# Handwritten Digit Recognition via Deformable Prototypes

Trevor Hastie
*Statistics and Data Analysis Research Department*
*AT&T Bell Laboratories*
*Murray Hill, New Jersey*

Robert Tibshirani
*Department of Preventive Medicine and Biostatistics*
*and Department of Statistics*
*University of Toronto*
*Toronto, Ontario*

Draft—22 July, 1992

©AT&T Bell Laboratories

**Abstract**

We present a new method for classifying handwritten characters, in particular digits. In our approach each character in the alphabet is represented by a prototype, which is a piecewise-linear curve. Prototypes are fitted to a point-set representation of the image by least squares, allowing a free affine transformation in $R^2$. Every prototype is fitted to a given image, and then classification is based on the matching statistics. The procedure is demonstrated on a large database of handwritten zipcodes.

## 1 Introduction

A current "hot" topic in the field of pattern recognition is the automatic reading of handwritten addresses and zip-codes from envelopes. Figure 1 shows the result of an automatic segmentation of a 5 digit handwritten zip code, into 5 pieces. The objective of the segmentation is to divide the zip code in such a way that each piece is likely to be a digit.

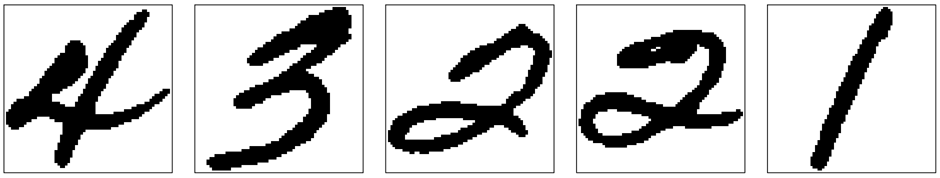

Figure 1: *Segmented zip code*

The problem addressed in this paper is that of recognizing the digit from segmented pieces like those in Figure 1. In this paper we propose an automatic procedure for this problem, and test it on a large database consisting of over 7000 training digits and 2007 test digits.

The data we use (kindly supplied by J. Bromley) are a subset of the Buffalo Postal-Address Image Database, constructed under contract to the United States Postal Service Office of Advanced Technology in 1987. Personnel from the State University of New York at Buffalo ran a data collection station at the Main Buffalo Post Office. The station consisted of an Eikonix 850 CCD digitizing camera, a Sun-3/260G workstation, a 689MB disk drive, and a half inch tape drive. The digits were extracted with an automatic segmentation program, and the correct classification of each image is known. These data are described in Le Cun et al. (1990), who implement an alternative digit classification procedure using a neural network.

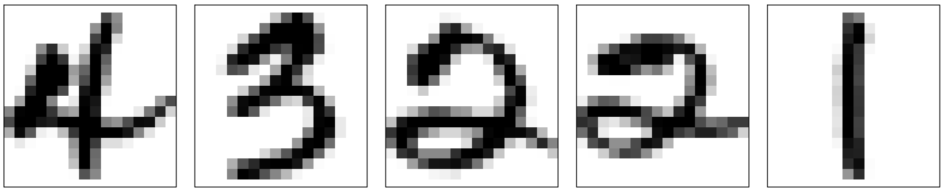

Figure 2: $16 \times 16$ *size and orientation normalized versions of the digits in Figure 1.*

Each of the binary images has a slightly different dimension depending on the size of the handwritten digits, but typically around $50 \times 50$ pixels per image. Any classifier would have to adjust for this size variation, and also for other spatial distortions such as rotations and shearing. Le Cun et al. (1990) correct for such distortions by an additional preprocessing step which normalizes the size and orientation of the images, and results in a reduced $16 \times 16$ grey-scale image. Figure 2 shows these normalized versions of the digits in Figure 1. Le Cun et al. (1990) use the 256 pixel values as the inputs to a multilayer-neural-network model, which is then trained to classify the images. They report overall misclassification rates on test data of just under 5%.

The size/orientation normalization step can be costly (LeCun et al. 1990), and our approach is designed to avoid it. We affinely transform a number of class *prototypes* to best match a given image, and assign a digit label based on the matching statistics.

Our procedure consists of the following steps:

1. The image is represented by a set of points with coordinates in $R^2$, corresponding to the dark pixels.

2. Each of a set of prototypes representing the digit classes are fitted to the points by least squares.

3. For each fitted prototype, statistics are computed that reflect the closeness of the match.

4. During the training phase, all the matching statistics are used as input to train a flexible classification procedure. During testing these features are fed to the trained classification procedure, and class predictions are made.

