An Efficient Classification Algorithm based on Pattern Range Tree Prototypes

Shreeranga P.R.
shreerangakamath@gmail.com

Akshat Vig
akshatvig19@gmail.com

Dr. V.S. Ananth Narayana
ananathvs1967@gmail.com

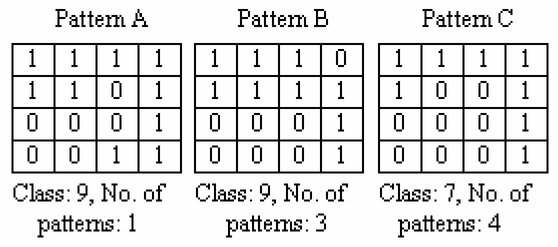
Dept. of Information Technology, National Institute of Technology Karnataka, Surathkal, India

Abstract

Abstraction based Pattern Classifier has drawn a lot of attention today. This type of classifier has two phases. They are: design phase, where the abstractions are created and classification phase, where the classification is done using these abstractions. Techniques like neural networks, genetic algorithms require very high design time. In other techniques like nearest neighbor classifier, the design time is near to zero but the classification time is predominantly high. Pattern Count Tree (PC- tree) based classifier was proposed as an abstraction based classifier that strikes a balance between the design time and the classification time. In this paper, we are going to propose a novel data structure called Pattern Range Tree (PR-tree) and a pattern classifier based on PR-tree. Experimental results presented in this paper show that PR-tree based classifier (PRC) is more efficient than PC-tree based classifier (PCC) in terms of storage space, processing time and classification accuracy.

1. Introduction

Data mining involves a vast variety of sub fields, one such challenging field is pattern recognition. Pattern recognition aims to classify data based on a priori knowledge or extracting statistical information from the patterns available. Generally, in neighborhood-based classifiers, for a test pattern, we find the extent of match with each of the training patterns and reduce our selection to the top k qualifying patterns. The neighborhood-based classifiers like PCC [3], k-nearest neighbor classifier (k-NNC) [4] classify the test pattern based on the number of representative training patterns selected for each class and the number of common features between the test and the selected training patterns. Neighborhood based classification algorithms' accuracy depends on the nature of training set to a large extent.


Pattern A

1	1	1	1
1	1	0	1
0	0	0	1
0	0	1	1

Class: 9, No. of patterns: 1

Pattern B

1	1	1	0
1	1	1	1
0	0	0	1
0	0	0	1

Class: 9, No. of patterns: 3

Pattern C

1	1	1	1
1	0	0	1
0	0	0	1
0	0	0	1

Class: 7, No. of patterns: 4

Figure 1. Training patterns

Consider for example, the training patterns for class '9' and '7' as shown in Figure 1. Here each pattern is a matrix of size 4X4 where a '1' indicates the presence of feature and a '0' indicates the absence of feature. Let us assume that all other classes also have such training patterns. Let us consider a test pattern, Pattern D as shown in Figure 2, to be classified. For a neighborhood based classifier like k-NNC with number of nearest neighbors k=8, assume that Pattern A (number of such patterns = 1), Pattern B (number of such patterns = 3) and Pattern C (number of such patterns = 4) are 8 nearest neighbors to Pattern D.

1	1	1	1
1	1	0	1
0	0	0	1
0	0	1	1

Figure 2. Test pattern

Table 1 illustrates the extent of match of Pattern D with different training patterns. A common feature between two patterns is either a nonzero value or a zero value in both the patterns at the same position. Weight for each training pattern (W_x), which is given in the last column of Table 1 is calculated as follows.

Let C_x = Number of common features between the test pattern and the training pattern X.

Table 1. Details of classification of pattern D				
Pattern X	Class	Number of common features with Pattern D (C_x)	Number of Patterns	Weight (W_x)
A	9	16	1	1
B	9	13	3	0
C	7	14	4	1/3

C_{min}=Minimum value of C_x among the 'k' C_x values.
C_{max}=Maximum value of C_x among the 'k' C_x values.
Weight for the training pattern X is given by:
$W_x = (C_x - C_{min})/(C_{max} - C_{min})$.

