Fast Implementations of Nearest Neighbor Classifiers

P. J. Grother, G. T. Candela and J. L. Blue

***Abstract*— Standard implementations of non-parametric classifiers have large computational requirements. Parzen classifiers use the distances of an unknown vector to all N prototype samples, and consequently exhibit O(N) behavior in both memory and time. We describe four techniques for expediting the nearest neighbor methods. replacing the linear search with a new *kd* tree method, exhibiting approximately O($N^{1/2}$) behavior; employing an L_∞ instead of L_2 distance metric; using variance ordered features; and rejecting prototypes by evaluating distances in low dimensionality subspaces. We demonstrate that variance-ordered features yield significant efficiency gains over the same features linearly transformed to have uniform variance. We qive results for a large OCR problem, but note that the techniques expedite recognition for arbitrary applications. Three of four techniques preserve recognition accuracy.**

***Index Terms*— Non-parametric classifiers, *kd* trees, k Nearest Neighbors, Distance metrics, Karhunen-Loève transform, OCR.**

I. Introduction

Statistical classifiers for OCR have been widely investigated. Using Karhunen-Loève (KL) transforms of normalized binary images it has been found that the non-parametric classifiers work better than many commonly used neural networks [1]. Indeed the simplicity and efficacy of the KNN method [2] has made it the algorithm of choice for prototyping pattern recognition tasks. However non-parametric classifiers in on-line recognition systems are expensive: the space and speed requirements rise linearly with the number of known samples, and that number is often necessarily large. Although extensive literature on faster methods exists, for example the papers in Dasarathy's volume [3], speed and memory requirements have historically mitigated against the practical use of non-parametric classifiers. However older research [4] recently applied to high dimensional searches has shown the aversion to non-parametric to often have been unjustified. The techniques we describe are unrelated to the traditional and very effective methods of condensing the sample sets (through deletion [5] or clustering [6]), which may still be applied in conjunction with our procedures.

Given that fast parametric classifiers, and neural networks can obtain very good classification results, the persistence of the neighbor methods is, in our case, due to their superior error rate performance and their simplicity. Thus this investigation details several methods of improving efficiency while preserving recognition performance. Section II.compares the KNN and PNN algorithms, while section III.describes their efficient implementation. Such methods confer a data dependency to the recognition speed, and section IV.discusses optimally efficient feature representations. Section V.describes the *kd* tree, and its use with KNN and PNN. The paper finishes with sections giving results for the various factors when applied to an OCR problem and the final conclusions.

II. Non-parametric Classification

The nearest neighbor methods employ n-dimensional feature vectors from N samples of known identity. Non-parametric classifiers usually use some function of the distances of an unknown to the known points. For practical OCR the number of training samples may exceed 10^6, each being represented in, say, a 40-dimensional space. An unknown, $\mathbf{y}$, is assigned that class whose discriminant value, $D_i(\mathbf{y})$, is maximum. Given $\mathbf{x}_j^{(i)}$ as the $N^{(i)}$ prototypical sample vectors of class i, the Parzen [7] discriminant value is

$$D_i(\mathbf{y}) = \frac{1}{N^{(i)}} \sum_{j=1}^{N^{(i)}} \kappa_i(\mathbf{y} - \mathbf{x}_j^{(i)})$$

where the form of the kernel function, κ, defines a non-parametric classification method and may be different for each class. The KNN and PNN algorithms considered in this paper use both the L_2 Euclidean squared distance and the L_∞ maximum metrics defined thus:

$$d(\mathbf{y}, \mathbf{x}) = (\mathbf{y} - \mathbf{x})^T (\mathbf{y} - \mathbf{x})$$

$$d(\mathbf{y}, \mathbf{x}) = \max_j |\mathbf{y}_j - \mathbf{x}_j| .$$

The Euclidean metric has useful analytic properties, and turns out to produce more accurate results. The latter metric is more efficient.

