On Using Prototype Reduction Schemes to Optimize *Locally* Linear Reconstruction Methods[*]

Sang-Woon Kim[†] and B. John Oommen[‡]

Abstract

This paper concerns the use of Prototype Reduction Schemes (PRS) to optimize the computations involved in typical k-Nearest Neighbor (k-NN) rules. These rules have been successfully used for decades in statistical Pattern Recognition (PR) applications, and are particularly effective for density estimation, classification, and regression because of the known error bounds that they possess. For a given data point of unknown identity, the k-NN possesses the phenomenon that it combines the information about the samples from *a priori* target classes (values) of selected neighbors to predict the target class of the tested sample, or to estimate the density function value of the given queried sample. Recently, an implementation of the k-NN, named as the Locally Linear Reconstruction (LLR) [2], has been proposed. The salient and brilliant feature of the latter is that by invoking a quadratic optimization process, it is capable of systematically setting model parameters, such as the number of neighbors (specified by the parameter, k) and the weights. However, the LLR takes more time than other conventional methods when it has to be applied to classification tasks. To overcome this problem, we propose a strategy of using a PRS to efficiently compute the optimization problem. In this paper, we demonstrate, first of all, that by completely discarding the points not included by the PRS, we can obtain a reduced set of sample points, using which, in turn, the quadratic optimization problem can be computed far more expediently. The values of the corresponding indices are comparable to those obtained with the original training set (i.e., the one which considers all the data points) even though the computations required to obtain the prototypes and the corresponding classification accuracies are noticeably less. The proposed method has been tested on artificial and real-life data sets, and the results obtained are very promising, and could have potential in PR applications.

[*]The second author was partially supported by NSERC, the Natural Sciences and Engineering Research Council of Canada. This work was generously supported by the National Research Foundation of Korea funded by the Korean Government (NRF-2011-0002517). We are extremely grateful to the anonymous Referees of the earlier version of the paper. Their queries helped improve the quality of this paper significantly. A preliminary *short* version of this paper was presented at AI'10, the 2010 Australasian Conference on Artificial Intelligence, Adelaide, Australia, in December 2010. The conference version has been extended by including more extensive results for additional data sets and comprehensively analyzing them in this paper.

[†]*Senior Member, IEEE.* Address : Dept. of Computer Science and Engineering, Myongji University, Yongin, 449-728 Korea. e-mail address : kimsw@mju.ac.kr.

[‡]*Chancellor's Professor*; *Fellow: IEEE* and *Fellow: IAPR.* Address: School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6. This author is also an Adjunct Professor with the University of Agder in Grimstad, Norway. e-mail address : oommen@scs.carleton.ca.

Keywords: *Prototype Reduction Schemes (PRS), k-Nearest Neighbor (k−NN) Learning, Locally Linear Reconstruction (LLR).*

1 Introduction

1.1 Motivation of the Problem

It is well known that the optimal classifier is the one that invokes the Bayes decision rule. If the *a priori* density functions were easily computable, and the class conditional densities were truly of a classical well-defined nature (for example, of the exponential family), the tasks of training and testing a pattern recognition/classification system would be trivial. In practice, however, these distributions are far from ideal, and consequently, the science and art of PR has had to develop various non-parametric methods for training and testing. The most elementary of these, and yet the most well-developed, constitute the Nearest Neighbor (NN) family of classifiers. Some strategies for speeding up the k-NN have been reported in the literature, e.g., in [6].

The idea behind the NN rules is age-old and is essentially encapsulated in the axiom that the information about a particular sample point can be gleaned from its nearest neighbors. Traditionally, the consequent decision rule merely performs a majority decision based on the decision of the closest k neighbors. The beauty of such a scheme is that the decision rule asymptotically attains the accuracy of the Bayes rule as the number of neighbors, k, is increased. More recently, to yield even more accurate results (for any given value of k), researchers have proposed that the neighbors need not be assigned equal weights. Rather, the question is that of modeling every feature point as a convex combination of its k neighbors, and from this perspective, the crucial question is that of determining the weights that are to be assigned to these neighbors.

Using the notation of [2], we formalize this as follows. Let $\{(x_i, y_i)\}_{i=1}^{n}$ be the reference set of training patterns, where $x_i \in \mathbb{R}^d$ is an input vector, and y_i is its target class label, where the class into which the classification is done yields $y_i \in \{1, \cdots, c\}$ when there are c classes. In the case of *regression*, y_i is a function value, where $y_i \in \mathbb{R}$. When a pattern x_{n+1} is to be tested, (i.e., its class is unknown, but it has to be assigned to one of the pre-determined classes), its predicted class label, $\hat{y}_{n+1}$, is calculated by merely considering the weights, w_i, $(i = 1, \cdots, k, w_i \geq 0, \sum_i w_i = 1)$, assigned to k-NNs of the test pattern, x_{n+1}.

The reader must observe that the crucial step in the above is the assignment of the above-mentioned corresponding weights. Typically, each neighbor is assigned a weight based on its distance between it and the test pattern, where a farther neighbor would be associated with a smaller weight (and vice versa), so that its influence on the final decision is less predominant[1]. From the perspective of these weights, one sees that this version of the k-NN is really a variation of locally weighted learning described in [26],

[1]Implementation-wise, this is sometimes achieved by using kernel functions, which decrease monotonically as the distance increases.

the *locality* being true because the class is predicted based on training patterns "closest" to the testing pattern, and the "weighting" nature being a consequence of these weights which rank the contribution of each neighbor in terms of its relative distance from the test pattern. The k-NN has been used in so-called "collaborative" filtering [27], where the system recommends target customers to a marketing agency, based on what "similar" customers prefer. As reported in [2], it has also been recently applied to financial forecasting, material engineering, image processing, face recognition, and intrusion detection [18], [19].

The most important paper, in this regard, which attempts to enhance this scheme is *probably* the one due to Kang and Cho [2], referred to as the Locally Linear Reconstruction (LLR) method. The fundamental, and rather brilliant idea behind the LLR, though simple, is quite intriguing, and it involves a quadratic optimization strategy explained presently. The salient feature of this scheme is that by invoking this optimization, one can systematically determine the model parameters, such as the number of neighbors (k) and the corresponding weights. However, the LLR, as proposed in [2], is computationally intensive.

To be more specific, the premise of the LLR [2] (which is actually akin to the Locally Linear Embedding (LLE) proposed in [3, 4]) is that if we can adequately describe a test pattern in terms of its local neighbors, we can also predict the target class, or estimate the function value of the test pattern as well. Thus, the LLR is able to preserve the locally linear topological properties in the space surrounding the test pattern. Indeed, it is a scheme by which the weights of the NNs are determined *collectively*, as opposed to *individually*, which is the case when kernel functions are used. This is what the LLR achieves. In addition, the LLR is expected to be robust to k. Thus, if we are given a sufficiently large value of k, the LLR can effectively identify the pertinent neighbors for reconstructing the test pattern.

The LLR procedure consists of three modules. The first step determines the k-NNs of the test pattern. One determines these by computing an appropriate (dis)similarity index between the patterns, which is, typically, a distance measure based on the context or characteristics of the data sets. The second step computes the associated weights of the neighbors, which effectively describes the test pattern in terms of the selected neighbors. Here, since we assume a locally linear topology around the test pattern, the LLR describes it in the form of a weighted linear combination of its neighbors. To find the best linear combination and its weights, w, we minimize the following reconstruction error:

$$Err(w) = \frac{1}{2}\left\| x_{n+1} - \sum_{j=1}^{k} w_j \tilde{x}_j \right\|^2 . \tag{1}$$

The final phase involves predicting the class of the target pattern.

The optimal weight, w, can be computed by minimizing the reconstruction error, $Err(w)$, with two additional constraints:

(1) $w_j \geq 0$ for all j, and

(2) $\sum_j w_j = 1$,

where this minimization problem can be solved by any quadratic programming (QP)[2] module. Once the weight is determined, the last step is to use the selected neighbors and their corresponding weights to predict the target class so as to achieve the classification or to estimate the function value of the target.

The most computationally intensive step for the LLR scheme, is the above-mentioned optimization procedure, achieved by a standard QP invocation. Indeed, as demonstrated in Theorem 2 (Computational complexity for LLR (Classification)) of [2], if n is the number of reference patterns, d is the number of attributes (features), and k is the number of NNs with $d > n$, then the computational complexities of conventional k-NN and LLR with standard QP are $\mathrm{O}(n \log n)$.

This is where our research comes into the picture: *The reason why we would like to intelligently reduce the number of sample points "n", is to reduce this prohibitive complexity.* More specifically, to tackle the computational burden, we propose a strategy of using a Prototype Reduction Scheme (PRS) to quickly and efficiently approximately compute the optimization problem. We formulate this as below.

1.2 Rationale for the Paper

We start with the premise that it is advantageous to compute the above mentioned optimization. As one sees, the primary drawback of k-NN learning is that there is no simple and formal scheme to determine the number of neighbors (k), and their associated weights. The number of neighbors is, in practice, often chosen empirically by cross-validation or by resorting to the opinion of domain experts. Regarding the weights associated with the neighbors, the rule of thumb so far is: *A more distant neighbor gets a smaller weight.* Thus, some kernel functions, which decrease monotonically as the distance increases (such as the inversion kernel, the exponential kernel and the Gaussian kernel), have been used. However, the literature states that there is no clear evidence that any of kernel functions is always superior to the others. Rather any one can outperform another on some particular data sets.

As opposed to this, we seek a strategy by which the associated computational burden can be reduced. Thus, in this paper, we propose a technique[3] for the fast computation of the reconstruction problem, and in particular, for the various classification applications. We advocate that rather than compute the reconstruction for the entire data set, the data be first reduced into a smaller representative subset using a PRS [7], [8], and that the reconstruction (classification) be achieved by invoking the corresponding method on *this* reduced data set. Thus, by completely discarding the points not included by the PRS, we can obtain a reduced set of sample points, using which, in turn, one can solve the quadratic optimization problem. The reader will observe, at once, that this can reduce the computational burden drastically, because the number of points chosen by the PRS is usually a small *fraction* of the total number of points found in the original data set. Our hypothesis, i.e., that the PRS can be effectively used to noticeably reduce the

[2]The details of why this is a QP problem is given in [2] and briefly explained in Setion 2.1.

[3]As a *prima facie* case, to justify the hypothesis of [2], we only consider the two-class problem. Indeed, the effective definition and computation of the measures for the multi-class problem are open.

computations and yet yield almost as accurate results, has been verified by testing on benchmark real-life and artificial data tests, as we shall presently explain.