Cumulative weight for a particular class is obtained by adding up the weights of the patterns among 'k' nearest neighbors corresponding to that class. For example, for class '9', there are totally 4 patterns among the 8 nearest neighbors. So, its cumulative weight is 1+0+0+0=1. For class '7', there are totally 4 patterns among the 8 nearest neighbors. So, its cumulative weight is 1/3+1/3+1/3+1/3=4/3. Considering these results, the neighborhood-based classifier will classify Pattern D into class '7', since it has the maximum cumulative weight (i.e., 4/3). But, actually Pattern D has an exact match with Pattern A of class '9'. The cause of this misclassification is the number of patterns with which the test pattern (Pattern D) had an exact match (Pattern A) or number of patterns closer to the test pattern was less in the training data as compared to the number of patterns of the class into which the test pattern was finally classified (Pattern C). We call such training patterns 'rare patterns', since the patterns similar or exactly same as them are rare in the training data. In Figure 1, Pattern A is rarer as compared to Pattern C. In this paper, we propose a classification algorithm that considers rareness of patterns in training data.

In PC-tree [3], there exists a node for every feature in the pattern. The structure we are going to propose need not maintain a node for every feature in the pattern. So, the structure proposed is more compact than PC-tree. We call this structure Pattern Range Tree (PR-tree). The proposed pattern classification algorithm uses this data structure as abstraction.

The organization of rest of the paper is as follows. We give a detailed description of the data structure, PR-tree, in section 2. Concept of rareness is discussed in section 3. We give PR-tree based classification algorithm in section 4 and the experiments in section 5. Section 6 concludes the paper.

2. Pattern Range Tree (PR-tree)

PR-tree is a data structure, which is used to store the training patterns in a compact manner. Each node of the tree consists of the fields shown in Figure 3. In the node 'FEATURE' field specifies the position of nonzero value of the pattern. 'CHILD' field represents the pointer to the following path. 'SIBLING' field represents pointer to the node which indicates subsequent other paths from the parent of the node

FEATURE	CHILD	SIBLING	FLAG

Figure 3. PR-tree

under consideration. 'FLAG' field value gives the information about the existence of the features between the 'FEATURE' field value of current node and the 'FEATURE' field value of its immediate parent. It can take either a positive value or a negative value. A negative value of 'FLAG' field signifies the absence of all the features between the 'FEATURE' field value of current node and 'FEATURE' field value of its immediate parent. The first node after the root node in a branch always stores a negative value in the 'FLAG' field, since it corresponds to the first feature in the pattern. A positive 'FLAG' field value signifies the presence of all features between the 'FEATURE' field value of current node and the 'FEATURE' field value of its immediate parent. The absolute value of 'FLAG' field indicates the number of patterns sharing the particular node. We illustrate the construction of PR-tree in Figure 4 corresponding to the patterns shown in Table 2. In Figure 4 the horizontal links indicate the 'CHILD' pointers and the vertical links indicate the 'SIBLING' pointers. In each node the 'FEATURE' field and the 'FLAG' field values are separated by '/'.

Table 2.Representation of patterns		
Pattern #	Class label	Features
1	0	1,2,3,4,5,8,9,12,13,14,15
2	6	1,5,9,10,11,12,13,14,15,16
3	7	1, 2, 3, 7, 11, 15

In Table 2 above, 'Features' column represents the positions of nonzero values of the pattern confined by 4X4 matrix. For example, the positions of nonzero values for Pattern A shown in Figure 1 are 1, 2, 3, 4, 5, 6, 8, 12, 15 and 16.

Figure 4 shows the PR-tree construction in three different stages for Pattern # 1, 2 and 3 of Table 2. The tree has the root represented by 'T'. Figure 4 (A) shows PR-tree after the insertion of Pattern # 1. In Pattern # 1, the features from '1' to '5' are consecutive. So, we create a node 'n_1' for feature '1' with value 1 in n_1's 'FEATURE' field and then a child node 'n_2' for feature '5' with value 5 in n_2's 'FEATURE' field. For the node n_1, the 'FLAG' field value is -1. This is because (i) it is the first feature of the pattern (ii) it is shared by only one pattern so far