A. KNN and PNN

The KNN algorithms consider only the k nearest neighbors, denoted by $\mathbf{c}_1, \cdots, \mathbf{c}_k$, as members of the set

$$\mathcal{S}_u = \{\mathbf{x}_j |\, d(\mathbf{y}, \mathbf{x}_j) \;\leq\; d(\mathbf{y}, \mathbf{c}_k)\} \tag{1}$$

where the subscript k denotes the k^{th} closest prototype. If the set is composed of $k_i \geq 0$ elements from each class i, then the traditional method classifies using $D_i(\mathbf{y}) = k_i$ so that the unknown takes the majority class. This algorithm can be regarded as the Parzen classifier with a top hat function kernel whose width is a random variable, and different for each class. However we prefer a distance-dependent continuous discriminant value offering superior classification and the possibility of low-confidence rejection. The discriminant function is

$$D_i(\mathbf{y}) = \sum_{\mathbf{x}_j^{(i)} \in \mathcal{S}_u} \frac{1}{1 + d(\mathbf{y}, \mathbf{x}_j^{(i)})} \tag{2}$$

PNN originated in the neural network literature [8] as a parallel implementation of a Parzen type classifier. Our implementation uses a radially symmetric kernel of width σ which most simply is fixed for all classes. Although class specific σ_i's potentially offer superior performance, our investigation of applying a conjugate gradient algorithm to minimize a mean square error function has yielded minimal classification gains for our hand print and machine print applications. The discriminant function is

$$D_i(\mathbf{y}) = \frac{1}{N^{(i)}} \sum_{j=1}^{N^{(i)}} \exp\left(\frac{-1}{2\sigma^2}\; d(\mathbf{y}, \mathbf{x}_j^{(i)})\right) \tag{3}$$

For squared L_2 distances the kernel is normal and if the discriminant values for each of the L classes are normalized by their sum, they become estimates of the *a posteriori* probabilities. The algorithm is slow due to the exponential: the many distant prototypes have negligible contribution. Thus the discriminant may be approximated by ignoring terms less than $e^{-\alpha}$ times the largest. If all the discriminant functions are accumulated, the j^{th} prototype contributing to *any* $D_i(\mathbf{y})$ is retained if it satisfies this condition:

$$\exp\left(\frac{-1}{2\sigma^2}\; d(\mathbf{y}, \mathbf{x}_j)\right) \;<\; \exp(-\alpha) \exp\left(\frac{-1}{2\sigma^2}\; d(\mathbf{y}, \mathbf{c}_1)\right) \tag{4}$$

where $\mathbf{c}_1$ is the closest prototype. This is most economically used in the distance domain whence the voting set is:

$$\mathcal{S}_u = \left\{\mathbf{x}_j |\, d(\mathbf{y}, \mathbf{x}_j) \;<\; 2\alpha\sigma^2 \;+\; d(\mathbf{y}, \mathbf{c}_1)\right\} \tag{5}$$

This approximates pure PNN with an error controlled by α, a lower bound of which is the log of the number of samples per class. Pathologically this number of only-just-ineligible samples could cause the classification to be changed. The α value used in the public-domain [9] implementation is set to 9 ensuring no hypothesis changes, yet still affording a five-fold speed increase over the naive algorithm. Small α yields speed and any accompanying loss in recognition is limited, since, as $\alpha \to 0$, PNN reverts to KNN with $k = 1$. Although the PNN and KNN discriminants have different form, hypotheses are identical and error versus reject performance is comparable. The KNN method with $k = 1$ defines the lower time *and* error rate bounds for this PNN variant as $\alpha \to 0$. As $\alpha \to \infty$ PNN defaults to its slow definition. Since the number of samples local to $\mathbf{y}$ is a random variable, PNN potentially has a recognition advantage over KNN.