The geometric aspect of this strategy is the following: Although the reconstructed samples are obtained by using the prototypes procured by invoking a PRS, these reconstructed points do not *individually* "optimally" represent their original counterparts. However, *collectively*, they are the best locations for the k-NNs of the points in the training set, which can, in turn, *collectively* represent the points for testing purposes too. This is truly an interesting feature.

1.3 Organization for the Paper

The paper is organized as follows. In Section 2, we briefly summarize the LLR proposed by Kang and Cho [2]. Thereafter, we provide a brief introduction to the families of PRSs. This leads us (in Section 3) to our proposed strategy to efficiently compute the optimization by using the PRSs. In Section 4, we provide experimental results for artificial and real-life benchmark data sets. Finally, Section 5 concludes the paper.

2 An Overview : Locally Linear Reconstruction and Prototype Reduction Schemes

2.1 Locally Linear Reconstruction

In this section, we explain the LLR [2] for pattern classification and recognition (as also considered for instance-based learning), and in particular for the k-NN. As mentioned earlier, the main idea behind the LLR originates from the concept of the Locally Linear Embedding (LLE) [3, 4], which is one of the widely-used non-linear dimensionality reduction techniques. In the case of the LLR, we attempt to enforce a general premise in the topological space for the k-NN by arguing that if it is possible to accurately describe the input vector for a given query by its neighboring reference patterns, it is also possible to predict (estimate) well the target class (value) of the query with a small error.

To initiate discussions in this regard, we first state the notation that we shall use (in a d-dimensional feature space), after which we shall formally describe the LLR.

- x_i is a "query" (the testing point, for example) in the feature space, and is a $d \times 1$ vector.
- $\widehat{x}_i$ is a re-constructed version of x_i, and is also a $d \times 1$ vector.
- y_i (or $\widehat{y}_i$) is the target (or predicted) class label of x_i (or $\widehat{x}_i$), and is a scalar.
- X^i_{NN} is a $d \times k$ matrix, and contains the d-dimensional k-NNs of x_i.

- $w_{i,NN}$ is a $k \times 1$ vector. It is the corresponding weight vector obtained from X^i_{NN}. The matrix W, which is the collection of $w_{i,NN}$'s, is the set of vectors sought for, and $w_{i,j}$ is the set of weights for x_j with regard to the sample point x_i. Observe that $w_{i,j}$ will be zero if x_j is not a neighbor of x_i.

- The matrix N is the neighborhood indicator matrix whose element $N_{i,j} = 0$ if x_j is not a neighbor of x_i, and is unity otherwise. For ease of notation, $N(i)$ will represent the NNs of x_i.

When a query is given, the method first selects the k-NNs of the query – the testing sample. Once these NN patterns have been selected, the set of weights corresponding to the neighbors are determined by minimizing the LLR error, $Err(W)$, defined as the sum of the errors E_i as follows:

$$\begin{aligned} Err(W) &= \frac{1}{2}\sum_i E_i \\ &= \frac{1}{2}\sum_i \left\| x_i - w^T_{i,NN} x_j \right\|^2 , \end{aligned} \tag{2}$$

where every x_j in the above equation is a NN of x_i, and $\|\cdot\|$ implies the 2-norm.

The weights, W, which minimize the reconstruction error, $Err(W)$, can be obtained by solving the above minimization problem. Also, since the constraints on the optimization problem differ depending on whether the learning task is a classification or regression problem, the corresponding procedures for solving them are different as well. In particular, for classification tasks, we need to impose two additional constraints on W, namely, that all the weights must be non-negative, and that the sum of the neighbors' weights must be *unity* for every query. Thus,

$$\begin{aligned} Err(W) &= \frac{1}{2}\sum_i \left\| x_i - w^T_{i,NN} x_j \right\|^2 \\ &= \frac{1}{2}\sum_i \left(x_i - X^i_{NN} w_{i,NN} \right)^T \left(x_i - X^i_{NN} w_{i,NN} \right) \\ &= \frac{1}{2}\sum_i \left\{ x_i^T x_i - 2x_i^T \left[X^i_{NN} \right] w_{i,NN} + w^T_{i,NN} \left[X^i_{NN} \right]^T \left[X^i_{NN} \right] w_{i,NN} \right\} . \end{aligned} \tag{3}$$

By examining Eq. (3), we see that we can obtain the weights for the k-NNs of x_i, $w_{i,NN}$, by solving the following optimization problem[4]:

$$\begin{aligned} &Min\ Err(w_{i,NN}) = \frac{1}{2} w^T_{i,NN} \left[X^i_{NN} \right]^T \left[X^i_{NN} \right] w_{i,NN} - x_i^T \left[X^i_{NN} \right] w_{i,NN}, \\ &\text{such that} \quad w_{i,NN} \geq 0, \ \sum_j w_{i,j} = 1 \ \ \forall i. \end{aligned} \tag{4}$$

[4]The quadratic programming problem, min $\frac{1}{2}\underline{U}^T \boldsymbol{H} \underline{U} + \underline{B}^T \underline{U}$, such that $\boldsymbol{A}\underline{U} \leq 0$, $\boldsymbol{A}_{eq}\underline{U} = \underline{b}_{eq}$, and $\underline{l}_b \leq \underline{U} \leq \underline{u}_b$, (where $\boldsymbol{H}$, $\boldsymbol{A}$, and $\boldsymbol{A}_{eq}$ are matrices, and $\underline{B}$, $\underline{b}_{eq}$, $\underline{l}_b$, $\underline{u}_b$, and $\underline{U}$ are vectors) defines a set of lower and upper bounds on the design variables, $\underline{U}$, so that the solution is in the range $\underline{l}_b \leq \underline{U} \leq \underline{u}_b$.

After obtaining the weights assigned to a sample point $\hat{x}_i$, corresponding to the query x_i given for classification, we can calculate the predicted class label of $\hat{x}_i$, $\hat{y}_i$, by a weighted sum of the samples of the NNs of x_i as follows:

$$\hat{y}_i = \sum_{j \in N(i)} w_{i,j} \; y_j. \tag{5}$$

As the reader will observe, although this strategy is expedient, it involves the non-trivial optimization which is computationally intensive. If we choose such a strategy, this cannot be avoided. But our position is that it need not be done for all the sample points, but merely for a smaller subset of points which *represent* them. Thus, our study involves the effectiveness of using PRSs to minimize the above computations for the entire set of data points.

2.2 Why the LLR works

It is pertinent to query why a LLR works in the first place. As argued in [2], primarily, a LLR can be more robust to the choice of the value of k than if one used kernel functions. In other words, a LLR can reduce the importance of determining the "optimal" k, because for any given k, it can attain to the optimal weights. Thus for classification problems, the essential neighbors that are needed for optimal reconstruction can identified with any given k, and so when k is small, even though the reconstruction error is not small enough, one can increase k (and optimize the weights) so that the reconstruction error is acceptable. The effect of this is that the set of weights w would be sparse when 'k' is large. Additionally, once the value of k is set, the LLR provides us with a formal procedure to assign the weights to the neighbors – unlike the conventional heuristic approaches. Finally, we observe that whenever we reconstruct a test pattern, a NN philosophy advocates that we *only* consider some of the neighboring patterns, and not *all* the reference patterns. Thus, the NN paradigm implicitly states that we are effectively relying on the local (and not global) topology of the feature space. It is clearly plausible that a LLR can improve the classification and regression performance of a k-NN learning because it not only considers the similarity between patterns (as kernel functions do), but also incorporates the local *topology*.

2.3 Prototype Reduction Schemes: State-of-the-Art

In non-parametric pattern classification which uses the NN or the $k-$NN rule, each class is described using a set of sample prototypes, and the class of an unknown vector is decided based on the identity of the closest neighbor(s) which are found among all the prototypes. To reduce the number of training vectors, various PRSs have been reported in the literature - two excellent surveys are found in [7], [8]. Rather than embark on yet another survey of the field, we mention here a *few* representative methods of the "zillions" that have been reported. One of the first of its kind is the Condensed Nearest Neighbor (CNN) rule [9]. The reduced set produced by the CNN, however, customarily includes "interior" samples, which can be completely eliminated, without altering the performance of the resultant classifier. Accordingly, other methods have been proposed successively, such as the Reduced Nearest Neighbor (RNN) rule, the Prototypes for Nearest Neighbor (PNN) classifiers [10], the Selective Nearest Neighbor (SNN) rule, [12], two modifications of the CNN [13], the Edited Nearest Neighbor (ENN) rule [11], and the non-parametric data reduction method [16]. Besides these, in [14], the Vector Quantization (VQ) and the Bootstrap techniques have also been reported as being extremely effective approaches to data reduction. Recently, Support Vector Machines (SVM) [17] have proven to possess the capability of extracting vectors that support the boundary between any two classes. Thus, they have also been used to satisfactorily represent the global distribution structure. A brief description of the three PRSs which we shall use follows.

2.3.1 The CNN

The CNN has been suggested as a rule that retains the basic approach of the NN philosophy to determine a consistent subset of the original sample set. However, this technique, in general, will not lead to a minimal consistent sample set, which is a set that contains a minimum number of samples that is able to correctly classify all the remaining samples in the given set.

Initially, the first pattern of the original training set T is copied to T_{CNN}. Then, the second pattern of T is classified by considering T_{CNN} as the reference set. If that pattern is correctly classified, it is moved to the set of patterns to be removed. Otherwise, it is moved to the reference set. This procedure is repeated for all the patterns of T. Once all the patterns have been considered for such a verification phase, the same procedure is repeated for the set R, which contains the patterns to be removed. This phase will be repeated until either the set R becomes empty (i.e. the reference set is equivalent to the original set), or no more patterns are left in R which have any effect on the classification. Once this pre-processing has been achieved, T$_{CNN}$ will be the reference set for the NN rule. The patterns that are moved to R will be discarded.

2.3.2 The PNN

The PNN algorithm [10], can be described as follows: Given a training set T, the algorithm starts with every point in T as a prototype. We now define two auxiliary sets A and B. Initially, set A is empty and set B is equal to T, where every prototype (data sample) has an associated weight of unity. The algorithm selects an arbitrary point in B and initially assigns it to A. After this, the two closest prototypes p in A and q in B of the same class are merged, successively, into a new prototype, p^*. This is done only if the merging will not degrade the classification of the patterns in T, where p^* is the weighted average of p and q. For example, if p and q are associated with weights W_p and W_q, respectively, p^* is defined as $(W_p p + W_q q)/(W_p + W_q)$, and is assigned a weight, $W_p + W_q$. After determining the new value of p^*, p from A and q from B are deleted, and p^* is included into A. Thereafter, the procedure is repeated until a static condition attains.

If either p and q are not of the same class, or if merging is unsuccessful, q is moved from B to A, and the procedure is repeated. When B becomes empty, the entire procedure is repeated by letting B be the final A obtained from the previous cycle, and by resetting A to be the empty set, until no new merged prototypes are obtained. The final prototypes in A are then used in a NN classifier. The bottom-up nature of this method is crucial to its convergence.

