 1991.

[50] R. E. Schapire, Y. Singer, and A. Singhal. Boosting and Rocchio applied to text filtering. In *Proceedings of the Twenty-First Annual lnternational ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 215–223, 1998.

[51] G.S. Sebestyen. *Decision-Making Process in Pattern Recognition.* New York: The Macmillan Company, 1962.

[52] H. A. Simon. Why should machines learn. In *R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, (Eds), Machine Learning: An Artificial Intelligence Approach*, pages 25–37, 1983.

[53] D. B. Skalak. *Prototype Selection for composite nearest neighbor classifiers.* PhD thesis, University of Massachusetts Amherst, 1997.

[54] D.B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 293–301, 1994.

[55] R. F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991.

[56] C. Stanfill and D. Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29:1213–1228, 1986.

[57] C.W. Swonger. Sample set condensation for a condensed nearest neighbor decision rule for pattern recognition. In *Watanabe, S., editor, Frontiers of Pattern Recognition*, pages 511–519. Academic Press, New York, NY, 1972.

[58] I. Tomek. An experiment with the edited nearest–neighbor rule. *IEEE Transactions on Systems, Man, and Cyberneics*, 6(6):448–452, 1976.

[59] H. Trri, P. Knotkanen, and P. Myllymäki. Probabilistic instance-based learning. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 158–166, 1996.

[60] A. van den Bosch. Instance-family abstraction in memory-based language learning. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 39–48, 1999.

[61] S. M. Weiss and C. A. Kulikowski. *Computer systems that learn : classification and prediction methods from statistics, neural nets, machine learning, and expert systems.* San Mateo, Calif. : M. Kaufmann Publishers, 1991.

[62] D. Wettschereck. A hybrid nearest–neighbor and nearest–hyperrectangle algorithm. In *Proceedings of the Seventh European Conference on Machine Learning*, pages 323–335, 1994.

[63] D. Wettschereck, D.W. Aha, and T. Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11:273–314, 1997.

[64] D. Wettschereck and T.G. Dietterich. An experimental comparison of the nearest–neighbor and nearest–hyperrectangle algorithms. *Machine Learning*, 19:5–27, 1995.

[65] D. L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cyberneics*, 2(3):431–433, 1972.

[66] D. R. Wilson. *Advances in Instance-Based Learning Algorithm.* PhD thesis, Department of Computer Science, Brigham Young University, 1997.

[67] D. R. Wilson and T. R. Martinez. In integrated instance-based learning algorithm. *Computational Intelligence*, 16(1):1–28, 2000.

[68] D. R. Wilson and T. R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38:257–286, 2000.

[69] D.R. Wilson and T.R. Martinez. Instance pruning techniques. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 403–411, 1997.

[70] Q. Xie, C.A. Laszlo, and R.K. Ward. Vector quantization technique for nonparametric classifier design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(12):1326–1330, 1993.

[71] T. P. Yunck. A technique to identify nearest neighbor. *IEEE Transactions on Systems, Man, and Cybernetics*, 6(10):341–346, 1976.

[72] J. Zhang. Selecting typical instances in instance-based learning. In *Proceedings of International Conference on Machine Learning*, pages 470–479, 1992.

