PR 1527
pp: 1--16 (col.fig.: Nil)
PROD. TYPE: COM
ED: Indira
PAGN: Suresh – SCAN: Radhika

PERGAMON

Pattern Recognition 000 (2001) 000–000

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/patcog

Learning good prototypes for classification using filtering and abstraction of instances

Wai Lam[a,*], Chi-Kin Keung[a], Charles X. Ling[b]

[a] *Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Shatin, Hong Kong*
[b] *Department of Computer Science, The University of Western Ontario, London, Ont., Canada N6A 5B7*

Received 18 July 2000; accepted 3 July 2001

Abstract

We propose a framework for learning good prototypes, called prototype generation and filtering (PGF), by integrating the strength of instance-filtering and instance-abstraction techniques using two different integration methods. The two integration methods differ in the filtering granularity as well as the degree of coupling of the techniques. In order to characterize the behavior of the effect of integration, we categorize instance-filtering techniques into three kinds, namely, (1) removing border instances, (2) retaining border instance, (3) retaining center instances. The effect of using different kinds of filtering in different variants of our PGF framework are investigated. We have conducted experiments on 35 real-world benchmark data sets. We found that our PGF framework maintains or achieves better classification accuracy and gains a significant improvement in data reduction compared with pure filtering and pure abstraction techniques as well as KNN and C4.5. © 2001 Published by Elsevier Science Ltd on behalf of Pattern Recognition Society.

Keywords: Instance-based learning; Prototype generation; Instance abstraction; Machine learning

1. Introduction

The nearest neighbor (NN) algorithm and its derivatives have been proven to perform well in pattern classification on many domains [1,2]. 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 generalization accuracy, they suffer from high storage requirement, computational cost and sensitivity to noise.

One method for solving this problem is to develop advanced data structure and search techniques to speed up NN searching [3]. If the number of data instances is very large, it still requires high computational cost. Another method is to reduce the large data set to a small, representative prototype set. Removing non-representative 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. One technique is known as 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.

The two techniques are used independently in the past. An initial examination on the integration of the two methods has been carried out by the authors [4]. In this paper, we conduct a thorough and in-depth investigation on the integrating technique. We propose a framework for discovering good prototypes, called prototype generation and filtering (PGF), which combines instance-filtering and instance-abstraction techniques by integrating the strength of both techniques using two different

* Corresponding author. Tel.: +852-2609-8306; fax: +852-2603-5505.
E-mail address: wlam@se.cuhk.edu.hk (W. Lam).

PII: S0031-3203(01)00131-5

integration methods. The two integration methods differ in the filtering granularity as well as the degree of the coupling of the components. In order to characterize the behavior of the effect of integration, we categorize different kinds of instance-filtering techniques according to the locations of instances retained or removed, namely, (1) retaining center instances, (2) retaining border instances and (3) removing border instances. The effects of using different kinds of filtering in different variants of our PGF framework are investigated. In experiments on 35 real-world benchmark data sets, the classification accuracy and data retention rate of each variant of our method are investigated. The results are compared with those of pure instance-filtering and pure instance-abstraction techniques as well as KNN and C4.5. Empirical results show that the PGF framework maintains or achieves better classification accuracy and gains a significant improvement in data reduction compared with existing methods.

2. Our proposed algorithm

2.1. Motivation

Some works have been done on selecting representative instances. In instance-filtering methods, editing rules are used to determine whether an instance should be retained as a prototype or not. These methods differ from search direction and locations of instances retained. For example, Hart proposes a condensed nearest neighbor (CNN) which is probably the earliest method to select representative instances [5]. 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 [6]. The edited nearest neighbor (ENN) algorithm proposed by Wilson eliminates instances misclassified by their k-nearest neighbors [7]. A noise-tolerant instance filtering called NTGrowth is proposed by Aha and Kibler [8]. Later, Aha et al. formalize NTGrowth to the well-know IB2 and IB3 algorithm which is based on CNN storing misclassified instances [9]. 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. Zhang introduces typical instance-based learning which stores typical instance in the region centers [10]. 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 [11]. 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 [12] and the integrated decremental instance-based learning which combines confidence and cross-validation accuracy in the distance measure [13].