2.3.3 The HYB

In designing NN classifiers, however, it seems to be intuitively true that prototypes near the separating boundary between the classes play more important roles than those which are more interior in the feature space. Thus, in creating or selecting prototypes, vectors near the boundaries between the classes have to be considered to be more significant, and the created prototypes need to be moved (or adjusted) towards the classification boundaries so as to yield a higher performance. Based on this philosophy, Kim and Oommen [20], [22] proposed a new hybrid approach (HYB) that involved two distinct phases, namely, those of selecting and adjusting [20], [21]. In the first phase, initial prototypes are *selected* or *created* by any of the conventional reduction methods mentioned earlier. After this selection/creation phase, the technique in [20], [21] suggests a second phase in which the proposed reduced prototypes are migrated to their "optimal" positions by *adjusting* them by invoking an LVQ3-type *learning* scheme. The relative advantages of the scheme in [20], [21] have been demonstrated on both artificial and real-life data sets.

2.3.4 A PRS for "Large" Datasets

To overcome the computational burden for "large" datasets, Kim and Oommen also proposed a recursive HYB mechanism in [22]. In [22], the data set is sub-divided recursively into smaller subsets to filter out the "useless" internal points. Subsequently, a conventional PRS (i.e., HYB) processes the smaller subsets of data points that effectively sample the entire space to yield *subsets* of prototypes – one set of prototypes for each subset. The prototypes, which result from each subset, are then coalesced, and processed again by the PRS to yield more refined prototypes. In this manner, prototypes which are in the interior of the Voronoi boundaries, and are thus ineffective in the classification, are eliminated at the subsequent invocations of the PRS. Thus, the processing time of the PRS is *noticeably* reduced.

Finally, we conclude this section by remarking that as researchers who have published extensively in this field [20, 21, 22, 23, 24], from our prior experience, we believe that it would be fair to say that of the "zillions" of PRSs available, the HYB [20] is the most efficient one for "small" datasets, and that the Recursive method described in [22] is the best for "large" data sets. This is when we use the computational time and the accuracy as a metric. The criteria here are, however, slightly different. In this present situation, we are looking for a scheme which yields the LLR approximations and yet the best possible accuracies, and for which the time (which can be considered to be preprocessing) is not so crucial. In this light, we believe that as an overall conclusion, we can say that the HYB seems to be particularly poor for almost all the artificial data sets for which the PNN and CNN are almost comparable. On the other hand, the HYB seems to be the most superior scheme for the real-life data sets, sometimes yielding an accuracy superior to that obtained by the LLR-processed entire data set.

2.3.5 More Recent Applications and Comparisons of PRSs

Changing now the emphasis, we observe that with regard to designing classifiers, PRSs can be employed as a pre-processing module to reduce the data set into a smaller representative subset, and they have thus been reported to optimize the design of KNS classifiers in [23], [24]. The details of these are omitted here as they are irrelevant.

This overview of the state-of-the-art of PRSs should be sufficient to help us proceed in formulating our solution to the problem at hand.

3 Schema for the Proposed Solution

Our goal is to "quickly" find out the class of a query point in the input feature space after reconstructing an approximated version of the corresponding sample using its NNs. However, rather than reconstruct the approximated data sample using the entire training set, we advocate that the data be first reduced into a smaller representative subset using a PRS, and that the data point be estimated by invoking a reconstruction scheme on *this* reduced data set. Clearly, one is interested in the classification accuracy of the consequent $k-$NN classifier.

The proposed scheme can be formalized as follows:

```
Algorithm 1 PRS_LLR
  Input: The training set, $T_1$, and test set, $T_2$.
  Output: Testing by utilizing a fast reconstruction of the approximated query point using a reduced data set rather than the entire training set.
  Assumption 1: The algorithm has access to a PRS, such as the CNN, PNN, or HYB (see [20], [21]).
  Assumption 2: The algorithm has access to the LLR algorithm mentioned previously [2].
  Method:
    Step 1: Select the representative set, $Y$, from the training set $T_1$ by resorting to a PRS.
    Step 2: Find the closest neighbors, $X^i_{NN}$, for a query $x_i \in T_2$ from $Y$, rather than from $T_1$.
    Step 3: Compute corresponding weight vector, $w_{i,NN}$, using the LLR algorithm and a $k_1$-NN rule.
    Step 4: Approximate the predicted class label, $\widehat{y}_i$, with LLR using $Y$, $w_{i,NN}$, and a $k_2$-NN rule.
  End Algorithm PRS_LLR
```

Figure 1 shows the schematic diagram of the process used for evaluating the proposed method.

We would like to emphasize that there are a few fundamental differences between what we propose and the original LLR method proposed in [2]. First of all, we observe that the computation of the LLR

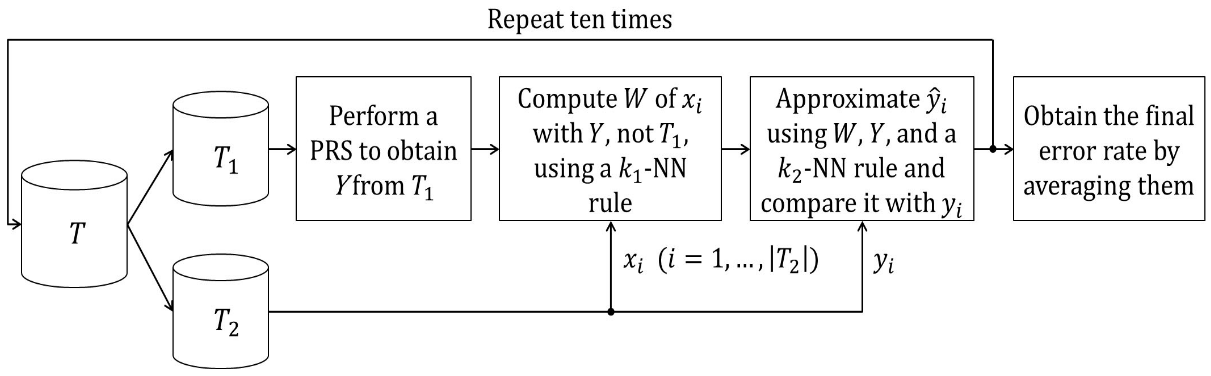
Repeat ten times
T
T_1
T_2
Perform a PRS to obtain Y from T_1
Compute W of x_i with Y, not T_1, using a k_1-NN rule
Approximate $\hat{y}_i$ using W, Y, and a k_2-NN rule and compare it with y_i
Obtain the final error rate by averaging them
x_i $(i = 1, \ldots, |T_2|)$
y_i

Figure 1: The schematic diagram of the process used for evaluating the proposed method. Here, a PRS is performed in a "off-line" mode. The details of the figure are discussed in the text.

weights does not involve the entire training set T, but a representative set, Y, derived from it using a PRS. Secondly, we note that the weights that are computed for the LLR involve a NN rule, using k_1 neighbors, where the latter is the pre-determined degree of the NN classifier used for the training phase. But once the reconstructed point is obtained, we now have the freedom of testing it using the most suitable NN classifier, which may not necessarily be a k_1-NN classifier. Indeed, as in any PR problem, given a training set, the practitioner has the freedom to choose the best NN classifier that suits his application. In the same vein, in our case, we choose the best "Testing" NN classifier (a k_2-NN classifier) for the application domain, using the modified "Training" set, Y, and the modified testing sample, $\underline{\widehat{X}}_i$. It turns out that usually, k_2 is quite distinct from k_1.

The reader should observe that we could have, indeed, used any one of a host of PRS schemes to achieve what we did[5]. Most of the PRSs involve a certain amount of computational time to obtain (select/create) the prototypes. The work of [21] gives an entire overview of the state of the art of PRSs. The issue here is not merely the reduction in the computational time for obtaining the prototypes, but the reduction in time that one could glean by using the reduced set of points *for the LLR itself.* In this regard we mention that, in particular, the recursive scheme proposed for large data sets [22] would be very beneficial, because, for such large sets, the computational time for the LLR could be prohibitive. Unfortunately, we have not included the corresponding experimental results or such datasets, because the computation of the LLR for these "very large datasets" is, understandably, infeasible

We shall now demonstrate the power of **Algorithm PRS_LLR**.

[5]Bezdek *et al* [7], who composed the second and more recent survey of the field, reported that there are "zillions!" of methods for finding prototypes (see page 1459 of [7]).

4 Experimental Set-Up, Results and Evaluation

4.1 Experimental Data

The proposed scheme has been tested and compared with the conventional LLR method reported in the literature. This was done by performing experiments on a number of data sets, as summarized in Table 1. In each case, the sample vectors of each data set were divided into two subsets of equal size $T1$ and $T2$ (typically, used for training and validation, alternatively). The computation was done on each subset and subsequently averaged. Figure 1 shows the schematic diagram of the process used for evaluating the proposed method.

Table 1: The artificial and real-life data sets used in the experiments to evaluate the effectiveness of computing the LLR from the original data set and their reduced prototypes. The sample vectors of each data set are "randomly" divided into two subsets (T_1; T_2) of equal size (typically, used for training and validation). The division and computation (i.e., training and validation) were repeated "ten" times on each subset and subsequently averaged.

Data type	Datasets	# of samples (T_1; T_2)	# of features	# of classes
Artificial data	Non_n1	100 (50; 50)	8	2
	Non_n2	1,000 (500; 500)	8	2
	Non_l1	100 (50; 50)	2	2
	Non_l2	1,000 (500; 500)	2	2
Real-life data	Iris2	100 (50; 50)	4	2
	Ionos	351 (176; 175)	34	2
	Sonar	208 (105; 104)	60	2
	Arrhy	452 (226; 226)	279	16

The Prototypes Obtained: Table 2 shows a comparison of the numbers of prototype vectors extracted (or selected) from the artificial and real-life data sets, namely, "Non_n1", "Non_n2", "Non_l1", "Non_l2", "Iris2", "Ionos", "Sonar", and "Arrhy", respectively, using the CNN, PNN, and HYB methods. The ten values for each data set are the numbers of prototype vectors extracted from the randomly divided training subsets, T_1's, respectively. The reader should observe that both Figure 1 and Table 2 report the proposed scheme as implemented using the **PRS_LLR** algorithm, which has been run for 10 times. By averaging the corresponding results, we were able to obtain the mean accuracy and standard deviation, which are the respective reported quantities.