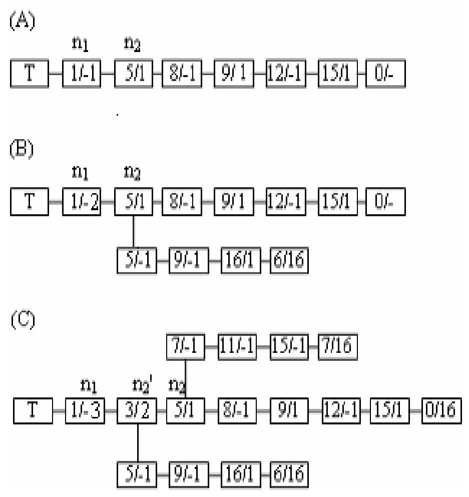


Figure 4. PR-tree construction stages

. For the node n_2 the 'FLAG' field value +1 indicates that all the features (i.e. '2', '3' and '4') between the 'FEATURE' field value of the present node (i.e., 5 of n_2) and the 'FEATURE' field value of its immediate parent (i.e., 1 of n_1) are present in Pattern # 1. For next feature '8' of Pattern # 1, we create a node with 'FEATURE' field value equal to 8. The 'FLAG' field value for this node is -1. This is because there are no features between the 'FEATURE' field value of its immediate parent node (n_2) and the 'FEATURE' field value of this node. This procedure is continued for the remaining part of Pattern # 1. In all the nodes created for Pattern # 1, absolute value of the 'FLAG' fields is 1. It indicates that the nodes are not shared by any other patterns so far. At the end of this branch we attach a node with '0' in the 'FEATURE' field which indicates that Pattern #1 belongs to class '0'. We call this node, the label node. The 'FLAG' field of this node is not assigned any value for now (indicated by '-'). The 'FLAG' field of the last node of each branch is assigned a value after the construction of the whole tree.

Figure 4 (B) shows PR-tree after the insertion of Pattern # 2. Pattern # 2 shares a common prefix ('1') with Pattern # 1. So, increment the absolute value of the 'FLAG' field for the node n_1. Now the new 'FLAG' field value for node n_1 is -2. The absolute value of the 'FLAG' field i.e., 2 indicates that two patterns share the node n_1. Add the remaining nodes for Pattern # 2 as the sibling branch of node n_2.

Figure 4(C) shows PR-tree after the insertion of Pattern # 3. Pattern # 3 contains a common prefix ('1', '2' and '3') with Pattern # 1. So we can share the nodes for these features with Pattern # 1. But we do not have a node for feature '3', since it is represented implicitly by the positive 'FLAG' field value of the node n_2. So a rearrangement is needed. That is, we need to insert a node (n_2') for feature '3' between nodes n_1 and n_2, with 3 in its 'FEATURE' field and +1 in its 'FLAG' field. After introducing this new node (n_2'), we should check whether the child node i.e., n_2 has any siblings. Since n_2 has a sibling, we should shift it such that it becomes the sibling of the node n_2'. After this rearrangement, increment the absolute values of the nodes representing the common prefix for Pattern # 3 and Pattern # 1. Add the rest of the nodes for Pattern # 3 as the sibling branch of the node n_2.

Use of the 'FLAG' field in label nodes and the values specified in Figure 4 (C) in those 'FLAG' fields will be described in section 3.1.

2.1. Algorithm for the construction of PR-tree

The Assignment of 'FLAG' field values of the label nodes in PR-tree constructed by the algorithm presented below is described in section 3.1.

INPUT:

T_r: Training Data Set.

OUTPUT:

PR-tree constructed for T_r.

