# Discovering Useful Concept Prototypes for Classification Based on Filtering and Abstraction

Wai Lam, *Member*, *IEEE*, Chi-Kin Keung, and Danyu Liu

**Abstract**—The nearest-neighbor algorithm and its derivatives have been shown to perform well for pattern classification. Despite their high classification accuracy, they suffer from high storage requirement, computational cost, and sensitivity to noise. We develop a new framework, called ICPL (Integrated Concept Prototype Learner), which integrates instance-filtering and instance-abstraction techniques by maintaining a balance of different kinds of concept prototypes according to instance locality. The abstraction component, based on typicality, employed in our ICPL framework is specially designed for concept integration. We have conducted experiments on a total of 50 real-world benchmark data sets. We find that our ICPL framework maintains or achieves better classification accuracy and gains a significant improvement in data reduction compared with existing filtering and abstraction techniques as well as some existing techniques.

**Index Terms**—Prototype learning, classification, instance abstraction, machine learning, data mining.

## 1 Introduction

The nearest-neighbor (NN) algorithm and its derivatives have been shown to perform well for pattern classification in many domains [9]. These algorithms store the entire training set and classify unseen cases by finding the class labels of instances which are closest to them. Despite their high classification accuracy, they suffer from high storage requirement and sensitivity to noise. Storing the entire data instances also leads to high-computational cost during the operational classification of a new data instance. This computational inefficiency becomes more serious if the classification needs to be done online or real time. These problems can be solved by reducing the large training set to a small, representative prototype set. Removing nonrepresentative and noisy instances can reduce storage requirement and computational cost while maintaining or even improving the classification accuracy. Two lines of research have been proposed to learn good prototypes [24]. One technique is known as an instance-filtering approach. Instance-filtering techniques reduce data set by retaining representative instances from the original data set. The other line of research can be regarded as instance-abstraction approach which reduces the data set by generating artificial prototypes summarizing representative characteristics of similar instances. In our previous work, a framework, called PGF, for integrating the two methods was proposed [23], [26]. Experimental results show that PGF gains improved performance over its components methods.

In this paper, we continue to pursue an extensive investigation of integrating the two methods. We identify different kinds of prototypes according to instance locality, i.e., the locations of instances retained. We consider three kinds of prototypes obtained by retaining, namely, 1) center instances, 2) border instances, and 3) nonborder instance, (i.e., removing border instances). We propose a new framework for discovering good prototypes, called ICPL (Integrated Concept Prototype Learner). Under this framework, two kinds of concept prototypes are separately learned by abstraction and filtering techniques. The concept prototypes are then integrated by maintaining a balance of different kinds of prototypes based on locality using specially designed integration methods. As for the abstraction component in ICPL, existing methods such as clustering require high-computational time; they are incapable of handling large data sets [13]. To mitigate this problem, we propose a novel abstraction method, based on typicality, with relatively low-computational cost.

In experiments on 35 real-world benchmark data sets, the classification accuracy and data retention rate of each variant of our ICPL are investigated. The performance is compared with those of pure instance-filtering and pure instance-abstraction techniques as well as some existing techniques. We have also conducted another comparative experiment with existing methods on 15 extra unseen data sets. Empirical results show that the ICPL framework maintains or achieves better classification accuracy and gains a significant improvement in data reduction compared with state-of-the-art existing methods.

Section 2 describes the related work. Section 3 describes the framework of ICPL. Section 4 presents two core components of ICPL, namely the abstraction component and the filtering component. Section 5 discusses the experiments using benchmark data. Section 6 summarizes our contributions and future work.

- *W. Lam and C.-K. Keung are with the Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Shatin, Hong Kong. E-mail: {wlam, ckkeung}@se.cuhk.edu.hk.*
- *D. Liu is with the University of Alabama at Birmingham, 820 18th Street South, Hixon Hall, Room 509, Birmingham, AL 35205. E-mail: liudy@cis.uab.edu.*

*Manuscript received 3 May 2001; revised 18 Sept. 2001; accepted 3 Oct. 2001. Recommended for acceptance by W. Freeman.*
*For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 114091.*

## 2 Related Work

### 2.1 Instance-Filtering

Instance-filtering methods make use of editing rules to select representative instances. Previous instance-filtering algorithms mainly differ from search directions and locations of instances selected. Search directions include instance inclusion, instance removal, and a combination of inclusion and removal. In terms of locations of instances selected, there are three broad categories retaining, namely, center instances, border instances, and nonborder instances.

Hart proposes a Condensed Nearest Neighbor (CNN) which is probably the earliest method to select representative instances [22]. CNN starts by randomly storing one instance for each class as the initial subset and stores instances misclassified by the current subset. A top-down variant of CNN, called Reduced Nearest Neighbor (RNN), is proposed by Gates which removes instance if the removal does not cause any misclassification of other instances [18]. The Edited Nearest Neighbor (ENN) algorithm proposed by Wilson eliminates instances misclassified by their *k*-nearest neighbors [37]. Zhang introduces Typical Instance-Based Learning which stores typical instance in the region centers [42]. Dasarathy proposes a new editing rule known as Minimal Consistent Set (MCS) by considering nearest unlike neighbor [10]. 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. Later, the synergy among NN editing and condensing techniques such as MCS has been exploited to minimize the computational costs [11]. Ferri et al. propose a modified optimal editing for NN classification [17].

A noise-tolerant instance filtering, called NTGrowth, is proposed by Aha et al. [2]. Later, Aha et al. formalize NTGrowth to the well-known IB2 and IB3 algorithms which are based on CNN storing misclassified instances [1]. IB2 is similar to CNN except that instances are normalized by the range of attributes and missing value are tackled while IB3 only accepts instances with a relatively high-classification accuracy compared with the frequency of the observed class. The two algorithms provide noise tolerance. Wilson and Martinez introduce an instance pruning technique, called RT3, removing an instance by considering its *associates*, instances in the current selected instance set having it as one of their *k*-nearest neighbors [38]. RT3 employs ENN to filter out noise first and removes an instance if most of its associates are correctly classified without it. They further refine this technique to form DROP1-DROP5 [39] and the Integrated Decremental Instance-Based Learning which combines confidence and cross-validation accuracy in the distance measure [40].

Filtering methods are usually simpler and faster. The behavior such as the convergence of simple instance-filtering algorithms has been investigated [24]. Despite their simplicity, some filtering methods can gain accuracy comparable or even superior to pure NN classifier and also retain a consistent subset. As filtering techniques select representative instances from the original instance set, the learned concept is represented in simple ways as the original instance. Thus, the learned concept is easily comprehended. Since filtering rules can be designed to filter in or out different instances such as border [2], [18], [22], [38], central points [33], [37], and even noise and outliers [2], [22], [38], one can easily apply different rules simultaneously or separately to filter instances. However, instance-filtering methods have some drawbacks. They assume that ideal examples can be found in the original data set. This limits the representation power of filtering methods. The generalization power is also limited by only selecting the original data. Some methods enforcing the edited subset to be consistent may result in overfitting [25]. Noisy instances at the border are usually selected by boundary instance retaining methods. For methods retaining central points, data reduction rate is usually low compared with instance-abstraction [33], [37]. Some methods are sensitive to the order of presentation of instances [1], [18], [22].

### 2.2 Instance-Abstraction

Another approach for finding representative instances is the instance-abstraction method which generates prototypes by abstracting the original instances. Chang's method learns representative instances by merging similar ones [6]. It iteratively merges two closest instances and summarizes them by taking the weighted average of them. Bezdek et al. modify Chang's method which averages instances using simple mean and merges instances of the same class only [3]. Bradshaw introduces the *Disjunctive Spanning* (DS) which merges instances with the ones they can be correctly classified [4]. Kibler and Aha improve DS by using an *adaptive threshold* to limit the distance between two merged instances [24]. Hamamoto et al. introduce a bootstrap technique which can generate bootstrap prototypes by locally combining original training instances [21].

An algorithm, called Nested Generalized Exemplar (NGE), is proposed by Salzberg which stores instances as hyperrectangles [29]. Wettschereck combines the NGE with KNN to form a hybrid algorithm [36]. However, this algorithm stores the entire data set in memory. Recently, an instance-abstraction algorithm, called FAMBL, in language learning task is proposed by van den Bosch. It forms hyperrectangles like NGE but a different instance merging procedure is used [34].

Datta and Kibler introduce the Prototype Learner (PL) which learns artificial instances for each class by generalization of representative instances in nominal domains [12]. Then, they propose the Symbolic Nearest Mean Classifiers (SNMC) [14] which attempts to learn a single prototype for each class using a modified Value Difference Metric proposed by Cost and Salzberg to weigh symbolic features [8]. SNMC uses k-means clustering to group instances of the same class and create artificial instances using cluster means. A technique known as *squashing* is proposed to scale down the data set by exploiting the statistical property of the instances [16]. However, this technique does not make use of the class label information if it is employed in classification problems. A kind of approach is known as Learning 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 [41]. Sato employs this technique for generating abstracted prototypes by discriminative learning [30].

Artificial prototypes generated by instance-abstraction techniques can be more representative than the counterparts obtained by instance-filtering techniques. Instance-abstraction technique usually remove border points so that smoother decision boundaries can be obtained [13]. They usually gain a larger data reduction rate than instance-filtering techniques. Instance-abstraction techniques typically has a

greater generalizing power. Through abstraction instances, the most common characteristic of a concept can be learned and the learned concept has less risk to overfit the original data. As for noise tolerance, abstraction methods can generalize away mislabeled instances in compact region by merging them with other instances rather than just selecting them as prototypes. On the other hand, artificial instances by abstraction and relabeling may be nonprototypical [4]. Therefore, learned prototypes are less reasonable and comprehensible than those selected by instance-filtering methods. For example, it is not easy to find an interpretation for the average value of discrete features computed. A good resubstitution performance is also not guaranteed by relabeling merged instances [24], [25]. The generated subset may even contain a mislabeled instance so that unseen cases will be misclassified. Besides, simple abstraction also fails to describe concepts formed by concave decision boundaries. This leads to an inferior classification accuracy in some domains. Though abstraction methods can generalize away mislabeled instances in compact regions, they cannot do so on outliers.

### 2.3 Approach Based on Integration

Domingos proposes an integrated technique, called the RISE algorithm, combining instance-based learning and rule induction [15]. In 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. Golding and Rosenbloom proposes a technique combining rule-based and case-based reasoning [19]. Their work focuses on the reasoning problem while our ICPL framework tackles the learning problem.

## 3 Our ICPL Framework

Instance abstraction and filtering methods learn prototypes in different ways leading to different natures of the prototypes learned. We differentiate different kinds of prototypes according to the locations of the instances retained. They are prototypes obtained by retaining, namely, 1) center instances, 2) border instances, and 3) nonborder instances. We investigate techniques integrating the strengths of combining different kinds of prototypes learned by abstraction and filtering.

To achieve this, we develop Integrated Concept Prototype Learning (ICPL) method. ICPL first performs filtering and abstraction on the training set separately. After this step, two kinds of concept prototypes, one for each method, will be learned. The abstraction component in ICPL, based on typicality, is good at retaining nonborder instances and conducts abstraction on those instances. The abstracted prototypes have the same representation as an instance. As for the filtering component in ICPL, we investigate different filtering algorithms for this component. Next, these two kinds of prototypes are integrated in an attempt to unify the strengths of the prototypes to form the final concept prototype set.

