Spatial Intent Recognition using Optimal Margin Classifiers

16.412J Cognitive Robotics
Final Project Report

Thomas Coffee
Shannon Dong
Shen Qu

5/11/05

Abstract

Human-robot collaboration will be crucial to the productivity and success of future space missions. A simple yet intuitive means of communication between two parties—such as communication through gestures—is critical to the success of such collaboration. Optimal margin classifiers can be used for the classification and recognition of such gestures. Two pattern input methods are used to test the behavior and performance of these classifiers. The first is a 2-dimensional computer mouse interface which allows for ease of control and visualization of the patterns. Patterns obtained through this input include circles, lines, and written numerals 2, 3, and 4. The second is a 6 degree-of-freedom hardware tracking device comparable to systems that may be integrated into actual spacesuits. Patterns for this input include gestures designed to convey the intentions of “come here,” “lift,” and “move.” These gestures are meant to mirror the ones actual astronauts may make to communicate with their robotic assistants. We demonstrate a basic linear optimal margin classifier based on support vector methods to efficiently learn and recognize input patterns from multiple categories. We test the algorithm’s capability not only to distinguish different patterns but also to differentiate same pattern made by different users. We further characterize the performance of this algorithm and its sensitivity to training corpus size and input sampling resolution. Finally, we discuss directions for further development of these algorithms to support flexible, intuitive astronaut collaboration with automated space systems.

Table of Contents

Introduction

The high costs of human spaceflight operations favor large investments to optimize astronauts' time usage during extravehicular activity. These have included extensive expenditures in training, tool development, and spacecraft design for serviceability. However, astronauts' spacesuits themselves still encumber more than aid and are the focus of several current research programs. Potential improvements face tight integration between suits and astronaut activities, resulting in many mechanical and computational challenges. One major area of work aims to alleviate the difficulties of conducting precise or prolonged movements within a pressurized garment. Powered prosthetic or other robotic assistance may provide a solution to this problem but creates key operational challenges. Standard digital or verbal user command interfaces are limited by low bandwidth and non-intuitive control structures and may prove incompatible with such devices.

Body language, on the other hand, is one of the most fundamental, natural forms of communication between human beings. Therefore, tactile control using, for example, hand or finger gestures seems far more suitable for controlling mechanical effectors, providing high speed and intuitive spatial relationships between command signals and desired actions. Flexibility and robustness in controllers like these will likely require personalized command recognition tailored to individual astronauts. The need for speed and natural facility will make this capability even more indispensable than in, say, speech recognition. Command recognition systems could adjust their interpretation rules as training data is accumulated, improving their precision and following long-term trends as astronauts develop their working behaviors throughout a career's worth of extravehicular activity.

In this project, we propose a relatively simple gesture-based spatial command recognition system as an analog to more advanced systems suitable for augmenting extravehicular activities with robotic assistance. The first section of this report conveys the key ideas behind the optimal margin classifier, the fundamental recognition method at the core our system. The second section then introduces two different pattern input interfaces used to collect both training and testing data for the algorithm.

We first we implemented a computer mouse interface for the preliminary analysis of our algorithm. Although this input method should be very different from any system that may be used by astronauts in space, it does provide a solid foundation towards the understanding of algorithmic behavior. Most users are very familiar with the operation of a mouse and can therefore achieve a relatively high degree of precision in the inputs without extensive practice. The 2D nature of the inputs also allows for easy visualization. Next we extended our user interface to an InterSense tracking device capable of detection spatial motion performed by the user. Gesture patterns collected through this device would be much closer to those astronauts may use to command their robotic assistants.

Finally, we conclude this report with sections on results and analysis of our data and possible extensions to our current system.

1. Optimal Margin Classifier[1]

The enveloping problem that a classifier must solve is how to accurately and efficiently separate one pattern from another given a set of training data (a corpus for each pattern in question). By maximizing the minimal distance from the training data to the decision boundary, an optimal margin classifier can achieve an errorless separation of the training data if one exists. Its strength lies in its ability to identify and eliminate outliers that are ignored by other classifiers such as those that minimize the mean squared error. Optimal margin classifiers are also less sensitive to computational accuracy because they provide maximum distance (error margin) between training set and decision boundary.

1.1 Representation

A pattern is represented by an n-dimensional vector **x**. Each **x** vector belongs to either class A or B, where A corresponds to some pattern of interest and B corresponds to either another pattern or possibly all other patterns in the universe. The training data consists of a set of vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_k, \ldots \mathbf{x}_p$; we label each $\mathbf{x}_k$ +1 or –1 to associate it with a class: for example, +1 for class A and –1 for class B. A training set containing *p* training points of vectors $\mathbf{x}_k$ and labels l_k would look like:

$$(\mathbf{x}_1, l_1), (\mathbf{x}_2, l_2), \ldots, (\mathbf{x}_k, l_k), \ldots, (\mathbf{x}_p, l_p), \qquad \text{where } \begin{cases} l_k = 1 & \text{if } \mathbf{x}_k \in \text{class A} \\ l_k = -1 & \text{if } \mathbf{x}_k \in \text{class B} \end{cases} \qquad (1)$$

Figure 1 gives a visual representation of a training set with five points $(p = 5)$ each a two-dimensional vector of the form $\mathbf{x}_k = [x_k, y_k]$.

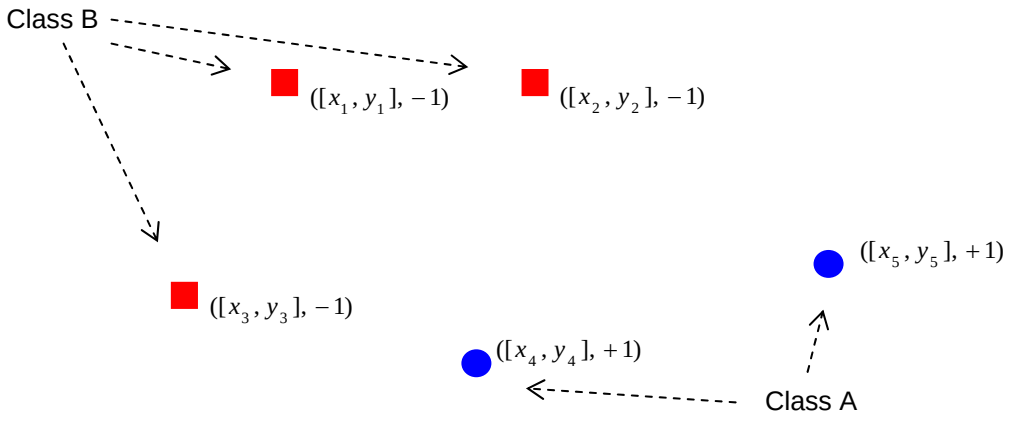

Figure 1: Example of a simple two-dimensional training set

1.2 Decision Function

Given the training data, the algorithm must derive a decision function (i.e. classifier) $D(\mathbf{x})$ to divide the two classes. The decision function may or may not be linear in **x**

depending on the properties of the corpus. For a classification scheme shown in Equation 1, a decision value of $D(\mathbf{x}) = 0$ would be on the boundary between the classes. Any new, unknown vector **x** can be classified by determining its decision value: a positive decision value places **x** in class A, and a negative value places **x** in class B.

The decision function is a multidimensional surface that optimally divides the different classes. In our simple pedagogical example, *D* is some linear function of *x* and *y* that divides the two classes in a way that maximizes the minimum distance between itself and the nearest points in the corpus, as shown in Figure 2.

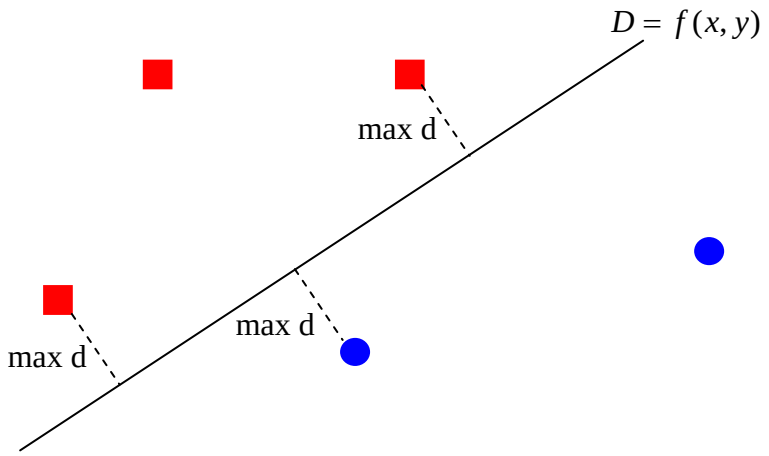

Figure 2: Decision function for the pedagogical example

The decision function of an unknown input vector **x** can be described in either direct space or dual space. In direct space, the representation of the decision function is:

$$D(\mathbf{x}) = \sum_{i=1}^{N} w_i \varphi_i(\mathbf{x}) + b, \tag{2}$$

where the φ_i are previously defined functions of **x,** and w_i and *b* are adjustable parameters of *D*. Here, *b* is called the bias, i.e. offset of *D*. In dual space, the decision function is given by:

$$D(\mathbf{x}) = \sum_{i=1}^{p} a_k K(\mathbf{x}_k, \mathbf{x}) + b, \tag{3}$$

where the a_k are adjustable parameters, and $\mathbf{x}_k$ are the training patterns. Equations 2 and 3, the direct space and dual space representations of *D*, are related to each other via the following relation:

$$w_i = \sum a_k \varphi_i(\mathbf{x}_k). \tag{4}$$

K is a predefined kernel function of the form

$$K(\mathbf{x}, \mathbf{x}') = \sum_i \varphi_i(\mathbf{x})\varphi_i(\mathbf{x}'). \tag{5}$$

There are many different options for the *K* function, depending on the structure of the training data. Since *D* is linear in all other parameters, the function *K* determines the complexity of *D*: *D* is linear if *K* is linear and *D* is exponential if *K* is exponential. For our purposes, we use a linear classifier *D* in its dual space representation with the corresponding *K* function:

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'. \tag{6}$$

1.3 Maximizing the Margin

Given *N*-dimensional vectors **w** and $\varphi(\mathbf{x})$, and bias *b*, $D(\mathbf{x})/\|\mathbf{w}\|$ is the distance between the hypersurface *D* and pattern vector **x**. We define *M* as the margin between data patterns and the decision boundary:

$$\frac{l_k D(\mathbf{x}_k)}{\|\mathbf{w}\|} \geq M \tag{7}$$

Maximizing the margin *M* produces the following minimax problem to derive the most optimal decision function:

$$\max_{\mathbf{w}, \|w\|=1} \min_k l_k D(\mathbf{x}_k). \tag{8}$$

Figure 3 shows a visual representation of obtaining the optimal decision function.

