Comparative Performance of Classification Methods for Fingerprints

G. T. Candela
R. Chellappa

U. S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Technology Administration
Gaithersburg, MD 20899

(Dr. Chellappa is a member of the
Department of Electrical Engineering
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742)

April 1993

Abstract

We report the results of several pattern classifiers as tested on NIST Special Database 4, which consists of fingerprint images produced from two rollings of each of 2000 different fingers. The fingerprints are to be classified into five broad classes known as Arch, Tented Arch, Left Loop, Right Loop and Whorl. The classifiers tested are drawn from traditional pattern recognition literature (minimum distance, maximum a posteriori, nearest neighbor) as well as neural network literature (multilayer perceptron, radial basis functions, probabilistic neural network). To enable a fair comparison of the classifiers, preprocessing steps such as enhancement and feature extraction are kept the same for all the classifiers. Classification accuracies for all the classifiers are tabulated for the case when they are trained on fingerprints from one rolling and tested on fingerprints from a different rolling. Variations of classifier accuracies as the feature vector dimension varies are studied. Computational and memory requirements of the different classifiers are also compared.

Contents

List of Figures

List of Tables

1 Introduction

This report compares the performance of several types of automatic classifiers as applied to the Pattern-level Classification Automation (PCA) problem for fingerprints. This is the problem of building a machine to classify fingerprint images into five broad classes known as Arch, Tented Arch, Left Loop, Right Loop, and Whorl. Figures 1 through 5 are example fingerprints of the five classes. An automatic fingerprint classifier will be useful as a component of an Automated Fingerprint Identification System, or AFIS. The purpose of an AFIS is to find out whether the individual represented by an incoming "search" fingerprint card is the same as an individual represented by one of a large database of "file" cards. First, the name and other textual data on the incoming card (sex, birthdate, etc.) are automatically searched for in a database of the corresponding data from file cards; very often, a candidate matching card is found by this "name search", and the match can then be confirmed or denied by comparison of the fingerprints on the search card and the candidate file card. However, if the name search turns up no candidate card, or if the candidate card's prints do not match those of the search card, then it is necessary to conduct a "technical search" to find out whether there is any file card (regardless of textual data) whose prints match those of the search card.

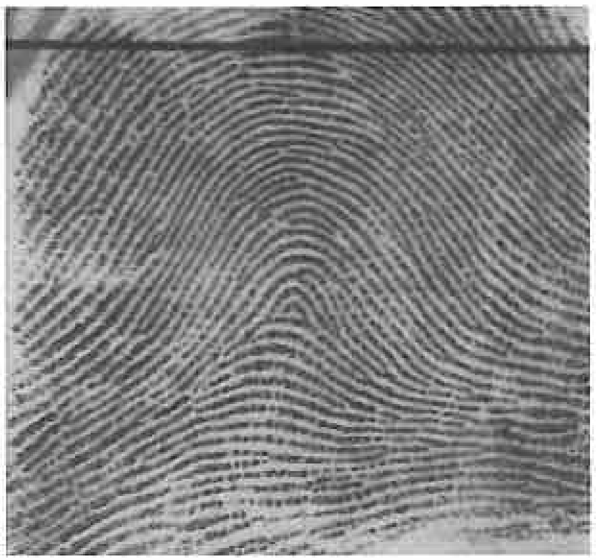

Figure 1: A fingerprint of the Arch class

Automated systems for comparing fingerprints have existed for many years. The Federal Bureau of Investigation uses an automatic "minutiae finder" to find the locations and directions of many small details (ridge endings and bifurcations) of the search fingerprints, and an automatic "minutiae matcher" to rapidly compare the resulting minutiae-lists with electronically stored minutiae-lists of the fingerprints from file cards. The matcher produces a small list of candidate matching cards, and a fingerprint examiner then manually compares the candidates with the search card to ascertain whether one of them is a positive match. If a match is found, a second examiner verifies the identification.

The automated technical search process currently used by the FBI precedes the minutiae-matching step with a manual classification step. Technicians classify each of the fingerprints on the search card, then assign a classification code to the card as a whole. The system used for manual classification is the so-called Henry system, with modifications and extensions introduced by the FBI. (The handbook [1] provides a complete description of the rules for manual fingerprint classification.) The file cards are stored in cabinets in the sequence defined by their Henry classifications, and their electronically stored minutiae-lists are also organized according to these classifications. Therefore, after a search card has been classified, it is not necessary to use the automatic matcher to compare its minutiae with those of *every* file card; rather, it is only necessary to compare its minutiae with those of cards of the same class. Because the Henry system separates the file cards into a very large number of classes, the number of minutiae-lists that the automatic matcher must compare with the search minutiae-list is often very much smaller than the total number of file cards. Manual classification therefore saves a large amount of processing time that would otherwise be used by the automatic matcher.

Classification is done manually because an automatic classifier of sufficient accuracy and efficiency has not yet been developed. The goal of the PCA project is to produce such an automatic classifier, albeit one that separates fingerprints into only five broad pattern-level classes rather than the much more numerous classes used in the Henry system. Automated classification using the Henry system may be achieved eventually, but a shorter-term goal is to produce an accurate classifier that uses only the five pattern-level classes. Note that although the PCA method uses only five classes for a single fingerprint, it separates a database of file cards into 9,765,625 ($= 5^{10}$) different card classes, each consisting of one of the possible ordered ten-tuples of fingerprint classes. One cannot assume that these classes will separate the file cards into 5^{10} *equal-sized* groups; nevertheless, the classification of a search card will often be that of one of the reasonably small groups, and therefore automatic pattern-level classification can be expected to save considerable minutiae-matching work.

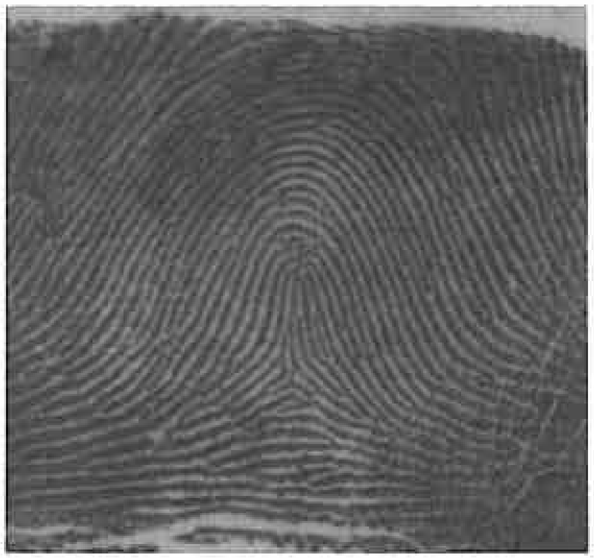

Figure 2: A fingerprint of the Tented Arch class

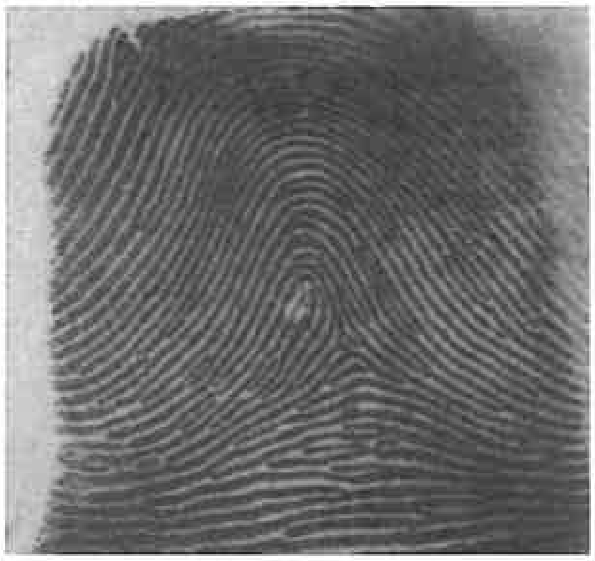

Figure 3: A fingerprint of the Left Loop class

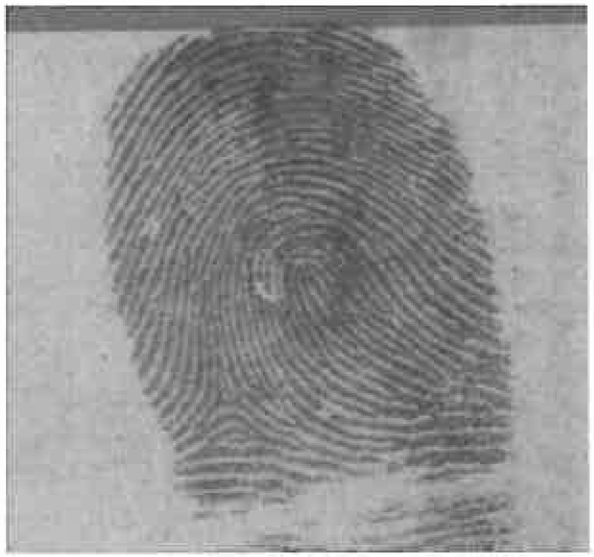

Figure 4: A fingerprint of the Right Loop class

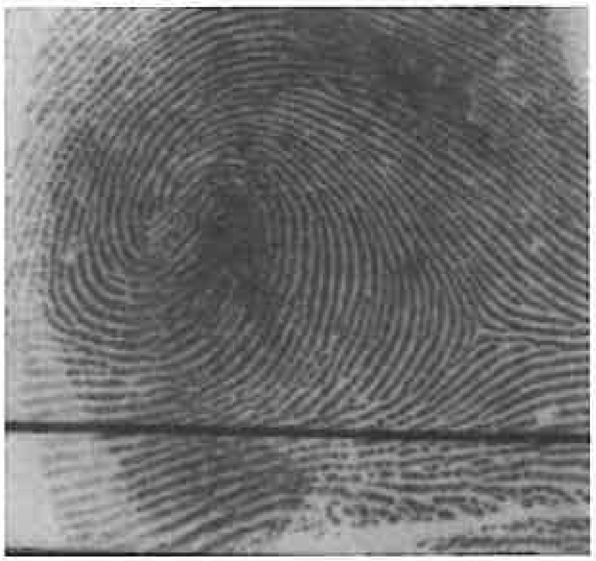

Figure 5: A fingerprint of the Whorl class

2 Previous Fingerprint Classification Work

Over the last fifteen years, at least four major approaches have been taken for automatic fingerprint classification. These are the *structural, syntactic, rulebased,* and *artificial neural network (ANN)* approaches. In the syntactic approach [2], one typically approximates ridge patterns [3] as a string of primitives and models the plausible strings from a class such as Tented Arch using production rules as in grammar. Depending on the type of grammar used one can see a significant drop in the number of production rules required. For example, a context free grammar would require fewer production rules than a regular grammar. When a new fingerprint whose identification is sought arrives, one extracts the string of primitives and passes it through a parser. A successful parser output indicates the class to which the fingerprint belongs. Whether the parsing is successful or not, a description of the input string is always generated. The more general the grammar is, the more complex the parser tends to be. Applications of the syntactic approach using more complex grammars such as stochastic grammars [4] (where probablities are associated with the production rules), tree grammars [5], and programmed grammars [6] for the fingerprint classification problem have been considered. The main stumbling block of the syntactic approach is that mechanisms for the inference of grammar from training samples have not been well understood [7, 8]. Recent advances in learning the structure of the grammar from training samples using neural nets [9] look promising.

At the Image Recognition Group of the National Institute of Standards and Technology, a massively parallel neural network fingerprint classification system has recently been implemented using a parallel computer of the SIMD variety (single-instruction-stream, multiple-data-stream) [10]. The work described in the present report is the result of modifications made to that system. More recently [11], a fingerprint matching and classification system that uses a hierarchical pyramid structure has been reported. The matching module takes two images as inputs and outputs a number between 0 and 1. This number, which reflects the degree of belief that the two images are from the same finger, is estimated using a probabilistic Bayesian approach. The network parameters (about 104 in number) are trained using a steepest descent procedure that minimizes a cross entropy measure. Impressive matching results are reported. It is interesting to note that the receptive fields of the learning filter at the bottom of the pyramid appear to be edge or ridge orientation detectors, but sometimes correspond to minutia detectors. This finding supports the choice of ridge direction components used as features in the NIST system. It would be interesting to compare the performance of the method described in [11] and the NIST system on the same dataset [12].

3 Experimental Fingerprint Database

The classifiers described in this report were trained and tested using feature vectors derived from the fingerprint images of NIST Special Database 4 [12]. This database consists of 8 bit per pixel graylevel raster images of two inked impressions ("rollings") of each of 2000 different fingers. The feature vectors used to train the classifiers were made from the 2000 first-rollings, and those used to test the classifiers were made from the 2000 second-rollings. Every fingerprint in the database has an associated class-label, assigned by experts. The two rollings of any finger have the same class, since the class of a fingerprint is not affected by variations that occur between different rollings of the finger.