ALGORITHM:

```
Let the root of the PR-tree be T.
For each pattern $t_i$ Є $T_r$
    Let $M_i$ be the sequence of positions of nonzero values in $t_i$
    If no branch starting from T corresponding to a prefix ofMi exists
    Then
        Create new branch by calling Makebranch($t_i$ ,$M_i$).
        Add it to T as a branch.
    Else
        Consider the branch $E_b$ with longest prefix pattern corresponding to $M_i$ among all the branches. Let prefix of $M_i$ corresponding to this branch be $M'_i$ and the rest of the sequence of values in $M_i$ be $M''_i$ (i.e., $M'_i$ concatenated with $M''_i$ should give $M_i$ )
        If there is no node in $E_b$ with 'FEATURE' field value equal to last value of $M'_i$
        Then
            Introduce a node corresponding to the last value of M'i and rearrange the tree appropriately.
        Else
            No rearrangement required.
        Endif
```

```
            Let $E'_b$ be the part of the branch $E_b$
            corresponding to $M'_i$. Put the Values of $M'_i$ in
            $E'_b$ by incrementing the absolute values of
            corresponding 'FLAG' fields without
            changing the sign. Let $E''_b$ be the part of the
            branch $E_b$ corresponding to $M''_i$. Create $E''_b$ by
            calling Makebranch($t_i$, $M''_i$). Add $E''_b$ as a new
            path following $E'_b$.
    Endif
Endfor
Makebranch (pattern t, sequence of positions of
nonzero values M)
Let B be the branch corresponding to M.
Initially B is empty.
For each value p in M
    If p is the position corresponding to the first value
    of a sequence of consecutive nonzero values in
    pattern t
    Then
        Create node X with 'FEATURE' field value
        equal to p and 'FLAG' field value equal to -1.
    Else
        If p is the position corresponding to the last
          value of a sequence of consecutive nonzero
          values in pattern t.
        Then
            create node X with 'FEATURE' field value
            equal to p and 'FLAG' field value equal to +1
        Else
            If p is the position corresponding to a nonzero
              value which is not a part of a sequence of
              consecutive nonzero values in pattern t
            Then
                create node X with 'FEATURE' field
                value equal to p and 'FLAG' field value
                equal to -1.
            Endif
        Endif
    Endif
    If node X is created with 'FEATURE'
      field value equal to p
    Then
        If B is empty
        Then
            Add node X as the first node in B
        Else
            Add node X as the child of the
            last node in B.
        Endif
    Endif
Endfor
Add the label node at the end of B with 'FEATURE'
field value equal to the class of the pattern t.
Return B
End Makebranch
```

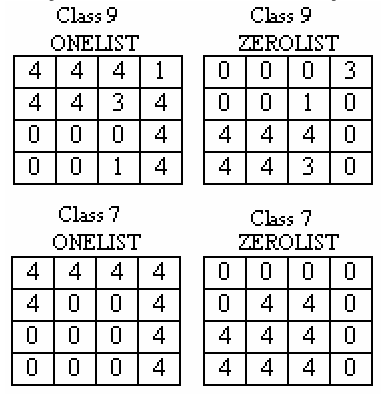
3. Rareness

Training data contains training patterns for each class. Some of the patterns in the training data may not have a close resemblance with the patterns in the training data of the same class. We call such patterns 'rare patterns'; since the patterns similar to them are rare in the training data and we call this property as 'rareness'. In Figure 1, if there were more patterns of class '9', similar to Pattern A, then they would have replaced some patterns in 8 nearest neighbors of Pattern D shown in Figure 2 and Pattern D would have been correctly classified into class '9' rather than class '7'. To handle this we use the concept of rareness.

3.1. Realization of rareness of a training pattern

One of the methods for finding the rareness of a training pattern is by calculating the number of common features between the training pattern under consideration and each of the training patterns of its class. Sum of these common features divided by the number of training patterns in the class will give the average number of common features between the training pattern under consideration and the training patterns in its class. Let the reciprocal of this quantity be 'R'. We call 'R', the 'rareness factor'. 'R' provides a measure of rareness.

Following example illustrates the calculation of 'R'. Consider the 4X4 patterns given in Figure 1 for class '9' and class '7'. For the sake of brevity let us assume that there are no other patterns in the training data other than the patterns shown in Figure 1 for class '9' and class '7'. We can compare each of these patterns with each pattern in the training data of the same class to find out the value of their 'R'. A simple way to do this is by using the matrices shown in Figure 5.



Class 9 ONELIST

4	4	4	1
4	4	3	4
0	0	0	4
0	0	1	4

Class 9 ZEROLIST

0	0	0	3
0	0	1	0
4	4	4	0
4	4	3	0

Class 7 ONELIST

4	4	4	4
4	0	0	4
0	0	0	4
0	0	0	4

Class 7 ZEROLIST

0	0	0	0
0	4	4	0
4	4	4	0
4	4	4	0

Figure 5. ONELIST and ZEROLIST matrices for patterns shown in Figure1.

