An efficient incremental mining algorithm for compact realization of prototypes

P. Viswanath, M. Narasimha Murty [1]

Department of Computer Science and Automation
Indian Institute of Science
Bangalore - 560 012
India
{viswanath,mnm}@csa.iisc.ernet.in

1 Introduction

There are two phases in pattern classification, *viz* design phase (abstractions are created/learnt), classification phase (abstractions are used to classify a test pattern). Classification based on neural networks [1] and genetic algorithms [2] needs more design time. Where as nearest neighbor classifiers [3] have zero design time, they need more classification time. PC-tree (pattern count tree) based classifier [4] strikes a balance between the above mentioned two categories.

PC-tree is a data structure originally used in incremental data mining to store the database. It can also be used in pattern classification for compact realization of prototypes [4]. PC-Tree based classifier has advantages over conventional classifiers(like nearest neighbor, k-nearest neighbor classifier) with respect to storage space, classification time, and classification accuracy(CA).

In this paper we propose a classification algorithm which is *superior* to PC-tree based classifier in *every aspect*. This classifier is based on a new data structure called *partitioned pattern count tree (PPC-tree).*

PPC-tree is more *compact* than PC-tree. This results in *reduction of space required and classification time*. The effective number of training patterns stored in PPC -tree is in orders more than the actual number of training patterns, this results in *increase of the CA*.

[1] Author to whom correspondence must be addressed.

2 The Data Structures

For completeness and comparison purposes PC-tree is briefly introduced. For details see [4]

2.1 Structure of PC-tree

PC-tree is a complete and compact representation of training patterns. Patterns are stored in a tree stucture (PC-tree) where each feature [2] occupies a node. Every training pattern is present in a path from root to leaf. Two patterns of a class can share a common node for their respective n^{th} feature if,
1. Their n^{th} feature is of same value, and
2. For all i, $(i < n)$, i^{th} feature values are same.
A node will have along with feature value a count indicating how many patterns are sharing that node. Many patterns sharing a common node in the tree leads to a compact representation of the training patterns
Example 1: Consider the following set of patterns shown in Table 1, which belong to a class (say with label i).The corresponding PC-Tree is shown in Figure 1. Each node of the tree is of format (*feature : count*) .

$Pattern\#$	Features					
1	a	b	c	x	y	z
2	a	b	d	x	y	z
3	a	e	c	x	y	u
4	f	b	c	x	y	v

Table 1 : Example Patterns

2.2 Structure of PPC-tree

In PPC-tree all training patterns for each class are vertically partitioned into p partitions according to some fixed partitioning criteria (which needs to be fixed for a given probelm). For each class, for each partition, a PC-tree is

[2] if the patterns are of binary valued features (say 0's and 1's), then position number of each 1 is enough to represent and results in more compact representation

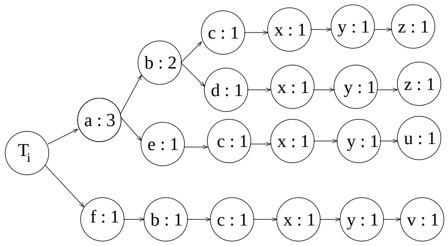

Fig. 1. PC-tree.

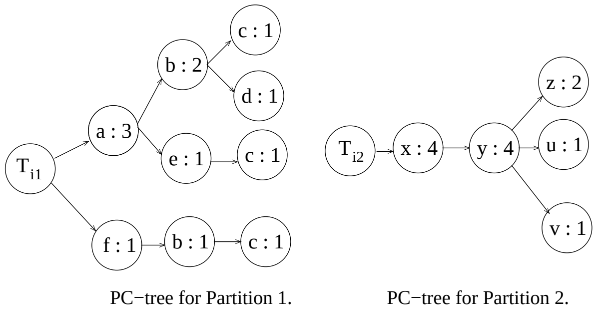

Fig. 2. Partitioned PC-tree.

built. This results in a more compact representation because for two patterns to share a common node for n^{th} feature, conditions required are-
1. Their n^{th} feature is of same value, and
2. For all i within the partition of n^{th} feature, $(i < n)$, i^{th} feature values are same.
The 2^{nd} condition is less restrictive when compared with that of PC-tree as it is limited to a partition only.
Example 2: Consider the same set of patterns as that of example 1 (shown in Table 1), let the partitioning criterion be: number of partitions, $p = 2$ and first 3 features go into partition 1, next 3 features into partition 2. The resulting PPC-tree is shown in Figure 2.

The number of nodes in PPC-tree is 16, but that in PC-tree is 22.

The important point to note is, even though 4 patterns are stored in the PPC-tree, the number of patterns that can be retrieved is 12, *viz:* (a,b,c,x,y,z), (a,b,c,x,y,u), (a,b,c,x,y,v), (a,b,d,x,y,z), (a,b,d,x,y,u) *etc.*; that is all combinations of sub-patterns across the partitions are possible.

2.3 Comparison of PPC-tree with PC-tree

(1) PPC-tree is more compact than PC-tree. This results not only in reduced storage space requirement but also in reduction of classification time. If each pattern is seen as a string of features then PC-tree does factoring out of common prefixes. In the case of PPC-tree, within each partition factoring out of common prefixes is done.

(2) PPC-tree can also be built by accessing the training patterns only once as that of PC-tree [4]. So there is no extra design time needed.
(3) Effective number of training patterns stored in PPC-tree is more than the actual number of training patterns. This can result in increased classification accuracy based on the suitability of the partitioning criteria.

