Overlap pattern synthesis with an efficient nearest neighbor classifier

P. Viswanath, M. Narasimha Murty, Shalabh Bhatnagar*

Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India

Abstract

Nearest neighbor (NN) classifier is the most popular non-parametric classifier. It is a simple classifier with no design phase and shows good performance. Important factors affecting the efficiency and performance of NN classifier are (i) memory required to store the training set, (ii) classification time required to search the nearest neighbor of a given test pattern, and (iii) due to the curse of dimensionality the number of training patterns needed by it to achieve a given classification accuracy becomes prohibitively large when the dimensionality of the data is high. In this paper, we propose novel techniques to improve the performance of NN classifier and at the same time to reduce its computational burden. These techniques are broadly based on: (i) overlap based pattern synthesis which can generate a larger number of artificial patterns than the number of input patterns and thus can reduce the curse of dimensionality effect, (ii) a compact representation of the given set of training patterns called overlap pattern graph (OLP-graph) which can be incrementally built by scanning the training set only once and (iii) an efficient NN classifier called OLP-NNC which directly works with OLP-graph and does implicit overlap based pattern synthesis. A comparison based on experimental results is given between some of the relevant classifiers. The proposed schemes are suitable for applications dealing with large and high dimensional datasets like those in data mining.

Keywords: Nearest neighbor classifier; Pattern synthesis; Compact representation; Data mining

1. Introduction

Nearest neighbor (NN) classifier is a very popular non-parametric classifier [1–3]. It is widely used because of its simplicity and good performance. It has no design phase and simply stores the training set. A test pattern is classified to the class of its nearest neighbor in the training set. So the classification time required for the NN classifier is largely for reading the entire training set to find the NN.[1] Thus the two major shortcomings of the classifier are that the entire training set needs to be stored and searched. To add to this list, its performance (classification accuracy) depends on the training set size.

Cover and Hart [4] show that the error for NN classifier is bounded by twice the Bayes error when the available sample size is infinity. However, in practice, one can never have an infinite number of training samples. With a fixed number of training samples, the classification error for 1-NN classifier tends to increase as the dimensionality of the data gets large. This is called the peaking phenomenon [5,6]. Jain and Chandrasekharan [7] point out that the number of training samples per class should be at least 5–10 times the dimensionality of the data. The peaking phenomenon with NN classifier is known to be more severe than other parametric classifiers such as Fisher's linear and quadratic

* Corresponding author. Fax: +91 80 23602911.

E-mail addresses: shalabh@csa.iisc.ernet.in (S. Bhatnagar), viswanath@csa.iisc.ernet.in (P. Viswanath), mnm@csa.iisc.ernet.in (N.M. Murty).

[1] We assume that the training set is not preprocessed (like indexed, etc.) to reduce the time needed to find the neighbor.

classifiers [8,9]. Duda et al. [3] mention that for non-parametric techniques, the demand for a large number of samples grows exponentially with the dimensionality of the feature space. This limitation is called the *curse of dimensionality*. Thus, it is widely believed that the size of training sample set needed to achieve a given classification accuracy would be prohibitively large when the dimensionality of data is high.

Increasing the training set size has two problems, viz., (i) space and time requirements get increased and (ii) it may be expensive to get training patterns from the real world. The former problem can be solved to some extent by using a compact representation of the training set like PC-tree [10], FP-tree [11], CF-tree [12], etc., while the latter by re-sampling techniques, like bootstrapping [13] which has been widely studied [14–18]. These two remedies are however orthogonal, i.e., they have to be followed one step after the other (cannot be combined into a single step).

An elegant solution to the above problems would be to find a *compact and generalized abstraction for the training set*. The abstraction being compact solves the space and time requirements problem. The generalization implies that not only the given patterns but also other new patterns are possible to be generated from it.

In this paper we propose (i) a novel pattern synthesis technique called *overlap-based pattern synthesis* (OLP-*synthesis*), (ii) a corresponding compact representation of the training set called OLP-*graph* and (iii) an efficient NN classifier called OLP-NNC.

The effectiveness of OLP-synthesis is established both informally and formally. The number of synthetic patterns generated by OLP-synthesis can be exponential in the number of original patterns.[2] As a result, the synthetic patterns cannot be explicitly stored.

To overcome the above problem, OLP-NNC directly works with the OLP-graph and avoids explicit synthetic pattern generation. That is, OLP-NNC implicitly does OLP-synthesis and the nearest neighbor for a given test pattern is found from the entire synthetic training set. So OLP-graph acts as a compact and generalized abstraction for the training set. Further, it can be incrementally constructed by scanning the training set only once, whereas the compact structures like FP-tree [11] require two database scans and cannot be constructed incrementally. Addition of new patterns and deletion of existing patterns from the OLP-graph can be done easily. Unlike compact representations like CF-tree [12], this is independent of the order in which the original patterns are considered. Hence OLP-graph is a suitable representation for large training sets which can vary with respect to time like in data mining applications.

Empirical studies show that (i) the space requirement for OLP-graph is smaller than that for the original training set and the rate at which its size increases with respect to the original set becomes smaller and smaller as the original set grows, (ii) the classification time needed by OLP-NNC using the synthetic dataset is of the same order of magnitude as that of conventional NN classifier using the input data set and (iii) the performance of OLP-NNC is better than the conventional NN classifier. We also compare our results with those of other classifiers like the Naive Bayes classifier and NN classifier based on the bootstrap method given by Hamamoto et al. [18]. Naive Bayes classifier can be seen as carrying on a pattern synthesis implicitly by assuming statistical independence between features.