Another approach for finding representative instances is the instance-abstraction method which generates prototypes by abstracting or averaging the original instances. Chang's method learns representative instances by merging similar ones. It iteratively merges two closest instances and summarizes them by taking the weighted average of them [14]. Bradshaw introduces the *disjunctive spanning* (DS) which merges instances with the ones they can be correctly classified [15]. Kibler and Aha improve DS by using an *adaptive threshold* to limit the distance between two merged instances [16]. An algorithm called nested generalized exemplar (NGE) is proposed by Salzberg which stores instances as hyperrectangles [17]. Wettschereck combines the NGE with KNN to form a hybrid algorithm [18]. However, this algorithm stores the entire data set in memory. Domingos also proposes an integrated technique, the RISE algorithm, combining instance-based learning and rule induction [19]. Under this algorithm, instances are treated as rules and data reduction is achieved using specific rules formed by generalization of instances. Datta and Kibler introduce the prototype learner (PL) which learns artificial instances for each class by generalization of representative instances in nominal domains [20]. Then they propose the symbolic nearest mean classifiers (SNMC) [21] 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 [22]. SNMC uses k-means clustering to group instances of the same class and create artificial instances using cluster means. Bezdek et al. modify Chang's method which averages instances using simple mean and merges instances of the same class only [23]. 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 [24]. A technique known as *squashing* is proposed to scale down the data set by exploiting the statistical property of the instances [25]. However, this technique does not make use of the class label information if it is employed in classification problems.

We observe that instance-filtering and instance-abstraction approaches can be beneficial to each other. Filtering methods do not conduct generalization on instances so that they usually cannot gain a satisfactory level of data reduction [7]. With the help of instance-abstraction methods, instances in compact regions can be generalized to a few or single prototypes leading to a significant improvement in data reduction rate. Also, the representative power of filtering methods will be limited if the truly representative instances cannot be found in 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.

Instance-filtering can assist instance-abstraction too. Non-prototypical instances will be formed if distant instances, especially for outliers and exceptions, are grouped in abstraction methods. To avoid this, specially designed filtering rules can be applied to remove outliers and exceptions first before applying abstraction. Filtering techniques can also be helpful in the middle of or after the abstraction process. We can design a filtering rule to remove any non-representative prototypes formed when the abstraction process is in progress.

We observe that there are two main factors affecting the performance of the integration of the two approaches. The first factor is the type of filtering techniques. We can classify filtering techniques into three types according to [11]. The first type of filtering methods retains central instances as representative instances in a cluster of data points. The second type of filtering retains border instances of a cluster as representative instances. The third type of filtering removes border instances and treats the remaining ones as representative. As abstraction techniques attempt to generalize similar instances in compact regions, they work differently on instances in different regions. For example, center instances will be generalized to a larger extent compared with border instances.

The second factor affecting the performance of the integration is the filtering granularity. 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. 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. Furthermore, the two factors will interact with each other leading to different behaviors of the integration algorithm.

We develop a general framework for the integration called PGF. Then we investigate different integration algorithms under our PGF framework.

2.2. The framework of our approach

A simple PGF framework has been first proposed by the authors in previous work [4]. In this paper, PGF is further developed into two variants which differ from the integration method of filtering and abstraction techniques. PGF consists of an instance-abstraction component and an instance-filtering component. We first describe the abstraction component and each of the three filtering methods used. Then we present two different ways to integrate the two components in our PGF framework. We will also illustrate the effect of different components using a hypothetic data set of two classes as shown in Fig. 1.

2.2.1. 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. Fig. 2 shows the pseudo-code of the instance abstraction component, called ABS. Let P be the current prototype set. At each iteration, two prototypes with the shortest distance are merged to form a new prototype. 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. After the algorithm terminates, the output prototype set will be used for classifying unseen cases using the simple NN algorithm.

There are many ways to develop the prototype set score function. As our objective is to learn prototypes to classify unlabeled instances, 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.

To measure the distance between instances with continuous and nominal feature types, we adopt a heterogeneous distance function similar to the one proposed by [11]. 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*) [26] is used to handle nominal features. The distance function vdm_i for feature i is defined as:

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

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

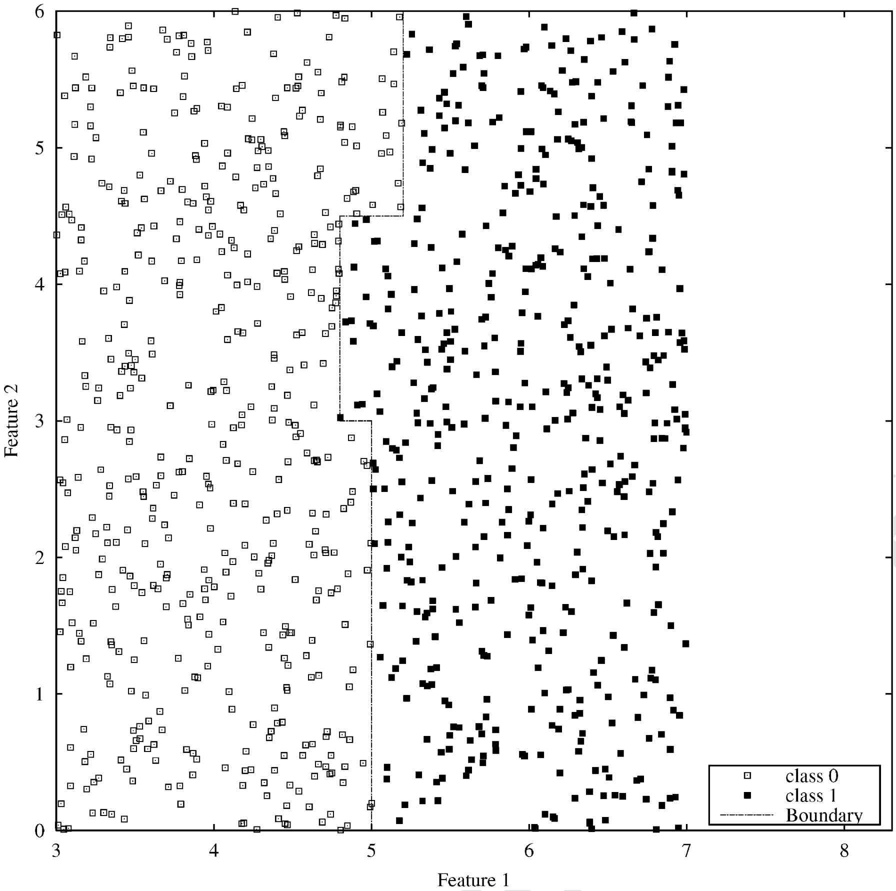

Fig. 1. A data set of two classes.

$\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_n)$, 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 $dist_i(x_i, y_i)$ equals to $vdm_i(x_i, y_i)$ for nominal features and $(x - y)$ for continuous features. 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.

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 [21]. These methods guarantee fully homogeneous prototypes but the entire data distribution

```
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).
  If (PROT_SET_SCORE(P) >= max_score)
    P' = P.
    max_score = PROT_SET_SCORE(P).
Return P'.
```

Fig. 2. The ABS component in PGF.

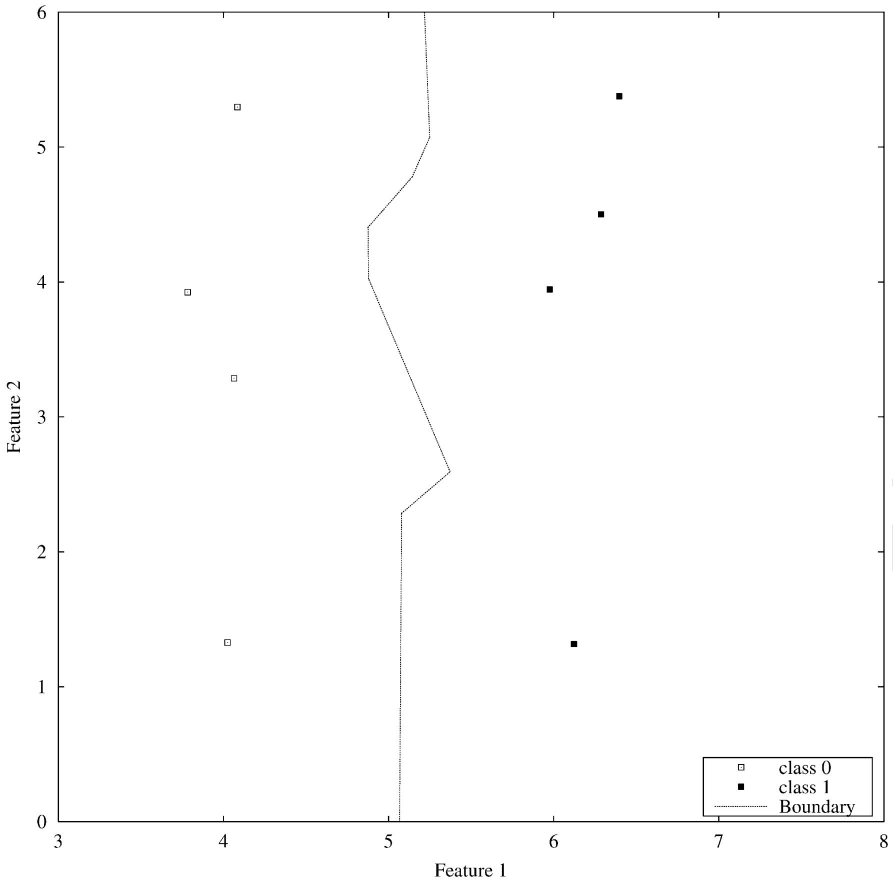

Fig. 3. The prototypes found by applying ABS on the data set in Fig. 1.

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 $(0 \leqslant \alpha \leqslant 1)$ 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. Fig. 3 illustrates the prototypes found by applying ABS on the data set in Fig. 1.