Fingerprints as they naturally occur are not distributed equally into the five classes. We have taken a summary of the NCIC classes[1] of fingerprints from more than 22.2 million cards and reduced the numbers contained therein to estimates of the true frequencies or *a priori* probabilities of the five PCA classes. The estimated probabilities are .037, .029, .338, .317, and .279, for the classes Arch, Tented Arch, Left Loop, Right Loop, and Whorl respectively.

The 2000 fingers represented in Special Database 4 are equally divided among the five classes. The database was produced this way, rather than by using a natural distribution, so as to increase the representation of the relatively rare, and also difficult, Arch and Tented Arch classes. This provides trainable classifiers with more examples with which to learn these problematical classes. The training set (first rollings) therefore has equally many prints of each class, so the testing set (second rollings) also has equally many prints of each class, since the database consists of rolling pairs.

4 Components of Classification System

Each of our experimental fingerprint classifiers consists of a set of components as shown in Figure 6. The ovals represent input and output data, the rectangles represent processing components, and the arrows represent the flow of data. The components do not necessarily correspond to separate devices or programs; they merely represent a separation of the processing into conceptual units, so that the overall structure may be discerned. The fingerprint image is a raster of 512 by 512 8-bit grayscale pixels, produced by scanning the fingerprint card with a CCD device or by some other method. The "Feature Extractor" component reduces the fingerprint from a raster of 262,144 bytes to a *feature vector*, a small ordered list of numbers that takes only about 448 bytes. The feature vector is intended to represent most of the information relevant to classification in a compact form. The next component is the bank of "Discriminant Functions". There are five discriminant functions, one for each class. Each one produces a single floating-point number, which tends to have a large value if the input fingerprint is of the class corresponding to that particular discriminant function. The five-tuple of values produced by the bank of discriminant functions is sent to two final components, the "Maximum Finder" and the "Rejector". The maximum finder merely finds

[1]The NCIC classification system separates fingerprints into a very numerous set of classes, which are basically the same as the classes used in a Henry system. The NCIC method for producing a card's class from the classes of its individual fingerprints is different than the summarizing method used in the Henry system.

which one of the discriminant values is highest, and assigns its class as the *hypothesized class* of the fingerprint, that is, the classification system's best guess as to the class. The rejector component computes a "confidence function" of the discriminant values, compares the resulting value with a preset threshold, and thereby decides whether the fingerprint should be accepted or rejected. Rejection of a fingerprint means that the classification system refuses to assign a class, because it cannot be sufficiently certain as to the correct class. Rejected fingerprints must be classified manually.

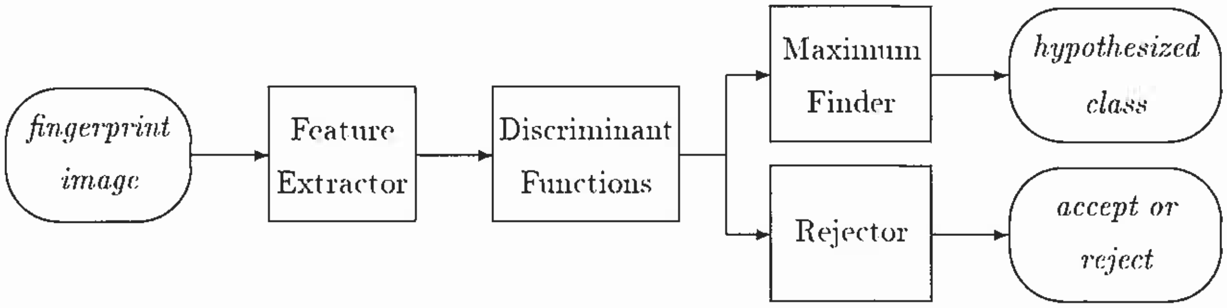

Figure 6: Components of Classification System

The following sections describe the classification system components in some detail. The feature extractor and rejector, while important components of the classification system, are peripheral to the main theme of this report, which is the comparison of different kinds of discriminant functions.

5 Feature Extractor

The classifiers described in this report take as their input a small vector of numerical features derived from a fingerprint raster image. The fingerprint is reduced to 112 features (not all of which need be used) as follows. First, it is subjected to an FFT-based filter that increases the relative power of dominant frequencies, increasing the ratio of signal (fingerprint ridges) to noise. Figure 7 shows the result of applying this filter to the Whorl example fingerprint of Figure 5. The local orientations of the ridges at 840 equally-spaced locations (a 28 by 30 grid) are then measured, using an orientation finder described in [10]. The orientation finder is based on a "ridge-valley" fingerprint binarizer described in [13]. It computes an orientation at the location of each pixel, then averages these basic orientations in nonoverlapping 16x16-pixel squares to produce the grid of 840 orientations. Figure 8 represents the output of this program, with the computed orientations indicated by the bars; the program has succeeded if the bars are parallel to nearby ridges. The resulting array of angles is sent to a *registration* program [14, 10], which finds a "core" feature. The plus sign in Figure 8 marks this core point, and the plus sign enclosed by a square marks the median of the core points of a sample of fingerprints. The array of pixel-specific ridge orientations is shifted horizontally and vertically so as to bring its core point to the median core point, and then it is averaged using an overlapping set of 16x16-pixel squares to produce an array of 3596 (58 x 62) orientations. The purpose of registration is to reduce differences between fingerprint images that result from differences in how the fingers were rolled, so that the differences that remain will be more likely to indicate true variations in fingerprint structure.

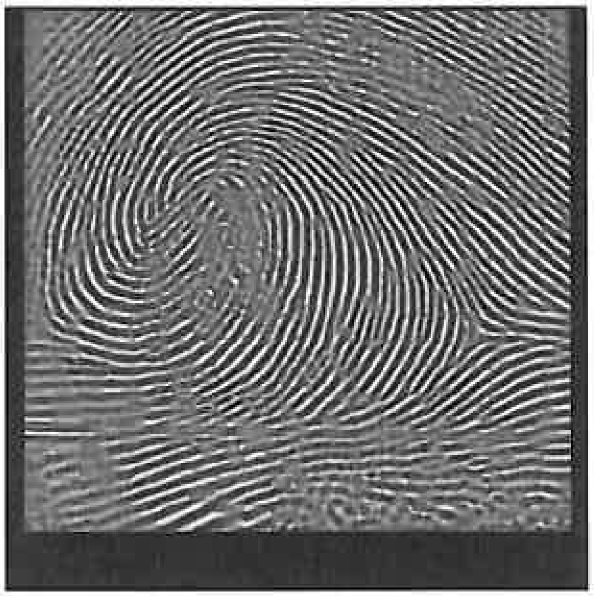

Figure 7: The Whorl example fingerprint after FFT-based filtering

Each of the 3596 orientations is represented, not as a ridge angle θ, but as a two-dimensional vector $(\cos(2\theta), \sin(2\theta))$. This representation seems prefererable because it takes account of the fact that ridges have an orientation (e.g. north-south) but not a sense (e.g. north as oppposed to south), and because it removes the artificial discontinuity that an angular representation always has at some point around its circle. The orientations are thus represented as a vector of 7192 floating-point numbers.

We had decided very early that the Karhunen-Loève (K-L) transform [15], described below, would probably be useful, because it could greatly reduce the dimensionality of feature vectors before they were sent to a classifier (that is, a bank of discriminant functions). Reducing feature dimensionality saves training time for classifiers such as multilayer perceptrons or radial basis functions, and saves memory for multiple-prototype classifiers such as the Probabilistic Neural Net. However, production of the K-L transform for 7192-dimensional input vectors would have been unfeasible on our computing equipment, because of the huge size of the required covariance matrix. Therefore, we first reduce the array of 3596 equally-spaced ridge orientations to a set of 840 orientations, arranged in a fixed unequally-spaced pattern that is intended to be optimal. The areas of the fingerprint raster that contained large numbers of the "cores" and "deltas" of a sample of fingerprints receive relatively denser spacings of orientations in the fixed pattern. (A description of cores, deltas, and the official rules for manual fingerprint classification can be found in [1]. The method of producing the unequally-spaced pattern is described in [16].) Figure 9 shows the unequally-spaced orientations for the Whorl example fingerprint. The K-L transform is applied to the vector of 1680 elements which this figure represents, reducing it to a vector of 112 elements, not all of which need be used. We have tried both equally- and unequally-spaced patterns of 840 orientations as the final features before the K-L transform, and the unequally-spaced patterns produced better classification results.

The K-L transform is an affine function produced using the mean vector and covariance matrix of a sample of the vectors whose dimension is to be reduced. (The production of this transform is also known as *principal factors* or *principal components* analysis.) The covariance matrix is subjected to a diagonalization program (made from EISPACK routines) to produce a set of largest eigenvalues and corresponding eigenvectors. To perform the K-L transform on an input vector of 1680 elements, the mean vector is subtracted from it and the resulting vector is then premultiplied by the matrix whose rows are the eigenvectors. In our case, the diagonalization program produced 112 eigenvectors. (It was set up to find all eigenvalues greater than 0.1 and their corresponding eigenvectors.) Each K-L feature is associated with an eigenvector and eigenvalue. In fact, the eigenvalues are equal to the sample variances of the corresponding features over the training set, and are therefore a measure of the "sizes" of the features. Figure 10 shows the top 112 eigenvalues; their rapid dropoff shows that the information content of the K-L features is concentrated in the first few features. Since the features associated with relatively small, later eigenvalues encode small amounts of the information contained in the original vectors, they are naturally expected to be less valuable than higher-ranking features, for purposes of classification. Therefore, if it is desired to use only a subset of k of the 112 K-L features, one obviously uses the first k that occur when they are arranged in decreasing order of eigenvalue. Figure 11 shows the mean vector; Figures 12 through 14 show the first three eigenvectors.

Figure 15 shows the full training set of 2000 examples represented using the first two K-L features, with the letters indicating the classes of the examples. It can be seen that even just two features can separate the classes somewhat: the A examples are concentrated on the right side, L below, R above, and W to the left, although the T examples are not very well separated from the others in this figure. Only two features are used in this illustration, for the simple reason that the paper on which the figure is printed is a two-dimensional surface; when the various kinds of discriminant functions were tested, they were allowed to use whatever number of features produced the best results, up to a maximum of 112 features. (The descriptions of discriminant functions are illustrated using diagrams of hypothetical class regions, also in two features.)

The quality of the features used with a classifier can have a major effect on the resulting classification accuracy. This report, however, is intended primarily to compare different versions of the classifier *per se*, that is, the last stage of the classification process. In order to make a fair comparison, all classifiers described were tested using the same features described above, although each classifier was allowed to use its particular optimal number of the features.

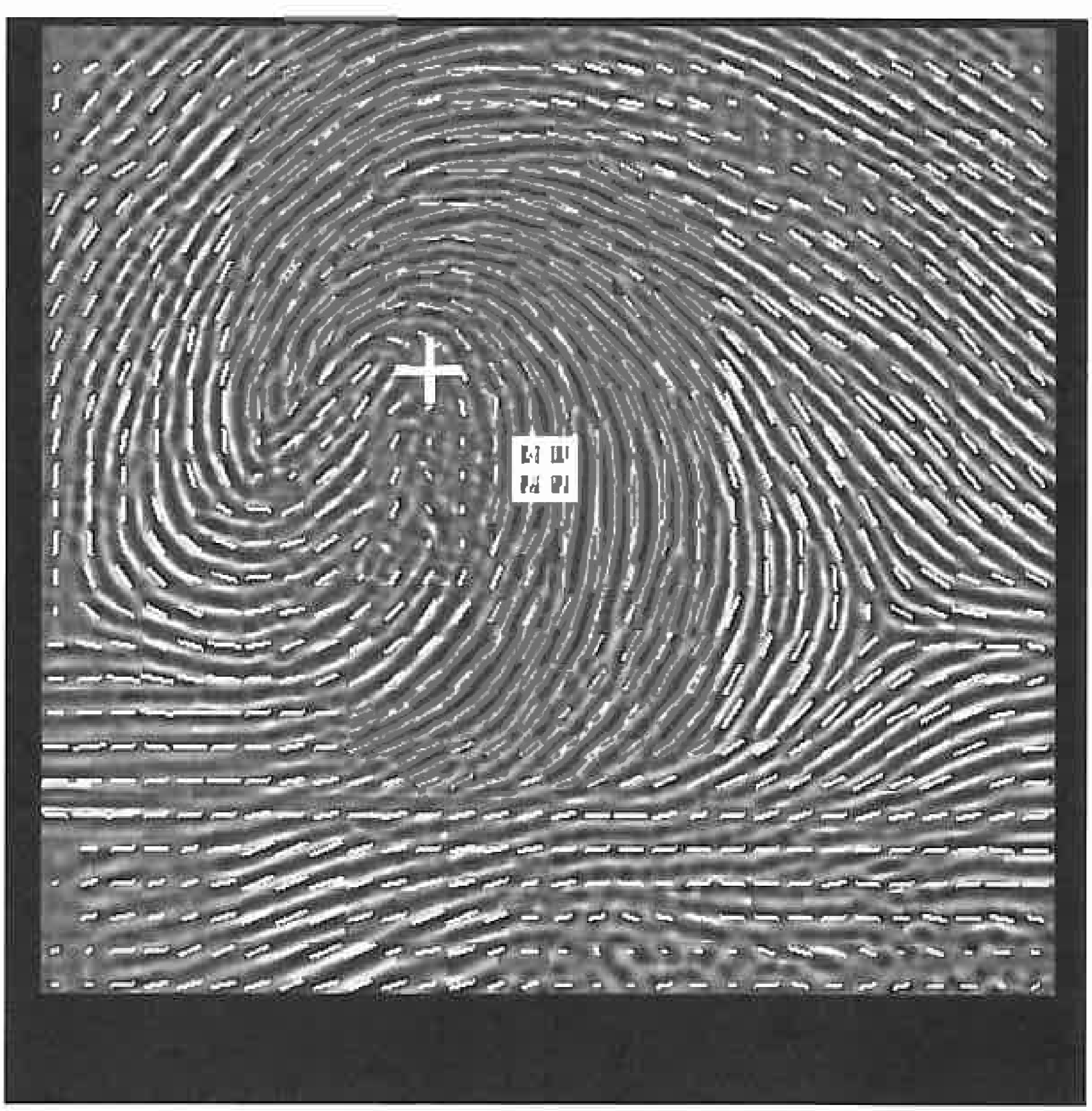

Figure 8: Ridge directions from filtered W example print. Plus sign marks core point; plus sign in square marks median of core points