III. Implementations

It is not necessary to calculate all the distances to an unknown in order to find the nearest prototype. Rather it is sufficient, during the computation of the distances, to retain only those prototypes that survive the membership conditions in equations 1 or 5. The distance metrics in section II. nominally require complete (that is, using all n feature components) calculation of dot products (L_2) and maxima (L_∞). However for KNN, once the distance has been found for the first k prototypes, it is possible to discard a subsequent prototype after *any* $n_d \leq n$ elements of the dot product or the maxima have been used; if the distance already exceeds the distance to the current k^{th} closest sample, then the prototype is too distant and $n - n_d$ elements are ignored. If the prototype is ultimately nearer than the k^{th} nearest neighbor then it is retained and the current $\mathbf{c}_k$ is discarded. The situation for PNN is analogous; any $n_d \leq n$ elements that put a prototype beyond the perimeter defined in equation 5 are sufficient to disqualify it. The use of a prototype according to these criteria is temporary; the eligibility is fixed only at the end of the search when $\mathbf{c}_1$ has been found. Since any of the k currently closest distances can only decrease during the search, prototypes that become ineligible remain permanently so.

The order in which the feature components are considered is irrelevant to the classification, but not to the computation time. For either metric the selection order can be arbitrary so that it is possible, without loss of generality, to use the elements in order of largest expected contribution to the distance calculation. For the L_2 metric let d_i^2 be the square of the i^{th} component of the difference vector $\mathbf{x}_j - \mathbf{y}$. Its expectation for $\mathbf{y}$ over all the training samples is

$$E_j(d_i^2) = E_j(x_i^2) - 2y_i E_j(x_i) + y_i^2$$

If the prototype set has zero mean, the second term disappears and the first term is just the variance. If further the testing vectors, $\mathbf{y}$, are identically distributed the expectation of the contribution of the i^{th} feature component to the distance sum is

$$E(d_i^2) = var(x_i) + var(y_i) = 2var(x_i)$$

That is compaction of feature variance increases the expected partial distance accumulations and thereby expedites

eligibility determination. This holds for arbitrary recognition applications. The KL transform has some powerful properties in this respect: if the N zero-mean sample vectors form the columns of the sample matrix $\mathbf{U}$, and the covariance $N^{-1}\mathbf{U}\mathbf{U}^T$ is diagonalized by eigenvectors Ψ, then the KL transforms are $\mathbf{V} = \Psi^T\mathbf{U}$. The covariance of these extracted features is

$$N^{-1}\mathbf{V}\mathbf{V}^T = N^{-1}\Psi^T\mathbf{U}\mathbf{U}^T\Psi = \Lambda$$

where the only non-zero elements of the Λ are the eigenvalues on its diagonal, usually obtained in decreasing order. Of all unitary transforms the KL transform optimally compacts variance into the leading features. These therefore contribute most, on average, to the distance sums, and yield briefest computation.

Given these details the methods retain their linear search character since all prototypes are considered and the algorithm still scales as $O(N)$. The naive implementations are also $O(n)$ with respect to dimensionality and, although these optimizations give significant data dependency, the speed remains closely linear in n.

IV. Variance Equalization

Variance ordering of the features (obtained by design, through KL transformation, direct sorting, or other methods) will improve the efficiency of nearest neighbor methods applied to arbitrary tasks. The overhead associated with any rearrangement of the data may be zero or quite considerable. The KL transform may be costly to the whole process, because, in the worst (complete) case, the expansion involves an $O(n^2)$ matrix-vector operation for each unknown. The Cosine and Discrete Fourier transforms approach the KL transform in terms of energy compaction and may be calculated using fast $O(n \log n)$ algorithms. In our applications the KL transforms *are* the features of our images; we have not contrived their use, we incur no overhead, and expedited classification is a fortunate by-product. Depending on the application, therefore, variance ordering maybe be beneficial to the whole process. To quantify the gains that are available, we compare the variance-ordered situation to the "average case" where the variances of the features are the same. To this end we transform our original features, $\mathbf{V}$, to a set, $\mathbf{Z}$, by applying a linear transformation, Φ. The new set $\mathbf{Z} = \Phi^T\mathbf{V}$ is obtained. The covariance matrix of these new features is