In Figure 5, for class '9' and class '7', each entry in the ONELIST matrix of a particular class indicates the number of training patterns of the corresponding class which have a nonzero value in the corresponding position. Similarly, each entry in the ZEROLIST of a particular class indicates the number of training patterns of the corresponding class, which have a zero value in the corresponding position. The following three steps calculate value of 'R' for a training pattern.

Step 1: Check the value at each position one by one in the training pattern. If the value is nonzero, take the corresponding entry from the ONELIST of the class of the training pattern. Else, if the value is zero we take the corresponding entry from the ZEROLIST of the class of the training pattern. Add these entries together to get the total number of common features between the training pattern under consideration and each of the training patterns in the class.

Step 2: Divide the final sum obtained in Step 1 by the number of patterns in the class of the training pattern.

Step 3: Calculate the reciprocal of the value obtained in Step 2. This value is 'R' for the training pattern.

Table 3. Illustration of calculation of 'R'

Pattern	Step 1	Step 2	Step 3
A	55	55/4	4/55
B	61	61/4	4/61
C	64	64/4	4/64

Table 3 illustrates Step 1, Step 2 and Step 3 for Pattern A, Pattern B and Pattern C shown in Figure 1. The column headed by Step 3 contains the value of 'R' for the corresponding patterns.

In Figure 4 (C), the values stored in the 'FLAG' field of the last node of each branch are the values obtained in Step 1 described above for the patterns represented in Table 2. These values can be calculated and stored in a single scan of the tree after the whole tree is constructed with the help of ONELIST and ZEROLIST. Using these values, 'R' can be calculated during classification.

4. PR-tree based Classifier (PRC)

So far in this paper, we have explained the concept of rareness and how rareness can cause misclassification of patterns. But, rareness need not be the cause of all the misclassifications. So, we need to use certain criteria to consider the rareness of patterns for classification. In this section, we present an algorithm, which does this. Here we find the 'k' nearest neighbors in the PR-tree for the test pattern. Then we find the cumulative weight corresponding to each class. Next, we find the difference between the highest and second highest cumulative weights. This gives a measure of closeness between the cumulative weights for the two classes. Then this difference is compared with a small constant called *threshold*. If the difference is greater than *threshold* then the test pattern is classified into the class with highest cumulative weight. Otherwise, the value of 'R' for each of the 'k' nearest neighbors is found out. Then, either (i) if the pattern with highest 'R' belongs to the class with second highest cumulative weight, the test pattern is classified into the class with second highest cumulative weight, or (ii) if the pattern with highest 'R' does not belong to the class with second highest cumulative weight, test pattern is classified into the class with highest cumulative weight.

The constant *threshold* is the maximum difference between the highest and the second highest cumulative weights of the classes, required to consider the rareness of the patterns for classification. Value of *threshold* needs to be calculated experimentally. Generally, this is close to 1. Very high values for the *threshold* may cause decrease in classification accuracy because of the over consideration of rareness. Minimum value for *threshold* is zero. Incase it is zero the algorithm just resolves the tie between the cumulative weights for two classes when the two highest cumulative weights are equal.

The classification algorithm proposed is given below.

INPUTS:

1. T_s – Test patterns
2. K – Number of nearest neighbors
3. NOC – Number of classes
4. T – PR-tree corresponding to the set of training patterns
5. *Threshold*

OUTPUT:

Class label, Label $_i$ for each test pattern s_i belonging to T_s according to the classification algorithm.