Fig. 1 shows our ICPL framework. Given a training set $T$, ICPL will first perform abstraction and filtering on it. Let the prototype sets learned from abstraction and filtering be $C_1$ and $C_2$, respectively, and $S$ be the resulting prototype set. At the beginning, a base prototype set $S$ is initiated by either one

Framework **ICPL**(Training Set $T$).
1 Set $C_1 =$ **Abstraction**$(T)$.
2 Set $C_2 =$ **Filtering**$(T)$.
3 Set $S = C_i$; i=1 or 2.
4 For each prototype $P$ in $C_{3-i}$ :
5 expand $S$ if $P$ is good.
6 For each prototype $P$ in $C_i$ :
7 refine $S$ if $P$ is redundant.
8 Return $S$.

Fig. 1. The ICPL framework.

of $C_1$ or $C_2$. Then, $S$ will be processed by two phases, namely, the *expansion* phase (Steps 4 and 5) and the *refinement* phase (Steps 6 and 7). During the expansion phase, each prototype in the counterpart prototype set will be examined. If the examined prototype is beneficial to $S$, then it will be added to $S$. After that, $S$ will be processed through the refinement phase. In the refinement phase, the initial prototypes in $S$ are examined and low quality prototypes are removed.

Some filtering methods are strong in detecting boundary instances so that complex concepts can be represented. In contrast, abstraction methods are strong in summarizing instances in compact regions to represent simple concepts leading to a low data retention rate. 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. Our ICPL framework attempts to select good prototypes and maintain a balance of different kinds of prototypes based on instance locality.

Our ICPL framework has four different variants of algorithms, namely, ICPL1, ICPL2, ICPL3, and ICPL4. The abstraction and filtering components used in the ICPL framework will be described in later sections. We first present the details of each variant of algorithms.

**ICPL1**. Fig. 2 shows the ICPL1 algorithm. Let the prototype sets learned from abstraction and filtering be $C_1$ and $C_2$, respectively, and $S$ be the resulting prototype set. In the integration, we first retain all prototypes in $C_1$. Each prototype learned from $C_2$ will then be examined whether it should be added to the integrated prototype set. We first classify the training set $T$ by the abstracted prototype set $S$ using simple, nearest neighbor method. Then, a temporary prototype set $Tmp$ containing both the current prototype set $S$ and the examined prototype $P$ is created and evaluated using the training set $T$. If an instance in $T$ is incorrectly classified by $S$ but is correctly classified by $P$ in $Tmp$ (i.e., have $P$ as its nearest neighbor and have the same class label as $P$), the number of correct classification of $P$ will be incremented. On the other hand, if the instance is correctly classified by $S$ but is incorrectly classified by $Tmp$ due to the addition of $P$, the number of incorrectly classification of $P$ will be incremented. $P$ will only be accepted if number of correct classification is larger then that of incorrect classification. It ensures that the additional instance $P$ contributes to the classification accuracy of the learned prototype set on the original training set positively and concepts unsuccessfully learned by abstraction will be complemented by prototypes learned from filtering. Note that ICPL1 contains only the expansion phase in the ICPL framework.

```
Procedure ICPL1(Training Set T).
1    Set C1 = Abstraction(T).
2    Set C2 = Filtering(T).
3    Set S = C1.
4    For each prototype P in C2 :
5        Set Tmp = S ∪ 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 ∪ P.
10   Return S.
```

Fig. 2. The ICPL1 algorithm.

**ICPL2**. Fig. 3 depicts the ICPL2 algorithm. It first retains all prototypes in $C_1$ and processes the filtering results $C_2$ similar to ICPL1. After all the prototypes in $C_2$ are processed, it starts the refinement phase where nonrepresentative prototypes are removed from the integrated prototype set by examining the abstracted prototypes in $S$. Similar to filtered prototypes, abstracted ones are also evaluated by the training set. When a prototype $P$ is examined in the refinement phase, a temporary prototype set $Tmp$ is also created with $P$ removed from the current prototype set $S$. $P$ is discarded if its removal does not hurt the classification accuracy of the current prototype set on training set. After the expansion phase, not all the prototypes learned by abstraction are essential to the formation of the class boundary. This pruning step helps eliminate those prototypes.

**ICPL3**. ICPL3 is similar to ICPL1 since it contains only the expansion phase. Unlike ICPL1 which retains abstraction results first at the beginning, ICPL3 retains all the filtered prototypes $C_2$ first, (i.e., change Step 3 of Fig. 2 to "Set $S = C_2$"). Using the filtered results, it then expands the final prototype set by including those good abstraction prototypes from $C_1$, (i.e., change Step 4 of Fig. 2 to "For each prototype $P$ in $C_1$"). Under this integration method, any useful concepts learned from abstraction will be retained in the final output prototype set.

**ICPL4**. Like ICPL2, ICPL4 includes both the expansion phase and refinement phase in the ICPL framework. However, instead of expanding the filtered prototypes, ICPL4 examines the abstracted ones first during the expansion phase, (i.e., change Step 3 and Step 4 of Fig. 3 to "Set $S = C_2$" and "For each prototype $P$ in $C_1$:" respectively). After adding abstracted prototypes, some of the filtered ones may become nonrepresentative. Therefore, in ICPL4, we refine the filtered prototypes so that the data retention rate of the result prototype set can be improved without hurting the classification accuracy, (i.e., change Step 10 of Fig. 3 to "For each prototype $P$ in $C_2$:").

## 4 Abstraction and Filtering Components in ICPL

We develop an abstraction method, called *Typical Prototype Abstraction* (TPA), which is designed for concept integration in ICPL. TPA abstracts typical instances to form prototype sets. It makes use of the *typicality* measure proposed by Zhang [42] in the abstraction process. We first conduct an in-depth investigation on how typicality can help abstraction. Then, we describe our TPA algorithm. Investigation of TPA on artificial data shows that it is able to generalize instances efficiently and tolerate the presence of noise.

### 4.1 Investigation on Typicality

Typicality was first proposed by Zhang to indicate how typical an instance with respect to the decision boundary is [42]. It makes use of the intraconcept similarity and the interconcept similarity to calculate the typicality of an

```
Procedure ICPL2(Training Set T).
1    Set C1 = Abstraction(T).
2    Set C2 = Filtering(T).
3    Set S = C1.
4    For each prototype P in C2 :
5        Set Tmp = S ∪ 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 ∪ P.
10   For each prototype P in C1:
11       Set Tmp = S \ 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 \ P.
16   Return S.
```

Fig. 3. The ICPL2 algorithm.

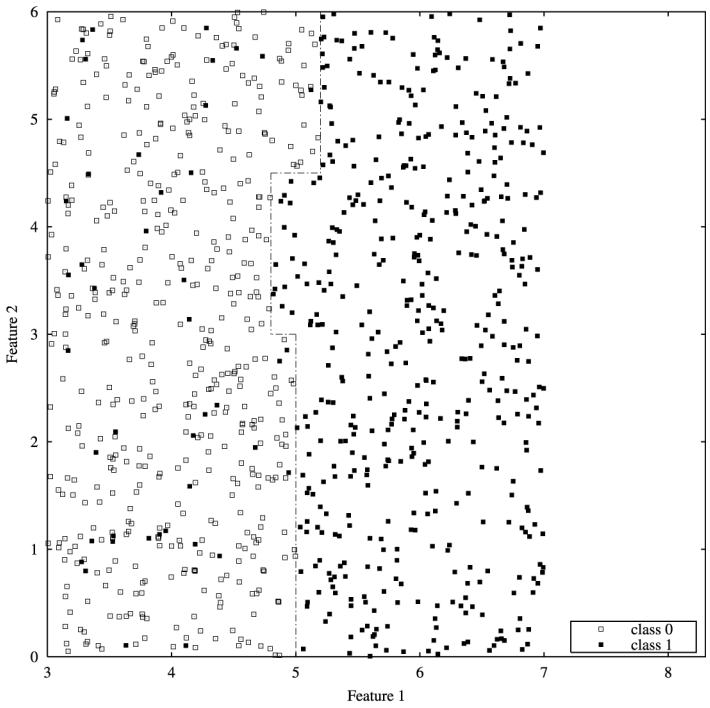



Fig. 4. A two-class data set with noise.

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 which is a heterogeneous distance function similar to the one proposed in [38] to measure the distance between instances with continuous and discrete feature types. We normalize all the continuous features by their feature ranges. Euclidean distance is employed to calculate distances between continuous feature values whereas a simplified version of Value Difference Metric (*vdm*) [32] is used to handle discrete features. To ensure an even contribution of continuous and discrete features, we normalize the distance of each feature by its maximum distance. For a 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.

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.

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

To aid the design of using typicality in ICPL, we conduct an in-depth analysis on typicality. Fig. 4 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. We deliberately introduce some noise to class 0 to investigate its effects on typicality. 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 Fig. 5. We can see clearly from Fig. 5 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 [42]. This information is useful in the design of our abstraction component in ICPL. Instances with large distance from the boundary 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 toward 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 Fig. 5 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.

### 4.2 Typicality in Abstraction

Motivated by above observations, we design a procedure, called *IdentifyBorder*, to identify border instances for each

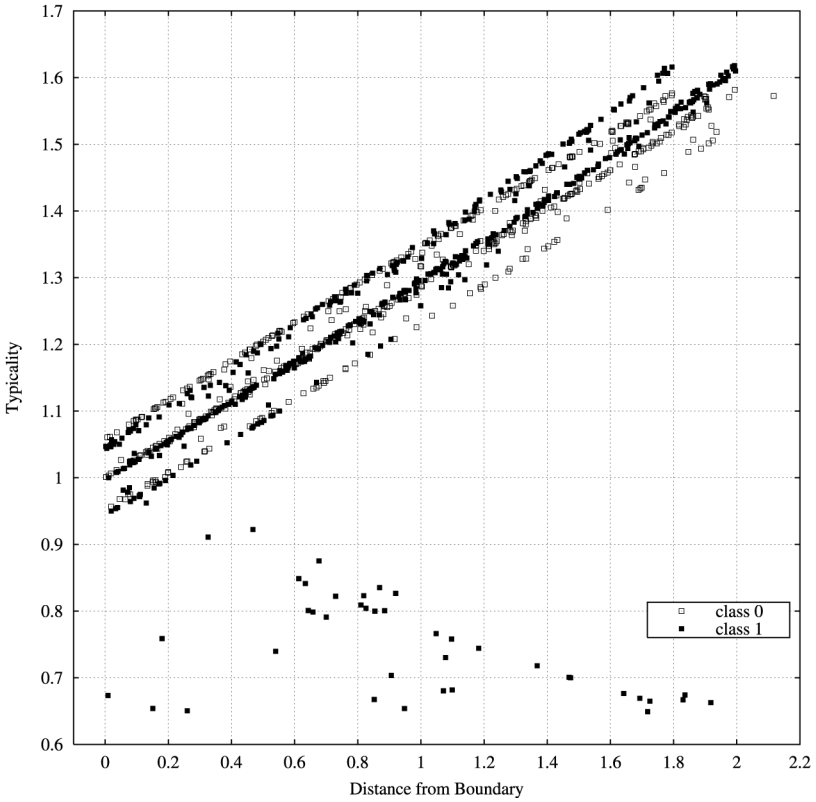



Fig. 5. Typicality versus distance from boundary.

class based on typicality. Fig. 6 shows the pseudocode of the procedure. 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, instances farthest from boundaries will be processed first and border or noisy ones will be handled at appropriate time during abstraction. Using the typicality calculated, the procedure can then focus on identifying nonborder 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 nonborder points of all classes. Besides, in order 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 processing, we can then perform abstraction on nonborder instances in descending order of their typicality in the TPA algorithm presented below.