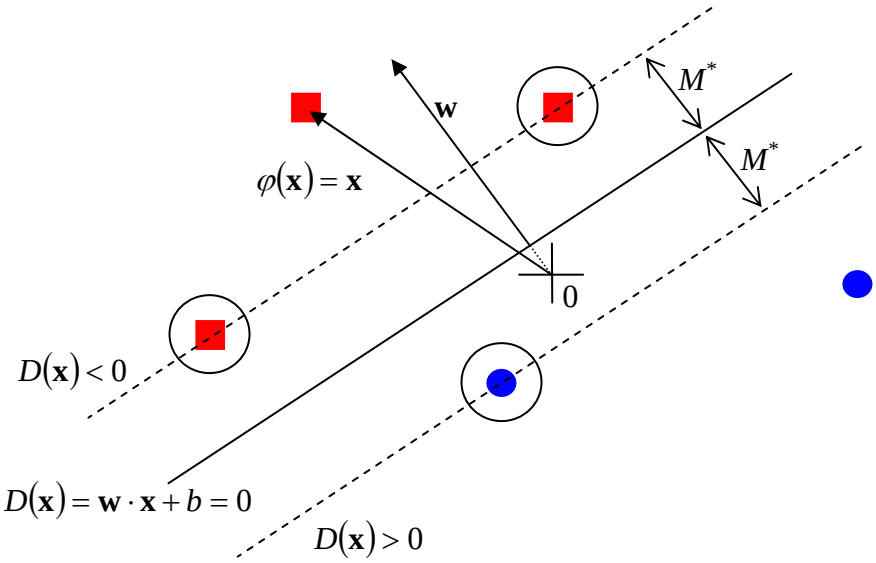

Figure 3: Finding the Decision Function. The decision function is obtained by determining the maximum margin *M**. Encircled are the three support vectors.

To find the optimal decision function, in the dual space, we optimize a Lagrangian which is a function of the parameters $\boldsymbol{\alpha}$ and bias b. It also encompasses the kernel function K via a $p \times p$ matrix $\boldsymbol{H}$, which is defined by $H_{km} = l_k l_m K(\mathbf{x}_k, \mathbf{x}_m)$. The Lagrangian is given by:

$$J(\boldsymbol{\alpha}, b) = \sum \alpha_k (1 - b l_k) - \frac{1}{2} \boldsymbol{\alpha} \cdot \boldsymbol{H} \cdot \boldsymbol{\alpha} \qquad (9)$$

$$\text{subject to } \alpha_k \geq 0, \qquad k = 1, 2, \ldots, p\,.$$

For some fixed bias b, we can find $\boldsymbol{\alpha}^*$ that maximizes J, and the decision function becomes

$$D(\mathbf{x}) = \sum_k l_k \alpha_k^* K(\mathbf{x}_k, \mathbf{x}) + b, \qquad \alpha_k^* \geq 0\,. \qquad (10)$$

1.3.1 A Note on Dimensionality

After the decision function is found, the points closest to the decision hyperplane are the support vectors. If the decision function has no restrictions, then there will naturally exist at least $n+1$ support vectors, where n is the dimension of each sample, unless there are fewer samples in the corpus than $n+1$, in which case all available samples are support vectors, and the optimization is reduced to a smaller dimension.

In our pedagogical example of Figure 2, $\mathbf{x}$ has dimensionality of 2, so there are $2+1=3$ support vectors. Had there been only two samples in the corpus, the problem would reduce to 1-D optimization, which dictates that the decision function D must be the perpendicular bisector of the two points.

We can similarly expand that idea to 3-D as shown in Figure 4 and to multi-dimensional hyperspaces. If there were only three points, then the problem would collapse to a 2-D optimization, even though the decision function is still a plane. Similarly, if there were only two points, the problem would be reduced to 1-D optimization, with the plane again being the perpendicular bisector of the two points.

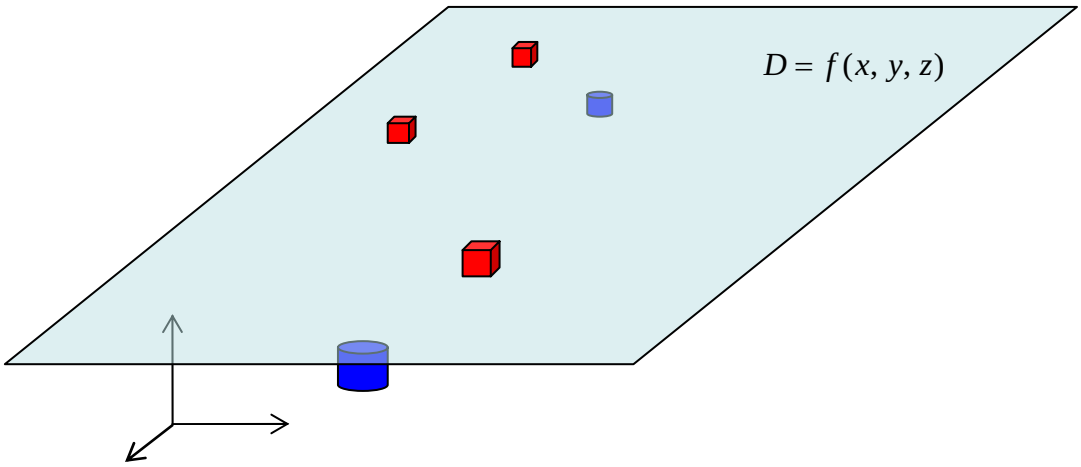

Figure 4: Example of expansion to multiple dimensions

The dimensionality of **x**, or the value *n*, is the product of the spatial dimensions of the pattern and the number of representative sampling points taken from the pattern. In our handwriting recognition implementation, the patterns are in 2-D, and each pattern is represented by 6 sampling points, as shown in Figure 5. Thus, each pattern is represented by a 12-dimensional vector; and without any restrictions on the decision function, there should be 13 support vectors.

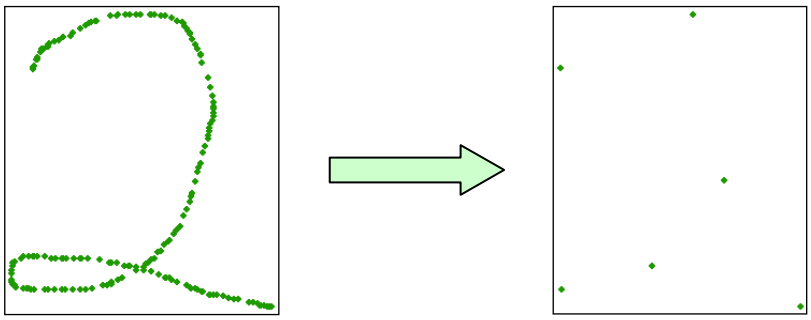

Figure 5: Sampling 6 representative points from a pattern

1.3.2 A Note on the Decision Function Bias

There are two ways to determine the bias *b* in the decision function (See Equation 10). One method is to set the value of *b* to some constant a priori. This works best when we know some information about the corpus. For example, we may know that the training data fits nicely on both sides of the x-axis and that it is fairly symmetric. In such a case,

we may want to set $b = 0$ a priori, as shown in Figure 6. When b is fixed to some constant value, the number of support vectors is decreased by 1. In fact, sometimes support vectors for decision functions with fixed b may be from the same class.

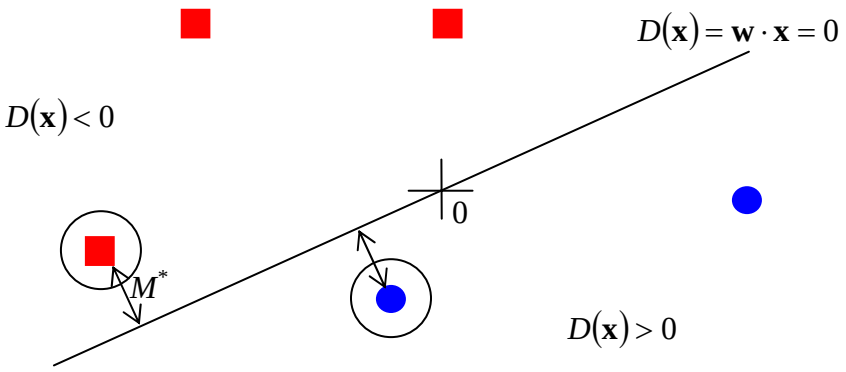

Figure 6: Decision function with bias set to 0

If we know that the corpus is not best described by 0 or some other fixed value, but we know that the corpus is nicely distributed, we may want a bias that enables the decision function to go through the midpoint of the centroids of the two classes. This can also be done by setting the origin of the hyperspace to be at that midpoint and set the bias to 0.

Finally, the bias b can be made to vary with the parameters $\boldsymbol{\alpha}^*$ if some of the support vectors are known. To obtain $\boldsymbol{\alpha}^*$ independent of the bias, see Vapnik[2]. Suppose we know the support vectors $\mathbf{x}_A \in$ class A and $\mathbf{x}_B \in$ class B. The decision values of the two support vectors are known: $D(\mathbf{x}_A) = +1$ and $D(\mathbf{x}_B) = -1$. Then the optimal bias b^* is given by:

$$b^* = -\frac{1}{2}\sum_{k=1}^{p} l_k \alpha_k^* \left[K(\mathbf{x}_A, \mathbf{x}_k) + K(\mathbf{x}_B, \mathbf{x}_k)\right]. \qquad (11)$$

2. Implementation Methods

The overall implementation of the project consisted of several main portions: the user input mechanisms, the input patterns, the optimal margin classifier algorithm, and our analytical tools. An overview of the components in the project implementation can be seen in Figure 7. The program controlling the 6D user input and DOF tracking device is written in Python. All other software components of this project are implemented in *Mathematica*.