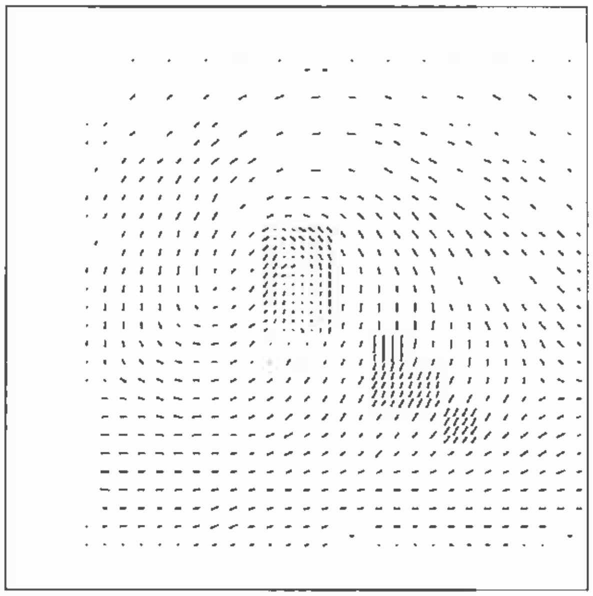

Figure 9: The 840 unequally-spaced ridge directions of the Whorl example print

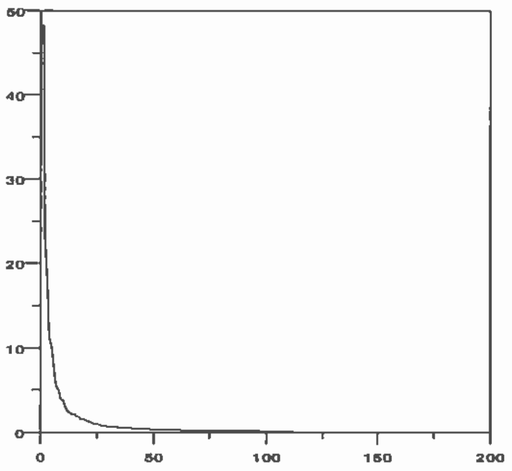

Figure 10: The top 112 eigenvalues of the covariance matrix

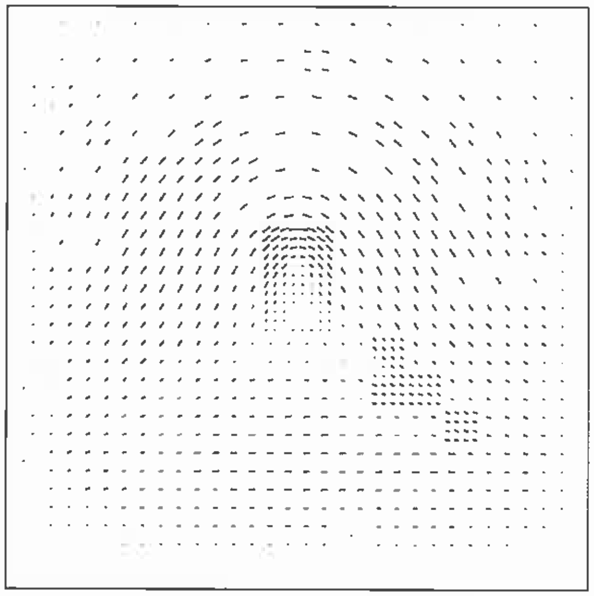

Figure 11: The mean vector

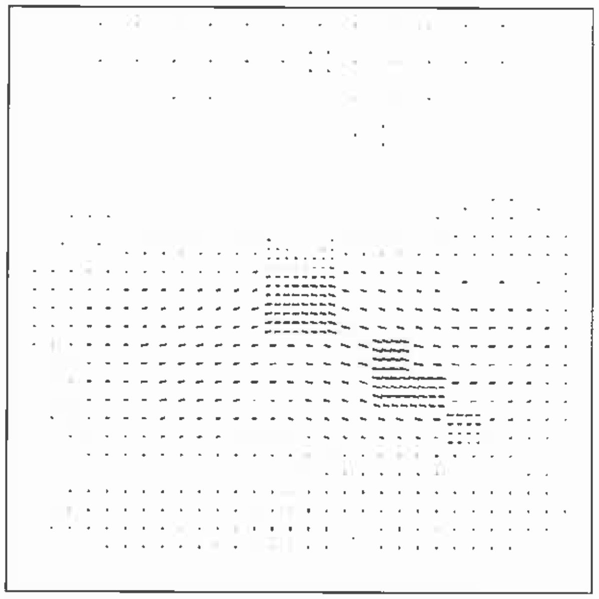

Figure 12: Eigenvector no. 1; eigenvalue is 48.35

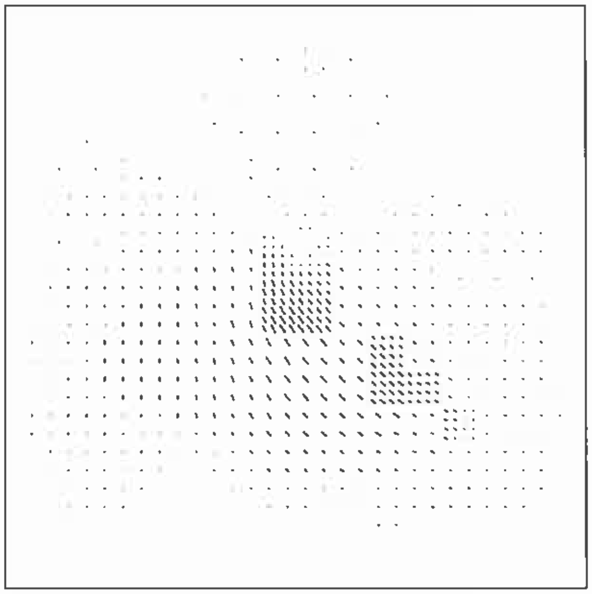

Figure 13: Eigenvector no. 2; eigenvalue is 23.33

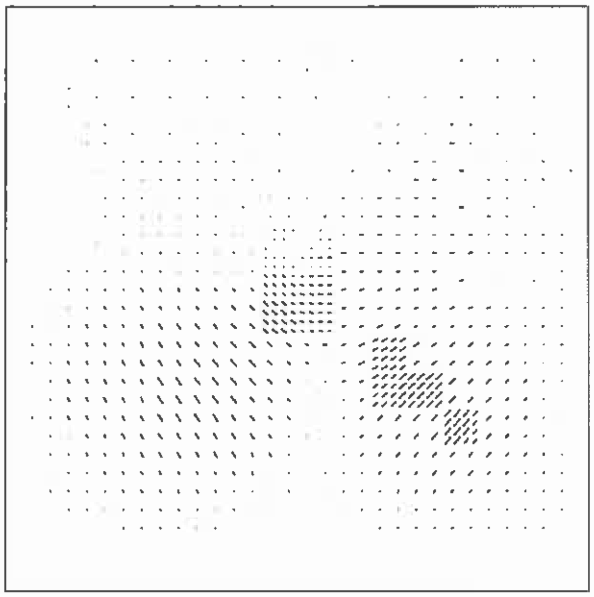

Figure 14: Eigenvector no. 3; eigenvalue is 16.92

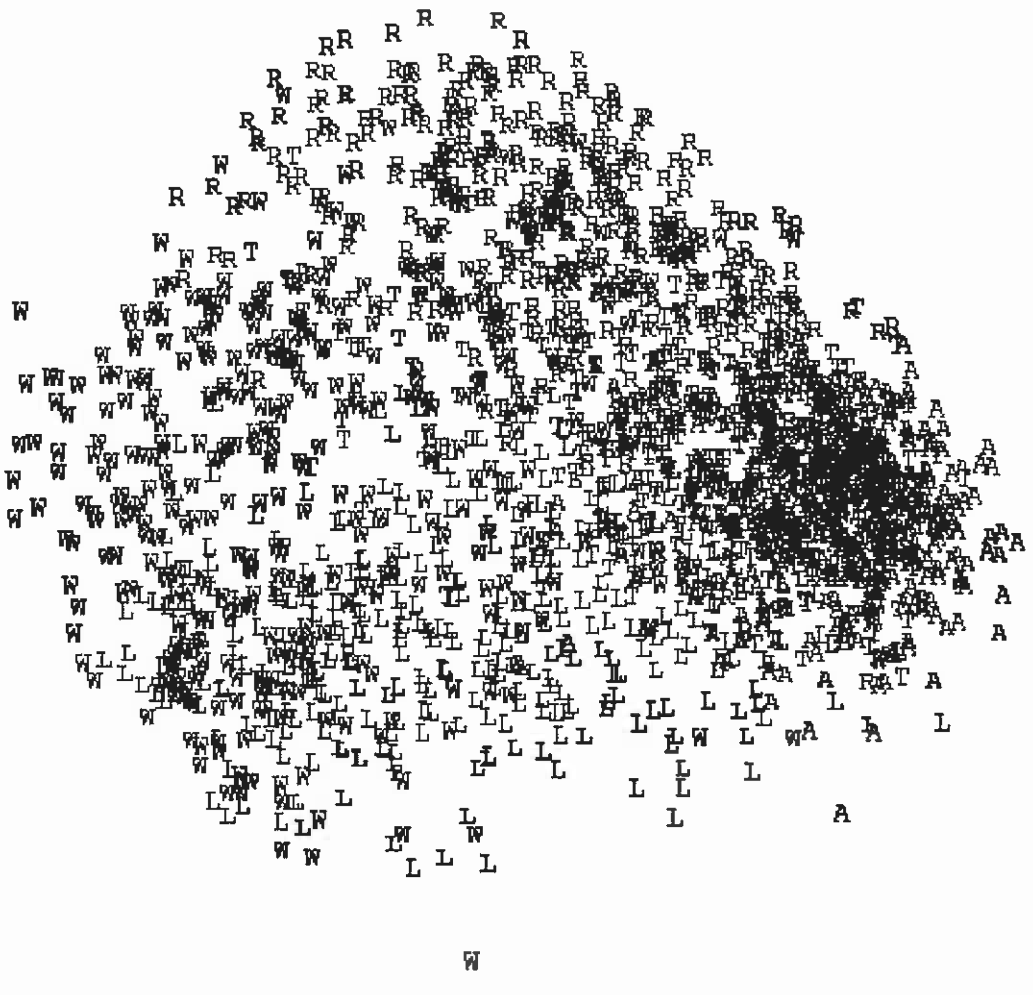

Figure 15: The full set of 2000 training examples, represented using only the first two K-L features. A = Arch, T = Tented Arch, L = Left Loop, R = Right Loop, W = Whorl.