For comparison purposes we apply our procedure to the normalized digit images which were used by the competing approaches. We emphasize, however, that our approach works equally well without the costly normalization step.

In the following sections we describe each step in detail, and compare our results to those obtained by others. While our classification results are within the acceptable ballpark, they are not the best seen on the particular test data set we use. Our procedure does have a speed advantage over some of the competing techniques, since each image has only to be matched to a small number of prototypes (13 at present) Section 6 discusses some alternative approaches to this problem. Although the focus here is on handwritten digit recognition, we hope that the proposed techniques will be applicable in other related areas.

## 2 Thresholding

As a pre-processing step, each image is converted to a binary point set. There are two cases of interest: the trivial case when the images are binary, and the less trivial case of greyscale images.

The unscaled images presented above are binary, and hence are easily converted to a point set. We simply assign lattice coordinates to the black pixels. In practice, it is convenient to thin this set down. Our procedure seems to work quite comfortably with 20-50 points; although it can work with more, the extra computational cost does not seem warranted. Our thinning filter consists of replacing each $3 \times 3$ block non-overlapping block of binary pixels by their average, which produces a greyscale image.

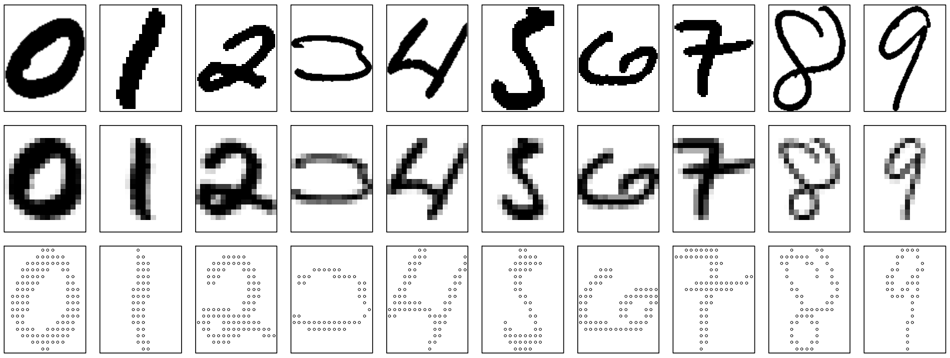

Figure 3: *A random selection of handwritten digits from the Buffalo database. The top row shows the digits as scanned, the second row shows the* $16 \times 16$ *size and orientation normalized versions of the same digits, and the bottom row shows the thresh-holded images represented as a point set.*

For a greyscale image we first discard all the white pixels. Next we compute a threshold, which we define to be the $\alpha$th quantile (default $\alpha = 0.4$) of the remaining greyscale values. All pixels with grey-scales greater than this threshold are retained as points with their corresponding lattice coordinates. By using the quantile as the threshold, we approximately produce the same number of points for a given digit class, even though the individual characters might have different thicknesses. Figure 3 shows some examples of the thresholding procedure.

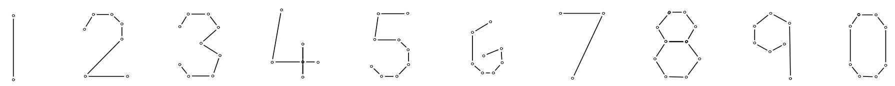

Figure 4: *Prototypes for the digits 0–9*

## 3 Fitting of prototypes

Each of the 10 digit-classes are represented by a prototype $P_j,\ j = 0, \ldots, 9$, each of which is a piecewise-linear two dimensional parametrized curve. These were constructed manually by tracing over enlarged printed digits using as few connected line-segments as possible. The 10 prototypes are displayed in Figure 4, with the knot points indicated. In some cases such as $P_4$, some backtracking would be required in order for the curve to remain continuous. Our procedure does not require continuity, so we can have breaks in the curve for cases such as this.

Our strategy is to deform the prototypes so that the image point-set matches the prototype as closely as possible. Denote by $X_{ik}$ the 2-vector consisting of the coordinates of the $k$th point for image $i$, $k = 1, 2, \ldots n_i$.