Appendix A

Detailed Information for Tested Data Sets

	Number of					
Data Set	**Instance**	**Class**	**Feature:**	continuous	symbolic	**Missing Value**
Automobile	205	7	25	15	10	yes
Auto-Mpg	398	5	7	4	3	yes
Audiology	226	24	69	0	69	yes
Balance-Scale	625	3	4	4	0	no
Breast-Cancer-W	699	2	9	9	0	yes
Car-Evaluation	1728	4	6	0	6	no
Credit-Screening	690	2	15	6	9	yes
Ecoli	336	8	7	5	2	no
Glass	214	6	9	9	0	no
Hepatitis	155	2	19	32	68	yes
Ionosphere	351	2	34	34	0	no
Iris	150	3	4	4	0	no
*Letter	20000	26	16	16	0	no
Liver	345	2	6	6	0	no
Monks-1	432	2	6	0	6	no
Monks-2	432	2	6	0	6	no
Monks-3	432	2	6	0	6	no
*Mushroom	8124	2	22	0	2	yes
New-Thyroid	215	3	5	5	0	no
*Nursery	12960	5	8	0	8	no
*Optdigit	5620	10	64	64	0	no
*Pendigit	10992	10	16	16	0	no
Pima-Indians-Diabetes	768	2	8	8	0	no
*Segmentation	2310	7	19	19	0	no
*Shuttle	14500	7	9	9	0	no
Sonar	208	2	60	60	0	no
Soyabean	683	19	35	0	19	yes
Tic-Tac-Toe	958	2	9	0	9	no
Voting	435	2	16	0	16	yes
Vowel	990	11	10	10	0	no
Wdbc	569	2	30	30	0	no
Wine	178	3	13	13	0	no
Wpbc	198	2	33	33	0	yes
Yeast	1484	10	8	8	0	no
Zoo	101	7	16	0	16	no

Table A.1: Detailed information for the 35 tested data sets in the UCI Repository.

* 1,500 instances are randomly selected in conducting experiments on PGF algorithms.

Appendix B

Detailed Experimental Results for PGF

	PGF1-ENN		PGF1-RT3		PGF1-ACC	
Data	accuracy	size	accuracy	size	accuracy	size
Ab	0.552 (0.078)	0.147	0.555 (0.154)	0.090	0.489 (0.094)	0.049
Am	0.789 (0.045)	0.226	0.797 (0.083)	0.049	0.766 (0.035)	0.032
Au	0.614 (0.113)	0.237	0.606 (0.113)	0.155	0.575 (0.168)	0.118
Ba	0.861 (0.047)	0.152	0.831 (0.069)	0.039	0.824 (0.084)	0.015
Bc	0.960 (0.041)	0.121	0.957 (0.031)	0.009	0.961 (0.041)	0.026
Ca	0.932 (0.036)	0.208	0.931 (0.019)	0.048	0.902 (0.022)	0.069
Cs	0.832 (0.044)	0.058	0.845 (0.040)	0.023	0.845 (0.042)	0.023