The rest of the paper is organized as follows: Section 2 describes pattern synthesis. Section 3 describes OLP-graph with its properties. OLP-NNC is explained in Section 4. Experimental results are described in Section 5 and conclusions in Section 6.

2. Pattern synthesis

Pattern synthesis can be seen as a method of generating artificial new patterns from the given training patterns. Broadly this can be done in two ways viz., *model-based pattern synthesis* and *instance-based pattern synthesis*.

Model-based pattern synthesis first derives a model based on the training set and uses this to generate patterns. The model derived can be a probability distribution or an explicit mathematical model like a Hidden Markov model. This method can be used to generate as many patterns as needed. There are two drawbacks of this method. First, any model depends on the underlying assumptions and hence the synthetic patterns generated can be erroneous. Second, it might be computationally expensive to derive the model. Another argument against this method is that if pattern classification is the purpose, then the model itself can be used without generating any patterns at all.

Instance-based pattern synthesis, on the other hand, uses the given training patterns and some of the properties about the data. It can generate only a finite number of new patterns. Computationally this can be less expensive than deriving a model. This is especially useful for non-parametric classifiers like NNC which directly use the training instances. Further, this can also result in reduction of the computational requirements of NNC.

We present in this paper an instance-based pattern synthesis technique called *overlap based pattern synthesis*, first providing key motivation. We also present an approximate version of the method.

2.1. *Overlap based pattern synthesis—main ideas*

Let F be the set of features. There may exist a three-block partition of F, say, $\{A, B, C\}$ with the following properties. For a given class, there is a dependency (probabilistic) among features in $A \cup B$. Similarly, features in $B \cup C$ have a dependency. However, features in A (or C) can affect those

[2] By *original patterns*, we mean the given training patterns (to contrast with the synthetic patterns).

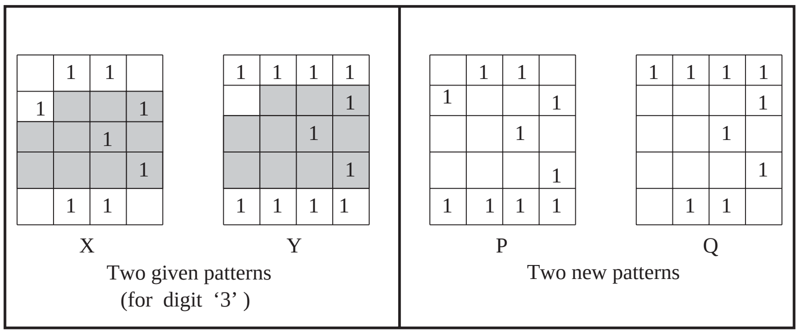

Fig. 1. Illustration of synthetic patterns. Note: The empty cells are assumed to have 0's. Shaded portion is the overlap.

in *C* (or *A*) only through features in *B*. Suppose now that we are given two patterns $X = (a_1, b, c_1)$ and $Y = (a_2, b, c_2)$ such that a_1 is a feature-vector that can be assigned to the features in *A*, *b* to the features in *B* and c_1 to the features in *C*, respectively. Similarly, a_2, b and c_2 are feature-vectors that can be assigned to features in *A*, *B*, and *C*, respectively. Then, our argument is that the two patterns (a_1, b, c_2), (a_2, b, c_1) are also valid patterns in the same class. If these two new patterns are not already in the class of patterns then it is only because of the finite nature of the set. We call this type of generation of additional patterns as *overlap-based pattern synthesis*, because this kind of synthesis is possible only if the two given patterns have the same feature-values for features in *B*. In the given example, feature-vector *b* is common between *X* and *Y* and, therefore, is called the *overlap*.

We present one simple example with hand-written digits which has geometric appeal also. Fig. 1 illustrates two given patterns (*X* and *Y*) for OCR digit '3', and two new patterns (*P* and *Q*) generated from the given patterns. In this example, the digits are drawn on a two-dimensional rectangular grid of size 5 × 4. A cell with '1' indicates presence of ink. Empty cells are assumed to have '0'. Patterns are represented row-wise. So the pattern $X = (0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0)$ and $Y = (1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1)$. Let the feature set be $F=\{f_1, \ldots, f_{20}\}$, and the three-block partition of *F* be $\{A, B, C\}$ which satisfies the earlier-mentioned properties with $A = \{f_1, \ldots, f_5\}$, $B = \{f_6, \ldots, f_{16}\}$ and $C = \{f_{17}, \ldots, f_{20}\}$, respectively. Then *X* and *Y* have an overlap and we can generate two new patterns, *P* and *Q*, as shown in Fig. 1.

2.2. *Overlap based pattern synthesis—formal procedure*

To describe the overlap-based pattern synthesis formally, we use the following notation and definitions. A pattern (data instance) is described as an assignment of values $X = (x_1, \ldots, x_d)$ to a set of features $F = (f_1, \ldots, f_d)$. As usual, we assume that all the data instances (including those in the training set) are drawn from some probability distribution over the space of feature vectors. Formally, for each assignment of values *X* to *F*, we have a probability $Pr(F = X)$. That is, a set of features is also seen as a random vector. Further, $X[f_i]$ is the value of feature f_i in instance *X*. That is, if $X = (x_1, \ldots, x_d)$, then $X[f_i] = x_i$. Let *A* be some subset of *F*. Given a feature vector *X*, we use X_A to denote the projection of *X* onto the features in *A*.