### 4.3 TPA—Abstraction Component in ICPL

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

```
Procedure IdentifyBorder(Training Set T).
1   For each instance I in T :
2       Find Typ(I), typicality of I.
3   For each class C :
4       Find T_mean(C), the mean of typicality of class C instances.
5       Find T_sd(C), the standard deviation of typicality of class C instances.
6       Sort class C instances in descending order of typicality.
7   For each class C :
8       For each instance I of class C in T :
9           If ( Typ(I) < ( T_mean(C) - T_sd(C) ) :
10              Set Bp(I) = TRUE, indicating I is a border point.
11          Else :
12              Set Bp(I) = FALSE, indicating I is not a border point.
13  Return Bp.
```

Fig. 6. The IdentifyBorder function.

```
Procedure TPA(Training Set T).
1    Set S = empty.
2    Set Bp = IdentifyBorder(T).
3    For each class C :
4         Set I = first instance of class C in T.
5         For each non-border instance I ( i.e. Bp(I) = FALSE ) :
6              If ( I is not processed ) :
7                   Set P = Merge(T, S, Bp, I).
8                   If ( no. of instances in P > 1 ) :
9                        Add P to S.
10                  End If.
11             End If.
12             Set I = next instance of class C in T.
13        End For.
14   End For.
15   Return S.
```

Fig. 7. The TPA algorithm.

tolerance. Besides, in order to handle large data sets, it should have low-computational complexity. We first present the TPA algorithm.

Fig. 7 shows the TPA algorithm. The algorithm learns prototypes in a bottom-up fashion. It begins with an empty prototype set, $S$. The *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. TPA then learns prototypes for each class by merging nonborder instances. Specifically, prototypes are represented by the mean of all the merged instances for continuous features. As for discrete feature, we calculate the distances between all the possible values of the feature and feature values of all the merge instances. Feature value with minimum total distance will be chosen to represent the prototype. Also, the majority class of the merged instances becomes the class label of the prototype. At the beginning, the first nonborder 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 nonborder instances in each class are processed.

The *Merge* function is described in Fig. 8. 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 either 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 usually 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. On the other hand, if $N$ is a border instance, the further neighbor of $I$ will usually be of different class of $I$ also. The merging process continues until one of the following two situations is encountered.

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 $S$.

### 4.4 Filtering Component In ICPL

Different types of filtering methods target at retaining instances in different locations leading to different behaviors when integrated with abstraction techniques. We investigate three filtering components in our ICPL framework as presented below. The distance measure employed in the filtering components is similar to the one as described in Section 4.1.

**Retaining Nonborder Instances**. The first filtering component is the ENN method introduced by Wilson [37]. 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 some border instances as they usually have neighbors of different classes resulting in smoother class boundaries. It mainly retains intermediate and center instances.

**Retaining Center Instances.** We develop a filtering technique, called ACC, which tries to find center instances of compact regions by considering the classification performance of each prototype in the prototype set [23]. 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 nonrepresentative instances such as outliers and exceptions, will be removed as they usually have lower accuracy.

**Retaining Border Instances**. The third filtering component is called RT2 proposed by Wilson and Martinez [38]. Initially, each instance is considered as a prototype. ENN is applied first to filter out noisy instances and smooth class boundaries. Then, the presentation order of instances is

```
Procedure Merge(Training Set T, Prototype Set S, Border Flag Bp, Instance I).
1   Set P = I.
2   Set N = nearest neighbor of I in T.
3   Set C = class of I
4   While ( (class of N not equal C) Or (Not Bp(N)) ) :
5       If ( class of N not equal C ) :
        # N may be either a border point or a noise
6           Set N = next nearest neighbor of I in T.
7           If ( class of N not equal C ) :
            # regard N as a border point and stop the merging process
8               Return P.
9           End If.
10      End If.
11      If ( Bp(N) ) :
12          Return P.
13      Else :
14          If ( N is not merged before ) :
15              Merge P and N.
16              Set N = next nearest neighbor of I in T.
17          Else :
18              Find M, the prototype containing N from S.
19              Merge P with M.
20              Remove M from S.
21              Return P.
22          End If.
23      End If.
24  End While.
25  Return P.
```

Fig. 8. The Merge function.

sorted in descending order by the distance of an instance to its nearest unlike neighbor. It ensures that 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. 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.

## 5 Experimental Results

### 5.1 Experimental Setup

ICPL1-4 were tested on 35 benchmark data sets from the Machine Learning Database Repository [27]. Table 1 shows the data sets and their corresponding codes used in this paper. These data sets are collected from different real-world applications in various domains, such as the city-cycle fuel consumption (Am), Wisconsin breast cancer (Bc) and the Iris plant database (Ir). For each data set, we randomly partitioned the data into 10 even portions. Ten trials derived from 10-fold cross-validation were conducted for every set of experiments. The average classification accuracy and data retention rate of 10-fold cross-validation were obtained for each data set, where

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

and

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

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

In these experiments, we have investigated the performance of different variants of our ICPL framework. Each variant is constructed by adopting a particular ICPL method with a different filtering component. We have also conducted rigorous statistical analysis to compare the performance of different variants of ICPL methods.

### 5.2 Analysis of ICPL

In the first set of experiments, we investigated the effect of different kinds of filtering components on our ICPL framework. ICPLx-ENN, ICPLx-ACC, and ICPLx-RT2 refer to our ICPLx framework using ENN, ACC, and RT2, respectively, as the filtering component. (ICPLx can refer to ICPL1, ICPL2, ICPL3, or ICPL4.) The details of the filtering components are described in Section 4.4.

**ICPL-ENN**. Table 2 depicts the summary and statistical analysis of 10-fold cross-validation of 35 data sets for ICPLx-ENN framework, the pure abstraction algorithm TPA, and the pure filtering algorithm ENN. It shows the average classification accuracy (acc) and the average data retention rate (size) of 35 data sets. The detailed performance on each data set is given in Table 7. Generally, the results in Table 2 show that all four ICPL algorithms achieve good classification accuracy ranging from 0.858 to 0.869. The data retention rate of ICPL1-ENN, ICPL2-ENN, and ICPL4-ENN are good ranging from 0.097 to 0.102. Considering both the accuracy and retention rate, ICPL2-ENN performs the best among the other three ICPLx-ENN algorithms. The statistical analysis in Table 2 makes use of ICPL2-ENN as the basis for comparison. B/W denotes the number of data set for which ICPL2-ENN is statistically better or worse than the corresponding ICPLx-ENN algorithm. S.B/W denotes the number of data set for which ICPL2-ENN is statistically significantly better or significantly worse than

TABLE 1
Data Sets and Their Codes

| 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 | | |

the corresponding ICPLx-ENN algorithm. Wcx denotes the one-tail Wilcoxon Signed Ranks test [7] presenting the confidence level of the difference between ICPL2-ENN and other algorithm on the performance.

The pure filtering algorithm, ENN, achieves similar accuracy as ICPL algorithms but the retention rate is extremely poor. As mentioned before, ENN is a filtering algorithm removing mostly border instances. This accounts for the large data retention rate for ENN. Let us examine the comparative performance of ICPL1-ENN with ENN and TPA more closely. Recall that ICPL1-ENN can be viewed as a learning algorithm based on pure abstraction (i.e., TPA) with an expansion process using the prototypes learned by ENN. Comparing ENN with ICPL1-ENN, we find that the retention rate improves significantly from 0.876 to 0.097 with similar accuracy. It demonstrates that the abstracted prototypes generated by TPA can remove nonrepresentative prototypes in center and intermediate regions of a class. Comparing TPA with ICPL1-ENN, we find that the accuracy improves significantly. It shows that incorporating center and intermediate prototypes generated from ENN does help the abstracted prototype from TPA to describe some complex concepts. If we compare the performance of ICPL2-ENN with ICPL1-ENN, we find that there is an improvement in retention rate from 0.097 to 0.088. The statistical analysis confirms that ICPL2-ENN is statistically better than ICPL1-ENN in terms of data retention rate for all 35 data sets. Wilcoxon test on data retention rate also shows that ICPL2-ENN is better than ICPL1-ENN with very high confidence level (99.50 percent). It shows that the refinement step in ICPL2-ENN helps the learning algorithm. Recall that ICPL3-ENN employs ENN to find the initial prototype set. Comparing ENN with ICPL3-ENN, it shows that ICPL3-ENN also exhibits an extremely high retention rate similar to ENN while the accuracy remains unchanged. The reason is that prototypes in center and intermediate regions generated by ENN already cover the characteristics of abstracted prototypes generated by TPA. Hence, ICPL3-ENN is not a good integration method for border retaining filtering algorithms such as ENN. Comparing ICPL4-ENN with ICPL3-ENN, we find that there is a significant improvement in data retention rate from 0.881 to 0.102 without loss in accuracy. It illustrates that the extra refinement step in ICPL4-ENN is useful in generating representative prototypes.