Let $P_j(s),\ s \in [0, 1]$ be the piecewise-linear (and piecewise-continuous) parametrized plane curve representing the prototype for digit $j$, with the scalar parameter $s$ indexing the points on the prototype. Note that for any fixed $s$, $P_j(s)$ is a two-vector. For each $j$ we can generate a family of affinely transformed prototypes

$$\mu + AP_j(s) \tag{1}$$

where $\mu$ is a 2-vector and $A$ a $2 \times 2$ real matrix. Then we seek values $\hat{\mu}, \hat{A}$ and $\hat{s}_1, \hat{s}_2, \ldots \hat{s}_{n_i}$ to minimize

$$G_i(j) = \frac{1}{n_i} \sum_{k=1}^{n_i} (X_{ik} - \mu - A_j P_j(s_k))^2 \tag{2}$$

Hence $\hat{\mu} + \hat{A}P_j(\hat{s}_k)$ is the point on the $j$th transformed prototype, closest to the image point $X_{ik}$.

Exact minimization of (2) is a difficult numerical problem: instead we settle for a local minimum. defined by an alternating algorithm. For fixed $s_1,\ s_2, \ldots s_{n_i}$, the minimizing values $\hat{\mu}$ and $\hat{A}$ are found by (multivariate) linear regression. For fixed $\mu$ and $A$, $\hat{s}_k$ is found by locating the closest line segment in the prototype to $X_{ik}$. The distance to a line segment is found by projection, which identifies either a point in the interior, or else one of the endpoints. An exhaustive search is made over all line-segments in the prototype, and the closest is identified. Since there are at most 11 line segments in our prototype set (8 has the most), this is no computational burden. This is similar to the algorithm of Hastie and Stuetzle (1989) for fitting principal curves to a set of points. The two steps are iterated until convergence: typically 3 or 4 iterations are sufficient. Figure 5 shows the iterations in a typical fitting operation.

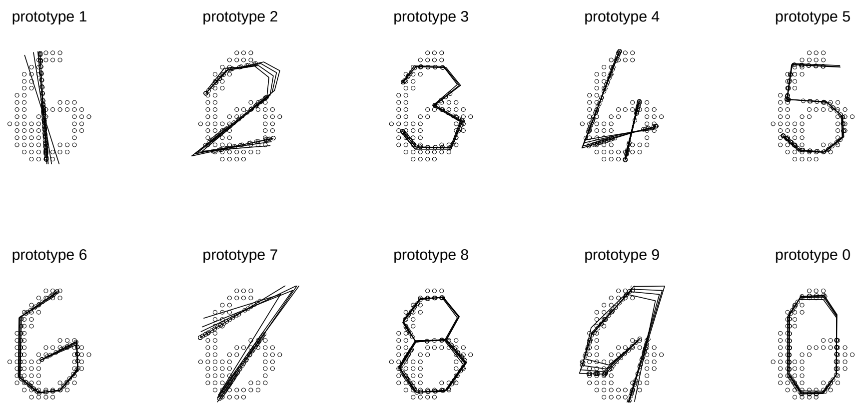


Figure 5: *Ten prototypes fitted to an image chosen at random from the training set. The different iterations are superimposed.*

In this manner, each of the 10 prototypes $P_j$, $j = 0, \ldots, 9$, were fit to all the training and test images.

There is not much of a computational burden in fitting a set of prototypes to an image. By adjusting the threshold parameter $\alpha$, we can control the number of points; in our case the median number is 60 with very few over 100. Each of the two steps in the alternating algorithm is linear in $n_i$ with a small multiplier, and we have found that a limit of 10 iterations is sufficient. The well fitting prototypes tend to converge quickly, and for those that fit poorly, it is not crucial that the fit be precise.

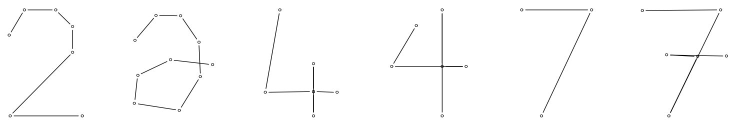

Figure 6: *Multiple Prototypes for some digits.*

### 3.1 Multiple Prototypes

There are two common ways a "4" is written—one is like the four printed here, the other is like the four in Figure 4. A characteristic that distinguishes them is the relative length of the legs. The affine transformations are not capable of mapping the one type into the other, and we found that this distinction accounted for a number of misclassifications. The former tend to be confused with "9"s. Similarly, some "7"s are crossed while others are not, and there is a looped versus non-looped "2". We augmented our set of prototypes with the additional version of each of these, and improved our misclassification rates by nearly 2% on the test data.