Let $\pi = \{A, B, C\}$ be a three-block partition of *F*. Let *X*, *Y* and *Z* be three given feature vectors. Then $W = (X_A, Y_B, Z_C)$ is a feature vector such that,

$$\begin{aligned} W[f_i] &= X[f_i], \quad \text{if } f_i \in A \\ &= Y[f_i], \quad \text{if } f_i \in B \\ &= Z[f_i], \quad \text{if } f_i \in C. \end{aligned}$$

Let $\mathbf{X} = \{X^1, X^2, \ldots, X^n\}$ be a set of patterns belonging to class ω_l. The synthetic set generated from $\mathbf{X}$ with respect to a given three-block partition $\pi = \{A, B, C\}$ of *F* is denoted as $\mathrm{SS}_\pi(\mathbf{X})$ and is obtained as follows:

(1) Initially $SS_\pi(\mathbf{X}) = \mathbf{X}$.
(2) For each pair of patterns (X^i, X^j) for $1 \leqslant i < j \leqslant n$, if $X^i_B = X^j_B$, add two patterns (X^i_A, X^i_B, X^j_C), (X^j_A, X^j_B, X^i_C) to $SS_\pi(\mathbf{X})$ if they are not already in it.

It is easy to see that $SS_\pi(\mathbf{X})$ and $\mathbf{X}$ are obtained from the same distribution provided for any assignment of values *a*, *b*, and *c* to the random vectors *A*, *B*, and *C*, respectively, $Pr(A = a \mid B = b, C = c, Class = \omega_l) = Pr(A = a \mid B = b, Class = \omega_l)$. That is, *A* is conditionally independent of *C*, given *B* and class is ω_l. Since we restrict our attention to one class at a time, we can equivalently state the rule as $Pr(A=a \mid B=b, C=c) = Pr(A=a \mid B=b)$. We will use the notation $I(A, C \mid B)$ to denote the conditional independence of two random variables *A* and *C* given random variable *B*.

If there is more than one such three-block partition, then we apply the synthesis technique sequentially (in some order). Let $\{\pi_1, \ldots, \pi_m\}$ be the *m* possible three-block partitions satisfying the conditional independence requirement, then a synthetic set that can be obtained is $SS_{\pi_m}(SS_{\pi_{m-1}}(\cdots(SS_{\pi_1}(\mathbf{X}))))$. For this kind of synthesis, three-block partitions can be seen as the building blocks and when applied in a sequence gives raise to the general form of the synthesis. That is, from given two original patterns, one can derive many synthetic patterns by applying the technique with respect to the given three-block partitions in a sequence.

2.3. *Overlap-based pattern synthesis—an approximate method*

If partitions, $\pi_i = \{A_i, B_i, C_i\}$ such that $I(A_i, C_i \mid B_i)$ is true for $1 \leqslant i \leqslant m$ are given, then it can be used for pattern synthesis. Unfortunately, there may not exist any such partition fully satisfying the conditional independence requirement. But there may exist partitions which approximately satisfy the requirement. Furthermore, finding either a true or an approximate partition might be very hard. In this section, we present one simple algorithm which provides a heuristic approach to dealing with this problem. The method is based on pairwise correlations between features. The partitions obtained by this method may not strictly satisfy the conditional independence requirement. Nevertheless, based on empirical results, the patterns generated using these partitions are shown to improve the classification accuracy of NNC.

Our intuition for constructing a candidate partition is based on the following: assume that a partition, $\pi = \{A, B, C\}$ does exist such that $I(A, C \mid B)$ is true. We can think that the features in A directly influence features in B and that in turn directly influence those in C. However, there is an indirect influence of features in A on features in C, via features in B. Therefore, features in A will tend to be quite strongly correlated with features in B, similarly features in B will be strongly correlated with features in C. But correlation between features in A and C will be weak. There is a well-known "folk-theorem" that probabilistic influence tends to attenuate over distance; that is, direct influence is stronger than indirect influence. (This has been shown both formally and empirically in certain special cases in [19,20].) Therefore, we use these heuristics to choose appropriate partitions. Further, we would like to find as many partitions as possible. This is done as explained below.

We obtain an ordering of features such that features which are close by (according to this ordering) are highly correlated than for distant ones. If $(f_1, f_2, \ldots, f_d)$ is an ordered set of features such that f_i is the ith feature then we define a criterion or cost $J = \sum_{\forall i,j} |cor(f_i, f_j)| \times |i - j|$ where $cor(f_i, f_j)$ is the correlation factor for the pair of features f_i and f_j. We find an ordering of features for which the cost J is minimum. We give two methods for finding such an ordering of features.

If the number of features is small such that performing an exhaustive search over all possible orderings is feasible, then this is done to find the best ordering of features having minimum J value. Otherwise, we (i) select a random ordering of features, (ii) randomly choose two distinct features f_i, f_j and swap their positions (to get a new ordering of features) if it decreases the cost. Step (ii) is repeated for a pre-specified number of times and the resulting ordering of features is taken.