$$N^{-1}\mathbf{Z}\mathbf{Z}^T = N^{-1}\Phi^T\mathbf{V}\mathbf{V}^T\Phi = \Phi^T\Lambda\Phi$$

where Λ is the diagonal eigenvalue matrix of the original V. The new covariance is not diagonal but its diagonal elements should all be equal to the mean of the original KL variances, $\bar{\lambda} = n^{-1}Tr(\Lambda)$. The simplest transformation achieving this is feature whitening using $\Phi = \bar{\lambda}^{1/2}\Lambda^{-1/2}$ but as distances are not invariant under this transformation KNN and PNN classify very poorly. This is avoided if we require the new basis Φ to be complete and orthonormal in which case the basis corresponds to a subtle rotation of the coordinate axes. Note that the above discussion does not apply to the L_∞ metric.

The following algorithm makes the columns of Φ sequentially. The first column is made by forming a random vector and normalizing each trial version of it to unit length. Subsequent columns are made by again generating a random vector and employing Gram-Schmidt to produce a new version of unit length and orthogonal to all previous columns. The vectors are made speculatively and although orthonormal by design, they must satisfy the intended equal variance condition, namely that the variance of the component each extracts from the original features, calculated using

$$var(z_i) = N^{-1}\phi_i^T\mathbf{V}\mathbf{V}^T\phi_i = \lambda_i\phi_i^T\phi_i,$$

must not differ from the goal variance, $\bar{\lambda}$, by more than a specified tolerance ϵ. If this is true then the basis vector is retained and the algorithm continues, otherwise another trial vector is generated and tested. The smaller the ϵ, the closer the variances of the new features will be to all being equal, but the longer it will take to produce the transform. The algorithm is primitive; for $\epsilon = 10^{-3}$, $\bar{\lambda} = 10.5$, a complete $n = 40$ basis was made using about 10^5 trial vectors and several cpu minutes.

V. *kd* Trees

Although there are large performance gains associated with the above implementation of the NN methods, the algorithms still have the character of a linear search methods. Alternatives exist; among the best are the "*kd* tree" methods [10], which often have average searching time proportional to $\log N$ [4]. However this excellent behavior is usually only realized when the sample points in the feature space are dense, $N \gg 2^n$. In our OCR example described later, the dimensionality is high so that $N \ll 2^n$, and the logarithmic behavior is not found. A brief description of our implementation, which is a slight variation on the original *kd* tree method, follows; details of the original may be found in [4].

Construction of the *kd* tree is done recursively. The top node contains all N points. The two children of this node each contain $N/2$ points. The left child node contains those points whose value of the t^{th} component is less than the global median; the right child node has the remainder. The order in which components are considered, $t = t_1, t_2, \cdots$, is determined for each subtree from the variance of the features contained therein. Each child node is then divided in half using the median of the component with the largest variance of the points in that subtree, and so on. The depth of the tree would therefore be $log_2 N$, which for our application is less than the value of n that gives optimal digit classification. However, the recursive building of the tree is limited by stipulating a minimum number of points per leaf node, m. This is implemented by stopping tree division when the number of points contained in a subtree is less than $2m$.

Construction of the tree takes time proportional to $NlogN$ and is generally inexpensive. A simpler alternative is to build the tree by splitting each node using the median of the next component in order, $t = 1, 2, \cdots, n$. In this case, global decreasing variance order, as obtained, for example, with the KL transform, will be important. However, we have found the preferred method of using variance ordered node division to give 25% greater speed in the classification phase.