Ec	0.860 (0.040)	0.101	0.872 (0.074)	0.036	0.806 (0.087)	0.034
Gl	0.588 (0.184)	0.058	0.570 (0.172)	0.047	0.523 (0.051)	0.033
He	0.813 (0.090)	0.027	0.819 (0.079)	0.033	0.805 (0.138)	0.019
Io	0.880 (0.077)	0.109	0.838 (0.089)	0.035	0.855 (0.065)	0.022
Ir	0.913 (0.090)	0.038	0.940 (0.054)	0.038	0.927 (0.112)	0.023
Le	0.710 (0.045)	0.335	0.659 (0.032)	0.189	0.521 (0.075)	0.084
Li	0.559 (0.121)	0.152	0.577 (0.081)	0.074	0.545 (0.089)	0.067
M1	0.919 (0.070)	0.238	0.928 (0.110)	0.151	0.826 (0.109)	0.173
M2	0.939 (0.079)	0.125	0.968 (0.022)	0.106	0.915 (0.044)	0.097
M3	0.948 (0.055)	0.055	0.950 (0.052)	0.055	0.914 (0.096)	0.065
Mu	0.997 (0.008)	0.011	0.996 (0.012)	0.009	0.993 (0.011)	0.010
Ne	0.926 (0.032)	0.060	0.889 (0.110)	0.031	0.852 (0.098)	0.028
Nu	0.847 (0.057)	0.156	0.834 (0.049)	0.074	0.841 (0.043)	0.089
Op	0.958 (0.027)	0.328	0.916 (0.036)	0.041	0.911 (0.018)	0.042
Pe	0.973 (0.030)	0.234	0.960 (0.031)	0.064	0.928 (0.025)	0.066
Pi	0.722 (0.063)	0.209	0.759 (0.121)	0.007	0.706 (0.116)	0.059
Se	0.948 (0.007)	0.236	0.936 (0.028)	0.071	0.911 (0.028)	0.076
Sh	0.984 (0.042)	0.209	0.974 (0.034)	0.022	0.981 (0.036)	0.061
Sn	0.833 (0.201)	0.472	0.697 (0.075)	0.107	0.716 (0.142)	0.051
Sb	0.889 (0.046)	0.221	0.867 (0.058)	0.090	0.757 (0.061)	0.069
Tt	0.881 (0.037)	0.326	0.859 (0.046)	0.136	0.821 (0.037)	0.083
Vo	0.919 (0.039)	0.104	0.915 (0.038)	0.025	0.924 (0.030)	0.025
Vw	0.959 (0.035)	0.252	0.914 (0.029)	0.198	0.632 (0.069)	0.096
Wd	0.954 (0.036)	0.195	0.949 (0.038)	0.014	0.933 (0.037)	0.037
Wi	0.938 (0.033)	0.112	0.948 (0.094)	0.032	0.932 (0.100)	0.021
Wp	0.747 (0.135)	0.033	0.703 (0.220)	0.056	0.728 (0.143)	0.019
Ye	0.560 (0.050)	0.067	0.516 (0.079)	0.074	0.524 (0.044)	0.067
Zo	0.920 (0.101)	0.077	0.900 (0.100)	0.089	0.830 (0.201)	0.077
Average	0.846	0.163	0.834	0.066	0.798	0.055