3 Algorithm for PPC-tree based classification(PPC-kNNC)

INPUTS:
(1) Labelled training patterns
(2) s - Test pattern
(3) p - Number of partitions
(4) c - Number of classes
(5) d - Dimensionality of patterns
(6) k - Number of nearest neighbours

OUTPUT:
(1) o_x - Label of the test pattern s.

ASSUMPTIONS:
(1) d/p is an integer.
(2) Partitioning citeria :
From $(k-1)(d/p)+1$ to $k(d/p)$ features of a pattern go into k^{th} partition. The order of features within a partition is same as that in the pattern.

STEPS:
(1) Construction of the PPC-tree:

For each class $i = 1$ to c
    For each partition $j = 1$ to p
        Construct PC-tree T_{ij}

Note: Refer [4] for details of constructing PC-tree.

(2) Classification:

Partition the test pattern, s into p partitions $s_1, s_2, \ldots, s_p$ by using the same partitioning criteria as in Assumption (2).

For each class $i = 1$ to c
    For each partition $j = 1$ to p

Classifier	Classification Time (secs)	CA (%)	Space to store training patterns (KB)	Number of Training Patterns
k−NNC	106	89.44	772.00	
PCC	99	89.56	393.14	2000
PPC−kNNC	87	92.65	339.36	
k−NNC	200	91.90	1544.00	
PCC	189	91.96	783.84	4000
PPC−kNNC	153	94.03	600.18	
k−NNC	329	93.55	2574.62	
PCC	294	93.61	1300.57	6670
PPC−kNNC	229	94.87	901.71	

k−NNC : k−nearest neighbor classifier (Ref.[3])
PCC : PC−tree based classifier (Ref.[4])
PPC−kNNC : Partitioned PC−tree based classifier

Fig. 3. Comparison of time, space and CA

For each branch b_l in PC-tree T_{ij}
Find the the squared Euclidean distance between b_l and s_j, let it be C_{ij}^l
Find k smallest distances in non-decreasing order; let them be $C_{ij}^1, C_{ij}^2, \ldots, C_{ij}^k$
// to find k-near distances within each class
For $m = 1$ to k
Find $C_i^m = \sum_{j=1}^{p} C_{ij}^m$
// to find k-nearest neighbors across all classes
Find k smallest distances in non-decreasing order among all C_i^m, for $i = 1$ to c and for $m = 1$ to k. Let them be $C^1, C^2, \ldots, C^k$, let the corresponding class labels (i value in C_i^m) be $o_1, o_2, \ldots, o_k$.
For $l = 1$ to k
Compute the weight, $w_l = (C^l - C^k)/(C^1 - C^k)$
For $n = 1$ to c
//Compute the total weight for each class,
$sum_n = \sum_{m=1}^{k}(w_m)$ where $(o_m == n)$
If sum_x is maximum for $x \in \{1, 2, \ldots, c\}$, then
Output (label $= o_x$).

4 Experiments

We conducted experiments using handwritten digit data [1,4,5]. The results are shown in Figure 3 The training data consists of 667 patterns for each class of digits labelled from 0 to 9, totalling to 6670 training patterns. The test data consists of 3333 patterns. We conducted experiments with 2000(200 $\times$ 10), 4000(400 $\times$ 10),and 6670 (667 $\times$ 10) training patterns seperately. We compared our results with k-NNC [3] (which is known to be very robust with

Classifier	CA (%)	Classifier	CA (%)
Leaders [5]	92.9	EALSM [1]	92.0
Medoids [5]	91.18	ALSM [1]	91.1
k−NNC [3]	93.55	GLSM [1]	91.0
GSM [1]	90.8	MLP [1]	90.4
PCC [4]	93.61	PPC−kNNC	94.87

Fig. 4. CA for various classifiers

zero design time) and with PC-tree based classifier(PCC) [4]. Because disk access time needed to construct PPC-tree is same as that of PC-tree, it is not shown in Figure 3. We have chosen the value of k to be 15 for all the three classifiers (it is known to give better results as for as classification accuracy is concerned) and the value of p is 3 for PPC-kNNC. It can be seen from Figure 3 that PPC-kNNC uses computational resources very efficiently and gives better classification accuracy.

It can also be observed that PPC-kNNC performs better with 4000 training patterns than that of PCC with 6670 training patterns.

Figure 4 shows the classification accuracy for the classifiers given in recent literature based on 6670 training patterns and 3333 test patterns. From this, we can infer that PPC-kNNC gives the best classification accuracy.

5 Conclusion

In this paper, a very compact data structure , PPC-tree for prototype storage and an efficient classification algorithm, PPC-kNNC are presented. PPC-tree can be built incrementally. It is observed that PPC-tree with PPC-kNNC reduces the processing time and memory requirements and gives better classification accuracy.

References

[1] M. Prakash, M. Narasimha Murty, Growing subspace pattern recognition methods and their neural networks models, IEEE Trans. Neural Networks 8 (1)(1997) 161-168.

[2] L.I. Kuncheva, L.C. Jain, Nearest neighbor classifier:Simultaneous editing and feature selection, Pattern Recognition Lett. 20 (1999) 1149-1156.

[3] R.O. Duda, P.E. Hart, Pattern Classification and Scene Analysis, Wiley, New York, 1973.

[4] V.S. Ananthanarayana, M.N. Murty, D.K. Subramanian, An incremental data mining algorithm for compact realization of prototypes, Pattern Recognition 34 (2001) 2249-2251.

[5] T. Ravindra Babu, M. Narasimha Murty, Comparison of Genetic Algorithms based prototype selection scheme, Pattern Recognition 34 (2001) 523-525.