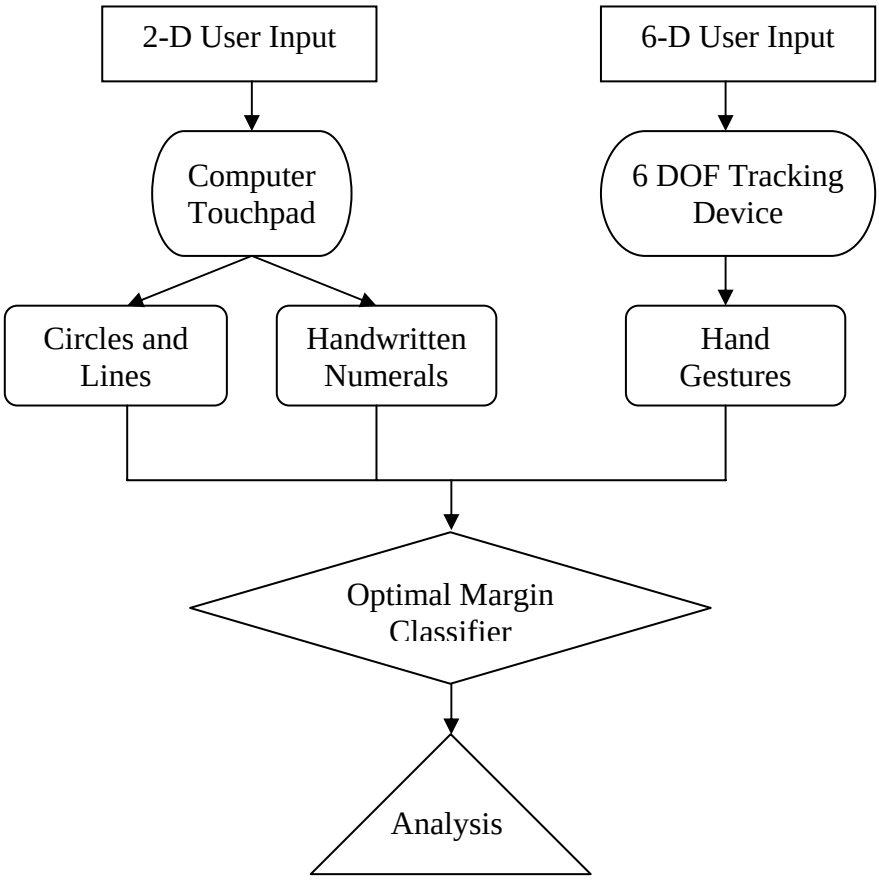

Figure 7: Project implementation overview

2.1 User Input Mechanisms

Two types of user inputs are used in our implementation: 2D planar inputs through a computer mouse and 6D spatial inputs through the InterSense tracking device.

2.1.1 Handwriting Input

The planar input mechanism is a two-dimensional drawing pad implemented using the *GUIKit* package provided by *Mathematica 5.1*. Initially, the program displays a pop-up

window for the user to input the desired symbol. After the user completes the input and closes the window, (x, y) positions of each point in the symbol is refreshed at machine speed and saved chronologically into a matrix.

In the user input window, the $(0,0)$ point is at the top left corner and the positive y-axis points downward, which is opposite from usual plotting convention; so when visualizing the data, we take the negation of the y values in order to upright the symbol.

During the actual user input process, the length of the input data may vary depending on the speed at which a symbol is drawn. However, in order to calculate the decision function between 2 classes, all **x** vectors in both corpuses for the classes must have an equal number of points (dimensions). Also, a typical user input symbol may lead to a data stream of 40 or 50 points, which is much more than what is needed to distinguish the classes. Therefore for most of our analysis, we sample a set 6 evenly spaced points from each symbol and run our optimization algorithm on a collection of such 6 point symbols. Part of our analysis also deals with the variation of this sample size and its effects on algorithm performance. Figure 8 shows the progression of the data from the user input to the sampled data matrix.

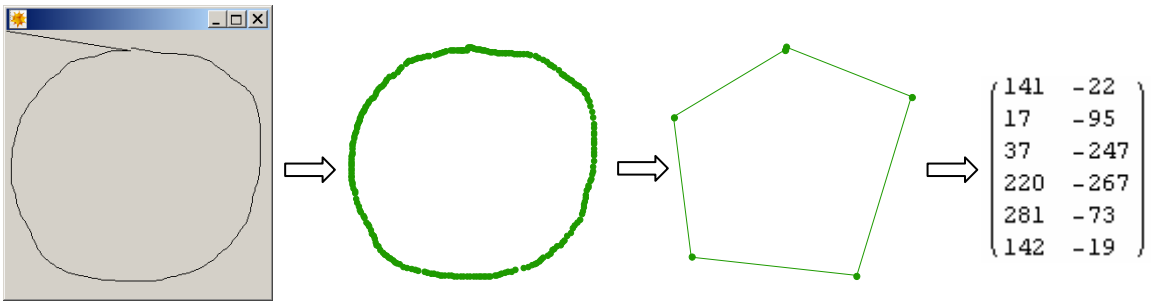

Figure 8 Evolution of a circle. a. The user inputs a two-dimensional figure into the drawing window using a mouse or keypad, **b.** Points are recorded at machine speed, **c.** Six points are sampled, and **d.** their (*x*, *y*) positions generate a 6×2 matrix.

Note that in the user input window, the visual feedback of the hand-drawn figure is continuous and starts at the $(0,0)$ point, but the $(0,0)$ point will not be part of the actual data record. Also note that Figure 8c appears to have 5 sample points instead of 6 only because the first and last points are almost coinciding. Once a sampled matrix as shown in Figure 8d is generated, it is transformed into a single data point in the $6 \times 2 = 12$ dimensional hyperspace. From there it will be fed into the optimal margin classification algorithm as discussed in Section 1.

2.1.2 Gesture Input

For spatial data, we use a unique piece of hardware called the InterSense tracking device[3], which consists of an inertial measurement unit (IMU) containing a gyro and an accelerometer to monitor angular travel and velocity, which mostly provides the pitch, yaw, and roll information. The tracking device uses two ultrasound beacons to

directionally position itself in 3-D space, providing (x, y, z) information. The aperture is held by the user like a wand at one end. Figure 9 shows the aperture setup.

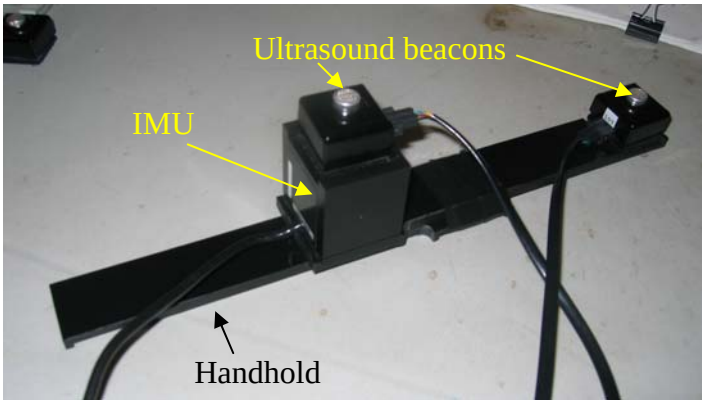

Figure 9: User interface for spatial inputs

The ultrasonic beacons transmit signals to receivers located in fixed locations several meters over head. There are four ultrasonic receivers situated on a crossbar that is fixed to the ceiling, as shown in Figure 10. The absolute spatial origin of the tracking system is calibrated to be the exact point on the ground directly below the center of the crossbar. When gathering gesture data, the user sat on a chair positioned over the origin, so that the range of (x, y, z) positions would be generally centered around $(0,0,1)$, in meters.

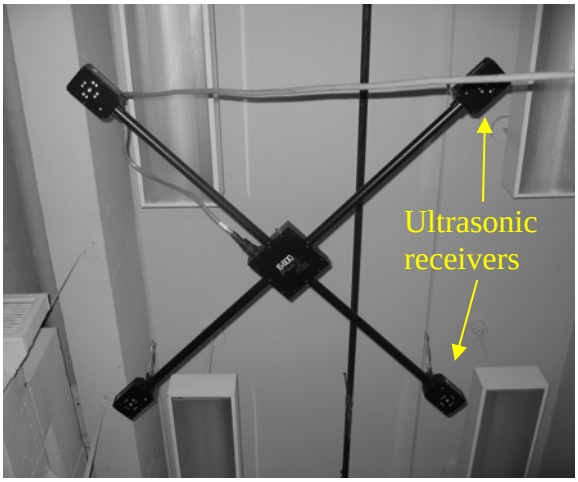

Figure 10: Ultrasonic receiver module crossbars on the ceiling

The ultrasonic transponder beacons are line-of-sight devices, meaning that they cannot operate when the line-of-sight between the transponder and receiver is blocked or if the transponder is tilted at a steep angle. The ultrasonic beacons do have about 70 degrees of sight from vertical, as shown in Figure 11, so the line-of-sight issue is usually not a problem. For our implementation, we kept to gestures that did not require turning the aperture upside-down.

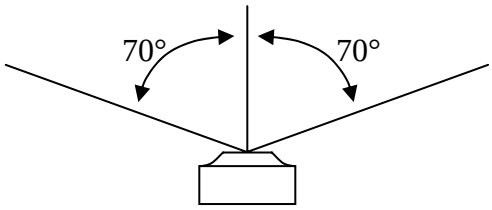

Figure 11: 140 degree conical range of ultrasonic beacons.

The data generated by the InterSense tracking device is accurate to a few millimeters, and each recorded point contains 6 variables $(x, y, z, \theta, \varphi, \phi)$. For each gesture pattern, we also extracted 6 representative sample points from the record of all points in the pattern, so each element of the gesture corpus is a 6×6 matrix, or in other words, the hyperspace is 36-dimensional.