Table B.1: The average classification accuracy and data retention rate (size) of 10-fold cross-validation for PGF1-ENN, PGF1-RT3, and PGF1-ACC. The standard deviation of classification accuracy is given inside the bracket.

	PGF2-ENN		PGF2-RT3		PGF2-ACC	
Data	accuracy	size	accuracy	size	accuracy	size
Ab	0.616 (0.125)	0.273	0.542 (0.117)	0.137	0.586 (0.163)	0.202
Am	0.774 (0.084)	0.346	0.772 (0.156)	0.090	0.786 (0.076)	0.101
Au	0.644 (0.271)	0.334	0.588 (0.154)	0.169	0.672 (0.135)	0.130
Ba	0.855 (0.045)	0.178	0.855 (0.060)	0.033	0.853 (0.041)	0.012
Bc	0.960 (0.049)	0.087	0.966 (0.062)	0.012	0.963 (0.037)	0.026
Ca	0.933 (0.021)	0.633	0.946 (0.020)	0.076	0.935 (0.019)	0.176
Cs	0.829 (0.061)	0.054	0.826 (0.040)	0.034	0.842 (0.041)	0.019
Ec	0.854 (0.100)	0.194	0.852 (0.084)	0.079	0.833 (0.074)	0.117
Gl	0.644 (0.128)	0.108	0.550 (0.283)	0.066	0.649 (0.213)	0.051
He	0.832 (0.121)	0.081	0.805 (0.097)	0.031	0.818 (0.097)	0.031
Io	0.858 (0.135)	0.203	0.872 (0.088)	0.060	0.874 (0.074)	0.035
Ir	0.933 (0.104)	0.115	0.907 (0.089)	0.050	0.933 (0.104)	0.073
Le	0.716 (0.081)	0.609	0.661 (0.057)	0.240	0.701 (0.059)	0.206
Li	0.570 (0.165)	0.271	0.620 (0.076)	0.078	0.585 (0.119)	0.072
M1	0.889 (0.074)	0.623	0.944 (0.092)	0.243	0.939 (0.082)	0.250
M2	0.957 (0.032)	0.488	0.960 (0.029)	0.165	0.951 (0.062)	0.120
M3	0.953 (0.067)	0.190	0.951 (0.036)	0.055	0.950 (0.081)	0.093
Mu	0.997 (0.009)	0.114	0.990 (0.010)	0.007	0.995 (0.008)	0.010
Ne	0.934 (0.113)	0.206	0.934 (0.087)	0.036	0.925 (0.075)	0.059
Nu	0.842 (0.031)	0.346	0.844 (0.024)	0.077	0.853 (0.035)	0.144
Op	0.951 (0.038)	0.350	0.919 (0.037)	0.059	0.946 (0.032)	0.114
Pe	0.979 (0.009)	0.417	0.954 (0.007)	0.071	0.972 (0.028)	0.104
Pi	0.709 (0.086)	0.223	0.716 (0.111)	0.026	0.715 (0.078)	0.046
Se	0.950 (0.012)	0.593	0.941 (0.016)	0.086	0.952 (0.015)	0.143
Sh	0.986 (0.039)	0.295	0.983 (0.042)	0.023	0.985 (0.042)	0.142
Sn	0.818 (0.103)	0.468	0.740 (0.062)	0.122	0.789 (0.090)	0.131
Sb	0.895 (0.034)	0.445	0.891 (0.054)	0.121	0.861 (0.068)	0.156
Tt	0.874 (0.045)	0.473	0.845 (0.032)	0.139	0.865 (0.061)	0.197
Vo	0.915 (0.070)	0.123	0.915 (0.033)	0.042	0.926 (0.047)	0.061
Vw	0.968 (0.040)	0.686	0.923 (0.045)	0.275	0.944 (0.039)	0.210
Wd	0.940 (0.051)	0.291	0.942 (0.052)	0.023	0.942 (0.053)	0.092
Wi	0.955 (0.035)	0.177	0.955 (0.025)	0.043	0.949 (0.050)	0.086
Wp	0.763 (0.148)	0.093	0.717 (0.080)	0.037	0.748 (0.120)	0.015
Ye	0.549 (0.052)	0.292	0.561 (0.041)	0.081	0.523 (0.056)	0.103
Zo	0.930 (0.108)	0.120	0.920 (0.071)	0.098	0.920 (0.101)	0.085
Average	0.851	0.300	0.837	0.085	0.848	0.103