Let $(f'_1, f'_2, \ldots, f'_d)$ be the final ordering of features (such that f'_i is the ith feature), derived from the above-mentioned process. For a given threshold correlation factor t $(0 \leqslant t \leqslant 1)$, we get d partitions $\pi_i = \{A_i, B_i, C_i\}$ for $(1 \leqslant i \leqslant d)$ as follows where d is the number of features. $A_i = (f'_1, f'_2, \ldots, f'_{i-1})$, $B_i = (f'_i, f'_{i+1}, \ldots, f'_{j-1})$ and $C_i = (f'_j, f'_{j+1}, \ldots, f'_d)$ such that $|cor(f'_i, f'_k)| < t$ for all $f'_k \in C_i$. It is possible for A_i or C_i to be empty. It is worth noting that always A_1 and C_d are empty. A concise representation of all partitions i.e., $(\pi_1, \pi_2, \ldots, \pi_d)$ is the list of integers $L = (|B_1|, |B_2|, \ldots, |B_d|)$ such that $|B_i|$ corresponds to the partition π_i with $A_i = (f'_1, f'_2, \ldots, f'_{i-1})$, $B_i = (f'_i, f'_{i+1}, \ldots, f'_{|B_i|+i-1})$ and $C_i = (f'_{|B_i|+i}, f'_{|B_i|+i+1}, \ldots, f'_d)$. L is called *the overlap lengths list*.

2.4. *An example*

Let the ordered set of features be $F = (f'_1, f'_2, \ldots, f'_6)$, $L = (2, 2, 3, 2, 2, 1)$ be the representation of partitions for a class and the given original patterns for the class be $\mathbf{X} = \{(a, b, c, d, e, f), (p, b, c, q, e, r), (m, n, c, q, e, o)\}$. Then $SS_{\pi_1}(\mathbf{X}) = \mathbf{X}$, since no two patterns in $\mathbf{X}$ have common values for first and second features simultaneously. $(SS_{\pi_2}(SS_{\pi_1}(\mathbf{X})) = \{(a, b, c, d, e, f), (a, b, c, q, e, r), (p, b, c, d, e, f), (p, b, c, q, e, r), (m, n, c, q, e, o)\}$, because for the two patterns in $\mathbf{X}$ viz., (a, b, c, d, e, f) and (p, b, c, q, e, r) the features in $B_2 = (f'_2, f'_3)$ have common values and so we get two additional synthetic patterns, viz., (a, b, c, q, e, r) and (p, b, c, d, e, f), respectively. Finally, the entire synthetic set, $SS_{\pi_6}(\ldots(SS_{\pi_2}(SS_{\pi_1}(\mathbf{X})))) = \{(a, b, c, d, e, f), (a, b, c, q, e, r), (a, b, c, q, e, o), (p, b, c, d, e, f), (p, b, c, q, e, r), (p, b, c, q, e, o), (m, n, c, q, e, r), (m, n, c, q, e, o)\}$.

3. A compact representation for synthetic patterns

For carrying out partition-based pattern synthesis, even though all possible partitions of the set of features are found, it would be a computationally hard job to generate all possible synthetic patterns. Further, the number of synthetic patterns that can be generated can be very large when compared with the given original set size. This results in increased space requirement for storing the synthetic set. In this section we present a data structure called *overlap pattern graph* (OLP-*graph*) which is a compact representation for storing synthetic patterns. OLP-graph, for a given collection of partitions of the set of features, can be constructed by reading the given original training patterns only once and is also suitable for searching the NN in the synthetic set.

For a given class of original patterns ($\mathbf{X}$), for a given collection of partitions of the set of features ($\{\pi_i \mid 1 \leqslant i \leqslant d\}$), overlap pattern graph (OLP-graph) is a compact data structure built by inserting each original pattern into it. But the patterns that can be extracted out of the OLP-graph form the synthetic set $SS_{\pi_d}(\ldots(SS_{\pi_1}(\mathbf{X})))$.

OLP-graph has two major parts, viz., (i) directed graph and (ii) header table. Every node in the graph has (i) a feature value, (ii) an adjacency list of arcs and (iii) a node-link. A path (consisting of nodes and arcs) of OLP-graph from one end to the other represents a pattern. If two patterns have an overlap with respect to a partition of the set of features, then these two patterns share a common sub-path. A node of the graph represents a feature for a pattern.

Header table and node links facilitate in finding the overlap. Header table consists of an entry for every feature. An entry in it points to a node in the graph which represents that feature for a pattern. The node link of a node points to another node which represents the same feature but for some other pattern. That is, a header table entry for feature f_i is a pointer to the head of a linked list of nodes. Each node in this linked list represents feature f_i for some pattern. This linked list is called *feature-linked-list of* f_i.

A pattern is progressively inserted feature by feature. For this purpose, for every feature f_i $(1 \leqslant i \leqslant d)$ a possible overlap with the existing patterns in the OLP-graph with respect to the partition π_i is looked for. If no overlap is possible then a new node is created for the feature.

3.1. An example

For the example presented in Section 2.4 the corresponding OLP-graph is shown in Fig. 2.

Node links which form *feature-linked-lists* are shown in dotted lines. A path consisting of nodes and arcs from the left end to the right end represents a pattern that can be extracted from the OLP-graph. Thus, all patterns that can be extracted out form the synthetic set $SS_{\pi_6}(\ldots(SS_{\pi_2}(SS_{\pi_1}(\mathbf{X}))))$.