The software that the InterSense tracking device runs on is called Vizard, which runs code in Python language. We programmed the software to run non-stop with a continuous timer and keyboard control of start and stop gesture sequences. The gesture sequence data would be instantaneously saved to a file that can be later analyzed by our algorithm code in *Mathematica*.

We selected the InterSense tracking device as our hardware input component because a similar device or device with similar principles could be integrated into a spacesuit for astronaut gesture recognition. For this application, the device can exploit the rigid nature of spacesuits. The receivers could be attached to the astronaut's helmet and/or parts of the upper torso of the spacesuit. The beacons could then be attached to part of the glove or even on a segment of the finger portion of the glove if the devices could be made more compact. In this case the (x, y, z) components would be recording the position of the astronaut's hand relative to his/her body, and all of the same principles discussed in this section would still apply.

2.2 Input Patterns of Interest

2.2.1 Circles and lines

In the 2-D user interface environment, the first test of the algorithm demonstrates distinction of two very different figures, such as circles versus lines. The circles in the corpus are all drawn in the same direction, in our case, counter-clockwise, and the lines are all drawn from the top right corner to the bottom left corner. Figure 12 shows a typical representation of the user inputs for circles and lines. Note again that the visual feedback starts at the $(0,0)$ point and connects to the first point in the figure.

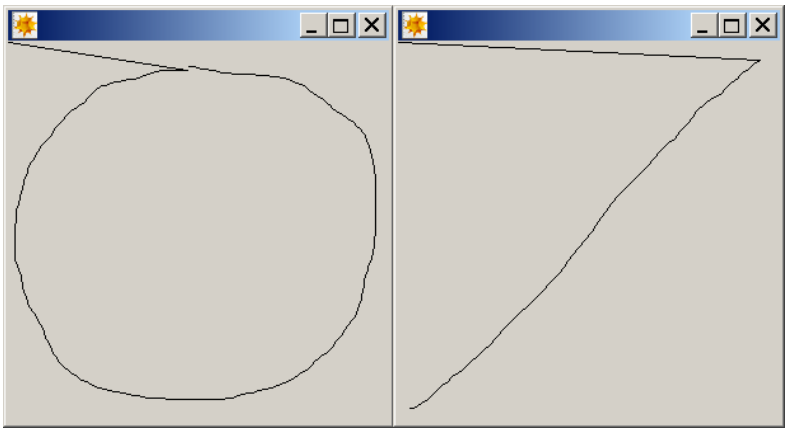

Figure 12: Typical user input circle and line

A corpus of 5 circles and 5 lines was used to determine the decision function. Twenty test inputs were implemented for evaluation. Upon deriving the decision function, we generated an interpolation sequence between a circle and a line to find the figure that has a decision value of 0, or in other words, a figure that sits on the class boundary between circles and lines for a certain representation of circles and lines. This pattern is shown in Figure 13. This figure can be a guideline for predicting which figures will be classified as circles and which will be classified as lines.

Figure 13: Between circle and line. This is a pattern that has decision value of 0 for a certain representation of circles and lines

2.2.2 Numerals

It seems that distinguishing two very different sets of figures from one another is trivial. We demonstrated the algorithm's ability to correctly classify different numerals, and following that, we decided to take on the challenge of distinguishing one person's handwriting from another's, as this distinction is sometimes difficult even for a human observer. This is implemented by two people generating corpuses of the numeral "2," "3," and "4," and running test inputs over the corpuses to determine the accuracy of the

classification. An example implementation is shown in Figure 14, where the corpuses of "2"s are generated by two different people, and 6 points are sampled out of each "2." Each person generated corpuses of around 20 elements for each numeral. Twenty test inputs were implemented for evaluation.

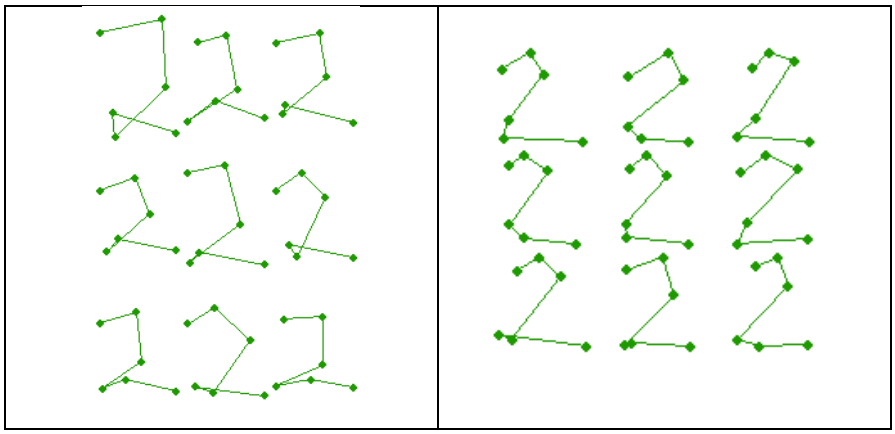

Figure 14: 2's vs. 2's. The sample set of the corpus of "2"s generated by person A is on the left and person B's is on the right

2.2.3 Gestures

Three different gestures were implemented for testing purposes, including the "come here" gesture, lifting gesture, and pointing gesture. Two different persons generated separate corpuses for all three gestures of 10 elements each. Twenty test inputs were implemented for evaluation.

3. Results and Analysis

In the following analysis, we verify the behavior of the optimal margin classification algorithm against theoretical intuition and assess the performance drivers in implementation relevant to our applications. We also attempt to explain any unusual results, particularly where they adversely affect performance. We use basic shape identification (circles and lines) to illustrate general features of the algorithm, then conduct a more thorough analysis using handwriting recognition, before moving to the more challenging application of recognizing human gestures. Unless stated otherwise, all gesture and figure recognition results reported use 6 sample points per pattern and 10 of each type of pattern per training corpus.

3.1 Basic Shapes: A First Look

As described earlier, two-dimensional figures are converted to patterns by uniform sampling of points from the cursor path and concatenation of these coordinates into higher-dimensional vectors. A linear optimal margin classifier then constructs a separating hyperplane in the pattern space that maximizes the distance from the hyperplane to the nearest pattern(s) in each distinct class (the support patterns).

We can observe the optimization behavior of the algorithm by examining the Euclidean distances from corpus patterns to the resulting hyperplane boundary. Figure 15 illustrates a typical example of this optimization: the corpus for each shape (shown in gray) contains five training examples, but the supporting patterns from each class lie at equal distances from the hyperplane and are barely distinguishable, while the support patterns for the two classes also lie at equal distances.

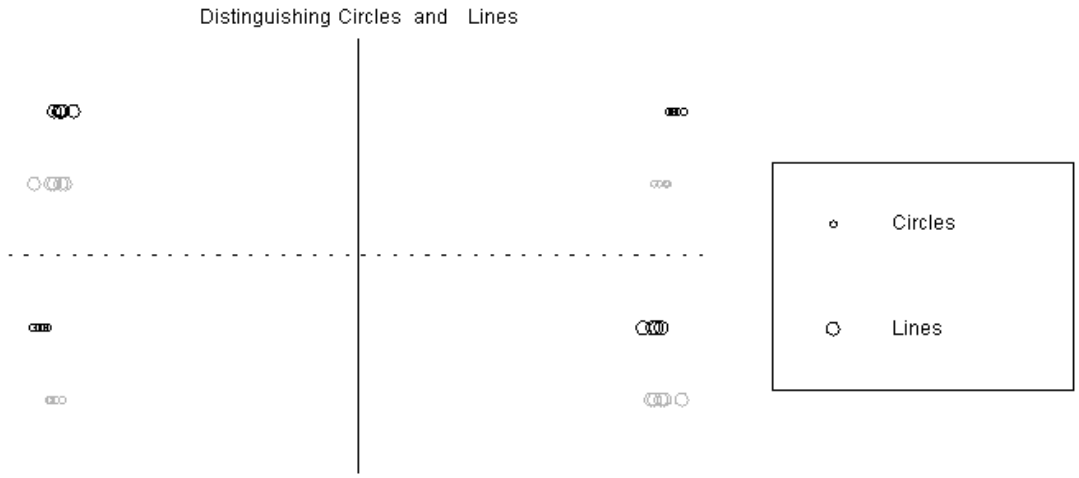

Figure 15: Euclidean distances from basic shapes to a separating hyperplane (solid line) in pattern space. The first horizontal segment illustrates the midpoint-bias classifier for circles, the second for lines; with only two classes, these classifiers produce mirror images. The classifier maximizes the minimum distance from the hyperplane to the pattern classes in the corpus (gray), which it then uses to separate independent test patterns (black).

Because our algorithm positions the hyperplane with a fixed bias (at the origin or at the midpoint of corpus centroids), in some cases the support patterns may lie all in the same class. In this case, the distance from the separating hyperplane to the nearest pattern in each class may no longer be equal despite the equidistance of the support patterns to the boundary. Figure 16 shows an example of this situation resulting from a zero-bias classifier applied to a corpus containing ten training samples each of circles and lines.

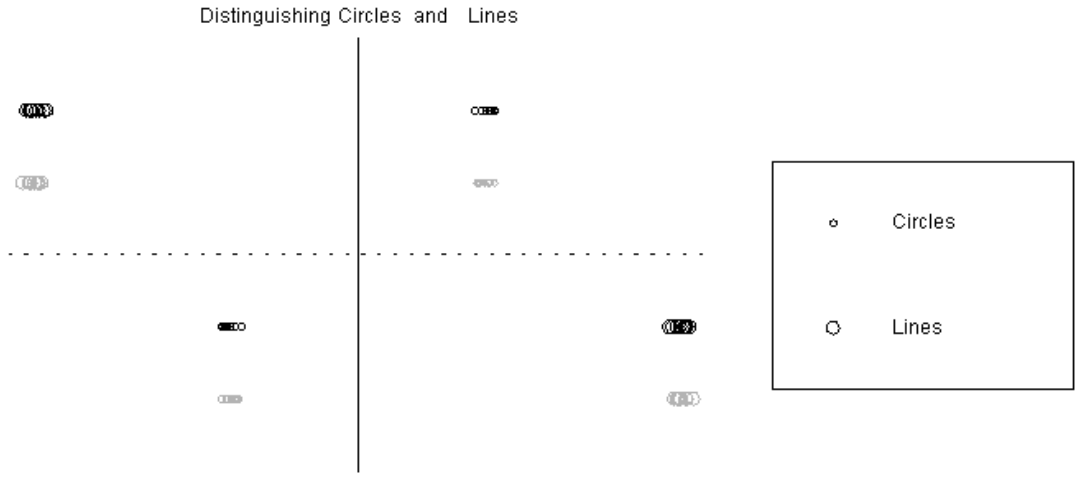

Figure 16: Distances to the separating hyperplane of a zero-bias classfier for circles and lines on a 20-sample corpus. When all support patterns lie in one class, the minimum distances to each class need not be equal.