Searching for k nearest neighbors in the kd tree achieves speed because calculating distances for entire sub-trees can be avoided. The method is applicable with both L_2 and L_∞ distances. In kd trees, rather than searching for the k closest points, it is more natural to search for points within distance δ of the unknown vector, $\mathbf{y}$, as follows. Start at the top node and suppose $y_{t_1} < x_{t_1}$ where the subscript t_1 indicates on which component the node was originally split. Put the left child node on a list to look at later. All the points in the right child node are at least $x_{t_1} - y_{t_1}$ distant from $\mathbf{y}$. If $x_{t_1} - y_{t_1} > \delta$, ignore the right child node; otherwise put it on the list. Continue searching by taking one node at a time off the list. If the node has no children, look at the distances of the point or points in the node and remember the k smallest distances. If the node has children, look at both child nodes and put one or both of them on the list.

If δ is excessively large, too few subtrees will be discarded and too many distances calculated, leading to a long search time; with δ too small, too many subtrees will be discarded and too few nearest neighbors will be found, though this calculation is fast. A reasonable approach is to estimate, δ, preferably on the small side. If fewer than k nearest neighbors are found, δ is doubled and a new search is made. During the search, after k distances less than δ have been calculated, δ can be reduced to the k^{th} largest distance, giving efficiency gains. To initialize δ, the centroid vectors of each prototype class can be used as trial points and their distances to the unknown calculated. If a fraction, say 0.5, of the smallest such distance is used as an estimate for δ, then the k near neighbors may be found in the search. However if less than the mandated k neighbors are ultimately found, the search is restarted. This effect is clearly data dependent and is determined by the distance distributions. Thus search time is a sensitive function of the initial δ. We report results for a simpler implementation; we initially use $\delta = \infty$. After k distances have been calculated δ assumes the k^{th} largest distance. The search finds the k nearest neighbors in one traversal.

For PNN the ideal choice of δ would be just the right hand side of the condition in 5. Initially $\delta = \infty$ and its value quickly decreases as successively closer prototypes, $\mathbf{c}$, are found. During the search δ is the closest distance so far plus $2\alpha\sigma^2$.

VI. Handprint OCR Results

Large databases are important to the meaningful appraisal of recognition algorithms. For this study we have used the digits from NIST Special Database 19 [11]. The training set contains 17231 examples of each of the 10 classes from 1831 writers; the testing data is 2314 samples per class from a further 250 writers. The features used throughout are KL transforms; i.e. the projections of the binary images onto the eigenvectors of the covariance matrix of the training images. An industry-academia OCR competition using this data is detailed in [12]. The sets contain many examples that do not define decision boundaries and are thus suitable candidates for prototype pruning, using Voronoi deletion techniques [5] [13] or clustering.

The results cover four axes of investigation: the L_2 versus L_∞ comparison, the kd tree versus linear search methods, the merits of KNN versus PNN, and the effect of variance-ordered over variance-equalized features. The tables use the definitions below. Specific values apply to all the tabulated results unless noted otherwise.

k	=	KNN's number of near neighbors, $k = 3$
σ	=	PNN's kernel width, $\sigma_{L_2} = 2$, $\sigma_{L_\infty} = 0.2$
α	=	PNN's deletion parameter in eq. 5, $\alpha = 2$
m	=	minimum allowed samples per kd tree leaf, $m = 10$
N	=	number of training samples available, $N = 172310$
N_s	=	number of samples searched, $N_s \leq N$, the equality holds
		for kd trees this is the sum of the patterns in the visited
N_r	=	number of samples retrieved for possible use i.e. those p
		considered, at any stage, to be close to the unknown, N_r
N_u	=	number of samples finally used in the discriminant sum
		KNN $N_u \equiv k$, for PNN $N_u \leq N_r$ is the mean number sa
n	=	the overall feature space dimension, $n = 40$
n_d	=	the mean number of components used in the distance su
		i.e. the dimension sufficient to determine sample eligibil
E	=	recognition error percent
t	=	recognition time in ms per unknown

Empirically error and speed performance is largely invariant over relatively wide ranges of n, m, σ and k. Table 1 compares the naive and fast implementations of KNN. By aborting the distance calculations, large efficiency gains are obtained. The ratio of the nominal dimensionality, n, to the actual dimension, n_d, gives a theoretical speed improvement since both metrics take time proportional to feature dimension. Although run time is reduced by a factor of four, the theoretical order of magnitude gains are not realized, due to the expense of the eligibility test. The number of components required decreases with increasing numbers of samples because, on average, a closer k^{th} nearest neighbor is present.