6 Discriminant Functions (Classifiers) of Various Types

This section provides descriptions for each of the types of discriminant functions we have tested. As noted earlier, the bank of discriminant functions is the classifier *per se*, as distinguished from other components of the classification system which can be thought of as pre- and post-processors. The classifiers we have used may be separated into four categories, bearing in mind that the category names are somewhat arbitrary and that some classifiers have attributes of more than one category. The "*Ad Hoc*" category contains the simple Euclidean Minimum Distance (EMD) classifier and a more advanced version, the Quadratic Minimum Distance (QMD) classifier. The "Parametric" category contains only one example, the Normal (NRML) classifier. The "Nearest Neighbors" category contains the Single Nearest Neighbor (1-NN) classifier and an improvement we call Weighted Several Nearest Neighbors (WSNN). Finally, the Neural Net category contains the Multi-Layer Perceptron (MLP), Radial Basis Functions classifiers of two types (RBF1 and RBF2), and the Probabilistic Neural Net (PNN).

For each type of discriminant function, one or more diagrams are provided showing the resulting hypothetical class regions in two-dimensional feature space. (More than one diagram is provided if the discriminant functions have an adjustable parameter, such as a number of hidden nodes.) These diagrams show the hypothesized classes that are assigned to each of a large number of regularly spaced feature vectors: each one is made using a square array of 512 by 512 points with (0,0) at the center and with the extent large enough to contain all the training feature vectors. It would have been better, perhaps, to make these diagrams for the optimal dimensionalities of feature space for each type of classifier - such as 28 or 112 dimensions - but we have decided to use just two dimensions because it would probably be difficult to produce satisfactory pictures of high-dimensional regions or boundaries. The reader should try to imagine class regions in high-dimensional feature spaces that are analogous to the regions depicted in the diagrams.

6.1 Notation

The notation below will be used in the descriptions of the discriminant functions.

N	$=$	number of classes. For fingerprint PCA, $N = 5$
$p(i)$	$=$	*a priori*: probability that a random fingerprint is of class i $(1 \le i \le N)$
$\hat{p}(i)$	$=$	an estimate of $p(i)$
n	$=$	number of features used
$\mathbf{R}^n$	$=$	the set of all n-tuples of real numbers = "feature space"
$\mathbf{x}$	$\doteq$	a "feature vector" representing a fingerprint $(\mathbf{x} \in \mathbf{R}^n)$
M_i	$=$	number of training prints of class i $(1 \le i \le N)$
$\mathbf{x}_j^{(i)}$	$=$	feature vector from j^{th} training print of class i $(1 \le i \le N, 1 \le j \le M_i)$ $(\mathbf{x}_j^{(i)} \in \mathbf{R}^n)$
$\boldsymbol{\mu}_i$	$=$	mean feature vector for class i $(1 \le i \le N)$ $(\boldsymbol{\mu}_i \in \mathbf{R}^n)$
$\mathbf{m}_i$	$=$	an estimate of $\boldsymbol{\mu}_i$
$\boldsymbol{\Sigma}_i$	$=$	covariance matrix for class i $(1 \le i \le N)$ $(\boldsymbol{\Sigma}_i$ is an $n \times n$ matrix$)$
$\mathbf{S}_i$	$=$	an estimate of $\boldsymbol{\Sigma}_i$
$d^2(\mathbf{x}, \mathbf{y})$	$=$	$(\mathbf{x} - \mathbf{y})^{\mathrm{T}}(\mathbf{x} - \mathbf{y})$ = squared Euclidean distance between $\mathbf{x}$ and $\mathbf{y}$ $(\mathbf{x}, \mathbf{y} \in \mathbf{R}^n)$
$D_i(\mathbf{x})$	$=$	i^{th} discriminant function $(1 \le i \le N, \mathbf{x} \in \mathbf{R}^n)$

6.2 *Ad Hoc* Classifiers

6.2.1 Euclidean Minimum Distance (EMD)

This is perhaps one of the simplest classifiers that one can design. Its discriminant functions are of the form

$$D_i(\mathbf{x}) = -d^2(\mathbf{x}, \mathbf{m}_i).$$

Classifying an unknown to the class of the highest-valued discriminant function is equivalent to using the class label of the estimated class-mean that is closest to the unknown in the sense of squared Euclidean distance. The resulting hypothetical class regions are convex polygons. Figure 16 shows the class regions when only two features are used. The estimated class mean vectors $\mathbf{m}_i$ are marked with plus signs.

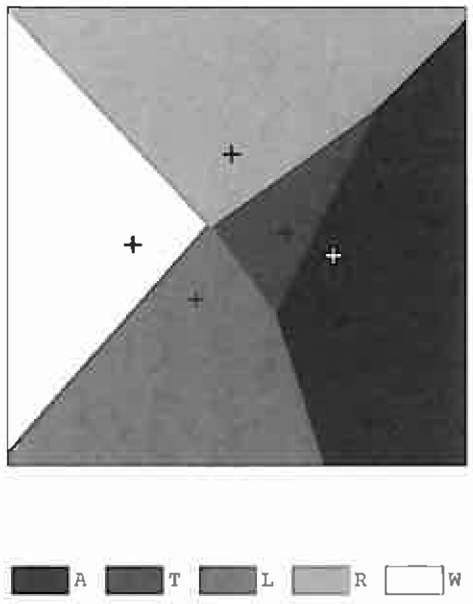

Figure 16: EMD class regions. Estimated class means are marked

6.2.2 Quadratic Minimum Distance (QMD)

In this classifier, the training examples of each class i are used to produce an estimate $\mathbf{S}_i$ of the class covariance matrix, as well as an estimate $\mathbf{m}_i$ of the class mean vector. Then the following discriminants are used:

$$D_i(\mathbf{x}) = -(\mathbf{x} - \mathbf{m}_i)^{\mathrm{T}}\mathbf{S}_i^{-1}(\mathbf{x} - \mathbf{m}_i).$$

This can be thought of as a form intermediate between EMD and the Normal (NRML) classifier (described next). Figure 17 shows the resulting class regions; their boundaries are quadratic.

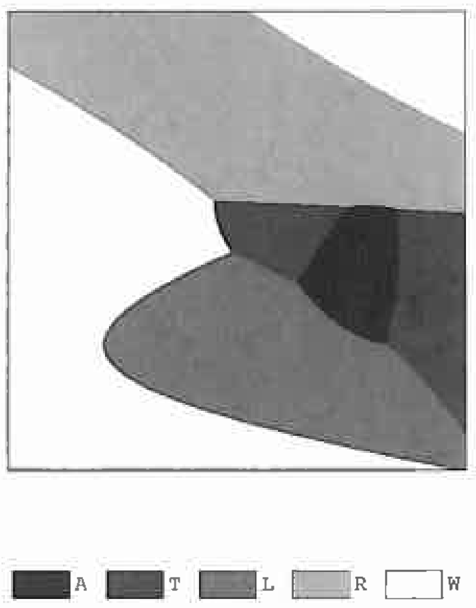

Figure 17: QMD class regions

6.3 Normal (NRML): A Parametric Classifier

This classifier is based on parametric density estimation that presupposes a multivariate normal distribution for each class of fingerprints. First, it will be useful to mention a few facts that pertain to any parametric classifier, using the following terminology:

$\lambda(i\|j)$	$=$	loss incurred by classifying to i a print that is of class j $(1 \leq i, j \leq N)$
$p(\mathbf{x})$	$=$	mixture density: for $S \subseteq \mathbf{R}^n$, $\int_S p(\mathbf{x})d\mathbf{x} = P(\mathbf{x} \in S)$
$p(\mathbf{x}\|i)$	$=$	conditional density: for $S \subseteq \mathbf{R}^n$, $\int_S p(\mathbf{x}\|i)d\mathbf{x} = P(\mathbf{x} \in S\|\mathbf{x}$ is from a class-i print$)$
$p(i\|\mathbf{x})$	$=$	*a posteriori* probability: for a particular $\mathbf{x}$, $p(i\|\mathbf{x}) = P(\mathbf{x}$ is from a class-i print$)$

Given a particular loss function $\lambda(i|j)$, the optimal or "Bayesian" classifier is the one that minimizes the expected loss. Suppose the "symmetric" loss function is used:

$$\lambda(i|j) = \begin{cases} 0 & i = j \\ 1 & \text{otherwise} \end{cases} .$$

This means that correct classifications produce no losses and that all kinds of incorrect classifications produce equal loss values of 1 unit. In this case, the Bayesian classifier is the one that classifies each unknown $\mathbf{x}$ to the class i for which the *a posteriori* probability $p(i|\mathbf{x})$ is highest. According to Bayes's rule [17],

$$p(i|\mathbf{x}) = \frac{p(i)p(\mathbf{x}|i)}{p(\mathbf{x})}.$$

Since the value of the mixture density $p(\mathbf{x})$ has no effect on which possible i value maximizes $p(i|\mathbf{x})$, $p(\mathbf{x})$ may be disregarded, resulting in the rule that one should classify $\mathbf{x}$ to the class i for which $p(i)p(\mathbf{x}|i)$ is highest.

When using the Normal classifier, one assumes that each class i of fingerprint feature vectors has conditional density function

$$p(\mathbf{x}|i) = (2\pi)^{-\frac{n}{2}}|\mathbf{\Sigma}_i|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^{\mathrm{T}}\mathbf{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i)\right),$$

where $\boldsymbol{\mu}_i$ and $\mathbf{\Sigma}_i$ are the mean vector and covariance matrix for class i. Therefore, the optimal classifier picks the i that maximizes

$$p(i)p(\mathbf{x}|i) = p(i)(2\pi)^{-\frac{n}{2}}|\mathbf{\Sigma}_i|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^{\mathrm{T}}\mathbf{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i)\right).$$

The above expression may be modified by discarding the $(2\pi)^{-\frac{n}{2}}$ term and taking the logarithm; these changes do not affect which of the classes i maximizes the expression. The rule, then, is that one should maximize

$$\log p(i) - \frac{1}{2}\log|\mathbf{\Sigma}_i| - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^{\mathrm{T}}\mathbf{\Sigma}_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i).$$

Finally, in order to produce the discriminant functions for the Normal classifier, one replaces the several $p(i)$ with estimates $\hat{p}(i)$ and replaces the class mean vectors $\boldsymbol{\mu}_i$ and class covariance matrices $\mathbf{\Sigma}_i$ with their estimates, the $\mathbf{m}_i$ (class sample mean vectors) and the $\mathbf{S}_i$ (class sample covariance matrices). The resulting discriminant functions are

$$D_i(\mathbf{x}) = \log \hat{p}(i) - \frac{1}{2}\log|\mathbf{S}_i| - \frac{1}{2}(\mathbf{x}-\mathbf{m}_i)^T\mathbf{S}_i^{-1}(\mathbf{x}-\mathbf{m}_i).$$

The hypothetical class regions for the Normal classifier are shown in Figure 18. Their boundaries are quadratic, as for the QMD classifier, but the Arch region has become much smaller and the Tented Arch region has disappeared completely.

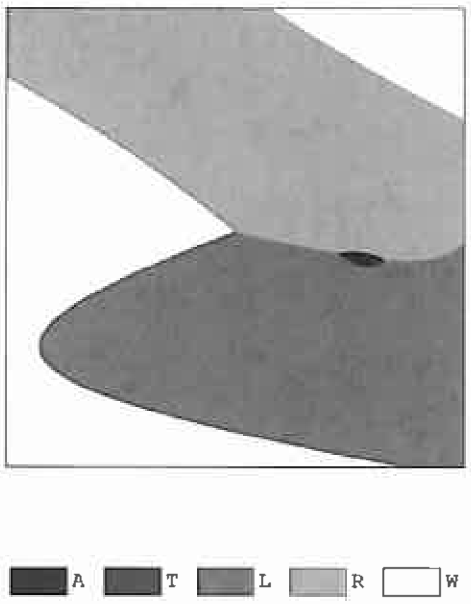

Figure 18: NRML class regions

6.4 Nearest Neighbor Classifiers

6.4.1 Single Nearest Neighbor (1-NN)

This can be thought of as a generalization of EMD. Instead of using just $\mathbf{m}_i$, the estimated mean vector for class i, as a single prototype for the class (as EMD does), the 1-NN classifier uses *all* of the class-i training examples as prototypes for the class. The hypothetical class produced by 1-NN for an unknown vector is simply the class of the closest prototype. This rule seems intuitively appealing, and Cover and Hart [18] have shown it to have good asymptotic behavior: under mild assumptions, its large-sample probability of error is bounded above by twice the Bayes (i.e. minimum possible) probability of error.

Discriminant functions of the following form may be used to implement the 1-NN classifier:

$$D_i(\mathbf{x}) = -\min_{1\leq j\leq M_i} d^2\left(\mathbf{x}, \mathbf{x}_j^{(i)}\right).$$

Figure 19 shows the class regions. Each region is the union of many convex polygons each containing a single prototype of the class; hence, a class region is a very complicated polygon, not necessarily convex or even connected.