Unequal (and thereby suboptimal) separation is an inherent risk of our fixed-bias approach. As we shall see, it rarely prevents effective separation of the corpus in our applications. However, there are pathological cases in which it can render the linear classifier impotent: these occur when the pattern clusters corresponding to the two classes to be separated both lie close to a ray emanating from the origin defined by the bias in pattern space.

As an example, we divide the coordinates of the circles in the corpus used above by a factor of five to produce a corpus of “scaled” circles lying between the origin and the original corpus. With zero bias, the optimizer fails to converge and the algorithm returns the solution shown in Figure 17, failing to separate either the corpus or the test data. In our examples, this weakness is rarely problematic, particularly since the likelihood of angular conjunction falls with increasing dimension of the pattern space for a given dispersion within classes.

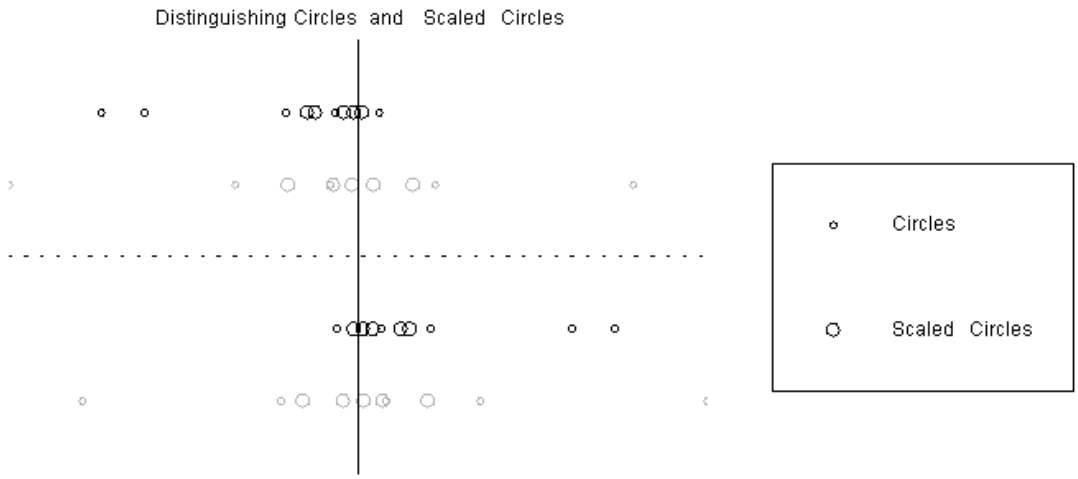

Figure 17: A pathological case constructed by scaling the circle corpus relative to the origin illustrates the potential pitfalls of fixed bias: the classifier fails to separate either the corpus (gray) or the test patterns (black). However, this problem rarely creates difficulties in our applied examples.

The related restriction to linear classifiers (that is, defined by hyperplane boundaries) limits the pattern space geometries that can be effectively handled by the algorithm: clearly, only classes that lie on either side of a hyperplane may be separated. Problems will arise whenever a class totally or partially "surrounds" another. This may have implications for separating specific patterns from generic background "noise," which can easily pervade the surrounding pattern space.

As an example, we optimize a classifier to distinguish between circles and patterns formed from uniformly random points within the two-dimensional canvas domain. Because the points sampled from circles lie at the outside edges of the domain, the class is not well surrounded by a small set (10) of random patterns, and can be barely separated from the noise (Figure 18a). If we generate alternative corpuses with smaller circles (constructed by shrinking the original circles toward their centers), we find that the noise increasingly surrounds the patterns of interest and prevents effective linear separation (Figure 18b,c).

Note that we have shrunk our sample resolution to two sample points per figure in this example in order to obtain a pattern space dimension of four, allowing the circles to be easily surrounded by a small number of random points; with larger dimension, many more random points are required to defeat hyperplane classifiers. In our applications, we rely on a sufficiently low dispersion among classes avoid this issue altogether.

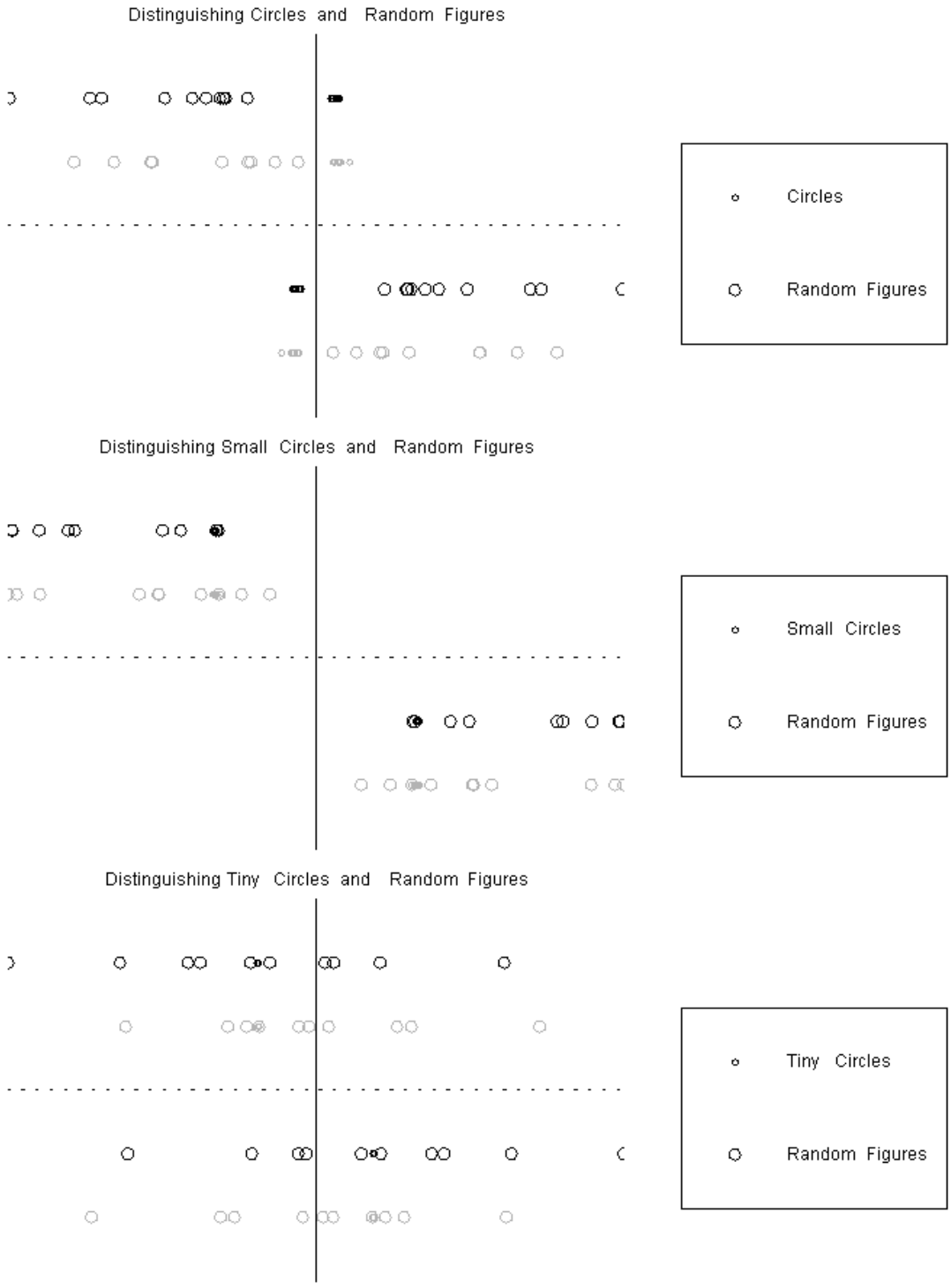

Figure 18: Distinguishing circles from random figure "noise" become increasingly difficult as the size of the circles relative to the canvas decreases and the circle classes become surrounded by the noise; the dispersion eventually overwhelms the algorithm. "Small" circles are reduced in size by a factor of 2, "tiny" ones by a factor of 5.

3.2 Handwriting Recognition: Performance Drivers

We used the domain of handwriting recognition as a comprehensive, realistic, yet relatively simple testbed to examine the capabilities and performance drivers of the algorithm. The less challenging task consisted of distinguishing different numerals ("2," "3," or "4") written by the same test subject. The more challenging task consisted of distinguishing the same numeral written by two different test subjects.

Distinguishing between numerals in our approach required constructing three optimized boundary hyperplanes, one to separate each numeral class from its complement in the training corpus. In these cases, midpoint-bias classifiers consistently placed the boundary hyperplane further from the class of interest than optimal, as one might expect (Figure 20). The midpoint bias is based on the assumption that the dispersion of the classes on either side of a boundary will be roughly equal, but the dispersion of a set of classes will be greater with respect to the distance between centroids than that of a single class, hence the complement of the class of interest will lie closer to the boundary than the class itself.

Zero-bias classifiers also showed strong asymmetries, but their direction depended upon the gestures themselves: those further separated from the others ("point," and "come" for Subject 1) enjoyed greater margins with respect to class boundaries, again as expected (Figure 19). However, the dispersion within classes was small enough that neither form of asymmetry prevented achieving perfect accuracy on both corpus and test data.

In distinguishing the identical numerals made by different subjects, we can see the difficulty of the problem in the greater dispersion of classes relative to the distance between the classes and the separating hyperplane (Figure 21). Though the corpus is still well separated, the test data spills over the boundary in several cases. A midpoint bias gives slightly more asymmetric separation boundaries (Figure 22), most likely due to differences in variability between the different subjects.