The first row of table 2 shows that the kd tree search

N	n_d	n/n_d	t_s/t_f	E
5000	3.83	10.4	5.5	2.7
10000	3.38	11.8	4.4	2.2
20000	3.03	13.2	4.0	1.9
40000	2.76	14.5	4.2	1.6
70000	2.58	15.5	4.3	1.3
100000	2.48	16.1	4.5	1.3
140000	2.40	16.7	4.5	1.2
172310	2.35	17.0	4.6	1.1

Table 1: Naive versus fast linear search for KNN using an L_2 distance metric. The fourth column gives measured speed improvement as the ratio of the slow and fast times.

	kd Tree		Linear	
	KNN	PNN	KNN	PNN
N_s/N	0.098	0.134	1.00	1.00
N_r	14.6	92.7	35.0	116.6
N_u	3	80.5	3	80.5
n_d	5.5	6.3	2.4	2.6
t	41	94	400	442

Table 2: Linear versus *kd* tree searches using L_2 distances. The error rates for KNN and PNN, 1.09% and 1.17%, are independent of the search method.

avoids retrieval of about 90% of the prototypes and saves corresponding time over the linear algorithm. Note that by using fewer samples the *kd* search requires additional components and temporarily finds relatively more of the N_s prototypes to be eligible. These increases in n_d and N_r/N_s negate some of the efficiency gains. Ultimately, both searching schemes produce the same result. The number of prototypes found during the search, N_r, is generally larger than the number, N_u, that finally remain eligible after final retention is determined by 1 and 5. The *kd* tree exhibits superior economy in this regard since the tree traversal more quickly yields close points.

Table 3 shows the effect of using KL transforms in place of variance-equalized features for both searching paradigms. The table also compares the two distance metrics. The *kd* tree method reaps greater benefit from KL variance-ordering than does the linear search; classification times fall by factors of about 4 and 1.5 respectively. As noted previously, *kd* trees succeed because many subtrees are never searched and the use of KL transforms cuts N_s by more than a factor of three, relative to their variance-equalized counterparts. Although the orthonormal transformation that produced the variance-equalized features gives invariant L_2 distances and identical classifications, we found that *kd* tree traversal considers (N_s) and retrieves (N_r) differing numbers of points when using the new features. This is due to the *kd* construction algorithm reordering prototypes while distributing them over the tree. With respect to feature dimension,

		Ordered		Equalized	
		L_2	L_∞	L_2	L_∞
	E	1.09	1.57	1.09	1.53
kd Tree	n_d	5.5	3.6	6.4	3.5
	N_s/N	0.098	0.014	0.328	0.044
	N_r	14.6	15.4	15.6	16.6
	t	41	6	165	16
		L_2	L_∞	L_2	L_∞
Linear	n_d	2.4	1.3	5.6	2.1
	N_r	35.0	35.1	35.0	35.1
	t	400	305	609	360

Table 3: Comparison of variance-ordered (KL) and variance-equalized ($\bar{\lambda}$) features using KNN and both searching schemes.

the L_∞ retention criterion is more efficient. For a KNN linear search with $k = 1$ using KL features, an average of just 1.1 coefficients are necessary to determine eligibility and, despite L_∞ distances not being invariant under rotation, classification is actually slightly superior for the variance-equalized features.

Empirically, speed increases obtained by using L_∞ (versus L_2) distances *and* using KL (versus variance-equalized) features scale closely with $n_d^{1/2} N_s$. The benefit of either optimization to the linear algorithm is much less pronounced since it gains only from a reduction in the number of dimensions required to determine retention, n_d, and not from any reduction in N_s, which is a constant equal to N.