2.2.2. *Instance-filtering component*

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 techniques in our PGF algorithm.

Removing Border Instances. The first one is the ENN method introduced by [7]. 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

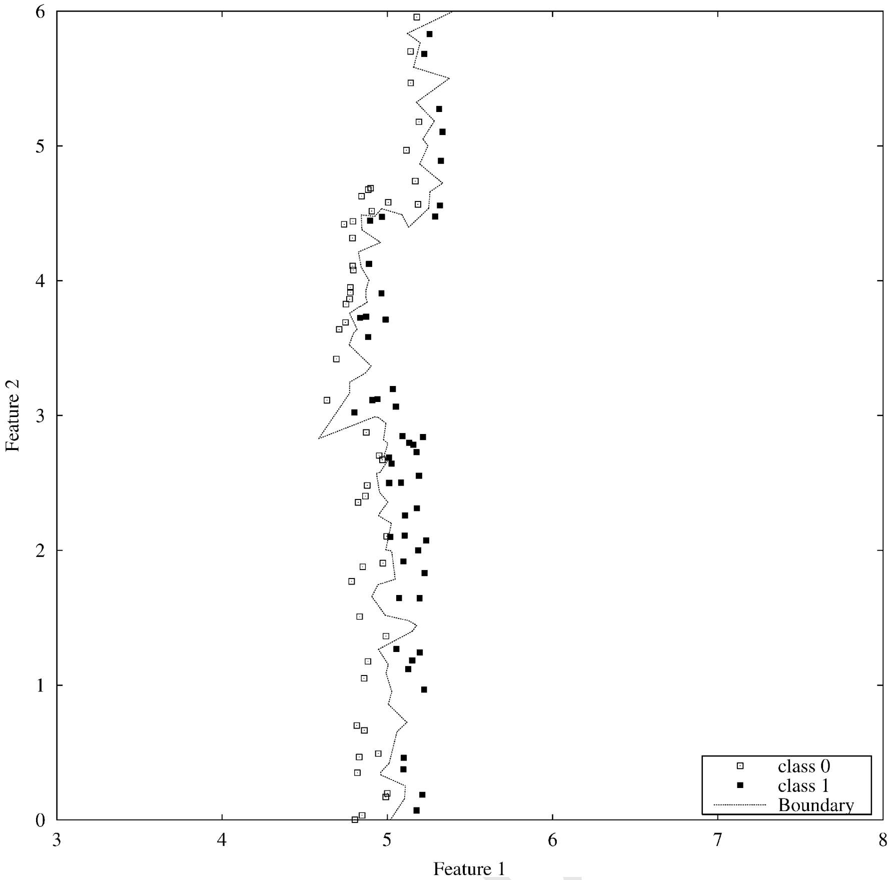

Fig. 4. The prototypes found by applying RT3 on the data set in Fig. 1.

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 [11]. 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. 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. Fig. 4 illustrates the prototypes found by applying RT3 on the data set in Fig. 1.

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 are classified by its NN. If it is correctly classified, classification accuracy of its NN 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.

2.2.3. *The PGF algorithm*

We propose two different integration algorithms which differ in the filtering granularity as well as the degree of coupling of the filtering component and the abstraction component.

*PGF*1. The first algorithm, called PGF1, conducts filtering on the original instances. As shown in Fig. 5, it first applies an instance-filtering method as a preprocessing step before prototype generation. Step 3 is the prototype generation based on ABS, our proposed instance-abstraction method. In prototype generation,

```
1  P = Training Set.
2  FILTER(P).
3  ABS component (Statements 2–10).
```

Fig. 5. The PGF1 algorithm.

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

Fig. 6. The PGF2 algorithm.

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. To achieve such a purpose, we add the procedure "FILTER(P)". just before the abstraction task. Thus, PGF1 essentially conducts filtering on the original instances.

*PGF*2. The second algorithm, called PGF2, conducts filtering on the intermediate prototypes in the process of prototype generation. As shown in Fig. 6, 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. The 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 reduction rate. On top of this, filtering can also remove noisy prototypes or instances and hence improving the classification accuracy. Fig. 7 depicts the prototypes found by applying PGF2 on the data set shown in Fig. 1. PGF2 can produce good abstraction prototypes at the bottom half of the figure where the decision boundary is smooth in this region. PGF2 is also able to produce good filtering prototypes at the upper half of the figure where the decision boundary is rugged in this region.

3. Empirical evaluation

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 [27] were tested in the experiments. 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). Table 1 shows the data sets and their corresponding code used in this paper.

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 of the data retention rate and the classification accuracy of 10-fold cross-validation were obtained for each data set. 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.