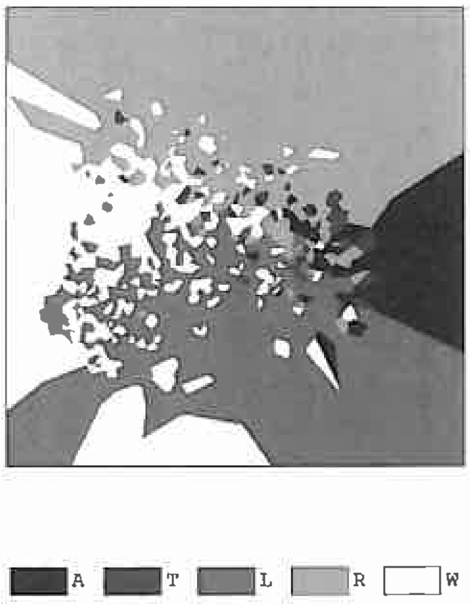

Figure 19: 1-NN class regions

6.4.2 Weighted Several Nearest Neighbors (WSNN)

After trying the 1-NN classifier, we naturally tried its well-known generalization the k-NN classifier, expecting improved results. This classifier finds the k nearest prototypes to the unknown and classifies it to the class that is most abundantly represented among these near neighbors. (For practical purposes, k may be optimized by simply finding which value produces the highest test scores.) Suprisingly, our experiments always produced lower scores for k values greater than 1 than for $k = 1$. However, results better than the 1-NN results were obtained with a slightly more elaborate version of k-NN, which may be called a Weighted Several Nearest Neighbor classifier. This classifier finds the closest prototype to the unknown, then defines the "neighboring" prototypes to be those whose squared Euclidean distance from the unknown is less than α times the squared-distance of the nearest prototype, where α is a constant. The number of "votes" received by class i is divided by the square root of the sum of squared-distances of class-i near neighbors from the unknown, so as to diminish the importance of neighbors that are relatively far away compared to other neighbors.

To put the above description in mathematical notation, let:

$$\begin{aligned}
\alpha &= \text{neighborhood-size factor} \\
\mathcal{S}_{\mathbf{x}}^{(i)} &= \text{the set of indices of class-}i\text{ training vectors that are} \\
&\ \text{in the }\alpha\text{-neighborhood of unknown vector }\mathbf{x} \\
&= \left\{ j \,\middle|\, 1 \le j \le M_i,\, d^2\left(\mathbf{x}, \mathbf{x}_j^{(i)}\right) < \alpha \min_{1 \le k \le N, 1 \le p \le M_k} d^2\left(\mathbf{x}, \mathbf{x}_p^{(k)}\right) \right\} \\
V_{\mathbf{x}}^{(i)} &= |\mathcal{S}_{\mathbf{x}}^{(i)}| = \text{number of "votes" for class } i
\end{aligned}$$

The discriminant functions are then

$$D_i(\mathbf{x}) = \begin{cases} V_{\mathbf{x}}^{(i)} \left(\sum_{j \in \mathcal{S}_{\mathbf{x}}^{(i)}} d^2\left(\mathbf{x}, \mathbf{x}_j^{(i)}\right) \right)^{-\frac{1}{2}} & \text{if } V_{\mathbf{x}}^{(i)} > 0 \\ 0 & \text{otherwise} \end{cases} .$$

Figure 20 shows the WSNN class regions resulting from α values of 3, 10, 50, and 90.

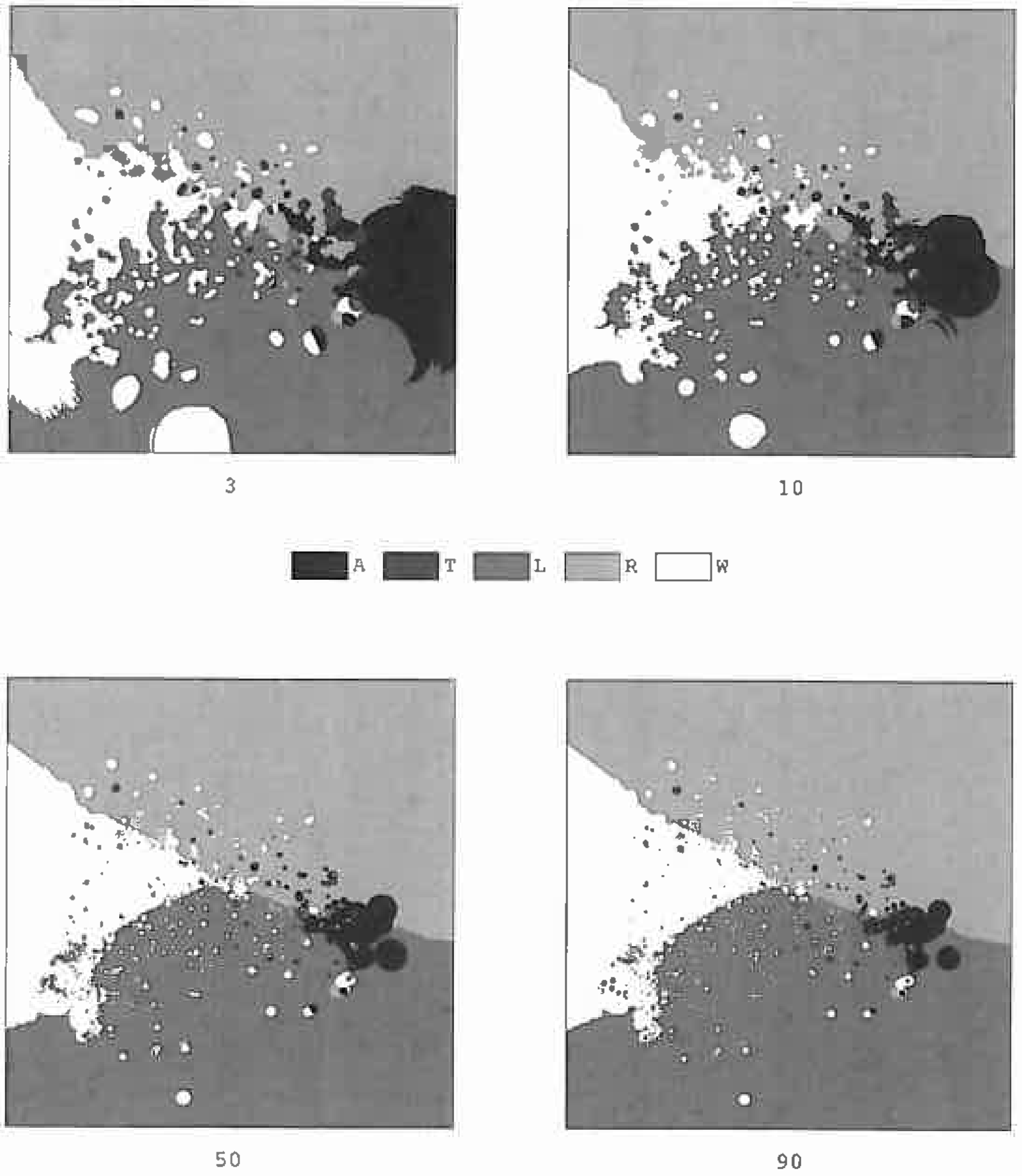

Figure 20: WSNN class regions, for α = 3, 10, 50, and 90

6.5 Neural Net Classifiers

6.5.1 Multi-Layer Perceptron (MLP)

This classifier is also known as a feedforward neural net. We have used an MLP with three layers (counting the inputs as a layer). It will be convenient to define the following notation:

$$\begin{aligned}
N^{(i)} &= \text{number of nodes in } i^{\text{th}} \text{ layer } (i = 0, 1, 2) \\
f(x) &= 1/(1 + e^{-x}) = \text{sigmoid function} \\
b_i^{(k)} &= \text{bias weight of } i^{\text{th}} \text{ node of } k^{\text{th}} \text{ layer} \\
w_{ij}^{(k)} &= \text{weight connecting } i^{\text{th}} \text{ node of } k^{\text{th}} \text{ layer to } j^{\text{th}} \text{ node of} \\
&\ (k-1)^{\text{th}} \text{ layer } (k = 1, 2; 1 \le i \le N^{(k)}; 1 \le j \le N^{(k-1)}) \\
\mathbf{x} &= (x_1, \ldots, x_n)^{\mathrm{T}} = \text{ a feature vector}
\end{aligned}$$

Note that

$$\begin{aligned}
N^{(0)} &= \text{number of input nodes} = n = \text{number of features} \\
N^{(1)} &= \text{number of hidden nodes} \\
N^{(2)} &= \text{number of output nodes} = N = \text{number of classes}
\end{aligned}$$

The discriminant functions are then of the form

$$D_i(\mathbf{x}) = f\left(b_i^{(2)} + \sum_{j=1}^{N^{(1)}} w_{ij}^{(2)} f\left(b_j^{(1)} + \sum_{k=1}^{N^{(0)}} w_{jk}^{(1)} x_k\right)\right).$$

For the training of the weights of this network, a reasonable procedure is the use of an optimization algorithm to minimize the mean-squared-error, over the training set, between the discriminant values actually produced and "target discriminant values" consisting of the appropriate strings of 1's and 0's as defined by the actual classes of the training examples. For example, if a training feature vector is of class 2, then its target vector of discriminant values is set to (0, 1, 0, 0, 0). It is more feasible to minimize this kind of an "error function" than to attempt to directly minimize the number of incorrectly classified training examples, since the latter number will take on only relatively few values and is a discontinuous "step function". In fact, the error function that is minimized is defined to contain an additional "regularization" term in addition to the mean-squared discriminant error. This term consists of a factor times the average of the squares of the weights. The motivation for including the regularization term is that the resulting weights will be somewhat controlled in magnitude, and that this will increase the generalization ability of the network. The goal of training is to produce a network that will "generalize" well, that is, accurately classify new examples that were not part of the training set; even if the training algorithm produces weights that result in perfect classification of the training examples, this is no guarantee that the network will generalize well.

Networks of the MLP type are the most commonly used "neural nets" in use today, and they are usually trained using a "backpropagation" algorithm [19]. However, we have used a "scaled conjugate gradient" training method instead [20, 21, 22, 23]. The procedure trains the network much faster than backpropagation and produces networks that generalize as well as or better than those trained by backpropagation. Figure 21 shows MLP class regions resulting from the use of 1, 2, 24, and 80 hidden nodes.