## 4 Matching Statistics

Let $\hat{G}_i(j)$ be the value of the criterion (2) when $P_j$ is fitted to image $i$. The most natural classification procedure would seem to be the minimum distance rule: classify image $i$ to be digit $j$ if $G_i(j) < G_i(j')$ for all $j' \neq j$. This approach resulted in a misclassification rate of 56.3% over the test set. Figure 7 shows two of the reasons for this poor performance.

The first panel shows the fit of $P_3$ to a "3", the second that of $P_8$ to the same "3". The goodness of fit for both of these (as measured by (2)) will be quite small, since part of $P_8$ looks like $P_3$. The least squares error does not reflect the fact some of $P_8$ is left *uncovered.* In the third panel, we show the fit of $P_8$ to a "1". As measured by (2), the fit will be judged a good one. Furthermore, most of $P_8$ is covered by the projected points. Because an affine transformation can "squash" any prototype to look like a "1", we cannot distinguish a "1" from other digits based on goodness of fit and coverage alone.

From these and other considerations, we extract several other features from each of the fits of a prototype to an image.

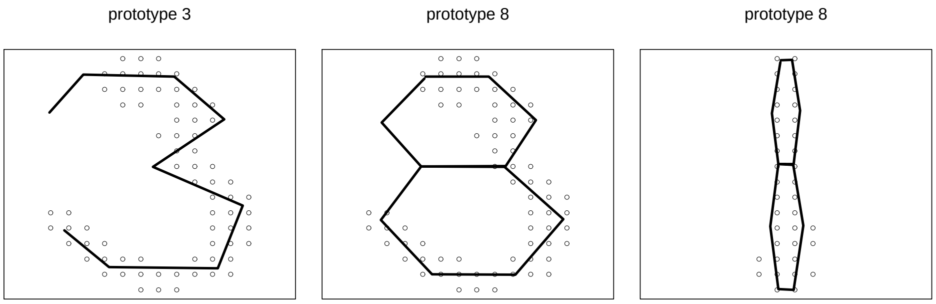


Figure 7: *$P_3$ and $P_8$ are fitted to a "3". As measured by the sum of squared deviations of the points from the prototype, both fits are equally good. But the "3" leaves much of $P_8$ unused. On the right, $P_8$ is squashed to fit a "1" The resulting fit is quite good but the severity of the transformation should be taken into account.*

### Coverage

Coverage measures the proportion of the prototype that is covered by the image. We define/compute it as follows: First all the (positive) gaps between adjacent fitted points on the prototype are computed, including the endpoints if it is not closed. The sum of the gaps is the arc-length of the transformed prototype[1]. We compute the 95% gap quantile, and deem all gaps greater than this quantile to be uncovered areas. We then compute the ratio of the total covered area to the arc-length. The coverage of $P_3$ in Figure 7 is 0.69 , while for $P_8$ it is 0.52; their sums-of-squares $G_i(j)$ are 0.33 and 0.35 respectively.

### Aspect ratio

This is simply the ratio of horizontal to vertical scaling of the prototype required to fit the image. This measure alone is sufficient to classify all the "1"s, when derived for the fit of, say, $P_8$. Any prototype with some width will have to be heavily squashed in the horizontal direction in order to adequately model a "1". In the first two panels the aspect ratio is about 1.25, while in the third $P_8$ is squashed with an aspect-ratio of 0.19 in order to make it fit a "1".

[1] Jumps at discontinuities are ignored in the calculations

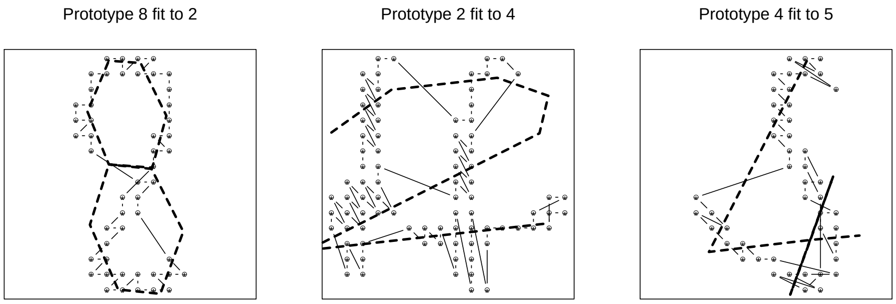


Figure 8: *Some situations where coverage and sum-of squares are not sufficient. The zigzag measure is illustrated by connecting all the points by straight lines in the sequence of their projections onto the prototype. The zigzag measure is the sum of these squared segments.*

### Rotation and Shear

These measures capture the other two aspects of the fitted transformation $\hat{A}_j$. We compute the QR decomposition of $\hat{A}_j$, a $2 \times 2$ matrix. Interpreting Q as a rotation, we retain the top left element which is the cosine of the rotation angle. The diagonal elements of $R$ represent the horizontal and vertical scaling, which are used to define the aspect ratio. We use the top-right element of $R$ to represent the shear.