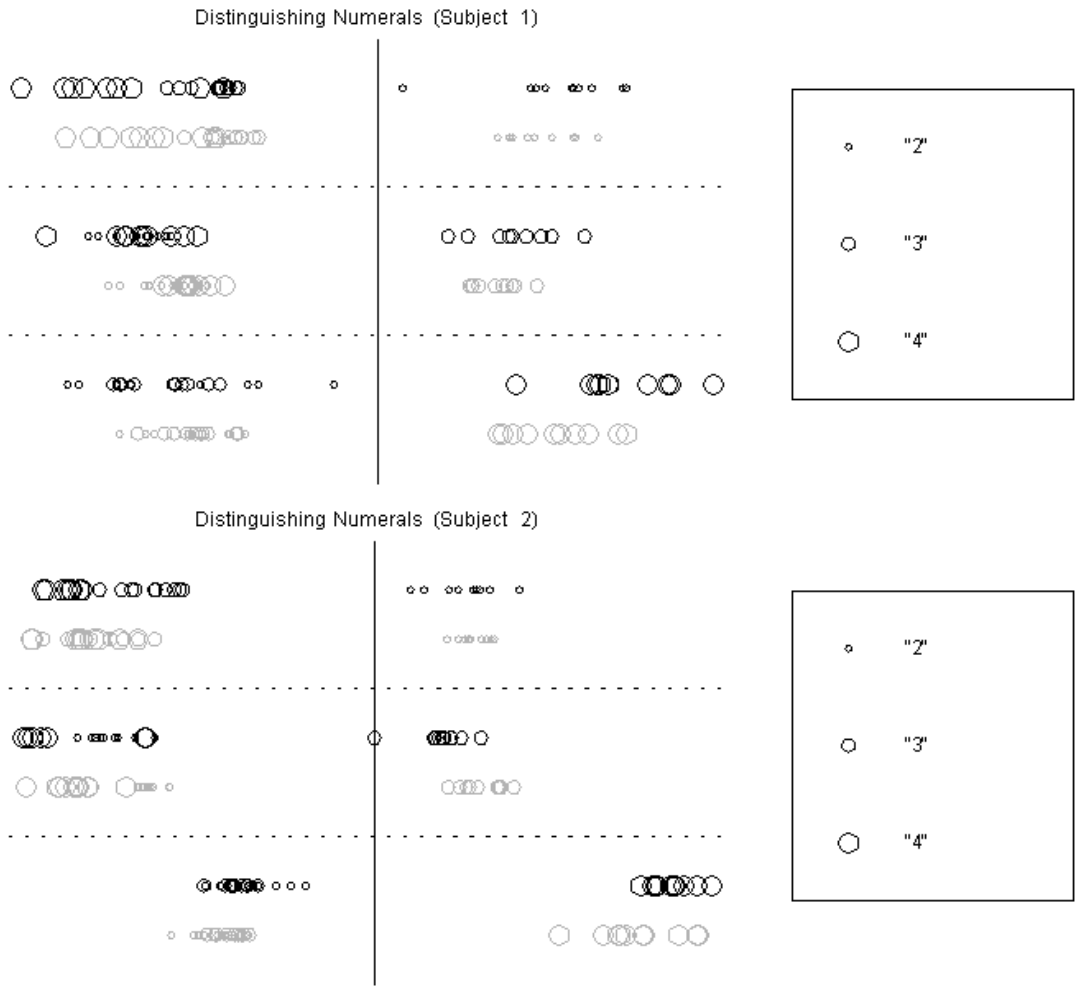

Figure 19: Hyperplane distance distributions for distinguishing numerals with zero bias for each subject. Asymmetries in minimum distance are governed by the relative proximity of classes.

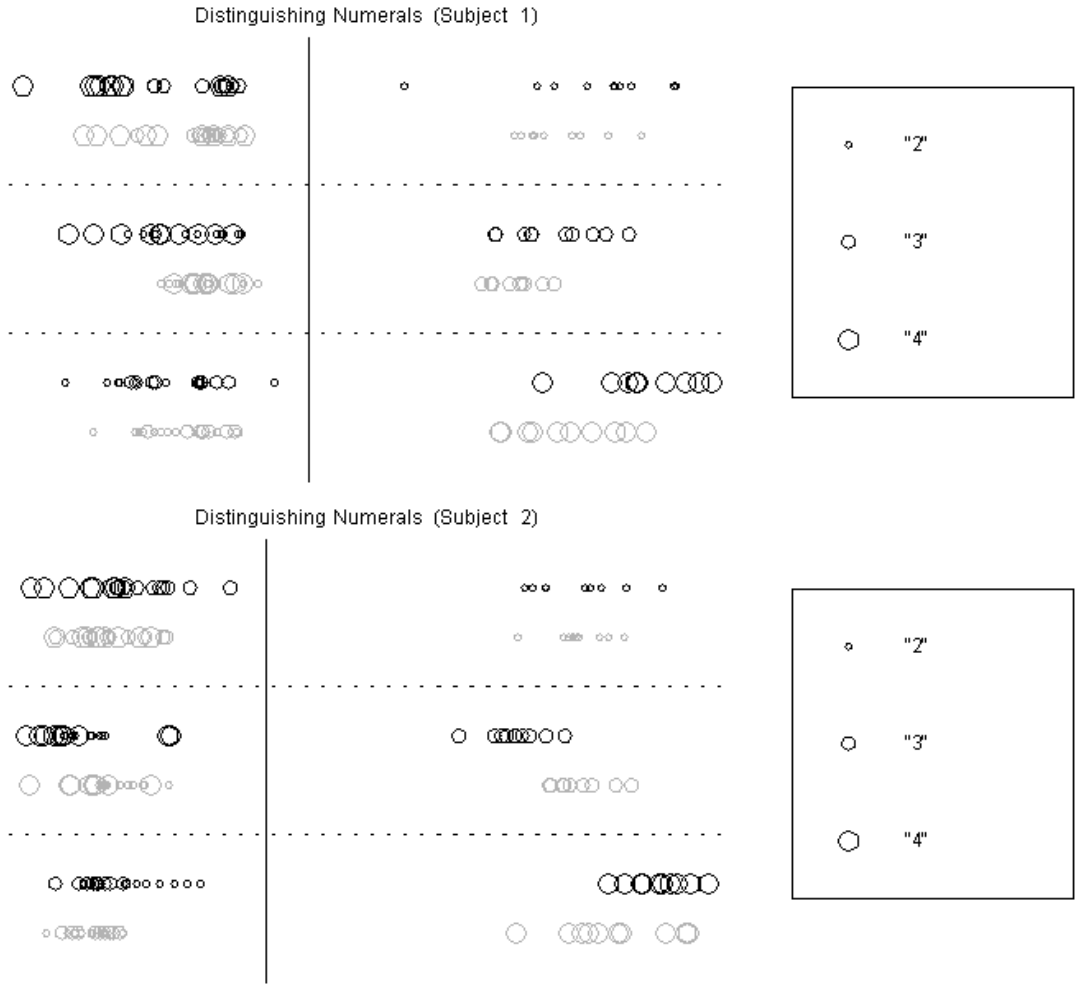

Figure 20: Hyperplane distance distributions for distinguishing numerals with midpoint bias for each subject. Hyperplanes tend to lie close to the complement of the distinguished class.

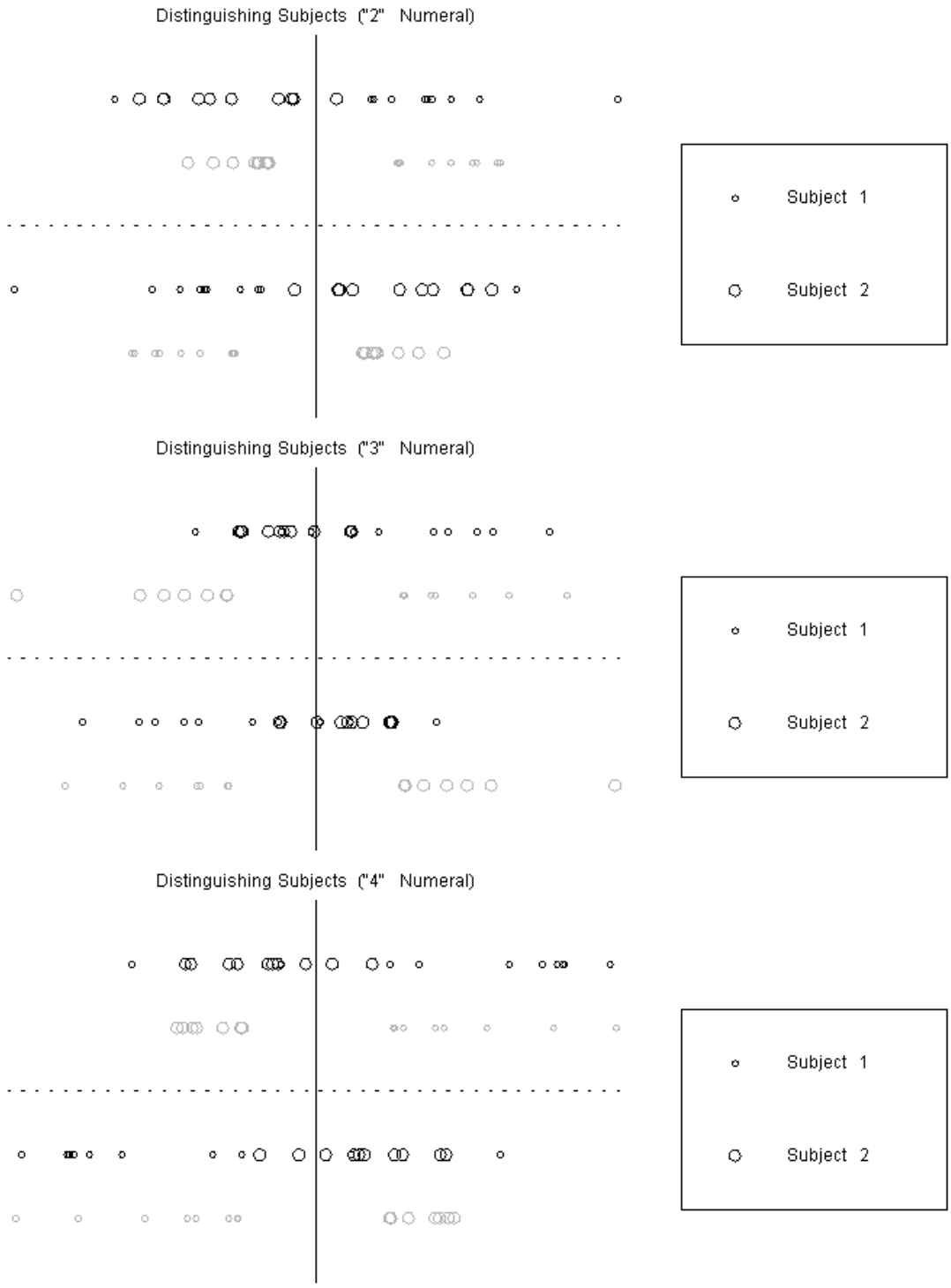

Figure 21: Hyperplane distance distributions for distinguishing subjects with zero bias for each numeral. The classifiers achieve near-optimal separation on the corpus, but variability is high enough to create overlap in the test data.