An important point to observe is that first and second patterns in the original set have an overlap according to the partition π_2 i.e., they have an overlap (or, same feature-values) for features in $B_2 = (f'_2, f'_3)$. In the OLP-graph, the node corresponding to *b* is shared but that corresponding to *c* is not. The reason is that if the node corresponding to *c* is also shared then $m \to n \to c \to d \to e \to f$ becomes a valid path, i.e., (m, n, c, d, e, f) becomes a synthetic pattern which can be extracted out of the graph. However, this is not a valid synthetic pattern according to overlap-based pattern synthesis (Section 2.2) based on the partitions given. To make this point clearer, a definition of the property *node-sharability* along with *sharable-node* is given below.

Node-sharability and sharable-node: For a given OLP-graph *G*, pattern *X*, feature f'_i and partition $\pi_i = \{A_i, B_i, C_i\}$, we say *node-sharability* $(G, X, f'_i, |B_i|)$ is true if there is a sub-path of nodes in *G*, $v_i \to v_{i+1} \to \cdots \to v_{i+|B_i|-1}$ such that feature value in v_j is equal to $X[f'_j]$ for $(i \leqslant j \leqslant i + |B_i| - 1)$, and v_i is a node in the *feature-linked-list* of f'_i. In this context, *sharable-node* $(G, X, f'_i, |B_i|)$ is v_i.

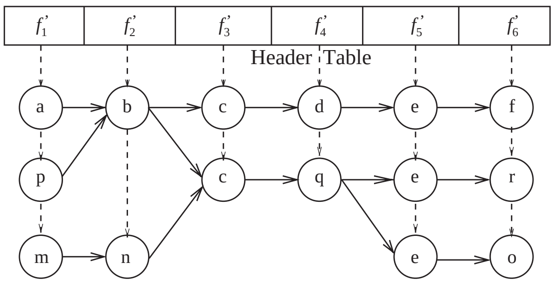

Fig. 2. Illustration of OLP-graph.

Let G_0 be the initial empty OLP-graph and G_i be the OLP-graph after inserting the *i*th pattern from the given original set. For the example $|B_2| = 2$ and $|B_3| = 3$. Then, *node-sharability* $(G_1, (p, b, c, d, e, f), f'_2, 2)$ is true and hence the node corresponding to *b* is shared. But *node-sharability* $(G_1, (p, b, c, d, e, f), f'_3, 3)$ is false, so a new node for *c* is created.

3.2. OLP-graph construction

An OLP-graph can be constructed for each class of patterns and for a given overlap lengths list $L = (|B_1|, |B_2|, \ldots, |B_d|)$ as described in the following method:

```
Build-OLP-graph($\mathbf{X}$, $L$)
{
//Let $G$ be the empty OLP-graph having header table
with all entries being empty.

  For each pattern $X$ in $\mathbf{X}$
  {
    Parent = NULL;
    For $i = 1$ to $d$
    {
      If (node-sharability($G, X, f_i, |B_i|$) is true)
      {
        $v$ = sharable-node($G, X, f_i, |B_i|$);
        Add-arc(Parent, $v$);
      }
      Else
      {
        $v$ = Create a new node;
        $v$.feature-value = $X[f_i]$;
        Add-to-feature-linked-list($v$, $f_i$);
        Add-arc(Parent, $v$);
      }
      Parent = $v$;
    }
  }
  Output($G$);
  }
```

The above method iteratively adds patterns from the original set to the already built OLP-graph.

The functions *Add-to-feature-linked-list* (v, f_i) appends node v to the feature-linked-list of f_i, and *Add-arc*(u, v) will add an arc to the adjacency list of arcs of node u pointing to node v provided such an arc already does not exist in the adjacency list.

3.3. Properties of OLP-graph

(1) The synthetic set $SS_{\pi_d}(SS_{\pi_{d-1}}(\cdots(SS_{\pi_1}(\mathbf{X}))))$ generated by doing overlap pattern synthesis and the one extracted from the corresponding OLP-graph are the same.
(2) For a given original set the OLP-graph is independent of the order in which the original patterns are considered. That is, the synthetic set generated from the OLP-graph is independent of the order in which the original patterns are arranged to build the OLP-graph.
(3) OLP-graph can be incrementally built.
(4) Addition or deletion of a pattern from the OLP-graph can be made in $O(n)$ time where n is the number of original patterns used to build the OLP-graph.

It is easy to see that the above properties are true.

3.4. Time complexity of build-OLP-graph

Let n be the number of patterns in the class for which the method is called. Since each pattern is considered only once, the method does a single scan over the training set. Because the patterns are considered to be stored in a disk (secondary storage medium), the number of times these are accessed from the disk is an important measure in applications (like data mining) where the total training set cannot be accommodated in the main memory.

The time complexity of the method is $O(n^2dl)$, where d is the dimensionality of patterns and l is the maximum element in L. The reason is that all patterns are considered once (total of n patterns), for each pattern every feature is considered once (total of d features), and for each feature its *feature-linked-list* is searched to check whether the *node-sharability* property holds or not. Each search for *node-sharability* takes at most $O(l)$ time. Feature-linked list for any feature can be at most of size n.

Since d is constant and $(l \leqslant d)$, the effective time complexity of the method is $O(n^2)$.