3.2. Results on PGF framework

Table 2 shows the average classification accuracy and data retention rate of 10-fold cross-validation across 35 real-world 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

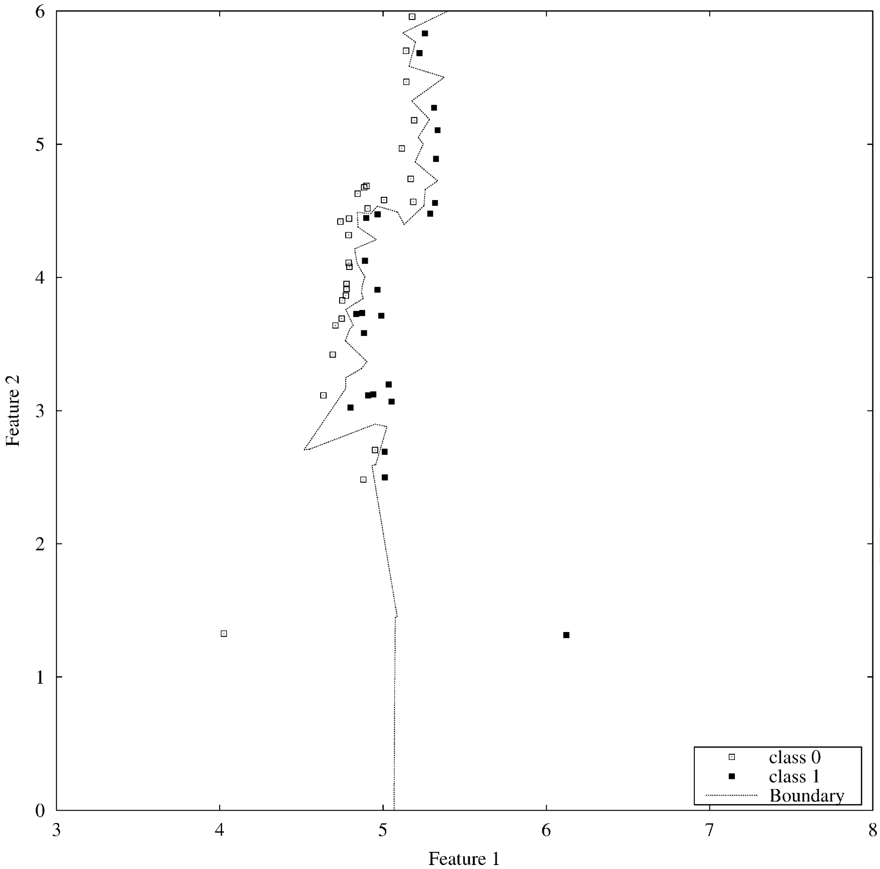

Fig. 7. The prototypes found by applying PGF on the data set in Fig. 1.

abstraction methods so that comparative analysis can be conducted. Table 3 shows the average classification accuracy and data retention rate of pure instance-filtering and instance-abstraction (ABS) methods, as well as C4.5 and KNN. The detailed performance of each algorithm for each individual data set can be found in Tables 4, 5, 6 and 7.

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.

3.2.1. *Analysis on PGF1*

*PGF*1–*ENN*. We investigate ENN and PGF1–ENN to analyze how the abstraction method can help ENN in PGF1. From Tables 2 and 3, it is found that the data retention rate of ENN is dramatically improved from 87.1% to 16.3% with less than 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 2 and 3 that the data retention rate of ABS is improved from 21.6% to 16.3% while keeping a similar classification accuracy.

*PGF*1–*RT*3. 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

Table 1
Data sets and their codes

Data set	Code
Automobile	Ab
Auto-Mpg	Am
Audiology	Au
Balance-scale	Ba
Breast-cancer-w	Bc
Car	Ca
Credit screening	Cs
Ecoli	Ec
Glass1	Gl
Hepati	He
Ionosphere	Io
Iris	Ir
Letter	Le
Liver	Li
Monk-1	M1
Monk-2	M2
Monk-3	M3
Mushroom	Mu
New-thyroid	Ne
Nursery	Nu
Optdigits	Op
Pendigits	Pe
Pima	Pi
Segmentation	Se
Shuttle	Sh
Sonar	Sn
Soyabean	Sb
Tic-tac-toe	Tt
Voting	Vo
Vowel	Vw
Wdbc	Wd
Wine	Wi
Wpbc	Wp
Yeast	Ye
Zoo	Zo

2.1% 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 2.4% 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 accuracy 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 is further refined by ABS to form more representative prototypes.