**ICPL-ACC**. Table 3 depicts the summary and statistical analysis of 10-fold cross-validation of 35 data sets for ICPLx-ACC framework, the pure abstraction algorithm TPA, and the pure filtering algorithm ACC. It shows the average classification accuracy (acc) and the average data retention rate (size) of 35 data sets. The detailed performance

TABLE 2
The Summary and Statistical Analysis of 10-Fold Cross-Validation of 35 Data Sets for ICPLx-ENN, Pure Abstraction Method, and Pure Filtering Method

| | ICPL1-ENN | | ICPL2-ENN | | ICPL3-ENN | | ICPL4-ENN | | Pure abstraction TPA | | Pure filtering ENN | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | acc | size | acc | size | acc | size | acc | size | acc | size | acc | size |
| Ave | 0.863 | 0.097 | 0.864 | 0.088 | 0.869 | 0.881 | 0.858 | 0.102 | 0.773 | 0.045 | 0.869 | 0.876 |
| B/W | 13/14 | 35/0 | - | - | 16/18 | 35/0 | 23/12 | 31/4 | 34/0 | 1/34 | 17/17 | 35/0 |
| S.B/W | 5/6 | 35/0 | - | - | 6/14 | 35/0 | 8/4 | 27/1 | 30/0 | 1/33 | 6/13 | 35/0 |
| Wcx | 71.55 | 99.50 | - | - | -82.88 | 99.50 | 95.59 | 99.50 | 99.50 | -99.50 | -75.67 | 99.50 |

*The columns "acc" and "size" denote the classification accuracy and the data retention rate, respectively. Ave denotes the average performance across 35 data sets. B/W denotes the number of data sets for which ICPL2-ENN is statistically better or worse than the corresponding ICPLx-ENN algorithm. Wex denotes the one-tail Wilcoxon Signed Ranks test [7] presenting the confidence level of the difference between ICPL2-ENN and other algorithm on the classification accuracy performance.*

TABLE 3
The Summary and Statistical Analysis of 10-Fold Cross-Validation of 35 Data Sets for ICPLx-ACC, Pure Abstraction Method, and Pure Filtering Method

| | ICPL1-ACC | | ICPL2-ACC | | ICPL3-ACC | | ICPL4-ACC | | Pure abstraction TPA | | Pure filtering ACC | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | acc | size | acc | size | acc | size | acc | size | acc | size | acc | size |
| Ave | 0.836 | 0.066 | 0.843 | 0.056 | 0.825 | 0.151 | 0.843 | 0.061 | 0.773 | 0.045 | 0.787 | 0.133 |
| B/W | 19/12 | 35/0 | - | - | 26/9 | 35/0 | 15/20 | 25/9 | 32/3 | 7/28 | 34/1 | 32/3 |
| S.B/W | 9/2 | 35/0 | - | - | 16/5 | 35/0 | 8/5 | 23/4 | 26/0 | 4/27 | 28/0 | 32/3 |
| Wcx | 99.27 | 99.50 | - | - | 99.50 | 99.50 | -57.32 | 99.50 | 99.50 | -99.50 | 99.50 | 99.50 |

*Notations used in the table share the same interpreation as those in Table 2.*

on each data set is given in Table 8. The results in Table 3 show that all four ICPL algorithms achieve good classification accuracy ranging from 0.825 to 0.843. The data retention rate of ICPL1-ACC, ICPL2-ACC, and ICPL4-ACC are good ranging from 0.056 to 0.066. Generally, ICPLx-ACC achieves a better retention rate than ICPLx-ENN with a slight inferior in accuracy. Considering both the accuracy and retention rate, ICPL2-ACC performs the best among the other three ICPLx-ACC algorithms. The statistical analysis in Table 3 makes use of ICPL2-ACC as the basis for comparison.

The pure filtering algorithm, ACC, achieves inferior accuracy and retention than ICPL algorithms. As mentioned before, ACC is a filtering algorithm retaining center instances. Therefore, the retention rate of ACC is quite satisfactory. However, center instances are not effective in describing complex class boundaries leading to inferior accuracy. By integrating with TPA as in the ICPL algorithms, both the accuracy and retention rate in ICPL algorithms improve. Let us examine the comparative performance of ICPL1-ACC with ACC and TPA more closely. Comparing ACC with ICPL1-ACC, we find that the accuracy improves from 0.787 to 0.836 and the retention rate improves from 0.133 to 0.066. It demonstrates that the abstracted prototypes generated by TPA can complement with center retaining prototypes generated by ACC. Comparing TPA with ICPL1-ACC, we find that the accuracy improves from 0.773 to 0.836. It shows that integrating ACC prototypes help the prototype set initially obtained by TPA. The reason is that abstracted prototypes generated by TPA and filtered prototypes generated by ACC have different characteristics which can handle different kinds of instances. If we compare the performance of ICPL2-ACC with ICPL1-ACC, we find that there is an improvement in accuracy from 0.836 to 0.843 and the retention rate from 0.066 to 0.056. The statistical analysis confirms that ICPL2-ACC is statistically better than ICPL1-ACC. In terms of accuracy, there are 19 and nine data sets for which ICPL2-ACC achieve a better and significantly better respectively than ICPL1-ACC. Conversely only 12 and two data sets for which ICPL1-ACC is better and significantly better, respectively, than ICPL2-ACC. In terms of retention rate, ICPL2-ACC is significantly better than ICPL1-ACC for all 35 data sets. Wilcoxon test also demonstrates that ICPL2-ACC is better than ICPL1-ACC both for accuracy and retention rate with high confidence level of 99.27 percent and 99.50 percent, respectively. It shows that the refinement step in ICPL2-ACC is effective. When we compare ICPL3-ACC with ACC, we find that ICPL3-ACC has a better accuracy than ACC. The accuracy of ICPL3-ACC and ACC is 0.825 and 0.787, respectively. Recall that ICPL3-ACC is similar to ACC with an additional step for adding TPA prototypes. It indicates that TPA prototypes can assist ACC providing a better model for classification. Comparing ICPL4-ACC with ICPL3-ACC, we find that there is a slight improvement in accuracy from 0.825 to 0.843 and obvious improvement in data retention rate from 0.061 to 0.045. It illustrates that the extra refinement step in ICPL4-ACC is useful in generating representative prototypes.

**ICPL-RT2**. Table 4 depicts the summary and statistical analysis of 10-fold cross-validation of 35 data sets for ICPLx-RT2 framework, the pure abstraction algorithm TPA, and the pure filtering algorithm RT2. It shows the average

TABLE 4
The Summary and Statistical Analysis of 10-Fold Cross-Validation of 35 Data Sets for ICPLx-RT2, Pure Abstraction, and Pure Filtering Method

| | ICPL1-RT2 | | ICPL2-RT2 | | ICPL3-RT2 | | ICPL4-RT2 | | Pure abstraction TPA | | Pure filtering RT2 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | acc | size | acc | size | acc | size | acc | size | acc | size | acc | size |
| Ave | 0.861 | 0.083 | 0.864 | 0.074 | 0.867 | 0.175 | 0.866 | 0.095 | 0.773 | 0.045 | 0.867 | 0.172 |
| B/W | 14/13 | 35/0 | - | - | 13/21 | 35/0 | 15/20 | 35/0 | 34/1 | 2/33 | 14/21 | 35/0 |
| S.B/W | 8/3 | 35/0 | - | - | 3/12 | 35/0 | 3/11 | 35/0 | 30/0 | 1/33 | 3/13 | 35/0 |
| Wcx | 80.43 | 99.50 | - | - | -94.17 | 99.50 | -71.27 | 99.50 | 99.50 | -99.50 | -92.08 | 99.50 |

*Notations used in the table share the same interpreation as those in Table 2.*

TABLE 5
The Average Classification Accuracy (Acc) and Data Retention Rate (Size) of 10-Fold Cross-Validation for ICPL (ICPL2-RT2) and Some Existing Algorithms

| | ICPL | | RT3 | | KNN | SVM | C4.5 |
|---|---|---|---|---|---|---|---|
| | acc | size | acc | size | acc | acc | acc |
| Ave | 0.864 | 0.074 | 0.861 | 0.138 | 0.875 | 0.878 | 0.842 |
| B/W | - | - | 17/17 | 35/0 | 15/19 | 11/24 | 17/18 |
| S.B/W | - | - | 8/10 | 35/0 | 6/15 | 7/17 | 15/15 |
| Wcx | - | - | 68.19 | 99.50 | -86.49 | -96.39 | 70.47 |

*Notations used in the table share the same imterpretation as those in Table 2.*

TABLE 6
The Average Classification Accuracy (Acc) and Data Retention Rate (Size) of 10-Fold Cross-Validation for ICPL (ICPL2-RT2) and Some Existing Algorithms on 15 Extra Data Sets

| | ICPL | | RT3 | | KNN | SVM | C4.5 |
|---|---|---|---|---|---|---|---|
| | acc | size | acc | size | acc | acc | acc |
| Ave | 0.851 | 0.057 | 0.846 | 0.089 | 0.849 | 0.844 | 0.857 |
| B/W | - | - | 9/6 | 14/1 | 6/9 | 7/8 | 5/10 |
| S.B/W | - | - | 3/1 | 14/1 | 4/2 | 4/5 | 3-8 |
| Wcx | - | | 83.75 | 99.50 | 58.00 | -52.00 | -86.25 |

*Notations used in the table share the same interpretation as those in Table 2.*

classification accuracy (acc) and the average data retention rate (size) of 35 data sets. The detailed performance on each data set is given in Table 9. The results in Table 4 show that all four ICPL algorithms achieve good classification accuracy

TABLE 7
The Average Classification Accuracy (Acc) and Data Retention Rate (Size) of 10-fold Cross-Validation for ICPL-ENN Framework of Each Data Set