### Zigzag and Template walk

There are some mismatched situations where both the sum-of-squares and coverage measures are within range for a digit, yet visually the fit is terrible. This can occur if the shape of the prototype is such that it is able to be close to all the points somewhere, and there are sufficient points to be able to cover it adequately. In this instance, points that are far apart in the image might project to nearby points on the prototype, or points that are close in the image might project to points that are far apart on the prototype. Figure 8 illustrates some of these situations. The measure we call *zigzag* illustrated in the figure is the total journey length in visiting the points in the sequence of their projections onto the prototype (in prototype order). This will emphasize cases where neighboring points on the prototype represent quite different areas of the image.

The *prototype walk* is the dual to zigzag. It measures the path length along the prototype in traveling between projected points in a sequence corresponding to a nearest-neighbor walk through the points. This latter measure is harder to compute since there is no well defined nearest neighbor path through the points. We use a simple approximation that exploits the raster nature of the points: traversing each row horizontally, we flag each pair with an immediate neighbor; the distance between the projections of all such pairs recorded. This is repeated using a vertical traversal, and the average pair distance is computed.

# 5 Training a Classifier based on Matching Statistics

All the measures can be viewed as "distances" of one sort or another, and if not, can be converted into distances. It might seem that these could be combined into a single "omnibus" distance for each prototype, and then used in a nearest-prototype classification scheme. Naive attempts at such combinations led to poor results. The "1" example illustrates why such an approach still does not use all the available information: the best feature for correctly identifying a "1" is the aspect ratio resulting from squashing any prototype but $P_1$ in order to fit it. Thus the measures associated with all the prototype fits carry potential discriminatory information. This led us to simply treat our measures as *features*, and feed them into an automatic classification procedure.

We report results using:

1. Linear Discriminant Analysis (LDA);
2. two extended versions of Linear Discriminant Analysis, each obtained by augmenting the original features with powers and cross-product terms and then fitting a LDA model:
   (a) LDA-1 uses the features and their squares, and
   (b) LDA-2 includes in addition some subjectively selected cross terms;
3. The Classification tree technique of Breiman et al. (CART, 1984) .
4. A flexible discriminant analysis (FDA) approach, described in Hastie, Tibshirani & Buja (1992), which represents the discriminant variables by additive models.

Table 1: *Classification results based on features obtained by fitting prototypes to handwritten digit images*

| Technique | Error Rates | | |
|---|---|---|---|
| | Training | Test-1 | Test-2 |
| LDA | 0.032 | 0.032 | 0.072 |
| CART | 0.041 | 0.077 | 0.124 |
| LDA-1 | 0.022 | 0.025 | 0.057 |
| LDA-2 | 0.014 | 0.023 | 0.055 |
| FDA | 0.014 | 0.029 | 0.060 |

Table 1 summarizes the results. In each case we trained the models on the first 4000 training images, and report the training error in column 1. We evaluated the models on two test sets: test-1 are 2000 additional images from the training set, but not used to train out models; test-2 are the 2007 test images designated as the "official" test set for these data. The LDA-2 method performed the best, with an error rate of 5.5% on the test data. Although the FDA technique selected fewer terms than the non-adaptive approaches, it had a slightly poorer misclassification rates. The classification tree had the worst performance of all. We suspect the reason is that it aims at interaction terms, and uses the data up rather quickly in modelling main effects; the success of the other methods indicate that the main effect terms are important.

The large difference between the Test-1 and Test-2 results can be explained. The former set is part of the training set, and has been "cleaned" of many poor examples. The latter set, on the contrary, includes about 40 extremely poor images that a human would be unable to classify. Although these represent a real life situation, they also put a limit on the performance we might expect from any classification scheme.