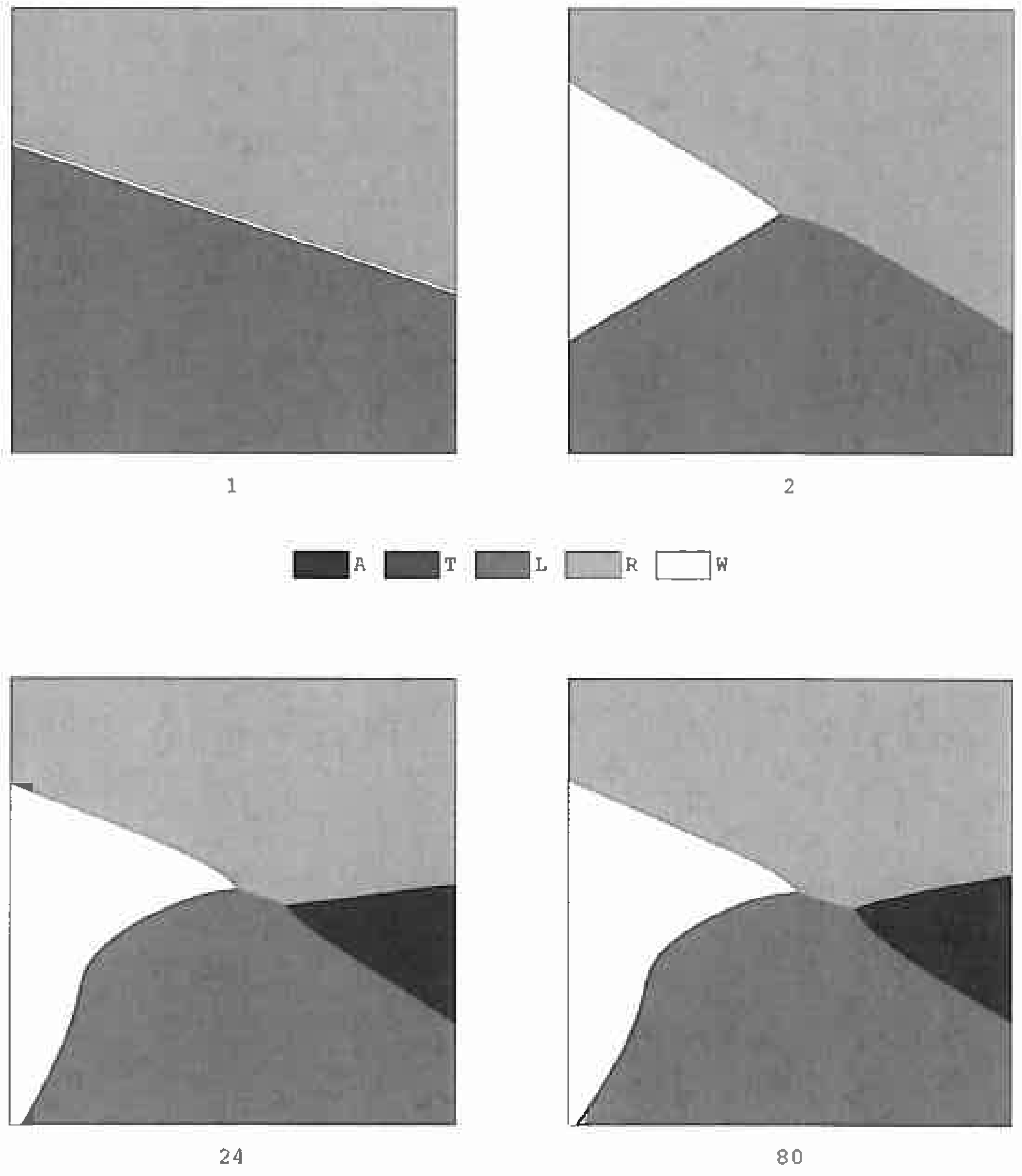

Figure 21: MLP class regions, for 1, 2, 24, and 80 hidden nodes

6.5.2 Radial Basis Functions (RBF1 and RBF2)

Neural nets of the Radial Basis Functions type get their name from the fact that they are built from radially symmetric Gaussian functions of the inputs. Actually, the RBF nets discussed here use Gaussian functions that are more general than radially symmetric functions: their constant potential surfaces are ellipsoids whose axes are parallel to the coordinate axes, whereas radially symmetric Gaussian functions have spherical constant potential surfaces. However, the name Radial Basis Functions has become customary for any neural net that uses Gaussian functions in its first layer.

We have experimented with RBF networks of two types, which will be denoted RBF1 and RBF2. The following notation will be convenient:

$$\begin{aligned}
N^{(i)} &= \text{number of nodes in } i^{\text{th}} \text{ layer } (i = 0, 1, 2) \\
\mathbf{c}^{(j)} &= \text{center vector of } j^{\text{th}} \text{ hidden node } (1 \le j \le N^{(1)}) \ (\mathbf{c}^{(j)} \in \mathbf{R}^n) \\
&= (c_1^{(j)}, \ldots, c_n^{(j)})^{\mathrm{T}} \\
\boldsymbol{\sigma}^{(j)} &= \text{width vector of } j^{\text{th}} \text{ hidden node } (1 \le j \le N^{(1)}) \ (\boldsymbol{\sigma}^{(j)} \in \mathbf{R}^n) \\
&= (\sigma_1^{(j)}, \ldots, \sigma_n^{(j)})^{\mathrm{T}} \\
\theta_j &= \text{bias weight of } j^{\text{th}} \text{ hidden node (RBF2 only)} \\
f(x) &= 1/(1 + e^{-x}) = \text{sigmoid function} \\
b_i &= \text{bias weight of } i^{\text{th}} \text{ output node } (1 \le i \le N^{(2)}) \\
w_{ij} &= \text{weight connecting } i^{\text{th}} \text{ output node to } j^{\text{th}} \text{ hidden node } (1 \le i \le N^{(2)}; 1 \le j \le N^{(1)}) \\
\mathbf{x} &= (x_1, \ldots, x_n)^{\mathrm{T}} = \text{ a feature vector}
\end{aligned}$$

Each hidden node computes a radial basis function. For RBF1, these functions are of the exponential form

$$o_j(\mathbf{x}) = \exp\left(- \sum_{k=1}^{N^{(0)}} \left(x_k - c_k^{(j)}\right)^2 / \left(\sigma_k^{(j)}\right)^2 \right),$$

and for RBF2, they are of the sigmoidal form

$$o_j(\mathbf{x}) = f\left(-\theta_j - \sum_{k=1}^{N^{(0)}} \left(x_k - c_k^{(j)}\right)^2 / \left(\sigma_k^{(j)}\right)^2 \right).$$

For either type of RBF, the i^{th} discriminant function is the following function of the radial basis functions:

$$D_i(\mathbf{x}) = f\left(b_i + \sum_{j=1}^{N^{(1)}} w_{ij} o_j(\mathbf{x}) \right).$$

Packing both layers of the neural net into a single equation for the discriminant function results in the following for RBF1:

$$D_i(\mathbf{x}) = f\left(b_i + \sum_{j=1}^{N^{(1)}} w_{ij} \exp\left(- \sum_{k=1}^{N^{(0)}} \left(x_k - c_k^{(j)}\right)^2 / \left(\sigma_k^{(j)}\right)^2 \right) \right).$$

and the following for RBF2:

$$D_i(\mathbf{x}) = f\left(b_i + \sum_{j=1}^{N^{(1)}} w_{ij} f\left(-\theta_j - \sum_{k=1}^{N^{(0)}} \left(x_k - c_k^{(j)}\right)^2 / \left(\sigma_k^{(j)}\right)^2\right)\right).$$

The centers $\mathbf{c}^{(j)}$, widths $\boldsymbol{\sigma}^{(j)}$, hidden-node bias weights θ_j (RBF2 only), output-node bias weights b_i, and output-node weights w_{ij} may be collectively thought of as the trainable "weights" of the RBF network. They are trained as follows. First, a k-means clustering program is applied to each class-set of the training patterns in turn, that is, each set of all training patterns that are of a particular class. This program separates the class-set into clusters of similar patterns. The mean vectors of these clusters are used as initial values for the center vectors $\mathbf{c}^{(j)}$. To initialize the width vectors $\boldsymbol{\sigma}^{(j)}$, we simply set all components of all width vectors to a single positive value: good values for this initial width component may be found by trial and error. The initial values of the output-node weights are set in such a way that each output node is connected with a positive weight to hidden nodes of its class (that is, hidden nodes whose initial center vectors are means of clusters from its class), and connected with a negative weight to hidden nodes of other classes.

The k-means algorithm and the above rules are used to set the initial values of the weights. Then, the weights are optimized using a conjugate-gradient algorithm, which is the same algorithm we have used to optimize the weights of the MLP network. Figure 22 shows RBF1 class regions resulting from the use of 1, 2, 4, and 6 hidden nodes per class, and Figure 23 shows RBF2 class regions for the same numbers of hidden nodes per class.