ALGORITHM:

```
For each pattern, $s_i$ belonging to $T_s$ let $n_i$ be the set of
    positions of non-zero values corresponding to $s_i$.
        For each branch $b_j$ belonging to PR-tree T
        count the number of common features
        between $n_i$ and $b_j$, let it be $c_{ij}$. (Note that the
        feature which is absent in both $n_i$ and $b_j$ is also
        counted as a common feature). Find K largest
        counts in non increasing order; let them be $c_{i1}$,
        $c_{i2}$, $c_{i3}$ …$c_{ik}$ and let corresponding branches in
        the PR- tree be $b_1$, $b_2$, $b_3$ … $b_k$.
        Let $o_1$, $o_2$, $o_3$ …ok be the labels associated
        with branches $b_1$, $b_2$, $b_3$ …$b_k$.
         Compute the weight $W_p$
        $= (c_{ip} - c_{ik}) / (c_{i1} - c_{ik})$, for each p = 1, 2, 3 …
        k.
        For n=1 to NOC
              $Sum_n = \sum^{k}_{m=1} (W_m)$ where ($o_m$==n)
        Endfor
        Label_x = x, if $Sum_x$ is highest, for each x = 1 to NOC
        Label_y = y, if $Sum_y$ is second highest,
        For each y = 1 to NOC
.       Difference = $Sum_x$ - $Sum_y$
        If 'Difference' is less than or equal to threshold
        Then
            Let $f_1$, f2, f3,…..$f_k$ be the 'FLAG' field values of the label nodes in the branches $b_1$, $b_2$, $b_3$, ….. $b_k$ respectively.
            Let $C_1$, C2, C3 …Ck be the class of the patterns corresponding to the branches $b_1$, $b_2$, $b_3$ …bk respectively.
            Compute the rareness factor $R_z$ = number of training patterns for class $C_z$ / $f_z$ ,
             For each z = 1, 2, 3 … k.
             Rarest = $o_d$, If $R_d$ is highest,
             For each d = 1 to k.
               If (Rarest == Label_y)
               Then
                   $Label_i$ = Label_y
                Else
                   $Label_i$ = Label_x
                Endif
        Else
        $Label_i$ = Label_x
        Endif
    Endfor
Endfor
```

5. Experiments

PCC [3] strikes a balance between the design time and the classification time unlike other classifiers like neural networks [1] and genetic algorithms [2]. PCC also has advantages over conventional classifiers (like nearest neighbor, k-NNC) with respect to storage space, processing time (PT) and classification accuracy (CA) [3] (processing time is the sum of design time and classification time). So, we chose to conduct experiments by comparing PRC with PCC for two different sets of handwritten digit data. Each set of data consists of a test set and training set of digits '0' to '9' written by different people. Value of the constant *threshold* is maintained equal to 1 for all experiments with PRC.

Table 5 provides the results of the experiments on first data set [3]. Test data consists of 3333 patterns. The dimension of test and training patterns is 16x12.

Table 5. Results with first data set

Classifier	Total processing time (in Sec)	Size of PR-tree (in Bytes)	CA (%)	No. of training patterns
PRC, k=16	62.36	29,94,784	94.449	6670
PCC, k=15	64.82	44,08,064	93.669	6670
PRC, k=16	38.30	18,45,504	92.619	4000
PCC, k=15	39.91	27,17,712	91.95	4000
PRC, k=16	19.54	9,57,216	90.57	2000
PCC, k=15	20.71	14,06,512	89.55	2000

Table 6 provides the results of the experiments on third data set (taken from MNIST handwritten digit database). Test data consists of 10,000 patterns. Test and training patterns are of the dimension 28X28.

Table 6 . Results with third data set

Classifier	Total processing time (in Sec)	Size of PR-tree (in Bytes)	CA (%)	No. of training patterns
PRC,k=11	4073.90	4,40,58,240	96.90	60,000
PCC,k=11	5034.90	18,98,02,800	96.55	60,000

6. Conclusion

In this paper, a compact data structure, PR-tree and an efficient classifier based on it, PRC are proposed. It is observed that PRC has advantages over other classifiers with respect to processing time, memory requirements and classification accuracy. The future direction of our research includes finding out the best criteria for determining the value of *threshold.*

7. References

[1] M. Prakash and M. Narsimha Murty, "Growing subspace pattern recognition methods and their neural network models". IEEE Transactions on Neural Networks 8 (1) (1997) 161-168.

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

[3] V.S. Ananth Narayana, M. Narsimha Murty and D.K. Subramaniam, "An Incremental Data Mining Algorithm for Compact realization of Prototypes", Pattern Recognition, 34:2249-2501,2001.

[4] E. Fix and J.L. Hodges, Jr. "Discriminatory analysis: Non-parametric Discrimination: Consistency Properties", Report Number 4, USAF School of Aviation Medicine, Randolph Field, Texas, 1952.