3.5. Space complexity of build-OLP-graph

The space required by the method is largely due to the space occupied by the OLP-graph G. G consists of a header table and a graph. The space required for the header table is $O(d)$, and that for the graph is $O(nd)$. Since each original pattern (total of n patterns) occupies a path in G, a path is of size d. So the effective upper bound on space complexity is $O(n)$. But empirical studies (Section 5) show that the actual space consumed by an OLP-graph is much smaller than that of the original patterns. The reason is that many patterns in a class can have an overlap and thus can share a common sub-path. Another important point to note from the empirical studies is that the rate of increase in the size of G decreases as n increases and for some $n_1 > n$, it can be assumed to become zero. That is, the size of G does not increase after adding a sufficient number of patterns to it. This is a useful property in cases where the data sets are large and grow with time, as in applications like data mining.

4. An efficient NN classifier using the synthetic patterns

For each class, even though OLP-graph is a compact representation for the synthetic set that can be generated by overlap pattern synthesis, the conventional NN classifier with the entire synthetic set takes a large amount of time. The reason is that the synthetic set can be exponentially larger (in size) as compared to the original set which depends on the overlap lengths list considered.

We propose a NN classifier called OLP-NNC which has classification time upper bound equal to $O(n)$ where n is the original training set size. OLP-NNC stores the partial distance computations in the nodes of the OLP-graph and avoids recomputing the same. This method is suitable for distance measures like Hamming distance, Squared Euclidean distance, etc., where the distance between two patterns can be found over its parts (called partial distance) and added up later to get the actual distance.

OLP-NNC first finds the distance between the test pattern and its NN within a synthetic set of a given class (represented by an OLP-graph). The class label assigned to the test pattern then is that for the NN or the pattern with the least distance. Algorithms 1 and 2 describe finding distance of NN within a class. Distance measure used is squared Euclidean distance. Note that NN found using either Euclidean or squared Euclidean distance measures are same.

Algorithm 1 Find-Min-Dist(Graph *G*, Test Pattern *T*)

```
{Let min-distance be an integer initialized to maximum possible value.}
for (each node $v$ in the feature-linked-list of $f_1$ in $G$) do
  $d$=Find-Dist($v, T, 1$);
  if ($d <$ min-distance) then
    min-distance $= d$;
  end if
end for
return(min-distance);
```

Algorithm 2 Find-Dist(Node v, Test Pattern T, Integer i)

```
if ($v$ is marked as visited) then return
   ($v \cdot partial\text{-}distance$);
else
   $d = (T[f_i] - v.feature\text{-}value)^2$;
   Find $L$= List of descendant nodes of $v$;
   if ($L$ is not empty) then
      for (each node $w$ in $L$) do
         $d = d + \text{Find-Dist}(w, T, i+1)$;
         if ($d < min\text{-}distance$) then
            $min\text{-}distance = d$;
         end if
      end for
      $v \cdot partial\text{-}distance = min\text{-}distance$;
   else
      $v \cdot partial\text{-}distance = d$;
   end if
   Mark $v$ as visited;
   return($v \cdot partial\text{-}distance$);
end if
```

5. Experiments

5.1. Datasets

We performed experiments with five different datasets, viz., OCR, WINE, THYROID, GLASS and PENDIGITS, respectively. Except the OCR dataset, all others are from the UCI Repository [21]. OCR dataset is also used in [22,10]. The properties of the datasets are given in Table 1. For OCR, THYROID and PENDIGITS datasets, the training and test sets are separately available. For the remaining datasets 100 patterns are chosen randomly as the training patterns and the remaining as the test patterns.

All the datasets have only numeric valued features. The OCR dataset has binary discrete features, while the others have continuous valued features. Except OCR dataset, all other datasets are normalized to have zero mean and unit variance for each feature and subsequently discretized as follows. Let a be a feature value after normalization, and a' be its discrete value. We used the following discretization procedure.

If $(a < -0.75)$ *then* $a' = -1$;
Else-If $(a < -0.25)$ *then* $a' = -0.5$;
Else-If $(a < 0.25)$ *then* $a' = 0$;
Else-If $(a < 0.75)$ *then* $a' = 0.5$;
Else $a' = 1$.

Table 1
Properties of the datasets used

Dataset	Number of features	Number of classes	Number of training examples	Number of test examples
OCR	192	10	6670	3333
WINE	13	3	100	78
THYROID	21	3	3772	3428
GLASS	9	7	100	114
PENDIGITS	16	10	7494	3498

5.2. Classifiers for comparison

The classifiers chosen for comparison purposes are as follows:
NNC: The test pattern is assigned to the class of its NN in the training set. The distance measure used is Euclidean distance. It has both space and classification time requirements equal to $O(n)$ where n is the number of original training patterns.
k-NNC: A simple extension of NNC, where the most common class in the k NN ($k \geqslant 1$) is chosen. The distance measure is Euclidean distance. Three-fold cross validation is done to choose the k value. Space and classification time requirements of the method are both $O(n)$ when k is assumed to be a small constant when compared with n.
Naive–Bayes classifier (*NBC*): This is a specialization of the Bayes classifier where the features are assumed to be statistically independent. Further, the features are assumed to be discrete valued. Let $X = (x_1, \ldots, x_d)$ be a pattern and l be a class label. Then the class conditional probability, $P(X \mid l) = P(x_1 \mid l) \times \cdots \times P(x_d \mid l)$. Here $P(x_i \mid l)$ is taken as the frequency ratio of number of patterns in class with label l and with feature f_i value equal to x_i to that of total number of patterns in that class. A priori probability for each class is taken as the frequency ratio of number of patterns in that class to the total training set size. The given test pattern is classified to the class for which the posteriori probability is maximum. OCR dataset is used as it is, whereas the other datasets are normalized (to have zero mean and unit variance for each feature) and discretized as done for the other classifiers. Design time for the method is $O(n)$, but the effective space and classification time requirements are $O(1)$ only.
NNC with bootstrapped training set (*NNC*(*BS*)): We used the bootstrap method given by Hamamoto et al. [18] to generate an artificial training set. The bootstrapping method is as follows. Let X be a training pattern and let $X_1, \ldots, X_r$ be its r NN in its class. Then $X' = (\sum_{i=1}^{r} X_i)/r$ is the artificial pattern generated for X. In this manner, for each training pattern an artificial pattern is generated. NNC is done with this new bootstrapped training set. The value of r is chosen according to a three-fold cross validation. Bootstrapping step requires $O(n^2)$ time, whereas space and classification time requirements are both $O(n)$.
OLP-NNC: This method is given in Section 4. The threshold correlation factor used in the overlap-based pattern synthesis is chosen based on a three-fold cross validation from