3.2.2. Analysis on 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. It results in significant improvement in data retention rate 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

Table 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

PGF1						PGF2					
PGF1–ENN		PGF1–RT3		PGF1–ACC		PGF2–ENN		PGF2–RT3		PGF2–ACC	
Acc.	Size	Acc.	Size	Acc.	Size	Acc.	Size	Acc.	Size	Acc.	Size
0.846	0.163	0.834	0.066	0.798	0.055	0.851	0.300	0.837	0.085	0.848	0.103

Table 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, pure abstraction method, C4.5 and KNN

Pure filtering						Pure abstraction		Other methods			
ENN		RT3		ACC		ABS		C4.5		KNN	
Acc.	Size	Acc.	Size	Acc.	Size	Acc.	Size	Acc.	Size	Acc.	Size
0.865	0.871	0.855	0.142	0.800	0.120	0.858	0.216	0.836	—	0.870	1.000

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 are 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 sets 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 1.8% 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 only 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 1.7% 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 will often be 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.

3.2.3. 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

Table 4
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

	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

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 the 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

Table 5
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

	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

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

Table 6
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

	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
Average	0.865	0.871	0.855	0.142	0.800	0.120	0.858	0.216

data retention rate and classification accuracy of ABS are degraded in PGF.

3.2.4. *Comparisons with other approaches*

In the second set of experiments, we compare PGF with existing algorithms, namely, C4.5 and KNN. In KNN, a range of k ($k = 1, 3, 5, 7, 9, 11, 13, 15, 20$) is tested and the best results are reported. Tables 2 and 3 show the average classification accuracy and data retention rate of 10-fold cross-validation of these algorithms across the same 35 data sets.

PGF (PGF2–ACC) performs slightly better than C4.5 in the average classification accuracy across all the data sets. When compared with KNN, PGF2–ACC stores only 10% of total data and gains a comparable accuracy. Hence, PGF2–ACC achieves comparable classification performance with state-of-the-art learning algorithms such as C4.5 and KNN. More importantly, PGF2–ACC

Table 7
The average classification accuracy and data retention rate (size) of 10-fold cross-validation for C4.5, KNN, PGF1–RT3, PGF2–RT3 and PGF2–ACC. The standard deviation of classification accuracy is given inside the bracket