From Table 2, for example, we can see that ten numbers of the prototype vectors selected with the CNN method for "Non_n1" dataset are $4, 11, 10, 9, 10, 10, 10, 7, 8, 4$, respectively. Each of them is considerably smaller than the size of the original data set (50 for Non_n1). Using the selected vectors as a representative of the training data set, we can reduce the cardinality of the dataset (and the consequential computations) without noticeably degrading the performance. The reduction of the classification processing time follows as a natural consequence. As an observation, we also mention that the reduction rate increased dramatically as the size of the data sets was increased.

Table 2: The number of prototype vectors extracted (or selected) from experimental data sets using the CNN, PNN, and HYB methods. The ten values for each data set are the numbers of prototype vectors obtained from the randomly-divided training subsets, T_1's, respectively.

Data types	Data names	Data sizes	Selection methods	Number of the selected prototypes									
				Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 8	Set 9	Set10
Artificial data	Non_n1	50	CNN	4	11	10	9	10	10	10	7	8	4
			PNN	4	7	9	6	9	8	10	6	7	4
			HYB	2	6	8	4	7	8	8	4	6	2
	Non_n2	500	CNN	54	52	53	68	65	57	60	64	68	65
			PNN	48	43	313	280	324	53	281	60	324	57
			HYB	56	49	50	73	63	54	69	61	58	63
	Non_l1	50	CNN	15	13	12	14	13	12	16	16	19	16
			PNN	13	12	11	11	12	15	15	16	17	14
			HYB	7	6	6	10	9	9	10	9	10	9
	Non_l2	500	CNN	121	99	105	103	113	118	100	98	89	92
			PNN	113	96	94	90	93	107	90	87	78	92
			HYB	69	59	58	62	62	68	58	53	45	56
Real-life data	Iris2	50	CNN	11	12	14	16	13	15	18	14	14	8
			PNN	7	10	12	14	11	11	11	11	11	6
			HYB	5	6	5	7	5	6	9	6	8	4
	Ionos	176	CNN	51	42	40	43	45	52	48	45	45	39
			PNN	40	26	30	45	42	41	49	32	29	38
			HYB	42	50	43	43	39	51	52	46	41	40
	Sonar	105	CNN	32	45	46	49	42	41	51	47	45	41
			PNN	31	31	32	34	25	27	34	30	32	32
			HYB	45	46	56	47	51	49	54	48	54	50
	Arrhy	227	CNN	38	30	21	23	25	23	24	26	23	26
			PNN	9	7	9	7	6	4	8	8	7	9
			HYB	54	59	60	59	55	54	56	54	52	49

Table 3: A comparison of the classification accuracies (and ± standard deviations) of *knnc* (where the cardinal numbers of the nearest neighbors, k_2's, are 1, 3, 5, 7, 9, 11, and 13 sample points) obtained from the artificial data sets, namely, "Non_n1", "Non_n2", "Non_l1", and "Non_l2", respectively, using the WHL, CNN, PNN, and HYB methods.

Datasets	PRS	NN: k_2=1	NN: k_2=3	NN: k_2=5	NN: k_2=7	NN: k_2=9	NN: k_2=11	NN: k_2=13
	WHL	93.60	96.80	95.20	95.00	95.20	95.00	94.60
		±2.45	±1.93	±2.52	±2.70	±2.69	±3.16	±3.13
Non_n1	CNN	93.60	94.00	74.40	57.40	42.40	5.00	–
		±2.45	±4.98	±39.29	±36.62	±40.39	±15.81	–
	PNN	93.40	94.00	70.20	42.40	19.80	–	–
		±2.83	±2.82	±39.53	±40.29	±34.45	–	–
	HYB	96.00	76.00	58.00	34.40	–	–	–
		±2.30	±40.16	±49.93	±46.52	–	–	–
	WHL	93.32	93.98	94.18	94.40	94.44	94.74	94.78
		±0.82	±0.59	±1.20	±0.93	±0.86	±1.02	±0.98
Non_n2	CNN	93.08	93.80	94.20	94.72	94.76	94.74	94.72
		±0.73	±0.64	±0.63	±0.80	±0.81	±0.82	±0.75
	PNN	92.90	94.02	94.32	94.60	94.58	94.64	94.70
		±0.99	±0.80	±0.80	±0.64	±0.69	±0.75	±0.84
	HYB	91.16	87.98	81.62	72.00	59.80	31.54	21.80
		±1.74	±2.40	±5.30	±7.07	±16.67	±20.76	±17.23
	WHL	89.00	87.60	84.20	80.80	76.80	75.00	70.80
		±2.86	±3.23	±6.21	±5.18	±6.87	±8.34	±9.34
Non_l1	CNN	87.00	77.00	65.20	56.80	57.20	57.00	45.20
		±3.43	±9.94	±7.49	±7.95	±7.95	±8.44	24.91
	PNN	85.20	74.00	61.20	57.00	61.40	56.20	32.00
		±3.55	±6.03	±7.25	±9.34	±9.33	±8.24	±29.30
	HYB	80.60	68.00	55.80	45.60	37.00	–	–
		±9.24	±8.43	±9.99	±25.02	±26.26	–	–
	WHL	89.50	90.84	91.26	91.52	91.38	91.56	91.22
		±0.62	±0.77	±0.55	±0.52	±0.95	±1.01	±0.98
Non_l2	CNN	88.42	86.20	85.80	85.16	84.40	84.96	82.82
		±0.70	±4.23	±3.55	±3.71	±4.51	±3.82	±4.63
	PNN	87.32	84.94	85.82	85.54	84.42	82.90	81.18
		±0.54	±2.58	±3.18	±3.68	±3.86	±3.40	±4.87
	HYB	89.30	85.62	77.50	69.84	66.28	63.26	61.52
		±1.22	±1.45	±3.16	±3.78	±3.82	±2.89	±4.95

The Classification Accuracies Obtained with Non_LLR: Prior to presenting the experimental results obtained with the formal PRS_LLR algorithm, in order to illustrate the functioning of the combination the PRS and LLR processes, we present a comparison of the classification accuracies of a k-nearest neighbor classifier (*knnc*, which is implemented with PRTools [28]) designed with the prototypes of Table 2, not T_1, and evaluated with the testing data sets T_2. Table 3 and Table 4 show the classification accuracies (and ± standard deviations) of *knnc* (where the cardinal numbers of the nearest neighbors, k_2's, are 1, 3, 5, 7, 9, 11, and 13 sample points) obtained from the artificial and real-life data sets, respectively. Here, the number of nearest neighbors used to decide the class label is referred to as k_2. So, when the number of prototype vectors is smaller than k_2, the classification was not done, and indicated by a "–" in the tables.

From Tables 3 and 4, we can see that the classification accuracies of the WHL method (i.e., with

Table 4: A comparison of the classification accuracies (and $\pm$ standard deviations) of *knnc* (where the cardinal numbers of the nearest neighbors, k_2's, are 1, 3, 5, 7, 9, 11, and 13 sample points) obtained from the real-life data sets, namely, "Iris2", "Ionos", "Sonar", and "Arrhy", respectively, using the WHL, CNN, PNN, and HYB methods.

Datasets	PRS	NN: k_2=1	NN: k_2=3	NN: k_2=5	NN: k_2=7	NN: k_2=9	NN: k_2=11	NN: k_2=13
	WHL	92.60	92.40	92.60	94.00	93.80	93.20	92.20
		±1.64	±2.27	±2.31	±2.10	±2.39	±3.55	±2.89
Iris2	CNN	90.20	87.20	82.20	76.40	66.00	62.80	45.60
		±2.89	±4.13	±11.21	±16.46	±26.93	±26.93	±33.68
	PNN	90.80	85.00	82.00	68.20	59.00	39.40	5.00
		±2.52	±6.61	±7.30	±27.07	±31.78	±28.62	±15.81
	HYB	91.40	80.20	54.80	20.40	5.00	–	–
		±1.89	±11.67	±24.76	±33.66	±15.81	–	–
	WHL	86.19	84.54	83.52	83.01	83.06	82.67	81.30
		±2.10	±1.87	±2.25	±1.63	±1.83	±1.99	±2.63
Ionos	CNN	86.07	81.81	79.82	72.44	65.90	58.35	54.09
		±1.82	±5.79	±3.20	±6.10	±10.86	±13.27	±15.11
	PNN	87.55	79.60	73.06	64.60	58.92	53.40	44.37
		±1.90	±6.66	±14.08	±18.93	±20.37	±16.74	±11.24
	HYB	84.94	84.14	73.63	68.86	64.20	61.98	60.73
		±1.17	±1.45	±5.03	±4.95	±7.18	±6.05	±6.00
	WHL	78.25	74.36	70.29	66.89	67.96	67.08	65.72
		±4.37	±5.68	±3.77	±4.74	±5.94	±4.65	±3.33
Sonar	CNN	76.11	65.43	65.92	63.88	61.94	60.00	60.29
		±5.16	±5.24	±5.20	±4.31	±5.23	±3.50	±6.01
	PNN	75.24	62.23	58.15	59.22	60.67	57.28	57.28
		±3.46	±6.85	±4.27	±3.48	±4.58	±3.93	±3.63
	HYB	68.8350	67.96	59.90	58.64	57.08	58.73	58.34
		±4.4243	±4.31	±6.14	±5.84	±6.33	±7.50	±5.79
	WHL	97.68	97.82	98.08	98.04	98.40	98.31	98.26
		±0.65	±0.84	±0.59	±0.66	±0.76	±0.62	±0.53
Arrhy	CNN	95.33	94.08	88.35	84.53	78.31	72.80	66.35
		±1.42	±3.59	±10.56	±14.07	±14.53	±15.59	±13.27
	PNN	97.73	71.64	53.91	38.97	13.73	–	–
		±0.60	±14.34	±21.21	±21.11	±22.11	–	–
	HYB	94.84	94.88	78.53	68.80	60.53	57.51	56.62
		±2.26	±1.74	±14.08	±13.20	±6.12	±4.89	±4.04

knnc designed with the entire data set T_1) and the CNN, PNN, HYB methods (i.e., *knnc* designed with the corresponding prototypes) are *almost* the same when $k_2 = 1$. However, especially, for Non_n1 and Non_l1, when k_2 increases, the classification accuracies of the both methods are different.

From Tables 3 and 4, it should also be observed that the classification accuracies of the HYB method are lower than those obtained by the CNN and PNN methods. The reasons for this observation can be explained as follows: First of all, in this experiment, to make things easy, among the HYB's four parameters[6], such as α, β, w, and η, we varied only the parameter w (i.e., the window length). The other parameters were fixed as constants. Further, the HYB consists of two steps: (1) Determining the support vectors (SVs) from T_1 using the SVM algorithm, and (2) adjusting the SVs using a LVQ3 algorithm. In this experiment, we utilized a polynomial kernel function of degree 1 for the SV-determination step (the *svm-train* given in [29] was employed), and a 1-nearest neighbor rule was used in the adjusting step of LVQ3. It was very interesting to observe that in Tables 3 and 4, the classification accuracy of the HYB decreased sharply as k_2 increased. Finally, in this experiment, the classification was performed by repeated the testing with "ten" subsets of T_1 for each data set. We computed the SVs from the individual "ten" subsets, but adjusted them (SVs) by modifying three of the four parameters determined by the use of a *single* subset.