Figure 1 shows the dependence of classification time on the number of prototypes used by the *kd* classifier. Given the log scales on both axes it is apparent that $t \propto N^\beta$ where β is less than unity and is a function of the feature space dimension, n. In the worst case, when n is large, searching takes time proportional to $N^{2/3}$. For small n the performance is markedly better, scaling as $N^{1/3}$. For the OCR case the dimensionality, $n = 40$, is large and the number of prototypes relatively small, so $N \ll 2^n$. This sparseness in the n-dimensional space is the cause of the logarithmic behavior [4] not being found. However while never as good as $\log N$, our results offer a substantial improvement over time proportional to N.

Table 4 shows that, using a value for α that preserves hypotheses, PNN employs many more neighbors in the discriminant calculation than KNN. The implication is that, although many samples contribute significantly, by an amount greater than $e^{-\alpha} d(\mathbf{y}, \mathbf{c}_1)$, the hypothesized class is determined sufficiently by the first few neighbors, and the remaining prototypes merely saturate the result. The number used in classification, N_u, increases approximately as α^2 for $\alpha > 1$. Since the number of eligible prototypes for PNN is a random variable, classification is relatively slow and time increases about linearly with α. Recognition error is ultimately very similar to KNN. For this data set PNN has only

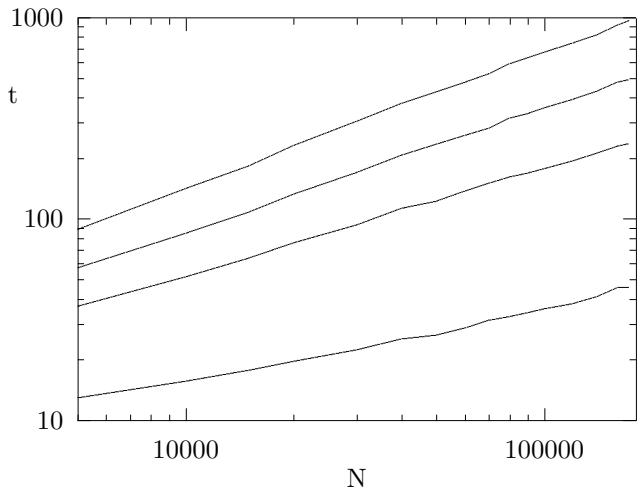

Figure 1: Classification time versus number of prototypes, for the *kd* method using KNN with L_2 distances. The curves, from top to bottom, correspond to $n = 40, 24, 16, 8$. Using a least squares fit their slopes are $0.67, 0.61, 0.53$ and 0.37 respectively.

KNN	k	1	2	3	5	7
	E	1.23	1.23	1.09	1.18	1.21
	n_d	2.1	2.3	2.4	2.5	2.6
	n_d ∘	4.7	5.3	5.6	6.1	6.4
	N_r	12	24	35	56	77
PNN	α	1.15	2.3	4.6	9.2	13.8
	E	1.17	1.16	1.17	1.17	1.17
	n_d	2.2	2.4	2.6	3.2	3.9
	n_d ∘	5.2	5.7	6.7	8.6	10.4
	N_r	21	40	117	400	851
	N_u	5	19	81	326	729

Table 4: KNN versus PNN for various k and α using the linear search and L_2 distances. The symbol ∘ denotes results for variance-equalized features.

its superior error versus reject performance to recommend it over the more constrained KNN scheme. It should be noted that PNN gives superior classification in other applications, for example in fingerprint recognition [14], in which only small training sets are available and density estimation is less accurate.

For variance-equalized features the mean number of components used is 4.7, double that for the KL transforms. The attendant increase in classification time is a factor of 1.5.

VII. Conclusions

The *kd* tree data structure gives rise to an order of magnitude increase in speed over our fast linear implementations of the non-parametric methods. Searching times scale much better than linearly with the number of prototypes. The dependence of time on the number of samples, N, and the feature dimensionality, n, means that real applications benefit from compact feature representation; Karhunen-Loève expansions are used very parsimoniously by the non-parametric classifiers. Partial distance calculations using these features afford a four-fold increase in classification speed over the naive definitions of these algorithms. The KL transform decreases the number of feature components used by the classifiers by more than a factor of two compared to the case where features have equal variance.