			PGF					
	C4.5	KNN	PGF1–RT3		PGF2–RT3		PGF2–ACC	
Data	Accuracy	Accuracy	Accuracy	Size	Accuracy	Size	Accuracy	Size
Ab	0.794 (0.156)	0.766 (0.076)	0.555 (0.154)	0.090	0.542 (0.117)	0.137	0.586 (0.163)	0.202
Am	0.776 (0.056)	0.771 (0.082)	0.797 (0.083)	0.049	0.772 (0.156)	0.090	0.786 (0.076)	0.101
Au	0.756 (0.064)	0.761 (0.102)	0.606 (0.113)	0.155	0.588 (0.154)	0.169	0.672 (0.135)	0.130
Ba	0.792 (0.066)	0.775 (0.066)	0.831 (0.069)	0.039	0.855 (0.060)	0.033	0.853 (0.041)	0.012
Bc	0.939 (0.041)	0.960 (0.014)	0.957 (0.031)	0.009	0.966 (0.062)	0.012	0.963 (0.037)	0.026
Ca	0.928 (0.012)	0.956 (0.016)	0.931 (0.019)	0.048	0.946 (0.020)	0.076	0.935 (0.019)	0.176
Cs	0.832 (0.054)	0.807 (0.047)	0.845 (0.040)	0.023	0.826 (0.040)	0.034	0.842 (0.041)	0.019
Ec	0.822 (0.060)	0.822 (0.095)	0.872 (0.074)	0.036	0.852 (0.084)	0.079	0.833 (0.074)	0.117
Gl	0.666 (0.083)	0.681 (0.300)	0.570 (0.172)	0.047	0.550 (0.283)	0.066	0.649 (0.213)	0.051
He	0.773 (0.182)	0.805 (0.186)	0.819 (0.079)	0.033	0.805 (0.097)	0.031	0.818 (0.097)	0.031
Io	0.900 (0.032)	0.866 (0.058)	0.838 (0.089)	0.035	0.872 (0.088)	0.060	0.874 (0.074)	0.035
Ir	0.953 (0.063)	0.947 (0.043)	0.940 (0.054)	0.038	0.907 (0.089)	0.050	0.933 (0.104)	0.073
Le	0.692 (0.043)	0.810 (0.034)	0.659 (0.032)	0.189	0.661 (0.057)	0.240	0.701 (0.059)	0.206
Li	0.642 (0.054)	0.632 (0.089)	0.577 (0.081)	0.074	0.620 (0.076)	0.078	0.585 (0.119)	0.072
M1	0.960 (0.084)	0.969 (0.039)	0.928 (0.110)	0.151	0.944 (0.092)	0.243	0.939 (0.082)	0.250
M2	0.625 (0.079)	0.993 (0.016)	0.968 (0.022)	0.106	0.960 (0.029)	0.165	0.951 (0.062)	0.120
M3	0.988 (0.033)	0.955 (0.045)	0.950 (0.052)	0.055	0.951 (0.036)	0.055	0.950 (0.081)	0.093
Mu	0.997 (0.006)	0.999 (0.002)	0.996 (0.012)	0.009	0.990 (0.010)	0.007	0.995 (0.008)	0.010
Ne	0.921 (0.081)	0.972 (0.031)	0.889 (0.110)	0.031	0.934 (0.087)	0.036	0.925 (0.075)	0.059
Nu	0.909 (0.018)	0.863 (0.024)	0.834 (0.049)	0.074	0.844 (0.024)	0.077	0.853 (0.035)	0.144
Op	0.824 (0.029)	0.962 (0.045)	0.916 (0.036)	0.041	0.919 (0.037)	0.059	0.946 (0.032)	0.114
Pe	0.914 (0.015)	0.987 (0.009)	0.960 (0.031)	0.064	0.954 (0.007)	0.071	0.972 (0.028)	0.104
Pi	0.694 (0.085)	0.706 (0.114)	0.759 (0.121)	0.007	0.716 (0.111)	0.026	0.715 (0.078)	0.046
Se	0.951 (0.015)	0.967 (0.016)	0.936 (0.028)	0.071	0.941 (0.016)	0.086	0.952 (0.015)	0.143
Sh	0.989 (0.045)	0.987 (0.050)	0.974 (0.034)	0.022	0.983 (0.042)	0.023	0.985 (0.042)	0.142
Sn	0.706 (0.094)	0.876 (0.152)	0.697 (0.075)	0.107	0.740 (0.062)	0.122	0.789 (0.090)	0.131
Sb	0.930 (0.034)	0.908 (0.053)	0.867 (0.058)	0.090	0.891 (0.054)	0.121	0.861 (0.068)	0.156
Tt	0.862 (0.036)	0.914 (0.027)	0.859 (0.046)	0.136	0.845 (0.032)	0.139	0.865 (0.061)	0.197
Vo	0.960 (0.021)	0.935 (0.031)	0.915 (0.038)	0.025	0.915 (0.033)	0.042	0.926 (0.047)	0.061
Vw	0.779 (0.046)	0.992 (0.016)	0.914 (0.029)	0.198	0.923 (0.045)	0.275	0.944 (0.039)	0.210
Wd	0.944 (0.031)	0.945 (0.028)	0.949 (0.038)	0.014	0.942 (0.052)	0.023	0.942 (0.053)	0.092
Wi	0.888 (0.081)	0.954 (0.054)	0.948 (0.094)	0.032	0.955 (0.025)	0.043	0.949 (0.050)	0.086
Wp	0.676 (0.168)	0.701 (0.108)	0.703 (0.220)	0.056	0.717 (0.080)	0.037	0.748 (0.120)	0.015
Ye	0.545 (0.049)	0.524 (0.054)	0.516 (0.079)	0.074	0.561 (0.041)	0.081	0.523 (0.056)	0.103
Zo	0.926 (0.101)	0.970 (0.034)	0.900 (0.100)	0.089	0.920 (0.071)	0.098	0.920 (0.101)	0.085
Average	0.836	0.870	0.834	0.066	0.837	0.085	0.848	0.103

can drastically reduce the data size to less than 10% of the original size on average.

4. Conclusions

We have presented a new prototype generation method, called PGF, which integrates the strength of instance-filtering and instance-abstraction techniques. We investigate classification performance and the data retention rate of different variants of PGF on 35 real-world benchmark data sets. We have also conducted experiments using pure filtering, pure abstraction, as well as C4.5 and KNN. PGF is found to be effective in reducing the data set size while maintaining or even improving the classification accuracy.

Acknowledgements

This work was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region (Project No. CUHK 4385/99E).

References

[1] B.V. Dasarathy, Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques, IEEE Computer Society Press, Los Alamito, CA, 1991.

[2] O. Maron, A.L. Ratan, Multiple-instance learning for natural scene classification, Proceedings of the 15th International Conference on Machine Learning, 1998, pp. 341–349.

[3] R.F. Sproull, Refinements to nearest-neighbor searching in k-dimensional trees, Algorithmica 6 (1991) 579–589.

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

[5] P.E. Hart, The condensed nearest neighbor rule, IEEE Trans. Inf. Theory 14 (3) (1968) 515–516.

[6] G.W. Gates, The reduced nearest neighbor rule, IEEE Trans. Inf. Theory 18 (3) (1972) 431–433.