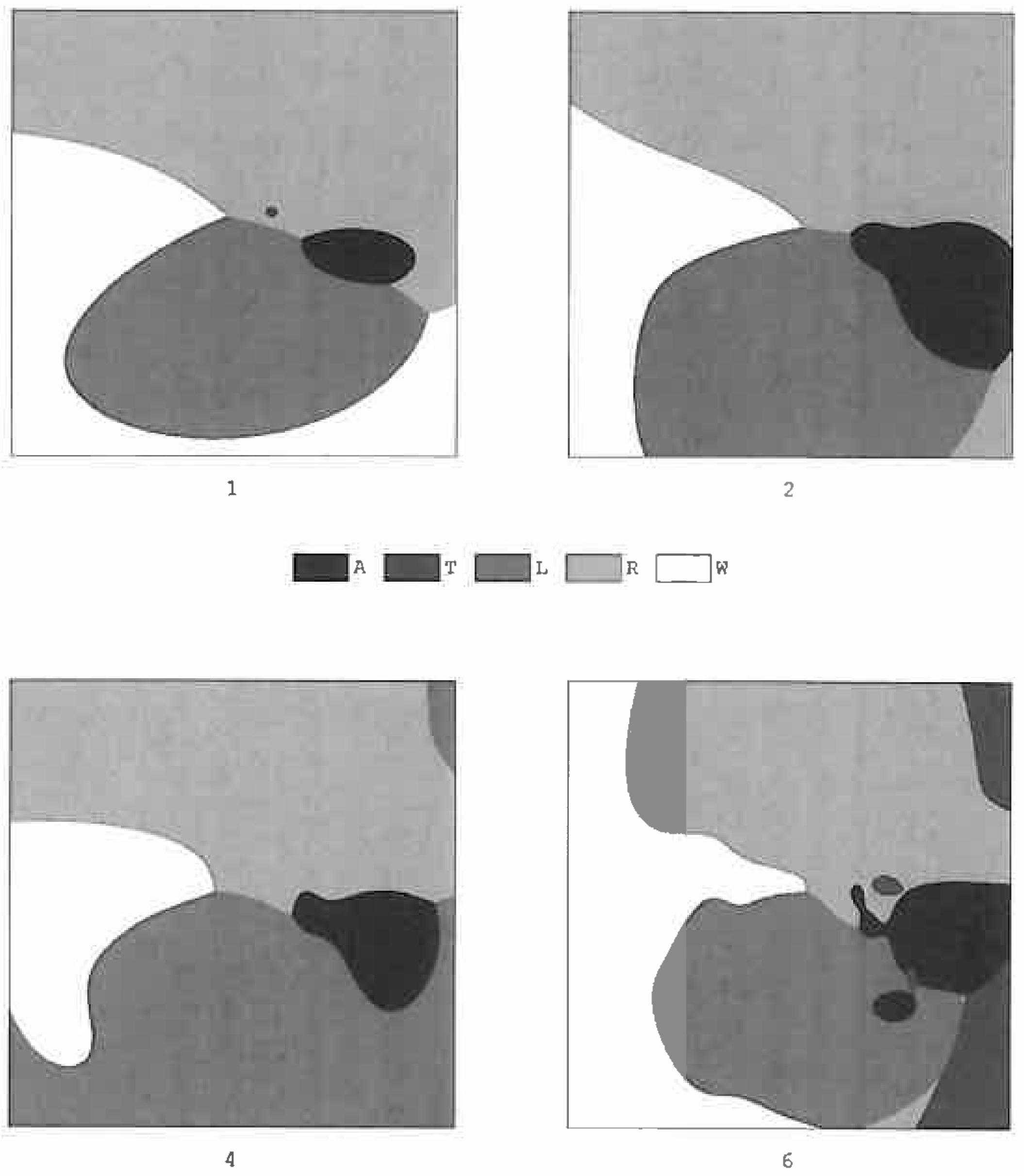

Figure 22: RBF1 class regions, for 1, 2, 4, and 6 hidden nodes per class

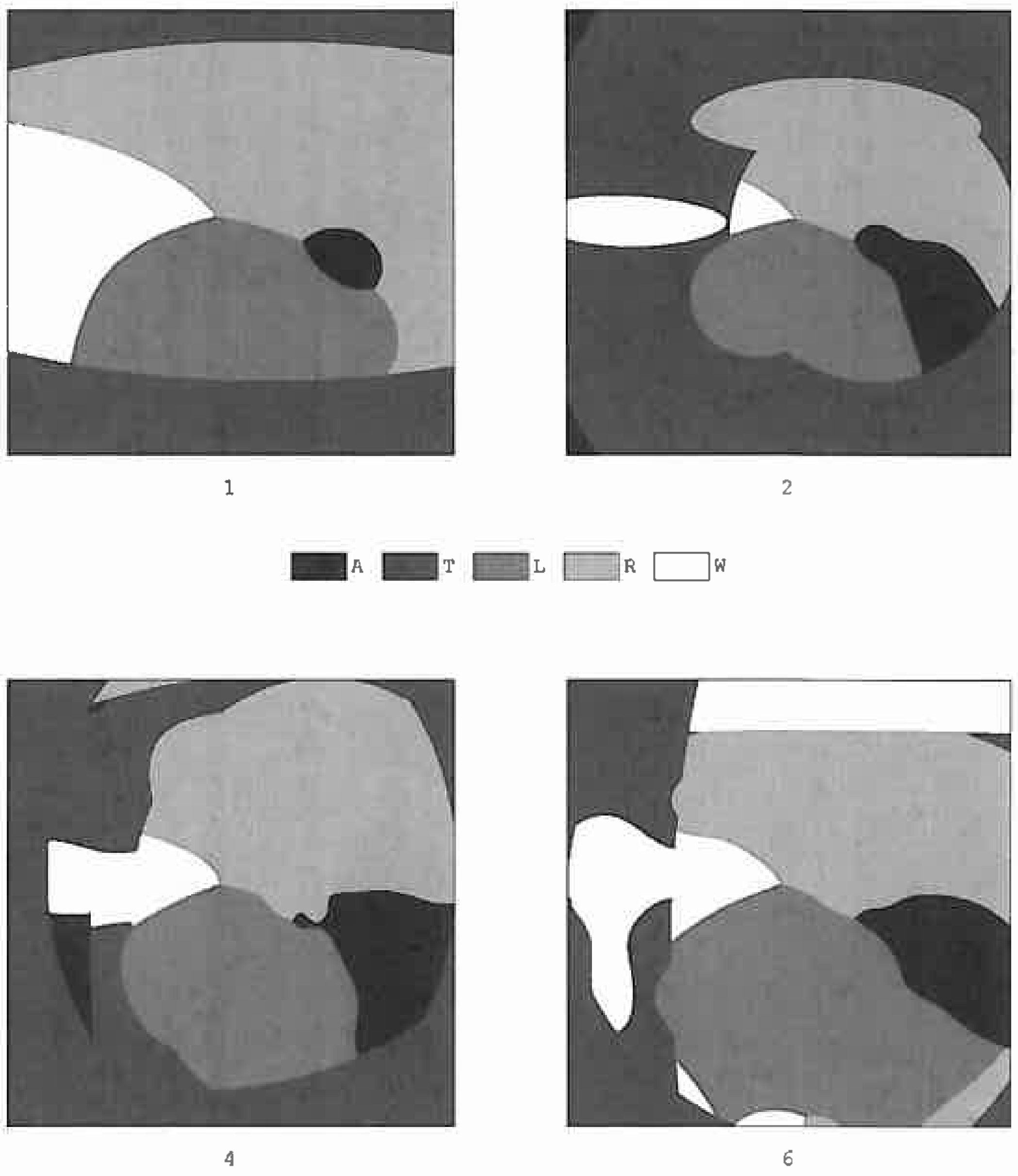

Figure 23: RBF2 class regions, for 1, 2, 4, and 6 hidden nodes per class

6.5.3 Probabilistic Neural Net (PNN)

This classifier is promulgated in a recent paper by Specht [24]. Each training example becomes the center of a kernel function which takes its maximum at the example and recedes gradually as one moves away from the example in feature space. An unknown $\mathbf{x}$ is classified by computing, for each class i, the sum of the values of the class-i kernels at $\mathbf{x}$, multiplying these numbers by compensatory factors involving the estimated *a priori* probabilities, and picking the class whose resulting discriminant value is highest. Many forms are possible for the kernel functions; we have obtained our best results using radially symmetric Gaussian kernels. The resulting discriminant functions are of the form

$$D_i(\mathbf{x}) = \frac{\hat{p}(i)}{M_i} \sum_{j=1}^{M_i} \exp\left(-\frac{1}{2\sigma^2} d^2\left(\mathbf{x}, \mathbf{x}_j^{(i)}\right)\right),$$

where σ is a scalar "smoothing parameter" that may be optimized by trial and error. Figure 24 shows the PNN class regions resulting from the use of σ values of 0.14, 0.42, 1.00, and 2.12. Notice that a small σ value produces very complex class regions similar to those of 1-NN, and that as σ is increased, the regions become simpler.

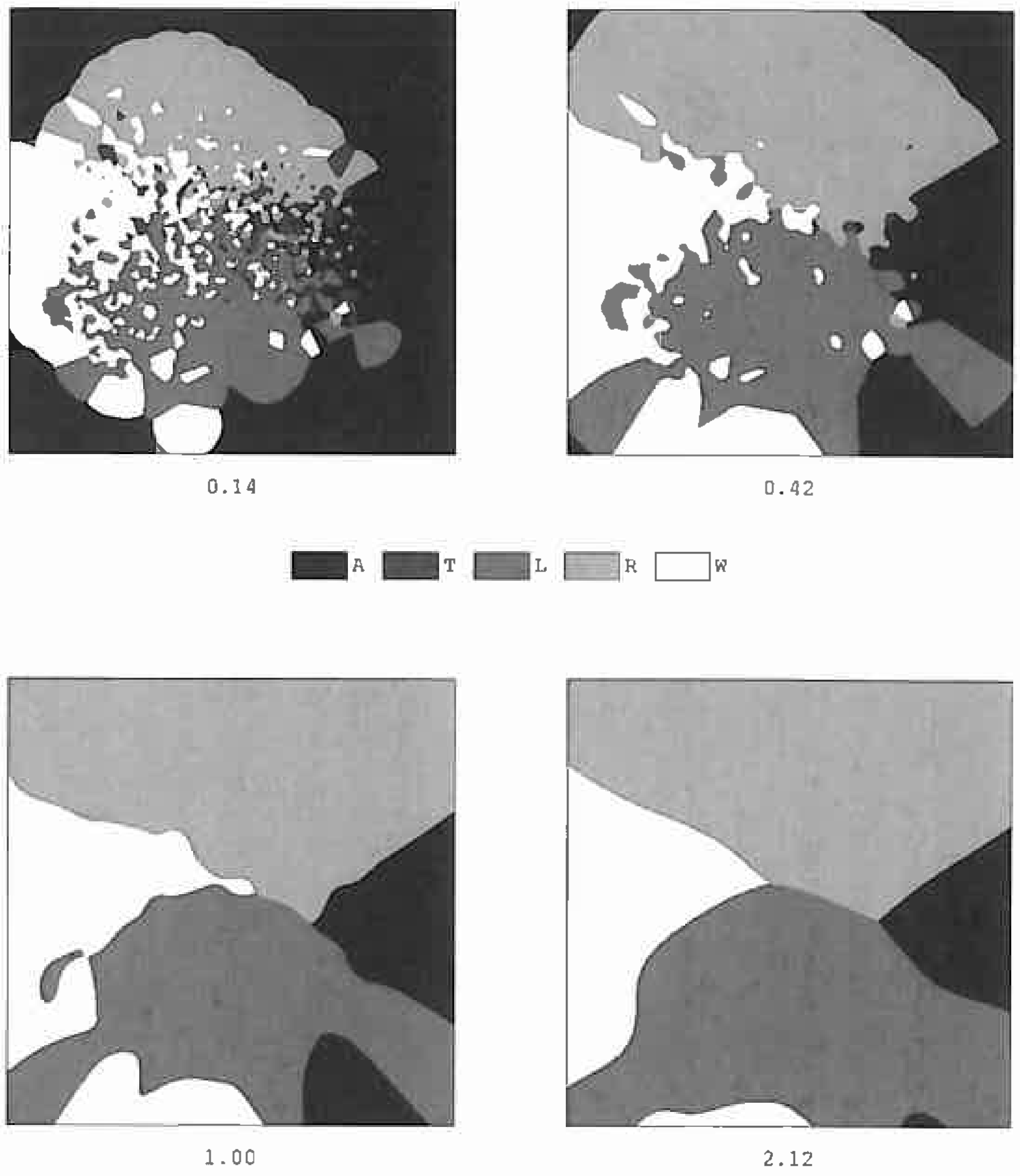

Figure 24: PNN class regions, for $\sigma = 0.14$, 0.42, 1.00, and 2.12

6.6 List of Discriminant Functions

The following list is included for comparison of the functional forms.

EMD:

$$D_i(\mathbf{x}) = -d^2(\mathbf{x}, \mathbf{m}_i)$$

QMD:

$$D_i(\mathbf{x}) = -(\mathbf{x} - \mathbf{m}_i)^{\mathrm{T}} \mathbf{S}_i^{-1} (\mathbf{x} - \mathbf{m}_i)$$

NRML:

$$D_i(\mathbf{x}) = \log \hat{p}(i) - \frac{1}{2} \log |\mathbf{S}_i| - \frac{1}{2} (\mathbf{x} - \mathbf{m}_i)^{\mathrm{T}} \mathbf{S}_i^{-1} (\mathbf{x} - \mathbf{m}_i)$$

1-NN:

$$D_i(\mathbf{x}) = - \min_{1 \le j \le M_i} d^2 \left(\mathbf{x}, \mathbf{x}_j^{(i)}\right)$$

WSNN:

$$D_i(\mathbf{x}) = \begin{cases} V_{\mathbf{x}}^{(i)} \left(\sum_{j \in \mathcal{S}_{\mathbf{x}}^{(i)}} d^2 \left(\mathbf{x}, \mathbf{x}_j^{(i)}\right)\right)^{-\frac{1}{2}} & \text{if } V_{\mathbf{x}}^{(i)} > 0 \\ 0 & \text{otherwise} \end{cases}, \text{ where}$$

$$\mathcal{S}_{\mathbf{x}}^{(i)} = \left\{ j \,\middle|\, 1 \le j \le M_i, d^2 \left(\mathbf{x}, \mathbf{x}_j^{(i)}\right) < \alpha \min_{1 \le k \le N, 1 \le p \le M_k} d^2 \left(\mathbf{x}, \mathbf{x}_p^{(k)}\right) \right\}, V_{\mathbf{x}}^{(i)} = |\mathcal{S}_{\mathbf{x}}^{(i)}|$$

MLP:

$$D_i(\mathbf{x}) = f \left(b_i^{(2)} + \sum_{j=1}^{N^{(1)}} w_{ij}^{(2)} f \left(b_j^{(1)} + \sum_{k=1}^{N^{(0)}} w_{jk}^{(1)} x_k \right)\right)$$

RBF1:

$$D_i(\mathbf{x}) = f \left(b_i + \sum_{j=1}^{N^{(1)}} w_{ij} \exp \left(- \sum_{k=1}^{N^{(0)}} \left(x_k - c_k^{(j)}\right)^2 / \left(\sigma_k^{(j)}\right)^2 \right)\right)$$

RBF2:

$$D_i(\mathbf{x}) = f \left(b_i + \sum_{j=1}^{N^{(1)}} w_{ij} f \left(-\theta_j - \sum_{k=1}^{N^{(0)}} \left(x_k - c_k^{(j)}\right)^2 / \left(\sigma_k^{(j)}\right)^2 \right)\right)$$

PNN:

$$D_i(\mathbf{x}) = \frac{\hat{p}(i)}{M_i} \sum_{j=1}^{M_i} \exp \left(- \frac{1}{2\sigma^2} d^2 \left(\mathbf{x}, \mathbf{x}_j^{(i)}\right)\right)$$

7 Maximum Finder and Rejector

The "Maximum Finder" component is very simple: it finds the i for which $D_i(\mathbf{x})$, the i^{th} discriminant function evaluated at the feature vector $\mathbf{x}$, is maximal, and outputs i as the hypothesized class of the fingerprint of which $\mathbf{x}$ is the feature vector.

Operating in parallel with the Maximum Finder, and using the same data – the outputs of the bank of discriminant functions – is the Rejector component. The motivation for a rejection mechanism is that by refusing to classify some "ambiguous" fingerprints (which must then be classified manually), the automatic classifier can achieve a lower substitutional error rate on those fingerprints which it does classify.

The rejectors used in our experiments are of the following form. The outputs of the discriminant functions are fed to a "confidence function", which produces a number that is treated as if it were a measure of reliability of the classification decision made by the Maximum Finder. In other words, fingerprints that produce high values of the confidence function are considered to be more likely to have been assigned correct classes than those that produce lower confidence values. The following confidence function (referred to as confidence function 2, for historical reasons) often produces good results: define its value to be the highest discriminant-function value minus the second-highest value. This is intuitively reasonable, since it will assign low confidence to fingerprints that are near a hypothetical class boundary. Another successful confidence function (number 4) is the same as function 2 except that it first "normalizes" the discriminant-function values by dividing them by their sum. To reject some fraction of the fingerprints presented, one simply sets a "confidence threshold" and then rejects all fingerprints whose confidence values are below the threshold. Higher thresholds result in rejecting greater percentages of the presented fingerprints and also result in correctly classifying higher percentages of the accepted fingerprints.

8 Comparative Error Rates of the Classifiers

8.1 Realistic Scoring Using Balanced Test Set

The test set used in these experiments – the second rollings of Special Database 4 – consists of an equal number (400) of each of the five classes of fingerprints. Because naturally occurring prints have a very unequal distribution into classes, it would be a mistake to use the percentage of this test set incorrectly classified as an estimate of the probability that a classifier will incorrectly classify a naturally occurring print. Instead, the following calculations are used to produce the test "score" (estimated probability of incorrect classification). For each class i, the number of the 400 class-i test prints (i.e. test prints whose actual class is i) that were incorrectly classified, is counted; this count is denoted w_i. Clearly, $w_i/400$ may be used as an estimate of the conditional probability of incorrect classification of a print given that the actual class of the print is i. Therefore,

$$\sum_{i=1}^{N} \hat{p}(i) w_i / 400$$

may be used as an estimate of the probability of incorrect classification. Accuracy "scores" mentioned below are these estimated probabilities, expressed as percentages.

Table 1 shows the lowest error rate that was obtained for each type of classifier. Also listed, for each classifier type, are the optimal settings that were found for other parameters: number of K-L features used; type of training set – the full "balanced" set of 400 prints of each class, or a "natural" set produced by discarding some of the training prints so as to cause the frequencies to be approximately equal to the estimated *a priori* probabilities; and, for some of the classfier types, another adjustable parameter or a number of hidden nodes. Table 2 shows, for each classifier type, the lowest error rates that were obtained for each of several numbers of features; it is clear that the optimal number of features is not the same for all of these classifier types.