Table 2
A comparison between the classifiers (showing classification accuracies (%))

Dataset	NNC	k-NNC	NBC	NNC(BS)	OLP-NNC
OCR	91.12	92.68	81.01	92.88	93.85
WINE	91.03	92.31	91.03	93.29	93.60
THYROID	92.44	93.70	83.96	93.35	93.23
GLASS	68.42	68.42	60.53	68.42	70.18
PENDIGITS	96.08	96.34	83.08	96.36	96.08

Table 3
A comparison of percentage accuracies and computational requirements for THYROID dataset for various threshold correlation factors

Classifier : OLP-NNC				
Threshold	CA (%)	Space (KB)	Design time (s)	Classification time (s)
0.0	92.44	173.95	20.24	16.17
0.1	93.23	144.70	9.68	10.23
0.2	92.79	86.11	7.22	8.44
0.3	92.65	5.23	2.96	1.85
0.4	92.65	4.94	1.30	0.69
0.5	92.65	3.46	1.23	0.55
0.6	92.65	3.04	1.07	0.47
0.7	92.65	2.36	0.92	0.38
0.8	92.65	1.45	0.78	0.27
0.9	92.65	1.36	0.55	0.20
1.0	92.65	1.35	0.38	0.15
Classifier : NNC				
—	92.44	158.42	0	14.48
Classifier : k-NNC				
—	93.70	158.42	0	16.52
Classifier : NBC				
—	83.96	1.26	0.28	0.12
Classifier : NNC(BS)				
—	93.35	158.42	23.26	15.06

$\{0.0, 0.1, \ldots, 1.0\}$. It has design time requirement equal to $O(n^2)$. Space and classification time requirements are both $O(n)$.

5.3. *Experimental results*

Table 2 gives a comparison between the classifiers. It shows the classification accuracy (CA) for each of the classifiers as a percentage over respective test sets. The parameter values (like k in k-NNC, r in NNC(BS) and threshold correlation factor in OLP-NNC) are chosen based on three-fold cross validation. Some of the observations are: (i) for OCR, WINE and GLASS datasets OLP-NNC outperforms the rest of the classifiers, (ii) OLP-NNC is better than to NNC for all datasets except PENDIGITS for which the CA for both OLP-NNC and NNC is the same, and (iii) OLP-NNC significantly outperforms NBC over all datasets. In fact, except for the WINE dataset, the difference in CA between OLP-NNC and NBC over all datasets is almost 10% or higher.

For OLP-NNC, there seem two ways to further reduce the computational requirements without degrading the CA much. First, one could increase the threshold correlation factor in doing the pattern synthesis. This would result in a compact OLP-graph structure that in turn would result in reduced space and classification time requirements. Table 3 demonstrates this for the THYROID dataset. The second way is to reduce the original training set size by taking only a random sample of it as the training set. Table 4 demonstrates this for OCR dataset. It can be observed that OLP-NNC even with (only) 2000 original training patterns outperforms NNC and is almost equal to k-NNC and NNC(BS) with 6670 training patterns with respect to CAs and clearly shows a significant reduction in both the space and classification time requirements. The difference is almost of an order of magnitude in favor of OLP-NNC, in terms of space requirements when compared with NNC, k-NNC and NNC(BS). Similar results are observed for the remaining datasets and hence are not reported.

Table 4
A comparison between the classifiers showing percentage accuracies and computational requirements for various training set sizes for OCR data set

Classifier	Number of training patterns	CA (%)	Space (KB)	Design time (s)	Classification time (s)
NNC	2000	87.67	772	0	98.01
	4000	90.16	1544	0	175.25
	6670	91.11	2575	0	306.53
k-NNC	2000	87.79	772	0	106.92
	4000	90.22	1544	0	200.21
	6670	92.68	2575	0	329.00
NBC	2000	80.71	15.36	4.02	0.46
	4000	81.03	15.36	5.28	0.51
	6670	81.01	15.36	7.26	0.49
NNC(BS)	2000	88.86	772	16.11	99.16
	4000	90.84	1544	34.23	172.74
	6670	92.88	2575	55.56	310.27
OLP-NNC	2000	92.44	311	6.03	91.74
	4000	92.89	473	10.26	145.51
	6670	93.85	629	22.31	205.05

6. Conclusion

Overlap-based pattern synthesis is an instance-based pattern synthesis technique which considers from the given training set, some of the properties about the data. This can result in reduction in both the curse of dimensionality effect and computational requirements for the NN classifier. Approximate pattern synthesis can be realized by considering pairwise correlation factor between the features and this is empirically shown to improve the classification accuracy in most cases. Synthetic patterns can be stored in a compact data structure called OLP-graph and can be used to find the NN of a given test pattern in $O(n)$ time, where n is the number of given original training patterns. The techniques described are suitable for large and high dimensional datasets.