### 5.1 Class probabilities and rejection rules

A rule that classifies an image to one of the 10 digits may not be sufficient for some purposes. For example, suppose the digit recognition procedure is called as a primitive by a parent algorithm designed to recognize the entire five-digit zip code. Then some measure of confidence about the predicted digit class will be useful. For this purpose, estimates of the class probabilities are required. Furthermore, one might want to discard images for which the digit classification is not clear in order to put an upper bound on the number of errors that the procedure makes. Again, class probabilities are needed for this purpose.

The discriminant analysis procedure used in the last section provides estimates of the class $j$ probability $\hat{p}_i(j)$ for each image $i$. The rule used for classifying image $i$ is to choose the class $j$ that maximizes $\hat{p}_j(i)$.

Based on the estimated class probabilities, a we define the rejection rule

$$\begin{array}{c} \text{If } \hat{p}_j(i) \geq \hat{p}_{j'}(i) \geq p_h(i) \text{ for } h \neq j, j' \\ \text{reject image } i \text{ if } \hat{p}_j(i) - \hat{p}_{j'}(i) \leq c \text{ for some } \; c > 0 \end{array} \tag{3}$$

Suppose we decide to limit the misclassification errors to say 2%. Then we find the cutoff value $c$ by starting at zero and increasing $c$ until the error rate on the retained images is 2%.

The results on the zip code data were (91,2,7) and (92,3,5). That is, if we reject 7% of the images, the error rate on the remaining images is 2%. If we reject 5% of the images, the error rate on the remaining images is 3%.

# 6 Discussion

LeCun, Boser et al. (1990), Boser, Guyon and Vapnik (1992), and Simard, Denker and LeCun (1992, Simard, personal communication) have developed classification procedures for this problem that use the greyscale pixel values as input. LeCun et al. employ a multi-layered neural network trained using backpropogation and pseudo-Newton acceleration. Using a training set of size 7300 and a test set of size 2007, they obtained a error rates of 1% and 4.9% on training and test sets. Using a rejection rule to achieve 99% correct classification, the rejection rate was 9%. Boser et al. (1992) develop a kind of non-linear discriminant analysis that aims at maximal separation of the classes. For the same training and test set used by LeCun, Boser et al. (1990), they report a test error rate of 4.6%. The new results from Simard have the most impressive misclassification rates of around 2.5% on the same test set. Their procedure is a nearest neighbor classifier that uses a distance metric invariant under local affine transformations of the scaled images. In this sense it is similar to ours, although their approach (unpublished at the time of this writing) is different. A drawback of their current implementation is that the distance measure is relatively expensive to compute, given that it has to be used to find the closest member of the (large) training set for each incoming image.

Although our approach has not yet achieved the lowest misclassification results, it is currently based on only 13 prototypes. Matching an image to each prototype takes an amount of computing time proportional to the number of discretized points, at most 100 per image. Computing the matching statistics takes a similar amount of time. Our experience suggests that we can afford to increase the number of prototypes by a factor of two or three, and expect to see much better results. One of the focuses of our current research is to develop methods for automatically generating prototypes from the training set, rather than designing them by hand.

Ripley (1992), discusses some related approaches to classification of image data, including fitting of prototypes and classification based on extracted features. He also describes a Bayesian framework for the problem; such an approach might be useful for our particular application but we have not yet explored this.

### Acknowledgements

We thank Daryl Pregibon, Yann LeCun, Henry Baird, John Denker and Michael LeBlanc for valuable discussion. John Overdeck was especially helpful in suggesting modified prototypes.

## References

Boser, B., Guyon, I. & Vapnik, I. (1992), A training algorithm for optimal margin classifiers, *in* 'Proceedings of COLT II', Philadelphia, Pa.

Breiman, L., Friedman, J.H., Olshen, R. & Stone, C.J. (1984), *Classification and Regression Trees*, Wadsworth.

Hastie, T. & Stuetzle, W. (1989), 'Principle curves', *Journal of the American Statistical Association* **84**(406), 502–516.

Hastie, T., Tibshirani, R. & Buja, A. (1992), Flexible discriminant analysis, submitted.

Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R., Hubbard, W. & Jackel, L. (1990), Handwritten digit recognition with a back-propogation network, *in* D. Touretzky, ed., 'Advances in Neural

Information Processing Systems', Vol. 2, Morgan Kaufman, Denver, CO.

Ripley, B. (1992), Classification and clustering in spatial and image data, To appear, proc. of '15 Jahrestagung von Gesellschaft fur Klassification', H. Goehl and M. Schader ed.

van Camp, D. (1991), Xerion neural net simulator. user's guide, Technical report, Dept. of Comp Sci, Univ. of Toronto.