The classification accuracies of Tables 3 and 4 will be compared with those of the PRS_LLR algorithms in subsequent sections.

The Classification Accuracies Obtained with PRS_LLR: In order to illustrate the functioning of the combination of the PRS and LLR processes, first of all, we present a graphical comparison of the classification accuracies of *knnc* (where the cardinal numbers of the nearest neighbors, k_2's, are 1, 3, 5, 7, 9, 11, and 13 sample points) designed in the feature spaces, reconstructed with the LLR algorithms (where the cardinal numbers of the nearest neighbors, k_1's, are 1, 3, 5, 7, 9, 11, and 13 sample points) for the artificial and real-life databases.

Figures 2, 3, 4, and 5 show the comparison of the classification accuracies (and $\pm$ standard deviations) (%) of the artificial databases, such as Non_n1 and Non_l2, and the real-life databases, such as Ionos and Arrhy, respectively. Here, the classifiers of *knnc* were designed in the feature space reconstructed with a PRS_LLR algorithm for the prototypes extracted from T_1, shown in Table 2, and evaluated with T_2. Also, when the number of prototype vectors is smaller than k_2 of *knnc*, the percentage classification accuracies (given on the vertical axes) are represented as "0" in the figures.

From the figures, it can be observed that the two error rates obtained with the WHL method and the CNN, PNN, and HYB methods are *almost* the same for the entire range of the values of k_2 when k_1 has small values. In other words, we can obtain the classification accuracy of WHL by using analogous classifiers designed with the small numbers of vectors (i.e., prototypes) extracted from T_1 using a PRS.

From Figure 2 (a), (b), (c), and (d) of Non_n1 (whose size is 50), for example, it can be observed

[6]The reader should kindly recall that the parameters of the HYB are sensitive to the characteristics of training data sets.

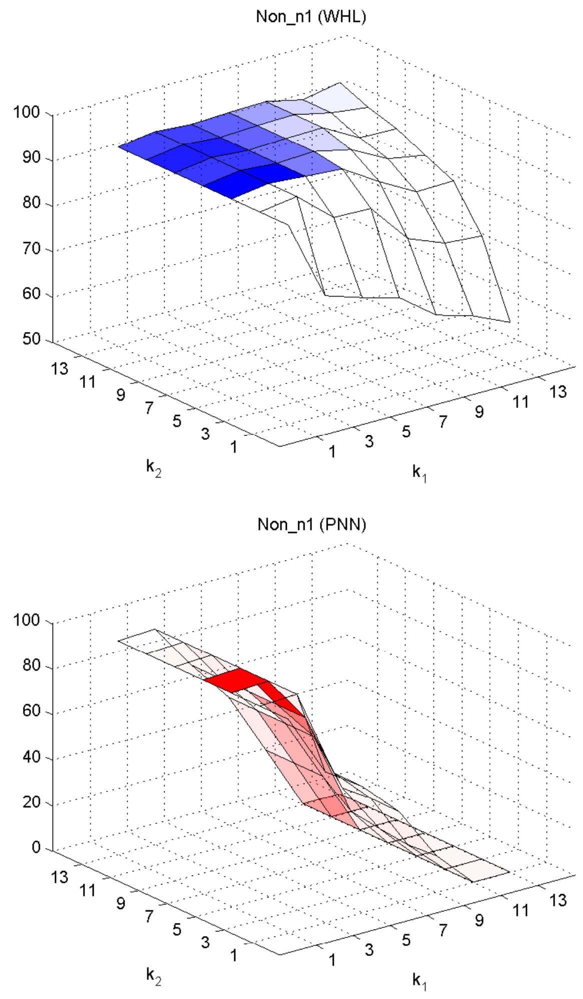

Figure 2: A comparison of the estimated error rates of *knnc* (where the cardinal numbers of the nearest neighbors, k_2's, are 1, 3, 5, 7, 9, 11, and 13 sample points), built in the feature spaces reconstructed with the LLR algorithm in which the cardinal numbers of the nearest neighbors, k_1's, are 1, 3, 5, 7, 9, 11, and 13 sample points. This is for the artificial database, Non_n1: (a) top left, (b) top right, (c) bottom left, and (d) bottom right. The figures (a) - (d) are obtained with the WHL, CNN, PNN, and HYB methods for Non_n1, respectively.

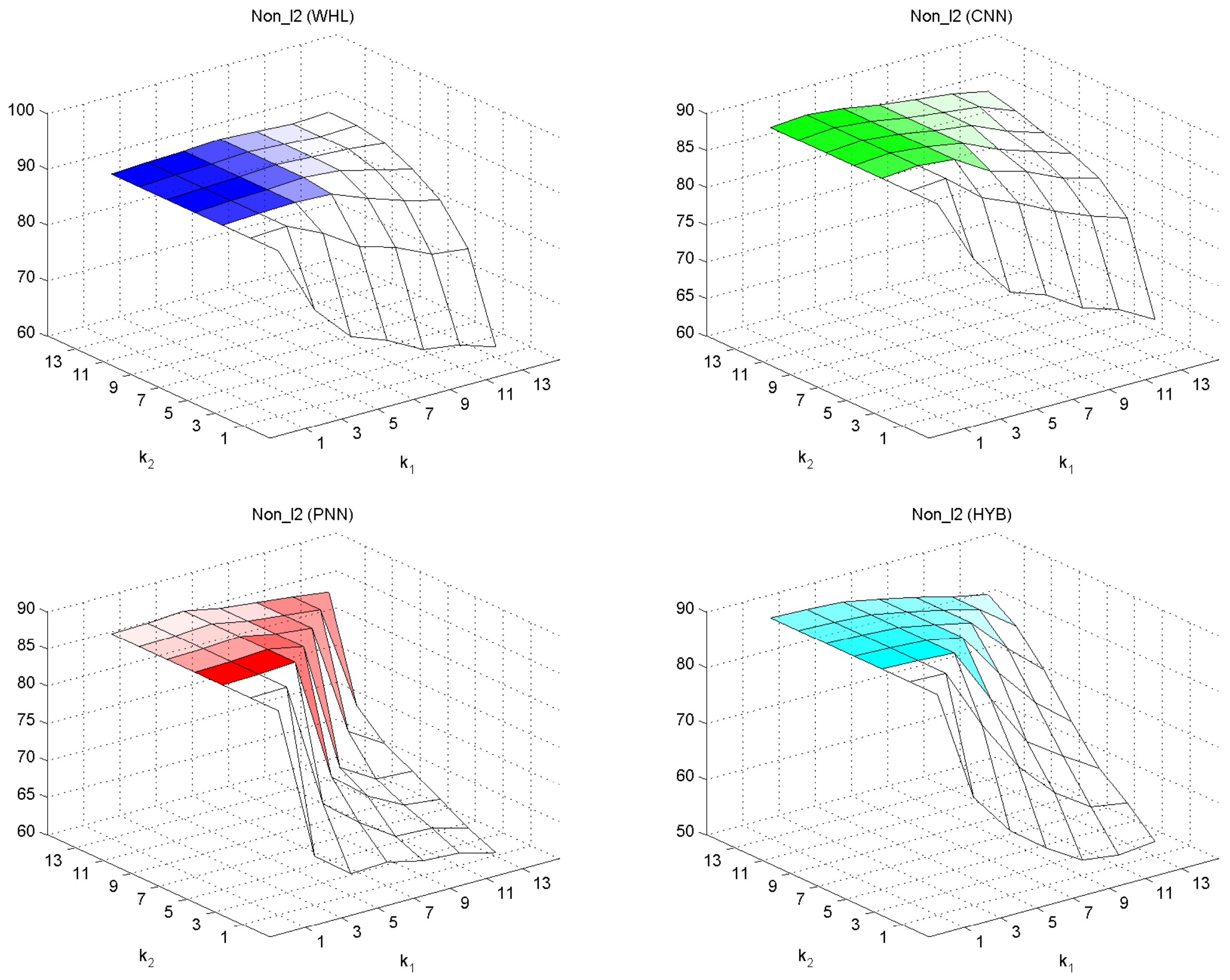

Figure 3: A comparison of the estimated error rates of *knnc* (where the cardinal numbers of the nearest neighbors, k_2's, are 1, 3, 5, 7, 9, 11, and 13 sample points), built in the feature spaces reconstructed with the LLR algorithm in which the cardinal numbers of the nearest neighbors, k_1's, are 1, 3, 5, 7, 9, 11, and 13 sample points. This is for the artificial database, Non_l2: (a) top left, (b) top right, (c) bottom left, and (d) bottom right. The figures (a) - (d) are obtained with the WHL, CNN, PNN, and HYB methods for Non_l2, respectively.