| | ICPL1-ENN | | | ICPL2-ENN | | | ICPL3-ENN | | | ICPL4-ENN | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | acc | std | size | acc | std | size | acc | std | size | acc | std | size |
| Ab | 0.605 | (0.081) | 0.210 | 0.620 | (0.080) | 0.193 | 0.639 | (0.091) | 0.784 | 0.649 | (0.064) | 0.205 |
| Am | 0.824 | (0.057) | 0.085 | 0.819 | (0.040) | 0.078 | 0.807 | (0.106) | 0.792 | 0.781 | (0.054) | 0.105 |
| Au | 0.650 | (0.105) | 0.161 | 0.650 | (0.102) | 0.149 | 0.680 | (0.175) | 0.785 | 0.672 | (0.182) | 0.153 |
| Ba | 0.887 | (0.037) | 0.053 | 0.885 | (0.033) | 0.049 | 0.864 | (0.044) | 0.784 | 0.851 | (0.036) | 0.098 |
| Bc | 0.967 | (0.044) | 0.013 | 0.969 | (0.038) | 0.010 | 0.967 | (0.039) | 0.953 | 0.957 | (0.030) | 0.024 |
| Ca | 0.938 | (0.016) | 0.061 | 0.949 | (0.013) | 0.051 | 0.939 | (0.015) | 0.959 | 0.949 | (0.012) | 0.062 |
| Cs | 0.842 | (0.054) | 0.045 | 0.842 | (0.054) | 0.041 | 0.828 | (0.052) | 0.816 | 0.817 | (0.047) | 0.070 |
| Ec | 0.860 | (0.075) | 0.071 | 0.857 | (0.073) | 0.060 | 0.860 | (0.065) | 0.810 | 0.848 | (0.094) | 0.077 |
| Gl | 0.672 | (0.334) | 0.185 | 0.663 | (0.304) | 0.165 | 0.709 | (0.280) | 0.709 | 0.690 | (0.299) | 0.169 |
| He | 0.824 | (0.125) | 0.077 | 0.817 | (0.143) | 0.065 | 0.856 | (0.157) | 0.810 | 0.831 | (0.187) | 0.079 |
| Io | 0.863 | (0.063) | 0.068 | 0.866 | (0.060) | 0.061 | 0.846 | (0.048) | 0.870 | 0.889 | (0.066) | 0.067 |
| Ir | 0.953 | (0.094) | 0.064 | 0.953 | (0.094) | 0.056 | 0.953 | (0.063) | 0.954 | 0.933 | (0.063) | 0.055 |
| Le | 0.702 | (0.060) | 0.201 | 0.699 | (0.060) | 0.192 | 0.743 | (0.071) | 0.829 | 0.695 | (0.075) | 0.209 |
| Li | 0.592 | (0.080) | 0.197 | 0.597 | (0.065) | 0.182 | 0.612 | (0.068) | 0.652 | 0.591 | (0.105) | 0.228 |
| M1 | 0.906 | (0.074) | 0.133 | 0.944 | (0.047) | 0.122 | 0.928 | (0.082) | 0.967 | 0.957 | (0.083) | 0.156 |
| M2 | 0.966 | (0.031) | 0.101 | 0.964 | (0.041) | 0.091 | 0.979 | (0.021) | 0.997 | 0.971 | (0.027) | 0.095 |
| M3 | 0.960 | (0.042) | 0.051 | 0.960 | (0.042) | 0.047 | 0.950 | (0.069) | 0.957 | 0.957 | (0.085) | 0.052 |
| Mu | 0.994 | (0.003) | 0.003 | 0.998 | (0.003) | 0.002 | 1.000 | (0.000) | 1.000 | 1.000 | (0.000) | 0.002 |
| Ne | 0.953 | (0.063) | 0.060 | 0.944 | (0.065) | 0.046 | 0.963 | (0.048) | 0.967 | 0.930 | (0.048) | 0.076 |
| Nu | 0.947 | (0.016) | 0.084 | 0.959 | (0.014) | 0.068 | 0.940 | (0.011) | 0.991 | 0.972 | (0.011) | 0.088 |
| Op | 0.966 | (0.008) | 0.044 | 0.964 | (0.006) | 0.041 | 0.984 | (0.010) | 0.987 | 0.957 | (0.019) | 0.046 |
| Pe | 0.986 | (0.004) | 0.028 | 0.985 | (0.005) | 0.025 | 0.993 | (0.006) | 0.994 | 0.983 | (0.005) | 0.027 |
| Pi | 0.734 | (0.107) | 0.122 | 0.722 | (0.113) | 0.112 | 0.752 | (0.094) | 0.715 | 0.695 | (0.064) | 0.137 |
| Se | 0.952 | (0.015) | 0.063 | 0.953 | (0.014) | 0.057 | 0.957 | (0.018) | 0.973 | 0.946 | (0.012) | 0.058 |
| Sh | 0.998 | (0.001) | 0.006 | 0.998 | (0.001) | 0.004 | 0.998 | (0.002) | 0.999 | 0.998 | (0.002) | 0.004 |
| Sn | 0.860 | (0.080) | 0.167 | 0.860 | (0.080) | 0.160 | 0.842 | (0.129) | 0.891 | 0.813 | (0.096) | 0.170 |
| Sb | 0.908 | (0.045) | 0.111 | 0.908 | (0.051) | 0.098 | 0.909 | (0.066) | 0.914 | 0.899 | (0.084) | 0.112 |
| Tt | 0.875 | (0.021) | 0.133 | 0.873 | (0.026) | 0.122 | 0.889 | (0.030) | 0.919 | 0.868 | (0.024) | 0.138 |
| Vo | 0.938 | (0.030) | 0.048 | 0.938 | (0.030) | 0.046 | 0.936 | (0.048) | 0.924 | 0.943 | (0.028) | 0.060 |
| Vw | 0.939 | (0.017) | 0.179 | 0.938 | (0.016) | 0.170 | 0.987 | (0.015) | 0.991 | 0.934 | (0.033) | 0.185 |
| Wd | 0.959 | (0.029) | 0.042 | 0.961 | (0.032) | 0.037 | 0.958 | (0.031) | 0.954 | 0.967 | (0.035) | 0.052 |
| Wi | 0.960 | (0.039) | 0.059 | 0.966 | (0.041) | 0.050 | 0.954 | (0.054) | 0.955 | 0.961 | (0.067) | 0.085 |
| Wp | 0.723 | (0.147) | 0.189 | 0.733 | (0.135) | 0.177 | 0.722 | (0.102) | 0.732 | 0.682 | (0.147) | 0.171 |
| Ye | 0.565 | (0.032) | 0.175 | 0.566 | (0.031) | 0.155 | 0.561 | (0.040) | 0.552 | 0.547 | (0.030) | 0.172 |
| Zo | 0.920 | (0.163) | 0.107 | 0.920 | (0.163) | 0.088 | 0.910 | (0.087) | 0.963 | 0.900 | (0.120) | 0.086 |
| Ave | 0.863 | | 0.097 | 0.864 | | 0.088 | 0.869 | | 0.881 | 0.858 | | 0.102 |

*The values inside the parenthesis denote the standard deviation.*

TABLE 8
The Average Classification Accuracy (Acc) and Data Retention Rate (Size) of 10-Fold Cross-Validation for ICPL-ACC Framework of Each Data Set

| | ICPL1-ACC | | | ICPL2-ACC | | | ICPL3-ACC | | | ICPL4-ACC | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | acc | std | size | acc | std | size | acc | std | size | acc | std | size |
| Ab | 0.586 | (0.114) | 0.153 | 0.620 | (0.095) | 0.136 | 0.572 | (0.138) | 0.158 | 0.575 | (0.128) | 0.112 |
| Am | 0.792 | (0.084) | 0.049 | 0.789 | (0.092) | 0.040 | 0.784 | (0.076) | 0.124 | 0.792 | (0.081) | 0.049 |
| Au | 0.658 | (0.146) | 0.134 | 0.650 | (0.124) | 0.125 | 0.685 | (0.116) | 0.209 | 0.725 | (0.108) | 0.126 |
| Ba | 0.847 | (0.047) | 0.029 | 0.859 | (0.036) | 0.026 | 0.781 | (0.050) | 0.119 | 0.830 | (0.049) | 0.052 |
| Bc | 0.966 | (0.039) | 0.014 | 0.966 | (0.047) | 0.010 | 0.963 | (0.029) | 0.127 | 0.960 | (0.062) | 0.012 |
| Ca | 0.912 | (0.033) | 0.048 | 0.933 | (0.027) | 0.035 | 0.921 | (0.019) | 0.126 | 0.925 | (0.027) | 0.033 |
| Cs | 0.855 | (0.048) | 0.029 | 0.854 | (0.045) | 0.022 | 0.836 | (0.042) | 0.118 | 0.845 | (0.037) | 0.034 |
| Ec | 0.815 | (0.105) | 0.045 | 0.815 | (0.105) | 0.039 | 0.795 | (0.121) | 0.115 | 0.825 | (0.086) | 0.055 |
| Gl | 0.639 | (0.239) | 0.125 | 0.630 | (0.228) | 0.109 | 0.654 | (0.163) | 0.166 | 0.644 | (0.240) | 0.114 |
| He | 0.810 | (0.259) | 0.042 | 0.810 | (0.259) | 0.033 | 0.817 | (0.189) | 0.117 | 0.798 | (0.156) | 0.050 |
| Io | 0.875 | (0.077) | 0.045 | 0.877 | (0.070) | 0.033 | 0.852 | (0.079) | 0.122 | 0.869 | (0.029) | 0.038 |
| Ir | 0.940 | (0.081) | 0.054 | 0.940 | (0.097) | 0.042 | 0.933 | (0.091) | 0.125 | 0.953 | (0.063) | 0.038 |
| Le | 0.641 | (0.059) | 0.134 | 0.637 | (0.059) | 0.126 | 0.631 | (0.047) | 0.169 | 0.621 | (0.045) | 0.126 |
| Li | 0.586 | (0.054) | 0.121 | 0.591 | (0.131) | 0.097 | 0.533 | (0.063) | 0.160 | 0.565 | (0.126) | 0.098 |
| M1 | 0.829 | (0.074) | 0.115 | 0.870 | (0.043) | 0.102 | 0.853 | (0.124) | 0.259 | 0.889 | (0.120) | 0.121 |
| M2 | 0.944 | (0.089) | 0.096 | 0.951 | (0.061) | 0.087 | 0.948 | (0.055) | 0.275 | 0.960 | (0.063) | 0.086 |
| M3 | 0.957 | (0.062) | 0.046 | 0.953 | (0.064) | 0.045 | 0.923 | (0.079) | 0.256 | 0.957 | (0.091) | 0.053 |
| Mu | 0.993 | (0.004) | 0.003 | 0.999 | (0.003) | 0.001 | 1.000 | (0.001) | 0.208 | 1.000 | (0.001) | 0.002 |
| Ne | 0.903 | (0.052) | 0.025 | 0.898 | (0.047) | 0.018 | 0.898 | (0.036) | 0.122 | 0.917 | (0.036) | 0.032 |
| Nu | 0.904 | (0.021) | 0.063 | 0.925 | (0.008) | 0.039 | 0.905 | (0.012) | 0.128 | 0.928 | (0.005) | 0.039 |
| Op | 0.957 | (0.020) | 0.028 | 0.958 | (0.015) | 0.025 | 0.961 | (0.014) | 0.103 | 0.957 | (0.018) | 0.025 |
| Pe | 0.977 | (0.005) | 0.021 | 0.976 | (0.005) | 0.019 | 0.980 | (0.009) | 0.115 | 0.977 | (0.012) | 0.018 |
| Pi | 0.716 | (0.083) | 0.072 | 0.732 | (0.080) | 0.058 | 0.712 | (0.090) | 0.125 | 0.734 | (0.076) | 0.066 |
| Se | 0.934 | (0.017) | 0.043 | 0.934 | (0.010) | 0.037 | 0.928 | (0.024) | 0.174 | 0.938 | (0.027) | 0.042 |
| Sh | 0.997 | (0.001) | 0.005 | 0.997 | (0.002) | 0.004 | 0.997 | (0.002) | 0.114 | 0.997 | (0.002) | 0.003 |
| Sn | 0.713 | (0.177) | 0.097 | 0.727 | (0.144) | 0.087 | 0.732 | (0.106) | 0.126 | 0.736 | (0.104) | 0.092 |
| Sb | 0.877 | (0.051) | 0.085 | 0.881 | (0.069) | 0.073 | 0.858 | (0.070) | 0.172 | 0.876 | (0.078) | 0.078 |
| Tt | 0.836 | (0.045) | 0.079 | 0.835 | (0.037) | 0.068 | 0.850 | (0.036) | 0.122 | 0.859 | (0.038) | 0.074 |
| Vo | 0.912 | (0.043) | 0.025 | 0.915 | (0.038) | 0.021 | 0.908 | (0.052) | 0.134 | 0.924 | (0.038) | 0.028 |
| Vw | 0.828 | (0.052) | 0.126 | 0.829 | (0.051) | 0.117 | 0.821 | (0.067) | 0.154 | 0.825 | (0.076) | 0.113 |
| Wd | 0.952 | (0.040) | 0.024 | 0.954 | (0.036) | 0.020 | 0.937 | (0.035) | 0.112 | 0.937 | (0.031) | 0.030 |
| Wi | 0.955 | (0.026) | 0.031 | 0.960 | (0.037) | 0.026 | 0.926 | (0.068) | 0.120 | 0.932 | (0.058) | 0.035 |
| Wp | 0.687 | (0.146) | 0.090 | 0.763 | (0.193) | 0.059 | 0.592 | (0.173) | 0.128 | 0.773 | (0.182) | 0.068 |
| Ye | 0.546 | (0.031) | 0.111 | 0.554 | (0.035) | 0.093 | 0.502 | (0.030) | 0.162 | 0.541 | (0.046) | 0.102 |
| Zo | 0.910 | (0.087) | 0.094 | 0.910 | (0.087) | 0.081 | 0.890 | (0.098) | 0.215 | 0.920 | (0.119) | 0.086 |
| Ave | 0.836 | | 0.066 | 0.843 | | 0.056 | 0.825 | | 0.151 | 0.843 | | 0.061 |

*The values inside the parenthesis denote the standard deviation.*

ranging from 0.861 to 0.867. The data retention rate of ICPL1-RT2, ICPL2-RT2, and ICPL4-RT2 are good ranging from 0.074 to 0.095. Generally, ICPLx-RT2 achieves better retention rate with similar accuracy as ICPLx-ENN and better accuracy with a slight inferior retention than ICPLx-ACC. Considering both the accuracy and retention rate, ICPL2-RT2 and ICPL4-RT2 achieve good performance among the four ICPLx-RT2 algorithms. Comparing with ICPL2-RT2 and ICPL4-RT2, we observe that ICPL4-RT2 has a slightly better accuracy, while ICPL2-RT2 clearly has a better retention rate. The statistical analysis in Table 4 makes use of ICPL2-RT2 as the basis for comparison.