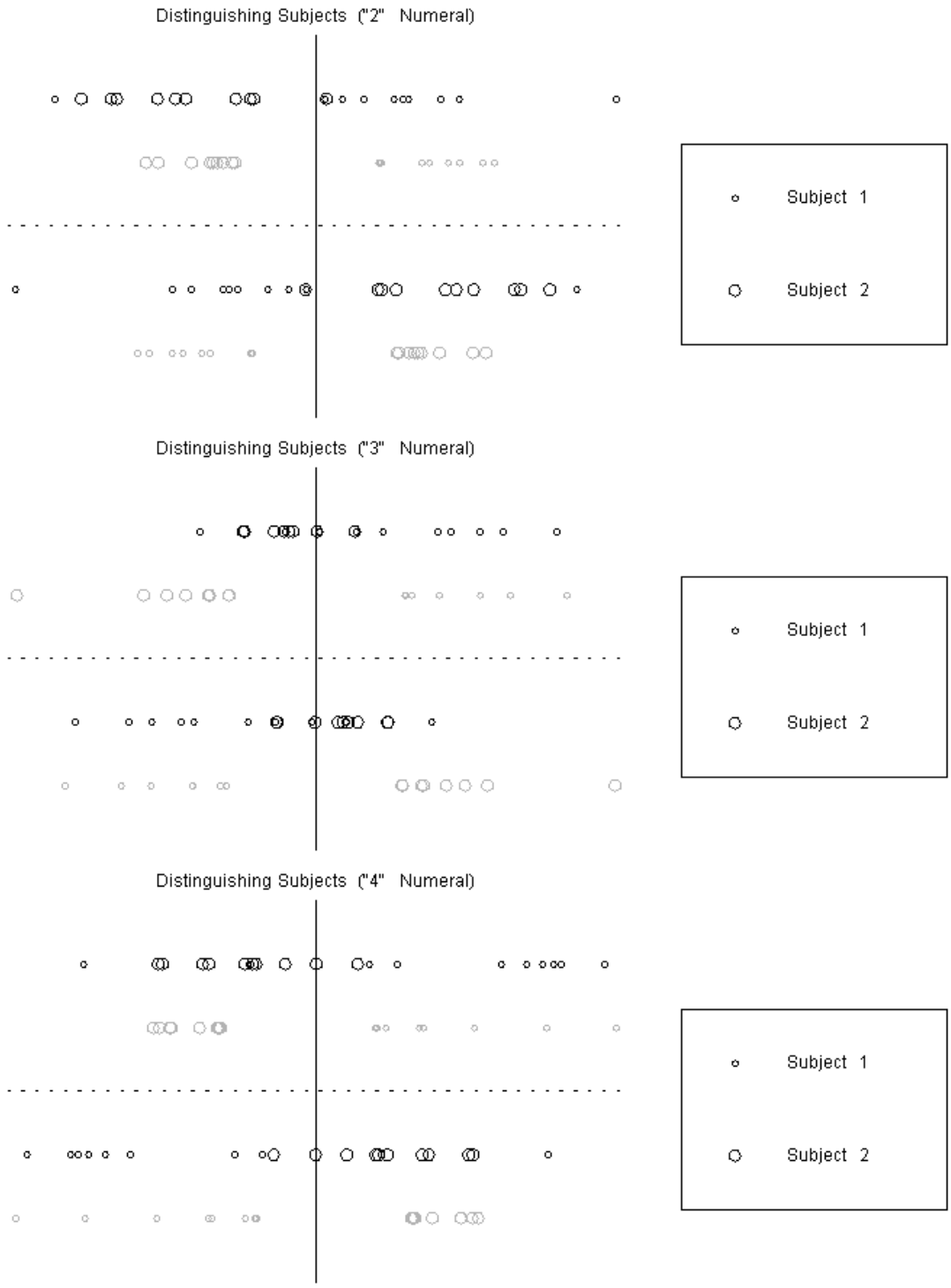

Figure 22: Hyperplane distance distributions for distinguishing subjects with midpoint bias for each numeral. The classifiers achieve slightly less symmetric class separation than the zero bias classifiers; again variability is high enough to create overlap in the test data.

We now analyze the accuracy of the two approaches on handwriting recognition tasks, and its variability with corpus size and pattern sampling resolution. The corpus size governs the dimensionality of the optimization problem to be solved in determining the optimal margin classifier: the number of optimization parameters is equal to the number of training patterns in the corpus. In the examples following, we test corpus sizes of 1, 2, 3, 5, 7, and 10, with a sample resolution of 6 points per figure. The number of support patterns determining the final classifier can be as large as one greater than the dimensionality of the pattern space, but can also be limited by the number of training patterns available (in which case the problem is reduced by projection to one of lower dimension).

The dimensionality of the pattern space is directly proportional to the sampling resolution: in this case, dealing with two-dimensional points, the representation of each pattern has dimension equal to twice the number of sample points. For instance, our results above using 6 sample points lead to a 12-dimensional pattern space. In the examples following, we test sampling resolutions of 2, 3, 5, 10, 20, 40, and 80 points per figure, with a corpus size of 5 patterns. All evaluations are done using 10 test patterns independent of the corpus.

The algorithm performs perfectly in distinguishing numerals from one another for all cases tested, so we will not consider these results further. Instead, we look at results for distinguishing between subjects writing identical numerals. Here the results for zero bias and midpoint bias are identical, reflecting that calculated midpoints lie near zero. Figure 23 shows the sensitivity of test pattern classification accuracy to corpus sample size; though the corpus is perfectly separated in all cases, increasing corpus size generally improves accuracy, with fluctuations attributable to the strong influence of individual patterns in a small corpus. With corpus sizes of 10, accuracies consistently reach the range of 80-90%. Figure 24 shows the sensitivity of accuracy to figure sample resolution, which generally levels off at fairly low sampling rates. This is not surprising given the limited amount of additional information provided beyond a fairly coarse sampling of a handwritten figure.

The desired approach for training will depend upon the relationship of computational effort required to each of these variables. These relations are shown in Figure 25. Computational effort appears to correlate with the ~4.5 power of corpus size; while still polynomial, this strong influence is prohibitive, and may motivate possible decompositional techniques discussed later on. Surprisingly, run time initially drops with increasing sample resolution, reflecting a quicker optimization process. This may be attributable to smoother local maxima resulting from the increased dimensionality and separation of classes in the pattern space, which make it faster to find the global maximum. These results suggest using as much sample resolution as available in the input data for recognition, but limiting the corpus size to that necessary to achieve the desired accuracy. We should note that the results presented here depend strongly upon the optimization algorithm employed (here a version of Nelder-Mead), and may change with the use of more specialized quadratic optimization techniques.