[7] D.L. Wilson, Asymptotic properties of nearest neighbor rules using edited data, IEEE Trans. Systems, Man, Cybern. 2 (3) (1972) 431–433.

[8] D.W. Aha, D. Kibler, Noise-tolerant instance-based learning algorithms, Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 1989, pp. 794–799.

[9] D.W. Aha, D. Kibler, M.K. Albert, Instance-based learning algorithms, Mach. Learning 6 (1991) 37–66.

[10] J. Zhang, Selecting typical instances in instance-based learning, Proceedings of the Ninth International Conference on Machine Learning, 1992, pp. 470–479.

[11] D.R. Wilson, T.R. Martinez, Instance pruning techniques, Proceedings of the 14th International Conference on Machine Learning, 1997, pp. 403–411.

[12] D.R. Wilson, T.R. Martinez, Reduction techniques for instance-based learning algorithm, Mach. Learning 38 (2000) 257–286.

[13] D.R. Wilson, T.R. Martinez, An integrated instance-based learning algorithm, Comput. Intell. 16 (1) (2000) 1–28.

[14] C.L. Chang, Finding prototypes for nearest neighbor classifiers, IEEE Trans. Comput. 23 (3) (1974) 1179–1184.

[15] G. Bradshaw, Learning about speech sounds: the Nexus project, Proceedings of the Fourth International Workshop on Machine Learning, 1987, pp. 1–11.

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

[17] S. Salzberg, A nearest hyperrectangle learning method, Mach. Learning 6 (1991) 251–276.

[18] D. Wettschereck, A hybrid nearest-neighbor and nearest-hyperrectangle algorithm, Proceedings of the Seventh European Conference on Machine Learning, 1994, pp. 323–335.

[19] P. Domingos, Unifying instance-based and ruled-based induction, Mach. Learning 24 (1996) 141–168.

[20] P. Datta, D. Kibler, Learning prototypical concept description, Proceedings of the 12th International Conference on Machine Learning, 1995, pp. 158–166.

[21] P. Datta, D. Kibler, Symbolic nearest mean classifier, Proceedings of the 14th National Conference of Artificial Intelligence, 1997, pp. 82–87.

[22] S. Cost, S. Salzberg, A weighted nearest neighbor algorithm for learning with symbolic feature, Mach. Learning 10 (1993) 57–78.

[23] J.C. Bezdek, T.R. Reichherzer, G.S. Lim, Y. Attikiouzel, Multiple-prototype classifier design, IEEE Trans. Systems, Man Cybern. 28 (1998) 67–79.

[24] A. van den Bosch, Instance-family abstraction in memory-based language learning, Proceedings of the 16th International Conference on Machine Learning, 1999, pp. 39–48.

[25] W. DuMouchel, C. Volinsky, T. Johnson, C. Cortes, D. Pregibon, Squashing flat files flatter, Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1999, pp. 6–15.

[26] C. Stanfill, D. Waltz, Toward memory-based reasoning, Commun. ACM 29 (1986) 1213–1228.

[27] C.L. Blake, C.J. Merz, UCI Repository of Machine Learning Database, Department of Information and Computer Science, University of California Irvine, Irvine, CA, 1998, http://www.ics.uci.edu/ ∼mlearn/MLRepository.html.

About the Author—WAI LAM received a Ph.D. in Computer Science from the University of Waterloo, Canada in 1994. He worked as a visiting Research Associate at Indiana University Purdue University Indianapolis in 1995 and as a Postdoctoral Fellow at Distributed Adaptive Search Laboratory in University of Iowa in 1996. Currently he is an Assistant Professor at Department of Systems Engineering and Engineering Management in the Chinese University of Hong Kong. His current interests include data mining, intelligent information retrieval, machine learning, reasoning under uncertainty, and digital library.

About the Author—CHI-KIN KEUNG is a M.Phil. student at the Department of Systems Engineering and Engineering Management, the Chinese University of Hong Kong. He received the Bachelor degree at the same department in 1998. His research interests are machine learning, artificial intelligence and data mining, especially for instance-based learning and classification.

About the Author—CHARLES LING obtained his B.Sc. in Computer Science at Shanghai JiaoTong Univ in 1985, and his M.Sc. and Ph.D. in 1987 and 1989 respectively, from Computer Science at University of Pennsylvania. Since then he has been a faculty member at Univ of Western Ontario (UWO), Canada. He is currently an Associate Professor, an adjunct Professor at U of Waterloo, and Director of Data Mining and E-commerce Laboratory. He is on-leave from UWO, and is a Visiting Researcher at Microsoft Research in China, leading log mining research projects. His research interests include data mining, machine learning, and cognitive science. He has also led successfully several data mining projects for banks and insurance companies in Canada.