Table B.2: The average classification accuracy and data retention rate (size) of 10-fold cross-validation for PGF2-ENN, PGF2-RT3 and PGF2-ACC. The standard deviation of classification accuracy is given inside the bracket.

	Pure Filtering						Pure Abstraction	
	ENN		RT3		ACC		ABS	
Data	accuracy	size	accuracy	size	accuracy	size	accuracy	size
Ab	0.640 (0.094)	0.763	0.621 (0.152)	0.319	0.489 (0.093)	0.084	0.723 (0.150)	0.535
Am	0.799 (0.110)	0.787	0.794 (0.067)	0.136	0.764 (0.086)	0.094	0.746 (0.056)	0.188
Au	0.680 (0.175)	0.773	0.667 (0.116)	0.247	0.579 (0.187)	0.147	0.725 (0.102)	0.406
Ba	0.864 (0.044)	0.784	0.837 (0.065)	0.103	0.818 (0.038)	0.091	0.779 (0.076)	0.103
Bc	0.967 (0.039)	0.953	0.958 (0.039)	0.034	0.965 (0.035)	0.122	0.964 (0.028)	0.091
Ca	0.939 (0.015)	0.959	0.952 (0.018)	0.112	0.901 (0.019)	0.114	0.954 (0.014)	0.349
Cs	0.823 (0.048)	0.815	0.826 (0.035)	0.085	0.833 (0.034)	0.101	0.822 (0.057)	0.022
Ec	0.866 (0.056)	0.808	0.878 (0.059)	0.108	0.818 (0.090)	0.092	0.828 (0.069)	0.163
Gl	0.719 (0.267)	0.695	0.672 (0.341)	0.211	0.532 (0.101)	0.059	0.584 (0.173)	0.068
He	0.856 (0.157)	0.809	0.856 (0.208)	0.093	0.825 (0.095)	0.097	0.819 (0.128)	0.049
Io	0.846 (0.048)	0.868	0.869 (0.047)	0.068	0.866 (0.032)	0.099	0.892 (0.061)	0.196
Ir	0.953 (0.063)	0.954	0.947 (0.114)	0.080	0.933 (0.091)	0.118	0.927 (0.087)	0.097
Le	0.740 (0.054)	0.815	0.696 (0.050)	0.282	0.524 (0.085)	0.099	0.774 (0.048)	0.456
Li	0.597 (0.067)	0.623	0.566 (0.110)	0.218	0.563 (0.092)	0.080	0.571 (0.098)	0.153
M1	0.928 (0.082)	0.966	0.971 (0.071)	0.296	0.831 (0.099)	0.247	0.975 (0.040)	0.303
M2	0.979 (0.021)	0.997	0.984 (0.014)	0.182	0.933 (0.070)	0.271	0.962 (0.074)	0.129
M3	0.950 (0.069)	0.957	0.955 (0.069)	0.086	0.939 (0.074)	0.243	0.960 (0.050)	0.128
Mu	0.998 (0.007)	1.000	0.998 (0.007)	0.015	0.992 (0.011)	0.149	0.997 (0.008)	0.011
Ne	0.963 (0.048)	0.967	0.948 (0.073)	0.111	0.833 (0.097)	0.115	0.944 (0.042)	0.153
Nu	0.844 (0.032)	0.857	0.837 (0.052)	0.150	0.840 (0.046)	0.101	0.850 (0.023)	0.271
Op	0.959 (0.036)	0.981	0.928 (0.027)	0.100	0.919 (0.023)	0.104	0.956 (0.045)	0.254
Pe	0.985 (0.014)	0.987	0.959 (0.016)	0.098	0.930 (0.033)	0.109	0.977 (0.012)	0.279
Pi	0.753 (0.104)	0.704	0.719 (0.097)	0.141	0.711 (0.101)	0.076	0.730 (0.043)	0.050
Se	0.956 (0.011)	0.967	0.953 (0.032)	0.105	0.919 (0.033)	0.147	0.965 (0.016)	0.325
Sh	0.985 (0.048)	0.996	0.983 (0.045)	0.037	0.982 (0.036)	0.115	0.987 (0.045)	0.214
Sn	0.833 (0.231)	0.860	0.812 (0.046)	0.226	0.716 (0.195)	0.097	0.866 (0.084)	0.552
Sb	0.909 (0.066)	0.913	0.889 (0.051)	0.155	0.760 (0.051)	0.132	0.908 (0.043)	0.439
Tt	0.887 (0.031)	0.916	0.876 (0.025)	0.181	0.824 (0.040)	0.100	0.896 (0.018)	0.418
Vo	0.936 (0.048)	0.924	0.922 (0.098)	0.062	0.913 (0.060)	0.136	0.915 (0.099)	0.075
Vw	0.987 (0.015)	0.989	0.956 (0.017)	0.293	0.635 (0.071)	0.110	0.971 (0.033)	0.252
Wd	0.958 (0.031)	0.954	0.952 (0.041)	0.056	0.950 (0.022)	0.106	0.938 (0.045)	0.191
Wi	0.954 (0.054)	0.951	0.938 (0.125)	0.114	0.887 (0.107)	0.101	0.938 (0.075)	0.112
Wp	0.733 (0.104)	0.712	0.723 (0.153)	0.138	0.738 (0.170)	0.071	0.747 (0.129)	0.018
Ye	0.562 (0.031)	0.528	0.544 (0.040)	0.173	0.522 (0.038)	0.076	0.505 (0.068)	0.404
Zo	0.910 (0.087)	0.963	0.931 (0.059)	0.169	0.830 (0.201)	0.199	0.920 (0.101)	0.109
Ave.	0.865	0.871	0.855	0.142	0.800	0.120	0.858	0.216

Table B.3: The average classification accuracy and data retention rate (size) of 10-fold cross-validation for pure filtering methods, namely, ENN, RT3, ACC, as well as the pure abstraction method. The standard deviation of classification accuracy is given inside the bracket.