Classifier	Error %	No. of features	Training set	Other parameter values
EMD	26.2	80	balanced	–
QMD	12.8	16	balanced	–
NRML	11.3	28	balanced	–
1-NN	9.0	96	natural	–
WSNN	8.9	96	natural	$\alpha = 1.09$
MLP	8.2	64	natural	64 hidden nodes
RBF1	8.3	112	natural	70 hidden nodes
RBF2	8.1	64	natural	110 hidden nodes
PNN	7.2	112	balanced	$\sigma = 2.26$

Table 1: Lowest error percentages for the various classifier types, and the parameters that produced them

	Number of features						
Classifier	16	32	48	64	80	96	112
EMD	26.9	26.6	26.4	26.3	26.2	26.3	26.3
QMD	12.8	15.6	18.0	20.1	20.7	21.6	23.0
NRML	13.5	12.8	16.8	18.1	19.7	20.7	23.0
1-NN	10.7	9.6	9.7	9.3	9.1	9.0	9.3
WSNN	10.3	9.3	9.1	9.1	8.9	8.9	9.0
MLP	9.1	8.8	8.6	8.2	8.2	8.4	8.5
RBF1	9.8	8.6	9.1	8.8	8.8	8.5	8.3
RBF2	10.7	9.5	10.7	8.1	8.8	8.4	8.2
PNN	9.0	7.9	7.5	7.6	7.4	7.3	7.2

Table 2: Error percentages for classifiers and numbers of features. NRML produced a smaller error percentage for a number of features not in the table: 11.3, for 28 features.

8.2 Error vs. Reject Curves

The above weighted scoring method can be extended to produce realistic scores at various levels of rejection, thereby producing an "error vs. reject" curve that is really a graph of an "estimated conditional probability of incorrect classification given acceptance" versus an "estimated probability of rejection". First, define some notation:

$$\begin{aligned}
p(r) &= \text{probability of rejection} \\
p(r|i) &= \text{conditional probability of rejection, given that actual class is } i \\
p(a) &= \text{probability of acceptance} = 1 - p(r) \\
p(a,w) &= \text{probability of acceptance and incorrect classification} \\
p(a,w|i) &= \text{conditional probability of acceptance and incorrect} \\
& \text{classification, given that actual class is } i \\
p(w|a) &= \text{conditional probability of incorrect classification given acceptance} \\
&= \frac{p(a,w)}{p(a)} \\
&= \frac{p(a,w)}{1-p(r)} \\
r_i &= \text{number of the 400 class-}i\text{ prints that were rejected} \\
w_i &= \text{number of the 400 class-}i\text{ prints that were accepted and incorrectly classified}
\end{aligned}$$

Clearly $r_i/400$ may be used as an estimate of $p(r|i)$, and $w_i/400$ as an estimate of $p(a,w|i)$. Therefore

$$p(r) = \sum_{i=1}^{N} p(i)p(r|i) \approx \sum_{i=1}^{N} \hat{p}(i)r_i/400,$$

$$p(a,w) = \sum_{i=1}^{N} p(i)p(a,w|i) \approx \sum_{i=1}^{N} \hat{p}(i)w_i/400, \text{ and}$$

$$p(w|a) = p(a,w)/(1-p(r)) \approx \frac{\sum_{i=1}^{N} \hat{p}(i)w_i/400}{1-\sum_{i=1}^{N} \hat{p}(i)r_i/400}.$$

Figures 25 through 27 are error vs. reject curves for some of the classifiers, produced by plotting the above estimates of $p(w|a)$ against the estimates of $p(r)$. The confidence functions that were used are indicated.

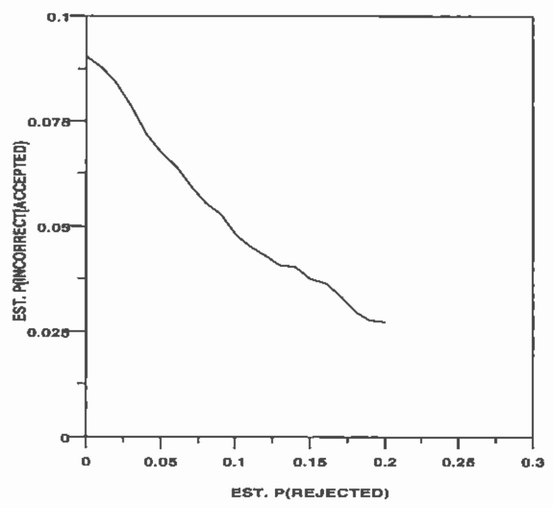

Figure 25: Error vs. reject for 1-NN; confidence function 2

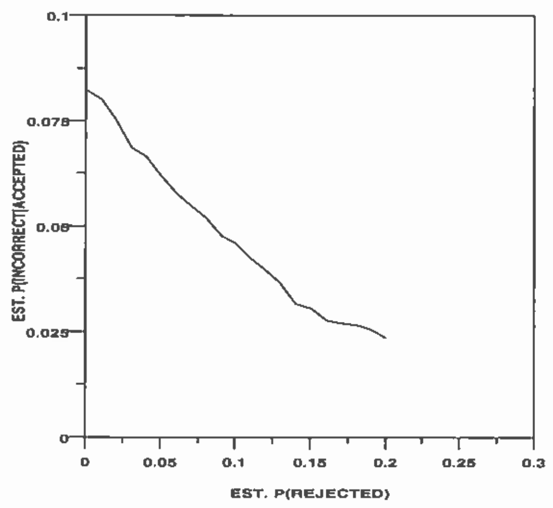

Figure 26: Error vs. reject, MLP, confidence function 2

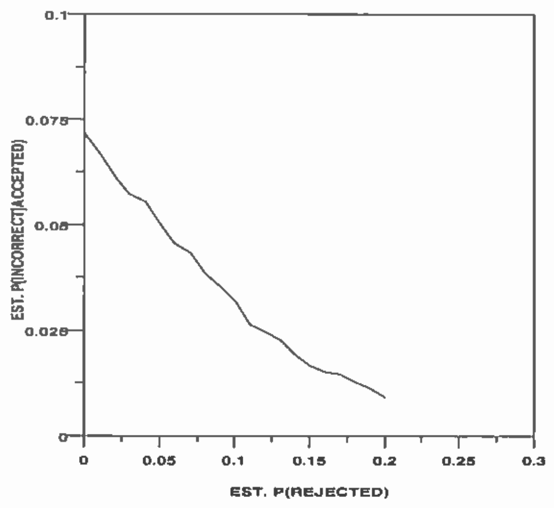

Figure 27: Error vs. reject, PNN, confidence function 4

9 Future Plans

The Image Recognition Group will continue work on automatic fingerprint classification (among other subjects), with the aim of improving the accuracy of the experimental classifiers while keeping memory and time costs under control. More generally, we will continue our attempts to characterize the difficulties inherent in automatic fingerprint classification, and will continue to increase our knowledge about the feasibility of various classification methods as applied to this task. The group is involved in other image processing work, in particular the classification of handwritten characters; cross-fertilization between the areas of fingerprint and character classification has proved beneficial so far, and will probably continue to be so. (The classifier algorithms discussed here are applicable to any classification task; they are not specific to fingerprints.)

Two of the possible areas of future research for this group can be mentioned here. One is the use of "decision tree" classifiers and their combination (discussed by Sethi [25]) with neural nets. Indications are that such combination tree/net methods have advantages compared to either tree or net methods alone. Another area of interest consists of methods for reducing the costs of many-prototype classifiers. In particular, classification memory and time can be saved by reducing the prototype set to a well-chosen subset of the training set, and time can be saved by using algorithms that do not require computing the distances of the unknown from *all* stored prototypes. Nearest-neighbor classifiers have been the subject of decades of research (see, for example, Dasarathy's collection of papers [26]), and it is likely that many of the cost-reducing techniques developed over the years can be applied to other classifiers involving numerous prototypes, in particular PNN.

10 Summary and Conclusions

Pattern-level classification of fingerprints is a nontrivial task even for humans; it appears to be very difficult to automate, even using a massively parallel computer. We have performed many experiments in fingerprint classification, using "statistical" and "neural net" approaches. When our features are used – that is, a fixed pattern of registration-corrected local ridge directions dimension-reduced by a K-L transform – the differences in classification accuracy resulting from varying the classifier *per se* between the types reported here, are significant but not dramatic. Our test results indicate that the accuracy differences are fairly small when certain "reasonable" classifiers are compared. Of course, it is possible that some other classifier algorithm that we have not tested, or that has not yet been invented, will produce much better results using the same features as input; research and testing must continue. The best accuracy results we obtained were for the Probabilistic Neural Net (PNN) method, which achieved an error rate of 7.2% (3.2% at a 10% rejection level). Improvements in accuracy seem likely to be attainable by using a larger training set and better features, even if the same PNN algorithm is used for the classifier itself. Production of special-purpose hardware is probably required if fingerprint classifiers are to be implemented at the lowest cost and with the highest possible classification speed; the algorithms discussed here are naturally parallel and their implementation in special hardware would probably not be very difficult.

References

[1] *The Science of Fingerprints.* U. S. Department of Justice. Washington. DC. 1984.

[2] K. S. Fu. *Syntactic Pattern Recognition.* Prentice-Hall. Englewood Cliffs. NJ. 1982.

[3] B. Moayer and K. S. Fu. A syntactic approach to fingerprint pattern recognition. *Pattern Recognition.* 7:1-23. 1975.

[4] B. Moayer and K. S. Fu. An application of stochastic languages to fingerprint pattern recognition. *Pattern Recognition.* 8:173-179. 1976.

[5] B. Moayer and K. S. Fu. A tree system approach for fingerprint pattern recognition. *IEEE Trans. on Computers.* 25:262-274. 1976.

[6] K. Rao and K. Balck. Type classification of fingerprints: A syntactic approach. *IEEE Trans. on Patt. Anal. Mach. Intell.*. 2:223-231. 1980.

[7] K. S. Fu and T. L. Booth. Grammatical inference: Introduction and survey - part i. *IEEE Trans. on Systems. Man. and Cybernetics.* 5:95-111. January 1975.

[8] K. S. Fu and T. L. Booth. Grammatical inference: Introduction and survey - part ii. *IEEE Trans. on Systems. Man. and Cybernetics.* 5:409-423. July 1975.

[9] C. L. Giles. C. B. Miller. D. Chen. H. H. Chen. G. Z. Sun. and Y. C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation.* 4(3):380. 1992.

[10] C. L. Wilson. G. T. Candela. P. J. Grother. C. I. Watson. and R. A. Wilkinson. Massively Parallel Neural Network Fingerprint Classification System. Technical Report NISTIR 4880. National Institute of Standards and Technology. July 1992.

[11] P. Baldi and Y. Chauvin. Neural networks for fingerprint matching and classification (abstract). In *Proceedings. Neural Information Processing Systems Conference. Vail. Colorado.* 1992.

[12] C. I. Watson and C. L. Wilson. Fingerprint database. *National Institute of Standards and Technology.* Special Database 4, **FPDB**, April 18. 1992.

[13] R. M. Stock and C. W. Swonger. Development and evaluation of a reader of fingerprint minutiae. *Cornell Aeronautical Laboratory.* Technical Report CAL No. XM-2478-X-1:13-17. 1969.

[14] J. H. Wegstein. An automated fingerprint identification system. *National Institute of Standards and Technology.* NBS Special Publication 500-89. Febuary 1982.

[15] Anil K. Jain. *Fundamentals of Digital Image Processing.* chapter 5.11. pages 163-174. Prentice Hall Inc., prentice hall international edition. 1989.

[16] C. L. Wilson, G. T. Candela, and C. I. Watson. Neural network fingerprint classification. *Journal of Artificial Neural Networks*, 1992. to be published.

[17] Nils J. Nilsson. *Learning Machines: Foundations of trainable pattern-classifying systems*, chapter 3, page 45. McGraw-Hill Book Company, 1965.

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

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 332:533–536, 1986.

[20] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Computer Journal*, 7:149–154, 1964.

[21] E. M. Johansson, F. U. Dowla, and D. M. Goodman. Backpropagation learning for multi-layer feed-forward neural networks using the conjugate gradient method. *IEEE Transactions on Neural Networks*, 1991.

[22] M. F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. Technical Report PB-339, Aarhus University, 1990.

[23] J. L. Blue and P. J. Grother. Training Feed Forward Networks Using Conjugate Gradients. In *Conference on Character Recognition and Digitizer Technologies*, volume 1661, pages 179–190, San Jose California, February 1992. SPIE.

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

[25] I. K. Sethi. Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78:1605–1613, 1990.

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