The L_∞ distance metric applied with the *kd* data structure yields another order of magnitude efficiency gain over the traditional L_2 norm. The improvement is much less pronounced in the linear search. The disadvantage of this metric is an increase in recognition error from 1.1% to 1.6%, although this may be traded for speed by using a larger training set.

PNN gives comparable recognition and speed to the KNN method. Given the necessity to optimize an extra parameter and the absence of any classification advantages associated with having a locally variable k, it is the authors' conclusion that PNN has little to commend it over KNN.

The relative efficacies of the techniques will depend on the data in question and, although specific pathologies therein can both help and hinder the classifier, we suggest that *kd* trees, alternative distance metrics, energy compaction and variance ordering, are potent means for expediting nearest-neighbor classifiers and worthy of investigation when speed is an issue. The traditional application of the sample condensation methods is still recommended as their action will usually be complementary to the techniques described.

References

[1] J. L. Blue, G. T. Candela, P. J. Grother, R. Chellappa, and C. L. Wilson. Evaluation of Pattern Classifiers for Fingerprint and OCR Applications. *Pattern Recognition*, 27(4):485–501, 1994.

[2] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Processing*, IT-13:21–27, 1967.

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

[4] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977.

[5] P. E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, IT-14:515–516, 1968.

[6] K. Fukunaga and R. R. Hayes. The reduced Parzen classifier. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11:423–425, 1989.

[7] K. Fukunaga. *Introduction to Statistical Pattern Recognition*, chapter 6, pages 254–299. New York: Academic

Press, second edition, 1990.

[8] D. F. Specht. Probabilistic neural networks. *Neural Networks*, 3(1):109–118, 1990.

[9] M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C. L. Wilson. NIST Form-Based Handprint Recognition System. Technical Report NISTIR 5469, National Institute of Standards and Technology, July 1994.

[10] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[11] P. J. Grother. Handprinted forms and characters database. Technical Report Special Database 19, **HFCD**, National Institute of Standards and Technology, March 1995.

[12] R. A. Wilkinson, J. Geist, S. Janet, P. J. Grother, C. J. C. Burges, R. Creecy, B. Hammond, J. J. Hull, N. J. Larsen, T. P. Vogl, and C. L. Wilson. The First Optical Character Recognition Systems Conference. Technical Report NISTIR 4912, National Institute of Standards and Technology, August 1992.

[13] G. T. Toussaint, B. K. Bhattacharya, and R. S. Poulsen. The application of Voronoi diagrams to nonparametric decision rules. In *Computer Science and Statistics:* 16^{th} *Symposium on the Interface, Atlanta, Georgia*, 1984.

[14] G. T. Candela and R. Chellappa. Comparative Performance of Classification Methods for Fingerprints. Technical Report NISTIR 5163, National Institute of Standards and Technology, April 1993.

Biographies

Patrick J. Grother received his B. Sc. degree in Physics from Imperial College, London. He subsequently received his M. Sc. degree in Computing and Image Processing from the same establishment. He is presently pursuing research in the Advanced Systems Division at the National Institute of Standards and Technology. His current interests include pattern recognition, numerical methods and image processing, as used for fingerprint and document processing.

Gerald T. Candela received a B.S. degree in mathematics from the University of Maryland in 1982. His research interests are in the areas of image processing and pattern recognition, particularly as applied to fingerprint classification.

James L. Blue received his A.B. in Physics and Mathematics from Occidental College and his Ph.D. in Physics from the California Institute of Technology. He currently heads a mathematical modeling group at the National Institute of Standards and Technology. He has done research in semiconductor devices, mathematical modeling, and numerical methods at Bell Telephone Laboratories. His current interests include computer algorithms and simulation, micromagnetics, and neural networks.