As mentioned before, RT2 retains border instances while center and intermediate instances not affecting the decision boundary are discarded. Hence, RT2 can gain a very high classification accuracy, and at the same time, a satisfactory retention rate. However, we find that RT2 retains unnecessarily large amount of instances to describe even a simple concept. With the integration of abstraction component TPA, we attempt to eliminate some prototypes improving the retention rate, while maintaining the high classification accuracy. Let us examine the comparative performance of ICPL1-RT2 with RT2 and TPA more closely. Comparing RT2 with ICPL1-RT2, we find that the retention rate improves from 0.172 to 0.083. It demonstrates that the abstracted prototypes generated by TPA can complement with the filtered prototypes generated by RT2. Comparing TPA with ICPL1-RT2, we find that the accuracy improves from 0.773 to 0.861 with a loss in retention rate from 0.045 to 0.083. It is due to the fact that the border retaining prototypes generated by RT2 can represent complex boundaries which complement the abstracted prototypes generated by TPA. If we compare the performance of ICPL2-RT2 with ICPL1-RT2, we find that there is a slight improvement in the retention rate

TABLE 9
The Average Classification Accuracy (Acc) and Data Retention Rate (Size) of 10-Fold Cross-Validation for ICPL-RT2 Framework of Each Data Set

| | ICPL1-RT2 | | | ICPL2-RT2 | | | ICPL3-RT2 | | | ICPL4-RT2 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | acc | std | size | acc | std | size | acc | std | size | acc | std | size |
| Ab | 0.630 | (0.075) | 0.215 | 0.624 | (0.058) | 0.196 | 0.625 | (0.053) | 0.380 | 0.634 | (0.059) | 0.206 |
| Am | 0.819 | (0.042) | 0.070 | 0.812 | (0.084) | 0.062 | 0.807 | (0.034) | 0.203 | 0.782 | (0.060) | 0.103 |
| Au | 0.685 | (0.118) | 0.141 | 0.685 | (0.107) | 0.131 | 0.725 | (0.108) | 0.273 | 0.729 | (0.115) | 0.142 |
| Ba | 0.851 | (0.056) | 0.027 | 0.848 | (0.053) | 0.026 | 0.839 | (0.034) | 0.172 | 0.834 | (0.043) | 0.086 |
| Bc | 0.970 | (0.052) | 0.013 | 0.968 | (0.055) | 0.009 | 0.960 | (0.030) | 0.060 | 0.957 | (0.022) | 0.021 |
| Ca | 0.938 | (0.016) | 0.061 | 0.946 | (0.020) | 0.053 | 0.948 | (0.036) | 0.136 | 0.952 | (0.021) | 0.063 |
| Cs | 0.852 | (0.048) | 0.038 | 0.851 | (0.045) | 0.035 | 0.832 | (0.052) | 0.178 | 0.835 | (0.056) | 0.063 |
| Ec | 0.854 | (0.083) | 0.060 | 0.846 | (0.102) | 0.053 | 0.869 | (0.092) | 0.173 | 0.857 | (0.113) | 0.068 |
| Gl | 0.677 | (0.289) | 0.174 | 0.677 | (0.279) | 0.154 | 0.686 | (0.254) | 0.275 | 0.696 | (0.304) | 0.173 |
| He | 0.864 | (0.076) | 0.059 | 0.864 | (0.076) | 0.052 | 0.855 | (0.231) | 0.169 | 0.837 | (0.159) | 0.065 |
| Io | 0.878 | (0.086) | 0.052 | 0.878 | (0.086) | 0.048 | 0.909 | (0.039) | 0.093 | 0.920 | (0.058) | 0.055 |
| Ir | 0.947 | (0.071) | 0.050 | 0.947 | (0.071) | 0.043 | 0.947 | (0.043) | 0.108 | 0.927 | (0.074) | 0.049 |
| Le | 0.693 | (0.049) | 0.188 | 0.692 | (0.048) | 0.179 | 0.719 | (0.049) | 0.314 | 0.692 | (0.066) | 0.198 |
| Li | 0.649 | (0.059) | 0.171 | 0.644 | (0.076) | 0.151 | 0.629 | (0.046) | 0.290 | 0.638 | (0.040) | 0.193 |
| M1 | 0.887 | (0.085) | 0.125 | 0.917 | (0.051) | 0.116 | 0.955 | (0.053) | 0.307 | 0.984 | (0.030) | 0.156 |
| M2 | 0.962 | (0.029) | 0.089 | 0.968 | (0.026) | 0.082 | 0.987 | (0.021) | 0.184 | 0.984 | (0.023) | 0.092 |
| M3 | 0.944 | (0.034) | 0.041 | 0.944 | (0.034) | 0.038 | 0.966 | (0.051) | 0.127 | 0.979 | (0.050) | 0.045 |
| Mu | 0.987 | (0.006) | 0.003 | 0.987 | (0.006) | 0.001 | 1.000 | (0.000) | 0.003 | 1.000 | (0.000) | 0.002 |
| Ne | 0.935 | (0.076) | 0.041 | 0.935 | (0.076) | 0.031 | 0.944 | (0.043) | 0.127 | 0.948 | (0.062) | 0.074 |
| Nu | 0.931 | (0.006) | 0.080 | 0.941 | (0.006) | 0.063 | 0.947 | (0.011) | 0.174 | 0.975 | (0.010) | 0.089 |
| Op | 0.959 | (0.020) | 0.026 | 0.958 | (0.020) | 0.023 | 0.966 | (0.017) | 0.060 | 0.961 | (0.023) | 0.039 |
| Pe | 0.976 | (0.007) | 0.017 | 0.976 | (0.006) | 0.015 | 0.986 | (0.004) | 0.042 | 0.983 | (0.003) | 0.026 |
| Pi | 0.713 | (0.087) | 0.097 | 0.716 | (0.111) | 0.087 | 0.733 | (0.079) | 0.236 | 0.698 | (0.094) | 0.131 |
| Se | 0.936 | (0.027) | 0.045 | 0.937 | (0.032) | 0.040 | 0.951 | (0.009) | 0.103 | 0.949 | (0.016) | 0.060 |
| Sh | 0.997 | (0.001) | 0.004 | 0.998 | (0.002) | 0.003 | 0.998 | (0.002) | 0.008 | 0.998 | (0.001) | 0.004 |
| Sn | 0.871 | (0.099) | 0.143 | 0.880 | (0.095) | 0.135 | 0.857 | (0.120) | 0.245 | 0.843 | (0.176) | 0.157 |
| Sb | 0.905 | (0.058) | 0.097 | 0.903 | (0.053) | 0.085 | 0.899 | (0.073) | 0.177 | 0.898 | (0.062) | 0.102 |
| Tt | 0.833 | (0.033) | 0.102 | 0.845 | (0.027) | 0.090 | 0.863 | (0.031) | 0.210 | 0.870 | (0.041) | 0.133 |
| Vo | 0.940 | (0.058) | 0.030 | 0.940 | (0.044) | 0.028 | 0.929 | (0.022) | 0.093 | 0.938 | (0.022) | 0.051 |
| Vw | 0.925 | (0.044) | 0.164 | 0.923 | (0.045) | 0.151 | 0.957 | (0.013) | 0.292 | 0.926 | (0.028) | 0.171 |
| Wd | 0.945 | (0.019) | 0.029 | 0.946 | (0.019) | 0.025 | 0.943 | (0.048) | 0.086 | 0.942 | (0.057) | 0.043 |
| Wi | 0.949 | (0.093) | 0.046 | 0.960 | (0.105) | 0.040 | 0.943 | (0.087) | 0.126 | 0.949 | (0.040) | 0.075 |
| Wp | 0.722 | (0.150) | 0.139 | 0.763 | (0.182) | 0.119 | 0.687 | (0.146) | 0.263 | 0.707 | (0.132) | 0.140 |
| Ye | 0.565 | (0.045) | 0.160 | 0.566 | (0.042) | 0.141 | 0.548 | (0.046) | 0.284 | 0.545 | (0.071) | 0.169 |
| Zo | 0.940 | (0.087) | 0.101 | 0.940 | (0.087) | 0.084 | 0.950 | (0.126) | 0.155 | 0.950 | (0.126) | 0.091 |
| Ave | 0.861 | | 0.083 | 0.863 | | 0.074 | 0.867 | | 0.175 | 0.866 | | 0.095 |

*The values inside the parenthesis denote the standard deviation.*

from 0.083 to 0.074 with a slight increase in accuracy. It shows that the refinement step in ICPL2-RT2 is effective. When we compare ICPL3-RT2 with RT2, we find that both the accuracy and the retention rate are of the same performance. It indicates that abstracted prototypes generated by TPA cannot help the filtered prototypes generated by RT2 to provide a better model for classification. It is probably due to the fact that abstracted prototypes can only represent simple boundaries which are already modeled by border retaining prototypes generated by RT2. Comparing ICPL4-RT2 with ICPL3-RT2, we find that there is an improvement in data retention rate from 0.175 to 0.095 with similar accuracy. It illustrates that the extra refinement step in ICPL4-RT2 is useful in substituting border retaining prototypes by a few abstracted representative prototypes.