References

[1] E. Fix, J. Hodges Jr., Discriminatory analysis: non-parametric discrimination: consistency properties, Report No. 4, USAF School of Aviation Medicine, Randolph Field, Texas, 1951.

[2] E. Fix, J. Hodges Jr., Discriminatory analysis: non-parametric discrimination: small sample performance, Report No. 11, USAF School of Aviation Medicine, Randolph Field, Texas, 1952.

[3] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, second ed., Wiley-interscience Publication, New York, 2000.

[4] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Trans. Inf. Theory 13 (1) (1967) 21–27.

[5] K. Fukunaga, D. Hummels, Bias of nearest neighbor error estimates, IEEE Trans. Pattern Anal. Mach. Intell. 9 (1987) 103–112.

[6] G. Hughes, On the mean accuracy of statistical pattern recognizers, IEEE Trans. Inf. Theory 14 (1) (1968) 55–63.

[7] A. Jain, B. Chandrasekharan, Dimensionality and sample size considerations in pattern recognition practice, in: P. Krishnaiah, L. Kanal (Eds.), Handbook of Statistics, vol. 2, North-Holland, 1982, pp. 835–855.

[8] K. Fukunaga, D. Hummels, Bayes error estimation using parzen and k-NN procedures, IEEE Trans. Pattern Anal. Mach. Intell. 9 (1987) 634–643.

[9] K. Fukunaga, Introduction to Statistical Pattern Recognition, second ed., Academic Press, New York, 1990.

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

[11] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proceedings of ACM SIGMOD International Conference of Management of Data, Dallas, Texas, USA, 2000.

[12] Z. Tian, R. Raghu, L. Micon, BIRCH: an efficient data clustering method for very large databases, in: Proceedings of ACM SIGMOD International Conference of Management of Data, 1996.

[13] B. Efron, Bootstrap methods: another look at the jackknife, Annu. Statist. 7 (1979) 1–26.

[14] A. Jain, R. Dubes, C. Chen, Bootstrap technique for error estimation, IEEE Trans. Pattern Anal. Mach. Intell. 9 (1987) 628–633.

[15] M. Chernick, V. Murthy, C. Nealy, Application of bootstrap and other resampling techniques: evaluation of classifier performance, Pattern Recognition Lett. 3 (1985) 167–178.

[16] S. Weiss, Small sample error rate estimation for k-NN classifiers, IEEE Trans. Pattern Anal. Mach. Intell. 13 (1991) 285–289.

[17] D. Hand, Recent advances in error rate estimation, Pattern Recognition Lett. 4 (1986) 335–346.

[18] Y. Hamamoto, S. Uchimura, S. Tomita, A bootstrap technique for nearest neighbor classifier design, IEEE Trans. Pattern Anal. Mach. Intell. 19 (1) (1997) 73–79.

[19] D. Draper, S. Hanks, Localized partial evaluation of belief networks, in: Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI'94), 1994, pp. 170–177.

[20] A.V. Kozlov, J.P. Singh, Sensitivities: an alternative to conditional probabilities for Bayesian belief networks, in: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI'95), 1995, pp. 376–385.

[21] P.M. Murphy, UCI repository of machine learning databases. Department of Information and Computer Science, University of California, Irvine, CA, 1994, http://www.ics.uci.edu/mlearn/MLRepository.html.

[22] T.R. Babu, M.N. Murty, Comparison of genetic algorithms based prototype selection schemes, Pattern Recognition 34 (2001) 523–525.

About the Author—P. VISWANATH received his M.Tech (Computer Science) from the Indian Institute of Technology, Madras, India in 1996. From 1996 to 2001, he worked as a faculty member at BITS-Pilani, India and Jawaharlal Nehru Technological University, Hyderabad, India. Since August 2001, he is working for his Ph.D. at the Indian Institute of Science, Bangalore, India. His areas of interest include Pattern Recognition, Data Mining and Algorithms.

About the Author—M. NARASIMHA MURTY received his Ph.D. from the Indian Institute of Science, Bangalore, India in 1982. He is a professor in the Department of Computer Science and Automation at the Indian Institute of Science, Bangalore. He has guided 14 Ph.D. students in the areas of Pattern Recognition and Data Mining. He has published around 80 papers in various journals and conference proceedings in these areas. He worked on Indo-US projects and visited Michigan State University, East Lansing, USA and University of Dauphine, Paris. He is currently interested in Pattern Recognition.

About the Author—SHALABH BHATNAGAR received his Ph.D. from the Indian Institute of Science, Bangalore in 1998. He has held visiting positions at the Institute for Systems Research, University of Maryland, College Park, USA; Free University, Amsterdam, Netherlands and the Indian Institute of Technology, Delhi. Since December 2001, he is working as an Assistant Professor at the Indian Institute of Science. His areas of interest include Performance Modelling and Analysis of Systems, Stochastic Control and Optimization with applications to Communication Networks and Semi-conductor Manufacturing. More recently he has also been interested in problems in Pattern Recognition and Evolutionary Algorithms.