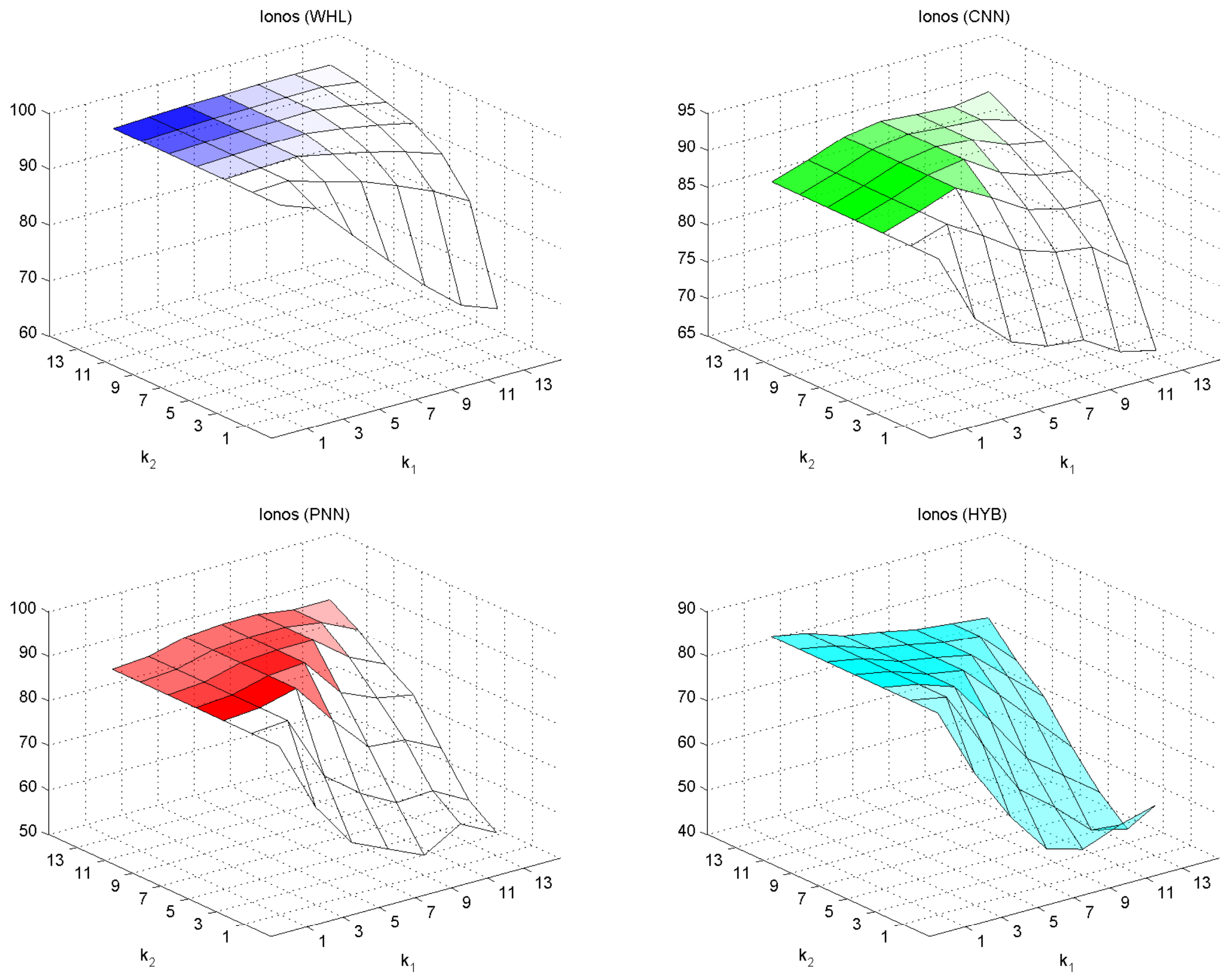

Figure 4: A comparison of the estimated error rates of *knnc* (where the cardinal numbers of the nearest neighbors, k_2's, are 1, 3, 5, 7, 9, 11, and 13 sample points), built in the feature spaces reconstructed with the LLR algorithm in which the cardinal numbers of the nearest neighbors, k_1's, are 1, 3, 5, 7, 9, 11, and 13 sample points. This is for the real-life database, Ionos: (a) top left, (b) top right, (c) bottom left, and (d) bottom right. The figures (a) - (d) are obtained with the WHL, CNN, PNN, and HYB methods for Ionos, respectively.

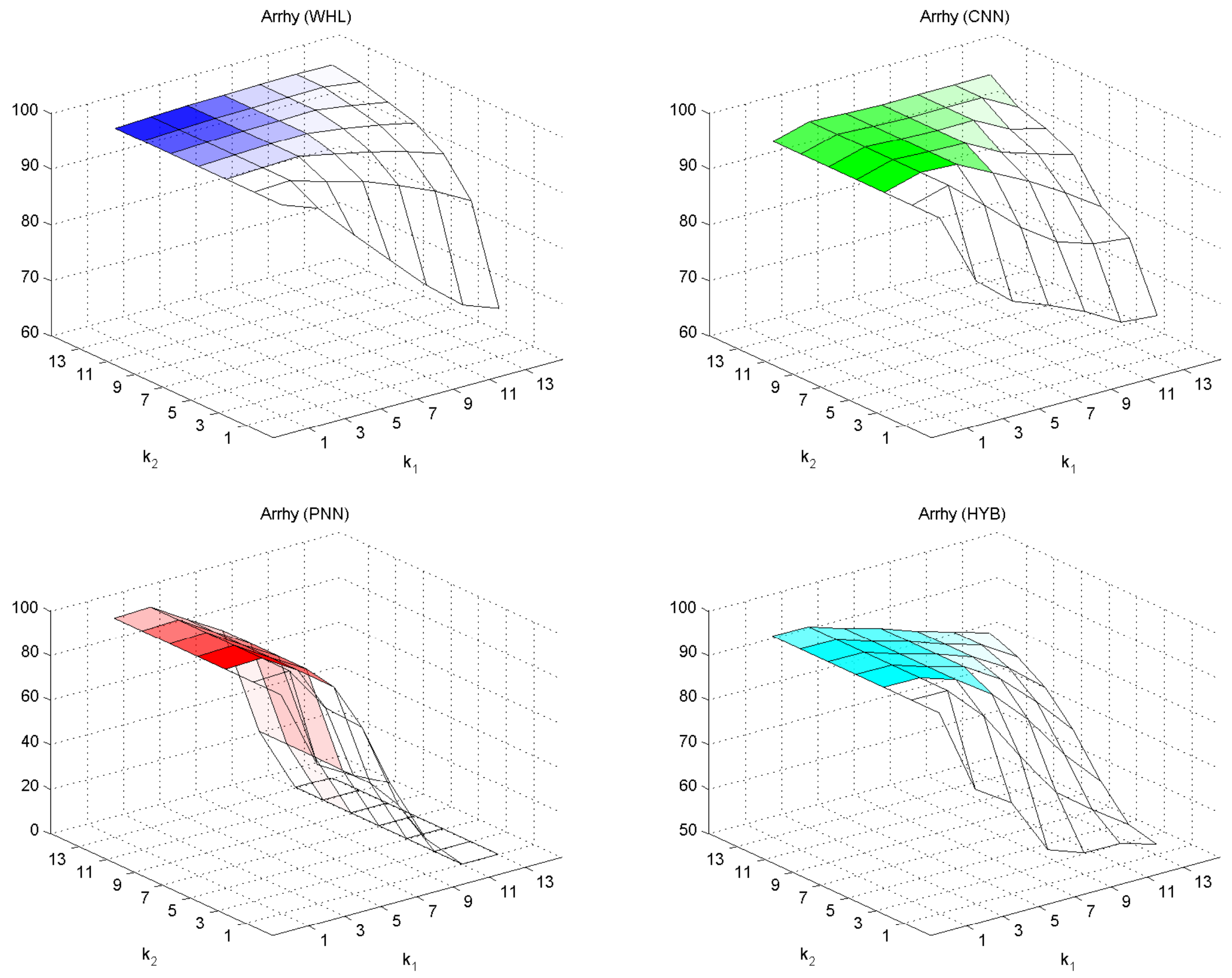

Figure 5: A comparison of the estimated error rates of *knnc* (where the cardinal numbers of the nearest neighbors, k_2's, are 1, 3, 5, 7, 9, 11, and 13 sample points), built in the feature spaces reconstructed with the LLR algorithm in which the cardinal numbers of the nearest neighbors, k_1's, are 1, 3, 5, 7, 9, 11, and 13 sample points. This is for the real-life database, Arrhy: (a) top left, (b) top right, (c) bottom left, and (d) bottom right. The figures (a) - (d) are obtained with the WHL, CNN, PNN, and HYB methods for Arrhy, respectively.

that the classification accuracies of *knnc* designed in the reconstructed feature space are *almost* the same when $k_1 = 1$, while the accuracies of *knnc* are different when $k_1 > 1$. However, from Figure 3 (a), (b), (c), and (d) of Non_l2 (whose cardinality is 500), it can be observed that the classification accuracies of *knnc* are *almost* the same for the entire range of k_1. In other words, the curved surfaces of the estimated error rates for the WHL, CNN, PNN, and HYB methods have almost identical shapes. We can also see a similar characteristic for the real-life data sets Ionos and Arrhy, as demonstrated in Figures 4 and 5. In the interest of brevity, a detailed description of these figures is omitted. The same characteristic can also be observed for the other databases, such as Non_n2, Non_l1, Iris2, and Sonar. Again, to avoid repetition, the details of the results for these databases is omitted.

In order to further investigate the characteristics of the two schemes, WHL and PRS_LLR, where the PRSs are CNN, PNN, and HYB, the experimental results were analyzed as shown in Tables 5 and 6 as well as Tables 7 and 8.

Tables 5 and 6 show the comparisons of classification accuracies (and $\pm$ standard deviations) (%) for the samples locally reconstructed with the artificial and real-life data sets and their prototypes extracted with the CNN, PNN, and HYB methods, respectively. On the other hand, Tables 7 and 8 show the comparisons of the processing CPU-times (and $\pm$ standard deviations) (seconds) required for the computations of the corresponding processes shown in Tables 5 and 6, respectively. In these tables, the entry "$-$" for the Non_n1, Non_l1, Iris2, and Arrhy data sets implies that classification was not performed for the reconstructed feature space of the k_1's values. The advantage of using PRS_LLR is clear and not re-iterated.

Apart from the results reported above, which demonstrates the advantage of resorting to a PRS prior to invoking a LLR, the most fascinating result is that for every data set there seems to be a specific pair of values k_1 and k_2 for which the classification is optimal. In other words, it is best to train the classifier with a k_1-NN classifier and test it with a k_2-NN classifier. The interesting point about this is that the value k_2 is not equal to k_1. Besides, the classification accuracy falls rather drastically if the testing classifier uses a k-NN rule, where $k > k_2$. This is definitely not obvious or intuitive.

We conclude this section by a note about the analysis of our scheme. Unfortunately, the theoretical analysis of this claim is open. The reason for this is that even though a "zillion" PRS methods have been reported (as Bezdek writes), we are not aware of a formal analysis of a single one that has been reported for more than three decades. The reason for the lack of analysis is probably because the problem could be inherently NP-hard. If we consider the PRS as a selection mechanism, the PRS reduces to being a scheme which chooses the best M of the N test samples, which we believe can be reduced to the graph partitioning problem. Thus, personally, we believe that such a formal analysis is not possible for all possible datasets, which is also probably why there is no single PRS which can be crowned to be the best scheme for all datasets.

Table 5: A comparison of the classification accuracies (and ± standard deviations) (%) for the samples locally reconstructed for the artificial data sets and their prototypes extracted with the CNN, PNN, and HYB methods, where each evaluation sample was reconstructed with the k_1 nearest neighbors of cardinalities *1, 3, 5, 7, 9, 11, 13.* The number in parenthesis in each entry represents the "order" k_2, of the corresponding "Testing" classifier, using which the respective accuracy was obtained.