**Summary**. In summary, our ICPL framework offers a promising model for integrating the strengths of abstraction and filtering of prototypes by maintaining a balance of different kinds of prototypes based on locality. As shown in the above results, the pure abstraction component TPA has a good generalization power which can abstract compact instances into a few prototypes but it is ineffective to describe complex concepts. It leads to its excellent data retention rate but moderate classification accuracy. However, with proper integration with filtering techniques, the accuracy improves with a little loss in retention rate. Nonrepresentative or redundant prototypes can be removed or refined. When we compare ICPL with pure filtering methods, we find that the accuracy of ICPL is generally better and the retention rate is definitely better. Among different variants of our ICPL algorithms, ICPL2 is better in terms of both accuracy and retention rate than ICPL1. Similar observations can be found for ICPL3 and ICPL4. ICPL4 is better than ICPL3 in general. This result illustrates that both the expansion and

TABLE 10
The Average Classification Accuracy (Acc) and Data Retention Rate (Size) of 10-Fold Cross-Validation for ICPL and Some Existing Algorithms of Each Data Set

| | ICPL | | | RT3 | | | KNN | | SVM | | C4.5 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | acc | std | size | acc | std | size | acc | std | acc | std | acc | std |
| Ab | 0.624 | (0.058) | 0.196 | 0.621 | (0.152) | 0.319 | 0.766 | (0.076) | 0.721 | (0.010) | 0.792 | (0.066) |
| Am | 0.812 | (0.084) | 0.062 | 0.794 | (0.067) | 0.136 | 0.771 | (0.082) | 0.827 | (0.004) | 0.939 | (0.041) |
| Au | 0.685 | (0.107) | 0.131 | 0.667 | (0.116) | 0.247 | 0.761 | (0.102) | 0.797 | (0.008) | 0.666 | (0.083) |
| Ba | 0.848 | (0.053) | 0.026 | 0.837 | (0.065) | 0.103 | 0.775 | (0.066) | 0.963 | (0.000) | 0.900 | (0.032) |
| Bc | 0.968 | (0.055) | 0.009 | 0.957 | (0.035) | 0.033 | 0.960 | (0.014) | 0.940 | (0.001) | 0.953 | (0.063) |
| Ca | 0.946 | (0.020) | 0.053 | 0.952 | (0.018) | 0.112 | 0.956 | (0.016) | 0.978 | (0.000) | 0.692 | (0.043) |
| Cs | 0.851 | (0.045) | 0.035 | 0.826 | (0.035) | 0.085 | 0.807 | (0.047) | 0.794 | (0.001) | 0.642 | (0.054) |
| Ec | 0.846 | (0.102) | 0.053 | 0.878 | (0.059) | 0.108 | 0.822 | (0.095) | 0.845 | (0.004) | 0.921 | (0.081) |
| Gl | 0.677 | (0.279) | 0.154 | 0.672 | (0.341) | 0.211 | 0.681 | (0.300) | 0.710 | (0.008) | 0.894 | (0.031) |
| He | 0.864 | (0.076) | 0.052 | 0.856 | (0.208) | 0.093 | 0.805 | (0.186) | 0.799 | (0.006) | 0.964 | (0.007) |
| Io | 0.878 | (0.086) | 0.048 | 0.869 | (0.047) | 0.068 | 0.866 | (0.058) | 0.875 | (0.001) | 0.694 | (0.085) |
| Ir | 0.947 | (0.071) | 0.043 | 0.947 | (0.114) | 0.080 | 0.947 | (0.043) | 0.947 | (0.002) | 0.962 | (0.014) |
| Le | 0.692 | (0.048) | 0.179 | 0.696 | (0.050) | 0.282 | 0.810 | (0.034) | 0.817 | (0.001) | 0.999 | (0.001) |
| Li | 0.644 | (0.076) | 0.151 | 0.566 | (0.110) | 0.218 | 0.632 | (0.089) | 0.693 | (0.010) | 0.706 | (0.094) |
| M1 | 0.917 | (0.051) | 0.116 | 0.971 | (0.071) | 0.296 | 0.969 | (0.039) | 1.000 | (0.000) | 0.779 | (0.046) |
| M2 | 0.968 | (0.026) | 0.082 | 0.984 | (0.014) | 0.182 | 0.993 | (0.016) | 0.984 | (0.000) | 0.944 | (0.031) |
| M3 | 0.944 | (0.034) | 0.038 | 0.955 | (0.069) | 0.086 | 0.955 | (0.045) | 0.775 | (0.008) | 0.888 | (0.081) |
| Mu | 0.987 | (0.006) | 0.001 | 1.000 | (0.000) | 0.003 | 1.000 | (0.000) | 1.000 | (0.000) | 0.676 | (0.168) |
| Ne | 0.935 | (0.076) | 0.031 | 0.948 | (0.073) | 0.111 | 0.972 | (0.031) | 0.948 | (0.004) | 0.545 | (0.049) |
| Nu | 0.941 | (0.006) | 0.063 | 0.968 | (0.008) | 0.167 | 0.991 | (0.004) | 0.877 | (0.004) | 0.756 | (0.064) |
| Op | 0.958 | (0.020) | 0.023 | 0.964 | (0.029) | 0.057 | 0.985 | (0.010) | 0.988 | (0.000) | 0.928 | (0.012) |
| Pe | 0.976 | (0.006) | 0.015 | 0.986 | (0.003) | 0.040 | 0.994 | (0.005) | 0.993 | (0.000) | 0.960 | (0.084) |
| Pi | 0.716 | (0.111) | 0.087 | 0.719 | (0.097) | 0.141 | 0.706 | (0.114) | 0.740 | (0.002) | 0.625 | (0.079) |
| Se | 0.937 | (0.032) | 0.040 | 0.949 | (0.008) | 0.091 | 0.969 | (0.015) | 0.967 | (0.000) | 0.988 | (0.033) |
| Sh | 0.998 | (0.002) | 0.003 | 0.998 | (0.001) | 0.007 | 0.999 | (0.001) | 0.885 | (0.000) | 1.000 | (0.000) |
| Sn | 0.880 | (0.095) | 0.135 | 0.812 | (0.046) | 0.226 | 0.876 | (0.152) | 0.846 | (0.002) | 0.979 | (0.004) |
| Sb | 0.903 | (0.053) | 0.085 | 0.889 | (0.051) | 0.155 | 0.908 | (0.053) | 0.899 | (0.001) | 0.930 | (0.034) |
| Tt | 0.845 | (0.027) | 0.090 | 0.876 | (0.025) | 0.181 | 0.914 | (0.027) | 0.978 | (0.000) | 0.862 | (0.036) |
| Vo | 0.940 | (0.044) | 0.028 | 0.922 | (0.098) | 0.062 | 0.935 | (0.031) | 0.947 | (0.001) | 0.960 | (0.021) |
| Vw | 0.923 | (0.045) | 0.151 | 0.956 | (0.017) | 0.293 | 0.992 | (0.016) | 0.973 | (0.000) | 0.926 | (0.101) |
| Wd | 0.946 | (0.019) | 0.025 | 0.952 | (0.041) | 0.056 | 0.945 | (0.028) | 0.963 | (0.001) | 0.776 | (0.056) |
| Wi | 0.960 | (0.105) | 0.040 | 0.938 | (0.125) | 0.114 | 0.954 | (0.054) | 0.961 | (0.003) | 0.794 | (0.156) |
| Wp | 0.763 | (0.182) | 0.119 | 0.723 | (0.153) | 0.138 | 0.701 | (0.108) | 0.738 | (0.008) | 0.832 | (0.054) |
| Ye | 0.566 | (0.042) | 0.141 | 0.544 | (0.040) | 0.173 | 0.524 | (0.054) | 0.602 | (0.001) | 0.822 | (0.060) |
| Zo | 0.940 | (0.087) | 0.084 | 0.931 | (0.059) | 0.169 | 0.970 | (0.034) | 0.960 | (0.002) | 0.773 | (0.182) |
| Ave | 0.863 | | 0.074 | 0.861 | | 0.138 | 0.875 | | 0.878 | | 0.842 | |

*The values inside the parenthesis denote the standard deviation. ICPL is taken from ICPL2-RT2.*

refinement steps in our ICPL framework are effective. When we compare ICPL2 and ICPL4, we find that ICPL2 has a slightly better retention rate with similar accuracy. This result suggests that using our TPA abstraction component as a starting prototype set will be more beneficial.

### 5.3 Comparisons with Existing Approaches

In the second set of experiments, we compare our ICPL framework with some existing state-of-the-art algorithms such as RT3, KNN, Support Vector Machine (SVM) and C4.5. We first compare results on the 35 data sets. After that, we also make comparison with existing algorithms on 15 extra unseen data sets.

#### *5.3.1 Results on 35 Data Sets*

Table 5 depicts the summary and statistical analysis of 10-fold cross-validation of 35 data sets for ICPL and some existing algorithms. The detailed performance of each data set is given in Table 10. The ICPL framework chosen in this comparative study is ICPL2-RT2.

RT3 and its derivatives, renamed as DROP1-DROP5, were intensively investigated in [39]. Among the five algorithms, DROP3 (it is also called the RT3 algorithm) is found to have the best mix of classification accuracy and data retention rate. DROP3 is also compared with 10 other prototype learners, namely, CNN [22], SNN [28], IB2, IB3 [1], ENN [37], Unlimited and All K-NN [33], and Explore [5]. Experiments on 31 real-world benchmark data sets show that DROP3 performs the best among these data reduction techniques considering both the average classification accuracy and data retention rate. Therefore, we choose RT3 in our comparative study. It is found from Table 10 that ICPL achieves a slightly higher classification accuracy than RT3. ICPL outperforms RT3 in classification

accuracy on 17 data sets while RT3 gains better performance on the other 17. Wilcoxon test on accuracy also suggests that the two methods achieve similar performance. 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 classification accuracy. We observe that ICPL retains only 7.4 percent in size of the original training set while RT3 stores 13.8 percent. Using the Wilcoxon test, we can see that ICPL has a significantly better performance in terms of data retention rate than RT3 does with high confidence level of 99.50 percent. These results support that our ICPL algorithm performs better than the state-of-the-art prototype learner, RT3, in learning representative prototypes.

When comparing ICPL with KNN, we observe that KNN has a slightly higher average classification accuracy. Among the 35 data sets, KNN performs better on classification accuracy on 19 of them while ICPL gains better results on other 15. 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 percent of the total instances are stored. For some data sets, such as Bc and Sh, ICPL obtains similar or even better classification accuracy than KNN with only less than 1 percent of the total instances. With significantly fewer prototypes, ICPL can classify unseen cases with lower computational cost and storage requirement than that of KNN.

We now compare ICPL with SVM which has been proven to perform well on the classification task in a variety of domains in the past few years [31], [35], [20]. In our experiments, we adopt traditional one-against-the-rest method to handle multiclass data sets. Polynomial kernel function was used in SVM. Experimental results show that our ICPL algorithm achieves a comparable classification accuracy than SVM does on average. Significantly better results for ICPL can also be found on some data sets such as Cs and He.

Contrasting ICPL with C4.5, we notice that ICPL performs better than C4.5 in average classification accuracy. The two algorithms achieve high classification accuracy on nearly the same number of data sets. Based on statistical test, ICPL is observed to obtain a significantly better classification accuracy than C4.5 does in 15 data sets. On some data sets, such as Ca and M1, ICPL performs significantly better than C4.5. These results suggest that the ICPL is strong in handling some kinds of data distribution.

#### 5.3.2 Results on 15 Unseen Data Sets

In order to make a more rigorous comparison on our ICPL (ICPL2-RT2) algorithm and existing algorithms, we have also conducted experiments on 15 extra unseen data sets also from the Machine Learning Database Repository [27].[1] From Table 6, we can see that ICPL performs equally well compared with previous experiments. ICPL outperforms RT3 on average classification accuracy with a significantly fewer prototypes. The average classification accuracy of ICPL, which retains only 5.7 percent of training instances, is even higher than that of KNN, which retains all the training instances. ICPL achieves comparable results with that of SVM. Unlike results on the previous experiment, C4.5 achieves the highest average classification accuracy. Similar results with previous data sets can be observed when comparing ICPL with SVM.

1. The data sets are Allhyper, German, Page-Blocks, Allrep, Heart-Disease, Satimage, Australian, Horse-Colic, Servo, CPU-Performance, Housing, Sick, Dis, Hypothyroid, Sick-Enthyroid.

## 6 Conclusions and Future Work

We have developed a new framework, called ICPL (Integrated Concept Prototype Learner), which integrates the strengths of concept prototypes learned from instance-filtering and instance-abstraction techniques. Three kinds of prototypes based on instance locality have been identified. ICPL attempts to select good prototypes and maintains a balance of different kinds of prototypes. The abstraction component, based on typicality, in ICPL is specially designed for concept integration. Extensive experiments on 35 benchmark data sets with statistical analysis have been conducted. The results indicate that the pure abstraction component TPA has a good generalization power which can abstract compact instances into a few prototypes but it has moderate classification accuracy. With proper integration with filtering techniques developed in ICPL, the accuracy improves with a little loss in retention rate. In summary, our ICPL framework performs favorably over many existing prototype learners in learning representative prototypes.

Apart from prototype selection, prototype-based learning algorithms also depend on similarity calculation. One future work will investigate the issue of similarity between prototypes. In many applications, data may contain both continuous and discrete features. A good similarity function is essential to the performance of prototype-based learning. Another future work is to focus on feature selection. In our current approach, prototypes are retained without any change in data representation. However, it will also store irrelevant features as well which will normally hurt the classification accuracy and storage requirement can be improved.

## Acknowledgments

The work described in this paper was partially supported by grants from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project Nos.: CUHK 4385/99E and CUHK 4187/01E).

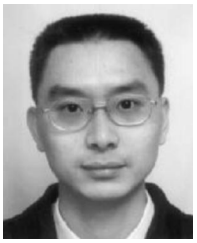
## References

[1] D.W. Aha, D. Kibler, and M.K. Albert, "Instance-Based Learning Algorithms," *Machine Learning*, vol. 6, pp. 37-66, 1991.
[2] D.W. Aha, D. Kibler, and M.K. Albert, "Noise-Tolerant Instance-Based Learning Algorithms," *Proc. Int'l Joint Conf. Artificial Intelligence*, pp. 794-799, 1991.
[3] J.C. Bezdek, T.R. Reichherzer, G.S. Lim, and Y. Attikiouzel, "Multiple-Prototype Classifier Design," *IEEE Trans. Systems, Man, and Cybernetics*, pp. 67-79, 1998.
[4] G. Bradshaw, "Learning about Speech Sounds: The NEXUS Project," *Proc. Fourth Int'l Workshop Machine Learning*, pp. 1-11, 1987.
[5] R.M. Cameron-Jones, "Instance Selection by Encoding Length Heuristic with Random Mutation Hill Climbing," *Proc. Eighth Australian Joint Conf. Artificial Intelligence*, pp. 293-301, 1995.
[6] C.L. Chang, "Finding Prototypes for Nearest Neighbor Classifier," *IEEE Trans. Computers*, vol. 23, no. 11, pp. 1179-1184, Nov. 1974.
[7] W.J. Conover, *Practical Nonparametric Statistics*. New York: John Wiley, 1971.

[8] S. Cost and S. Salzberg, "A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Feature," *Machine Learning,* vol. 10, pp. 57-78, 1993.
[9] B.V. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* Los Alamitos, Calif.: IEEE CS Press, 1991.
[10] B.V. Dasarathy, "Minimal Consistent Set (MCS) Identification for Optimal Nearest Neighbor Decision Systems Design," *IEEE Trans. Systems, Man, Cybernetics,* vol. 24, no. 3, pp. 511-517, 1994.
[11] B.V. Dasarathy, J.S. Sanchez, and S. Townsend, "Nearest Neighbour Editing and Condensing Tools-Synergy Exploitation," *Pattern Analysis and Applications,* vol. 3, no. 1, pp. 19-30, 2000.
[12] P. Datta and D. Kibler, "Learning Prototypical Concept Description," *Proc. 12th Int'l Conf. Machine Learning,* pp. 158-166, 1995.
[13] P. Datta and D. Kibler, "Learning Symbolic Prototypes," *Proc. 14th Int'l Conf. Machine Learning,* pp. 75-82, 1997.
[14] P. Datta and D. Kibler, "Symbolic Nearest Mean Classifier," *Proc. 14th Nat'l Conf. Artificial Intelligence,* pp. 82-87, 1997.
[15] P. Domingos, "Unifying Instance-Based and Rule-Based Induction," *Machine Learning,* vol. 24, pp. 141-168, 1996.
[16] W. DuMouchel, C. Volinsky, T. Johnson, C. Cortes, and D. Pregibon, "Squashing Flat Files Flatter," *Proc. Fifth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 6-15, 1999.
[17] F.J. Ferri, J.V. Albert, and E. Vidal, "Considerations About Sample-Size Sensitivity of a Family of Edited Nearest-Neighbor Rules," *IEEE Trans. Systems, Man, and Cybernetics—Part B,* vol. 29, no. 4, pp. 667-672, 1999.
[18] G.W. Gates, "The Reduced Nearest Neighbor Rule," *IEEE Trans. Information Theory,* vol. 18, no. 3, pp. 431-433 1972.
[19] A.R. Golding and P.S. Rosenbloom, "Improving Accuracy by Combining Rule-Based and Case-Based Reasoning," *Artificial Intelligence,* vol. 87, pp. 215-254, 1996.
[20] Y. Guermeur, A. Elisseeff, and H. Paugam-Moisy, "A New Multiclass SVM Based on a Uniform Convergence Result," *Proc. IEEE-INNS-ENNS Int'l Joint Conf. Neural Networks,* pp. 183-188, 2000.
[21] Y. Hamamoto, S. Uchimura, and S. Tomita, "A Bootstrap Technique for Nearest Neighbor Classifier Design," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 1, pp. 73-79, Jan. 1997.
[22] P.E. Hart, "The Condensed Nearest Neighbor Rule," *IEEE Trans. Information Theory,* vol. 14, no. 3, pp. 515-516, 1968.
[23] C.K. Keung and W. Lam, "Prototype Generation Based on Instance Filtering and Averaging," *Proc. Fourth Pacific-Asia Conf. Knowledge Discovery and Data Mining,* pp. 142-152, 2000.
[24] D. Kibler and D.W. Aha, "Comparing Instance-Averaging with Instance-Filtering Learning Algorithm," *Proc. Third European Working Session on Learning,* pp. 63-80, 1988.
[25] L.I. Kuncheva and J.C. Bezdek, "Nearest Prototype Classification: Clustering, Genetic Algorithms, or Random Search?" *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 28, no. 1, pp. 160-164, 1998.
[26] W. Lam, C.K. Keung, and C.X. Ling, "Learning Good Prototypes for Classification Using Filtering and Abstraction of Instances," *Pattern Recognition,* vol. 35, no. 7, pp. 1491-1506, July 2002.
[27] P.M. Murphy and D.W. Aha, "UCI Repository of Machine Learning Database," Dept. of Information and Computer Science, Univ. of Calif., Irvine, 1994.
[28] G.L. Ritter, H.B. Woodruff, and S.R. Lowry, "An Algorithm for a Selective Nearest Neighbor Decision Rule," *IEEE Trans. Information Theory,* vol. 21, no. 6, pp. 665-669, 1975.
[29] S. Salzberg, "A Nearest Hyperrectangle Learning Method," *Machine Learning,* vol. 6, pp. 251-276, 1991.
[30] A. Sato, "A Learning Method for Definite Canonicalization Based on Minimum Classification Error," *Proc. Int'l Conf. Pattern Recognition (ICPR),* pp. 199-202, 2000.
[31] B. Schölkopf, C. Burges, and V. Vapnik1995. "Extracting Support Data for a Given Task," *Proc. First Int'l Conf. Knowledge Discovery and Data Mining,* pp. 252-257,
[32] C. Stanfill and D. Waltz, "Toward Memory-Based Reasoning," *Comm. ACM,* vol. 29, pp. 1213-1228, 1986.
[33] I. Tomek, "An Experiment with the Edited Nearest-Neighbor Rule," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 6, no. 6, pp. 448-452, 1976.
[34] A. van den Bosch, "Instance-Family Abstraction in Memory-Based Language Learning," *Proc. 16th Int'l Conf. Machine Learning,* pp. 39-48 1999.
[35] J. Weston and C. Watkins, "Multi-Class Support Vector Machines," Technical Report CSD-TR-98-04, Dept. of Computer Science, Royal Holloway, Univ. of London, Egham, Surrey TW20 0EX, England, 1998.
[36] D. Wettschereck, "A Hybrid Nearest-Neighbor and Nearest-Hyperrectangle Algorithm," *Proc. Seventh European Conf. Machine Learning,* pp. 323-335, 1994.
[37] D.L. Wilson, "Asymptotic Properties of Nearest Neighbor Rules Using Edited Data," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 2, no. 3, pp. 431-433 1972.
[38] D.R. Wilson and T.R. Martinez, "Instance Pruning Techniques," *Proc. 14th Int'l Conf. Machine Learning,* pp. 403-411, 1997.
[39] D.R. Wilson and T.R. Martinez, "Reduction Techniques For Instance-Based Learning Algorithm," *Machine Learning,* vol. 38, pp. 257-286, 2000.
[40] D.R. Wilson and T.R. Martinez, "In Integrated Instance-Based Learning Algorithm," *Computational Intelligence,* vol. 16, no. 1, pp. 1-28, 2000.
[41] Q. Xie, C.A. Laszlo, and R.K. Ward, "Vector Quantization Technique for Nonparametric Classifier Design," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 15, no. 12, pp. 1326-1330, Dec. 1993.
[42] J. Zhang, "Selecting Typical Instances In Instance-Based Learning," *Proc. Int'l Conf. Machine Learning,* pp. 470-479 1992.

**Wai Lam** received the PhD degree in computer science from the University of Waterloo, Canada, in 1994. He worked as a research associate at Indiana University-Purdue University Indianapolis in 1995 and as a postdoctoral fellow at the Distributed Adaptive Search Laboratory at the University of Iowa in 1996. Currently, he is an associate professor in the Department of Systems Engineering and Engineering Management at the Chinese University of Hong Kong. His current interests include data mining, intelligent information retrieval, machine learning, reasoning under uncertainty, and digital library. He is a member of the IEEE.

**Chi-Kin Keung** is a M.Phil. student in the Department of Systems Engineering and Engineering Management, at the Chinese University of Hong Kong. He received the bachelor degree from the same department in 1998. His research interests are machine learning, artificial intelligence and data mining, especially for instance-based learning and classification.



**Danyu Liu** received a bachelor's degree and a master's degree from Nanjing University of Science and Technology. He was a PhD candidate in the Chinese University of Hong Kong. He is currently a PhD student at the University of Alabama at Birmingham.