Datasets	PRS	NN: k_1=1	NN: k_1=3	NN: k_1=5	NN: k_1=7	NN: k_1=9	NN: k_1=11	NN: k_1=13
Non_n1	WHL	93.60 (1) ±2.46	94.80 (3) ±2.35	94.20 (5) ±1.99	94.60 (7) ±2.12	94.80 (9) ±2.53	93.00 (11) ±2.87	94.60 (13) ±2.99
	CNN	93.60 (1) ±2.46	95.00 (3) ±1.70	73.60 (5) ±38.86	74.40 (7) ±39.31	57.60 (9) ±49.63	9.60 (11) ±30.36	– –
	PNN	93.40 (1) ±2.84	94.20 (3) ±3.71	73.80 (5) ±38.96	55.40 (7) ±47.84	28.00 (9) ±45.13	– –	– –
	HYB	96.00 (1) ±2.31	77.40 (3) ±40.81	58.00 (5) ±49.96	39.00 (7) ±50.35	– –	– –	– –
Non_n2	WHL	93.20 (1) ±0.78	93.34 (3) ±0.74	93.88 (5) ±0.73	93.44 (7) ±0.59	93.78 (9) ±1.09	93.32 (11) ±0.99	93.20 (11) ±0.67
	CNN	92.98 (1) ±0.75	93.34 (3) ±0.86	92.28 (5) ±0.83	91.96 (7) ±1.36	91.20 (9) ±1.72	91.16 (11) ±1.42	90.86 (13) ±1.64
	PNN	92.82 (1) ±0.98	93.10 (3) ±1.10	92.60 (5) ±1.39	92.12 (7) ±1.54	91.88 (9) ±1.63	92.02 (11) ±1.56	91.70 (13) ±1.86
	HYB	91.20 (1) ±1.76	91.28 (3) ±1.82	90.90 (5) ±1.85	90.82 (7) ±2.08	90.70 (9) ±1.55	90.74 (11) ±1.84	90.24 (13) ±1.85
Non_l1	WHL	89.00 (1) ±2.87	89.40 (3) ±3.13	88.40 (5) ±3.24	86.80 (7) ±4.54	86.00(9) ±4.42	82.40 (11) ±3.75	82.40 (13) ±6.65
	CNN	87.00 (1) ±3.43	85.80 (3) ±3.05	84.40 (5) ±4.09	80.20 (7) ±4.76	80.00 (9) ±5.42	78.20 (11) ±4.76	61.00 (13) ±32.66
	PNN	85.20 (1) ±3.55	84.40 (3) ±4.40	79.60 (5) ±5.06	77.40 (7) ±4.99	76.80 (9) ±5.35	74.80 (11) ±10.42	46.00 (13) ±39.90
	HYB	80.60 (1) ±9.24	76.60 (3) ±8.85	69.00 (5) ±7.32	54.40 (7) ±30.53	47.80 (9) ±33.60	– –	– –
Non_l2	WHL	89.50 (1) ±0.63	89.86 (3) ±0.90	90.08 (5) ±0.74	90.50 (7) ±1.25	90.12 (9) ±1.24	89.80 (9) ±0.82	90.08 (13) ±1.03
	CNN	88.42 (1) ±0.71	88.80 (3) ±1.21	88.32 (5) ±1.34	87.48 (7) ±1.45	86.88 (9) ±1.96	86.32 (11) ±2.48	85.36 (13) ±2.07
	PNN	87.32 (1) ±0.54	87.44 (3) ±0.82	87.76 (5) ±0.74	86.80 (7) ±1.29	86.26 (9) ±1.07	85.70 (11) ±1.59	85.04 (13) ±1.72
	HYB	89.30 (1) ±1.23	89.04 (3) ±0.98	88.64 (5) ±1.48	87.36 (7) ±1.47	85.96 (9) ±2.05	84.28 (11) ±2.84	82.96 (13) ±3.34

Table 6: A comparison of the classification accuracies (± standard deviations) (%) for the samples locally reconstructed for the four real-life data sets and their prototypes extracted with the CNN, PNN, and HYB methods, where each evaluation sample was reconstructed with the nearest neighbors of cardinalities *1, 3, 5, 7, 9, 11, 13*. The number in parenthesis in each entry represents the "order" k_2, of the corresponding "Testing" classifier, using which the respective accuracy was obtained.

Datasets	PRS	NN: k_1=1	NN: k_1=3	NN: k_1=5	NN: k_1=7	NN: k_1=9	NN: k_1=11	NN: k_1=13
	WHL	92.60 (1)	92.60 (3)	92.60 (5)	92.80 (7)	92.60 (9)	93.20 (11)	93.40 (13)
		±1.65	±0.97	±1.65	±2.35	±3.27	±3.16	±3.27
Iris2	CNN	90.20 (1)	90.80 (3)	90.20 (5)	90.40 (7)	81.20 (9)	81.40 (11)	63.00 (11)
		±2.90	±2.53	±5.12	±4.70	±28.89	±29.03	±43.59
	PNN	90.80 (1)	92.00 (3)	91.20 (5)	81.40 (7)	72.40 (9)	63.60 (11)	9.40 (13)
		±2.53	±1.89	±2.86	±28.72	±38.25	±43.97	±29.73
	HYB	91.40 (1)	92.40 (3)	82.00 (5)	27.60 (7)	8.80 (7)	–	–
		±1.90	±2.27	±28.94	±44.45	±27.20	–	–
	WHL	86.19 (1)	86.42 (3)	87.16 (5)	87.67 (7)	87.90 (9)	87.90 (11)	88.07 (13)
		±2.11	±2.13	±1.94	±1.78	±1.74	±2.06	±1.65
Ionos	CNN	86.08 (1)	87.67 (3)	89.49 (5)	90.28 (7)	89.89 (9)	89.60 (11)	90.34 (13)
		±1.82	±1.74	±1.40	±1.90	±2.29	±3.43	±3.44
	PNN	87.56 (1)	88.24 (3)	90.34 (5)	91.02 (7)	91.08 (9)	89.94 (11)	90.00 (13)
		±1.90	±2.46	±1.54	±1.36	±1.78	±2.64	±2.46
	HYB	84.94 (1)	83.47 (3)	80.68 (5)	79.32 (7)	77.67 (9)	76.81 (11)	75.97 (13)
		±1.18	±3.69	±2.13	±3.32	±3.17	±3.44	±3.93
	WHL	78.25 (1)	78.74 (3)	79.03 (5)	79.32 (7)	80.78 (9)	80.97 (11)	81.17 (13)
		±4.37	±4.47	±4.49	±4.71	±4.71	±4.07	±4.39
Sonar	CNN	76.12 (1)	76.60 (3)	78.16 (5)	78.84 (7)	79.32 (9)	80.19 (11)	79.61 (13)
		±5.16	±3.53	±4.63	±4.99	±4.91	±5.48	±5.14
	PNN	75.24 (1)	75.34 (3)	76.99 (5)	76.41 (7)	76.89 (9)	76.70 (11)	76.99 (13)
		±3.46	±6.18	±4.95	±4.84	±4.66	±4.67	±5.57
	HYB	68.84 (1)	69.51 (3)	68.64 (5)	68.84 (7)	69.13 (9)	69.32 (11)	69.90 (13)
		±4.42	±5.38	±5.42	±5.42	±4.29	±4.67	±4.80
	WHL	97.69 (1)	98.00 (3)	98.27 (5)	98.36 (7)	98.53 (9)	98.62 (11)	98.53 (13)
		±0.66	±0.84	±0.82	±0.63	±0.79	±0.74	±0.89
Arrhy	CNN	95.33 (1)	97.16 (3)	96.84 (5)	96.62 (7)	96.71 (9)	96.71 (11)	96.80 (11)
		±1.42	±1.67	±2.01	±1.96	±2.19	±2.41	±2.36
	PNN	97.73 (1)	98.36 (3)	88.49 (5)	78.98 (7)	29.56 (9)	–	–
		±0.61	±0.66	±31.10	±41.63	±47.59	–	–
	HYB	94.84 (1)	94.76 (3)	92.13 (5)	89.96 (7)	87.11 (9)	84.80 (11)	82.40 (13)
		±2.27	±2.21	±3.03	±3.80	±5.33	±5.78	±6.47

Table 7: A comparison of the processing CPU-times (and ± standard deviations) (seconds) required for the samples locally reconstructed for the artificial data sets and their prototypes. Here, the prototypes were extracted with the CNN, PNN, and HYB methods, respectively. Thereafter, each evaluation sample was reconstructed with the nearest neighbors of cardinalities *1, 3, 5, 7, 9, 11, 13*.

Datasets	PRS	NN: k_1=1	NN: k_1=3	NN: k_1=5	NN: k_1=7	NN: k_1=9	NN: k_1=11	NN: k_1=13
	WHL	1.15	1.19	1.28	1.36	1.47	1.50	1.64
		±0.04	±0.03	±0.04	±0.05	±0.08	±0.06	±0.05
Non_n1	CNN	1.01	1.06	0.90	0.93	0.74	0.14	–
		±0.07	±0.09	±0.47	±0.49	±0.64	±0.44	–
	PNN	0.97	1.02	0.89	0.71	0.37	–	–
		±0.06	±0.06	±0.47	±0.61	±0.60	–	–
	HYB	0.94	0.81	0.64	0.45	–	–	–
		±0.07	±0.43	±0.55	±0.58	–	–	–
	WHL	11.91	12.22	13.12	13.96	15.51	16.03	17.31
		±0.05	±0.13	±0.21	±0.19	±0.31	±0.30	±0.40
Non_n2	CNN	11.19	11.46	12.33	12.93	13.42	14.07	14.56
		±0.03	±0.13	±0.24	±0.32	±0.17	±0.33	±0.32
	PNN	11.30	11.64	12.39	13.41	14.29	14.73	15.56
		±0.14	±0.26	±0.22	±0.27	±0.70	±0.48	±0.88
	HYB	11.21	11.53	12.24	12.96	13.67	14.16	14.91
		±0.07	±0.14	±0.16	±0.22	±0.20	±0.25	±0.31
	WHL	1.14	1.15	1.20	1.27	1.40	1.51	1.63
		±0.01	±0.00	±0.02	±0.03	±0.05	±0.07	±0.09
Non_l1	CNN	1.04	1.06	1.09	1.17	1.23	1.31	1.14
		±0.01	±0.02	±0.02	±0.04	±0.04	±0.06	±0.61
	PNN	1.04	1.06	1.10	1.18	1.26	1.35	0.91
		±0.01	±0.01	±0.02	±0.03	±0.07	±0.06	±0.78
	HYB	1.00	1.01	1.07	0.95	0.90	–	–
		±0.03	±0.03	±0.05	±0.51	±0.63	–	–
	WHL	11.65	11.67	12.25	13.51	14.97	16.31	18.06
		±0.02	±0.03	±0.11	±0.20	±0.29	±0.21	±0.31
Non_l2	CNN	11.13	11.33	11.90	12.66	13.57	14.34	15.48
		±0.06	±0.06	±0.06	±0.12	±0.24	±0.37	±0.47
	PNN	11.12	11.26	11.99	12.76	13.91	14.81	16.20
		±0.05	±0.03	±0.10	±0.15	±0.27	±0.16	±0.33
	HYB	11.14	11.32	11.97	12.72	13.68	14.51	15.75
		±0.13	±0.11	±0.19	±0.24	±0.35	±0.39	±0.51

Table 8: A comparison of the processing CPU-times (and ± standard deviations) (seconds) required for the samples locally reconstructed for the four real-life data sets and their prototypes. Here, the prototypes were extracted with the CNN, PNN, and HYB methods, respectively. Thereafter, each evaluation sample was reconstructed with the nearest neighbors of cardinalities *1, 3, 5, 7, 9, 11, 13*.

Datasets	PRS	NN: k_1=1	NN: k_1=3	NN: k_1=5	NN: k_1=7	NN: k_1=9	NN: k_1=11	NN: k_1=13
	WHL	1.20	1.22	1.24	1.28	1.33	1.39	1.48
		±0.05	±0.04	±0.05	±0.05	±0.05	±0.05	±0.06
Iris2	CNN	1.15	1.18	1.19	1.24	1.16	1.19	0.97
		±0.02	±0.02	±0.02	±0.01	±0.41	±0.42	±0.67
	PNN	1.15	1.17	1.18	1.10	1.00	0.91	0.14
		±0.03	±0.04	±0.02	±0.39	±0.53	±0.63	±0.43
	HYB	1.11	1.13	1.05	0.36	0.13	–	–
		±0.03	±0.02	±0.37	±0.58	±0.40	–	–
	WHL	4.27	4.33	4.34	4.42	4.51	4.58	4.71
		±0.04	±0.05	±0.02	±0.05	±0.05	±0.05	±0.05
Ionos	CNN	4.17	4.22	4.27	4.33	4.44	4.56	4.64
		±0.02	±0.02	±0.03	±0.02	±0.04	±0.03	±0.05
	PNN	4.18	4.21	4.26	4.33	4.41	4.55	4.66
		±0.02	±0.01	±0.02	±0.03	±0.04	±0.04	±0.02
	HYB	4.19	4.54	4.39	4.40	4.50	4.65	4.77
		±0.03	±0.60	±0.17	±0.10	±0.19	±0.17	±0.22
	WHL	2.48	2.50	2.53	2.58	2.60	2.67	2.72
		±0.01	±0.01	±0.02	±0.03	±0.02	±0.04	±0.04
Sonar	CNN	2.40	2.45	2.49	2.52	2.56	2.61	2.67
		±0.01	±0.02	±0.02	±0.03	±0.02	±0.02	±0.02
	PNN	2.40	2.47	2.51	2.53	2.59	2.63	2.70
		±0.01	±0.03	±0.03	±0.01	±0.03	±0.03	±0.02
	HYB	2.42	2.49	2.50	2.54	2.61	2.67	2.74
		±0.03	±0.07	±0.03	±0.02	±0.03	±0.04	±0.04
	WHL	5.51	5.53	5.58	5.64	5.72	5.82	5.97
		±0.03	±0.02	±0.02	±0.05	±0.02	±0.05	±0.02
Arrhy	CNN	5.34	5.39	5.41	5.49	5.65	5.74	5.87
		±0.04	±0.03	±0.01	±0.04	±0.05	±0.02	±0.03
	PNN	5.30	5.39	4.92	4.46	1.71	–	–
		±0.03	±0.02	±1.73	±2.35	±2.75	–	–
	HYB	5.36	5.40	5.48	5.58	5.75	5.84	5.99
		±0.03	±0.02	±0.04	±0.02	±0.04	±0.03	±0.05

5 Conclusions

In this paper, we have considered how we can use the principles of Prototype Reduction Schemes (PRSs) to optimize the computations involved in the well-known families of k-Nearest Neighbor (k-NN) rules. Although k-NN rules have been extensively studied, recently, an implementation of the k-NN, named as the Locally Linear Reconstruction (LLR) [2], which invokes a quadratic optimization process, has been proposed. The latter method is capable of systematically setting model parameters, such as the number of neighbors (k) and the weights. Our aim, in this paper, was to optimize the computation time required for the LLR by using a PRS. We have proposed a strategy of using a PRS to efficiently compute the optimization problem. We have demonstrated that by completely discarding the points not included by the PRS, we can obtain a reduced set of sample points, using which, in turn, the quadratic optimization problem can be computed. The accuracies of the proposed method is comparable to those obtained with the original training set (i.e., the one which considers all the data points) even though the computations required are noticeably less (the proposed method sometimes requiring only about 50% of the time). The proposed method has been tested on artificial and real-life data sets, and the results obtained are quite promising, and could have potential in PR applications.

An avenue for further research involves developing alternate stochastic learning methods by which the query sample can be estimated accurately and quickly when only the prototype set is considered. We are currently investigating how this can be achieved.

References

[1] A. K. Jain, R. P. W. Duin and J. Mao, "Statistical pattern recognition: A review", *IEEE Trans. Pattern Anal. and Machine Intell.*, vol. PAMI-22, no. 1, pp. 4 - 37, Jan. 2000.

[2] P. Kang and S. Cho, "Locally linear reconstruction for instance-based learning", *Pattern Recognition*, vol. 41, pp. 3507–3518, 2008.

[3] S. T. Roweis and L.K.Saul, "Nonlinear dimensionality reduction by locally linear embedding", *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[4] S. T. Roweis and L.K.Saul, "Think globally, fit locally: unsupervised learning of nonlinear manifolds", *Journal of Machine Learning Research*, vol. 4, pp. 119–155, Jun. 2003.

[5] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning", *Artifical Intelligence Review*, vol. 11, no. 5, pp. 11–73, 1997.

[6] T. Liu, A. Moore, A. Gray, "Efficient exact k-NN and nonparametric classification in high dimensions", *Proc. of Neural Information Processing Systems*, (2003).

[7] J. C. Bezdek and L. I. Kuncheva, “Nearest prototype classifier designs: An experimental study”, *International Journal of Intelligent Systems*, vol. 16, no. 12, pp. 1445 - 11473, 2001.

[8] B. V. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, Los Alamitos, 1991.

[9] P. E. Hart, “The condensed nearest neighbor rule”, *IEEE Trans. Inform. Theory*, vol. IT-14, pp. 515 - 516, May 1968.

[10] C. L. Chang, “Finding prototypes for nearest neighbor classifiers”, *IEEE Trans. Computers*, vol. C-23, no. 11, pp. 1179 - 1184, Nov. 1974.

[11] P. A. Devijver and J. Kittler, “On the edited nearest neighbor rule”, *Proc. 5th Int. Conf. on Pattern Recognition*, Miami, Florida, pp. 72 - 80, Dec. 1980.

[12] G. L. Ritter, H. B. Woodruff, S. R. Lowry and T. L. Isenhour, “An algorithm for a selective nearest neighbor rule”, *IEEE Trans. Inform. Theory*, vol. IT-21, pp. 665 - 669, Nov. 1975.

[13] I. Tomek, “Two modifcations of CNN”, *IEEE Trans. Syst., Man and Cybern.*, vol. SMC-6, no. 6, pp. 769 - 772, Nov. 1976.

[14] Q. Xie, C.A. Laszlo and R. K. Ward, “Vector quantization techniques for nonparametric classifier design”, *IEEE Trans. Pattern Anal. and Machine Intell.*, vol. PAMI-15, no. 12, pp. 1326 - 1330, Dec. 1993.

[15] K. Fukunaga, *Introduction to Statistical Pattern Recognition, Second Edition*, Academic Press, San Diego, 1990.

[16] K. Fukunaga and J. M. Mantock, “Nonparametric data reduction”, *IEEE Trans. Pattern Anal. and Machine Intell.*, vol. PAMI-6, no. 1, pp. 115 - 118, Jan. 1984.

[17] C. J. C. Burges, “A tutorial on support vector machines for pattern recognition”, *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121 - 167, 1998.

[18] X. -Y. Jing, H. -S. Wong, D. Zhang, “Face recognition based on discriminant fractional Fourier feature extraction”, *Pattern Recognition Lett.*, vol. 27, no. 13, pp. 1465–1471, 2006.

[19] S. Jiang, X. Song, H. Wang, J.-J. Han, Q.-H. Li, “A clustering-based method for unsupervised intrusion detections”, *Pattern Recognition Lett.*, vol. 27, no. 7, pp. 802–810, 2006.

[20] S. -W. Kim and B. J. Oommen, ”Enhancing prototype reduction schemes with LVQ3-type algorithms”, *Pattern Recognition*, vol. 36, no. 5, pp. 1083 - 1093, 2003.

[21] S. -W. Kim and B. J. Oommen, "A Brief Taxonomy and Ranking of Creative Prototype Reduction Schemes", *Pattern Analysis and Applications Journal*, vol. 6, no. 3, pp. 232 - 244, December 2003.

[22] S. -W. Kim and B. J. Oommen, "Enhancing Prototype Reduction Schemes with Recursion : A Method Applicable for "Large" Data Sets", *IEEE Trans. Systems, Man, and Cybernetics - Part B*, vol. SMC-34, no. 3, pp. 1384 - 1397, June 2004.

[23] S. -W. Kim and B. J. Oommen, "On using prototype reduction schemes to optimize kernel-based nonlinear subspace methods", *Pattern Recognition*, vol. 37, no. 2, pp. 227 - 239, 2004.

[24] S. -W. Kim and B. J. Oommen, "On using prototype reduction schemes and classifier fusion strategies to optimize kernel-based nonlinear subspace methods", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 3, pp. 455 - 460, March 2005.

[25] C. L. Blake and C. J. Merz, *UCL Machine Learning Databases*, Irvine, CA: University of California, Department of Information and Computer Science. Can also be downloaded (as of June 2005) from http://www.ics.uci.edu/mlearn/MLRepository.html.

[26] T. M. Mitchell, *Machine Learning*, McGraw-Hill, Singapore, 1997.

[27] M. O'mahony, N. Hurley, N. Kushmerick, G. Silvestre, "Collaborative recommendation: a robustness analysis", *ACM Trans. Internet Technol.*, vol. 4, no. 4, pp. 344–377, 2003.

[28] R. P. W. Duin, P. Juszczak, D. de Ridder, P. Paclik, E. Pekalska, and D. M. J. Tax: *PRTools 4: a Matlab Toolbox for Pattern Recognition.* Technical Report, Delft University of Technology, The Netherlands, (2004). Can also be available at http://prtools.org/.

[29] R. -E. Fan, P.-H. Chen, C. -J. Lin: Working set selection using the second order information for training SVM. *Journal of Machine Learning Research*, 6 1889–1918 (2005). Can also be referred (as of July 2010) to http://www.csie.ntu.edu.tw/∼cjlin/libsvm.