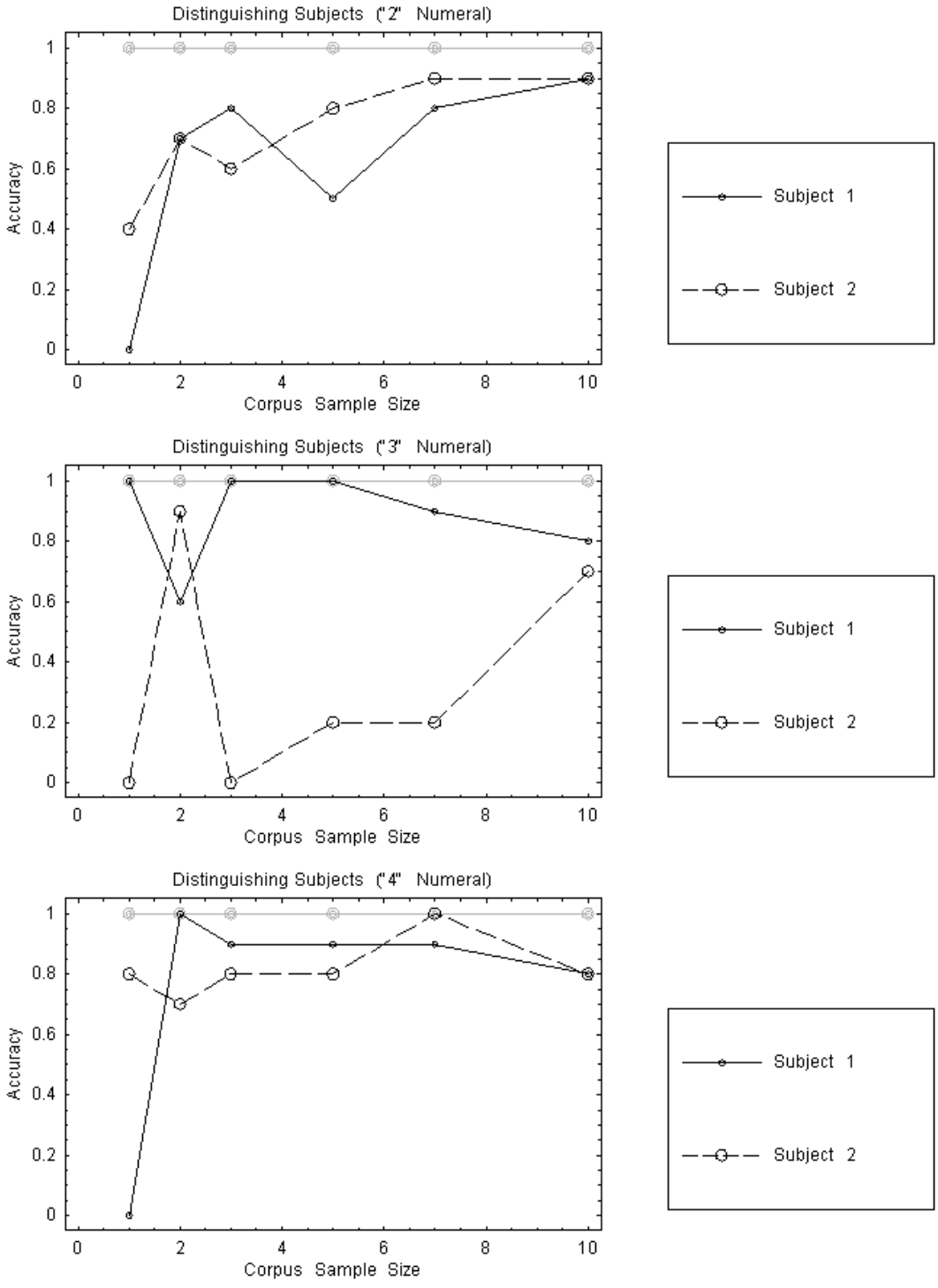

Figure 23: Accuracy versus corpus size for subject classification (zero and midpoint offset results identical). Results on test patterns (black) generally improve with increasing corpus size, though the corpus is perfectly separated in all cases (gray). Fluctuations may be attributed to the strong influence of individual patterns in small corpuses.

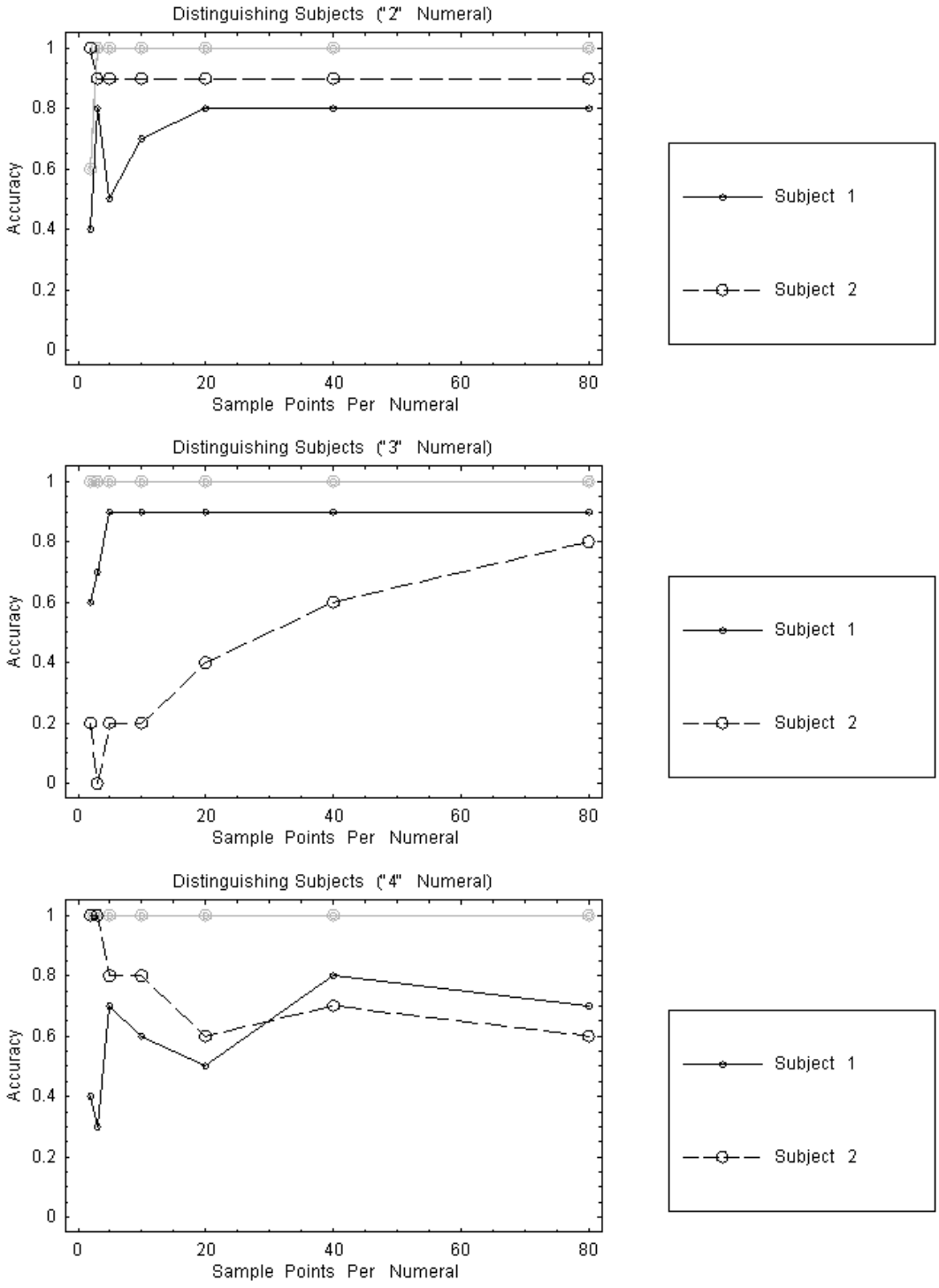

Figure 24: Accuracy versus sample resolution for subject classification (zero and midpoint bias results identical). Results on test patterns (black) seem to reach limiting accuracy at fairly low resolutions, though the corpus is perfectly separated in all cases (gray). Fluctuations may be attributed to the strong influence of individual patterns in small corpuses.

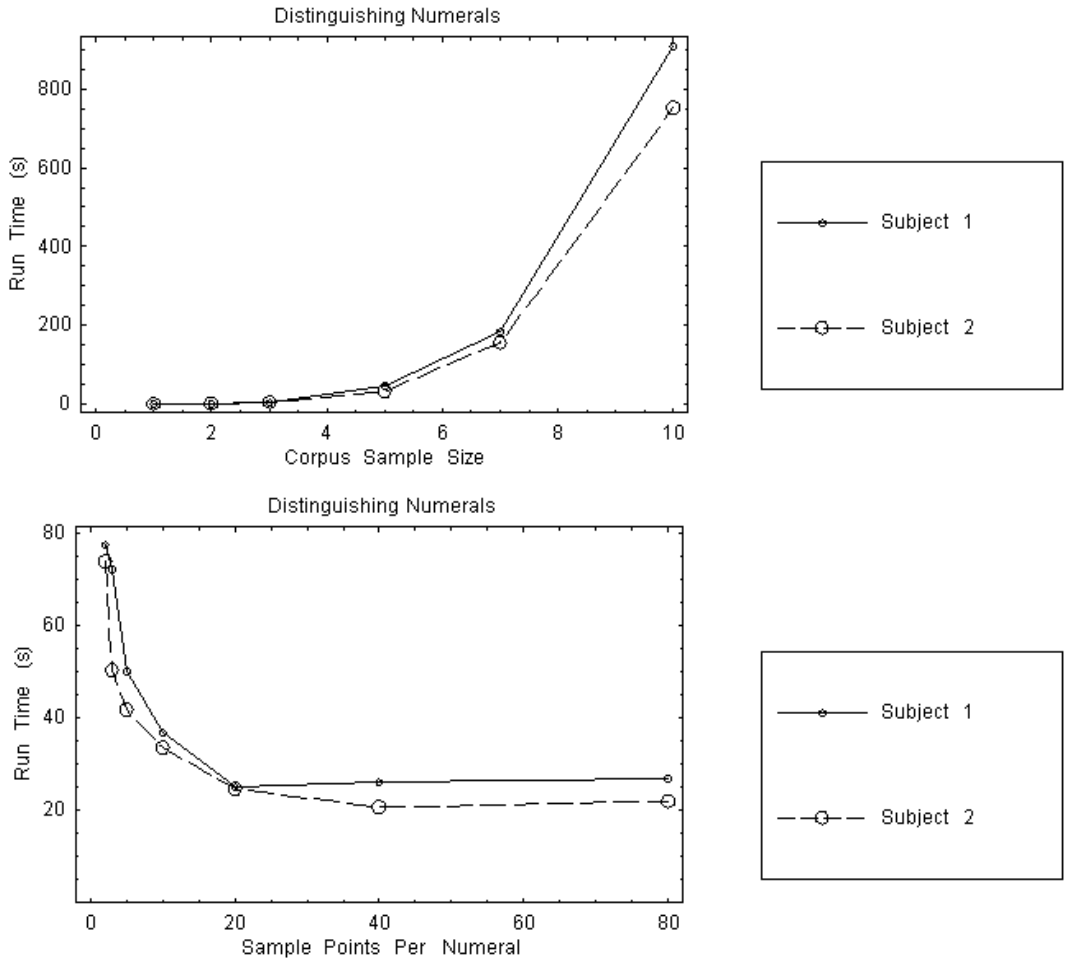

Figure 25: The computational effort required to compute the optimal margin classifiers for varying corpus size and sample resolution. Using our optimizer, run time appears to increase as the ~4.5 power of corpus size, while it drops to a limiting value with increasing sample resolution.

3.3 Gesture Recognition: Putting it All Together

Most of the patterns observable in our results for gesture recognition are familiar from handwriting recognition, so we will summarize them and focus on the noticeable differences in this more advanced application.

First, note that variability in patterns (with respect to the hyperplane margin provided by the classifier) and asymmetry in class separation are large enough that distinct gestures are no longer recognized with perfect accuracy (Figure 26). In this case, the results for zero and midpoint bias are identical, reflecting a midpoint very close to zero in each case. These results suggest finding a mechanism to move the bias closer to the class of interest in multi-class separation.

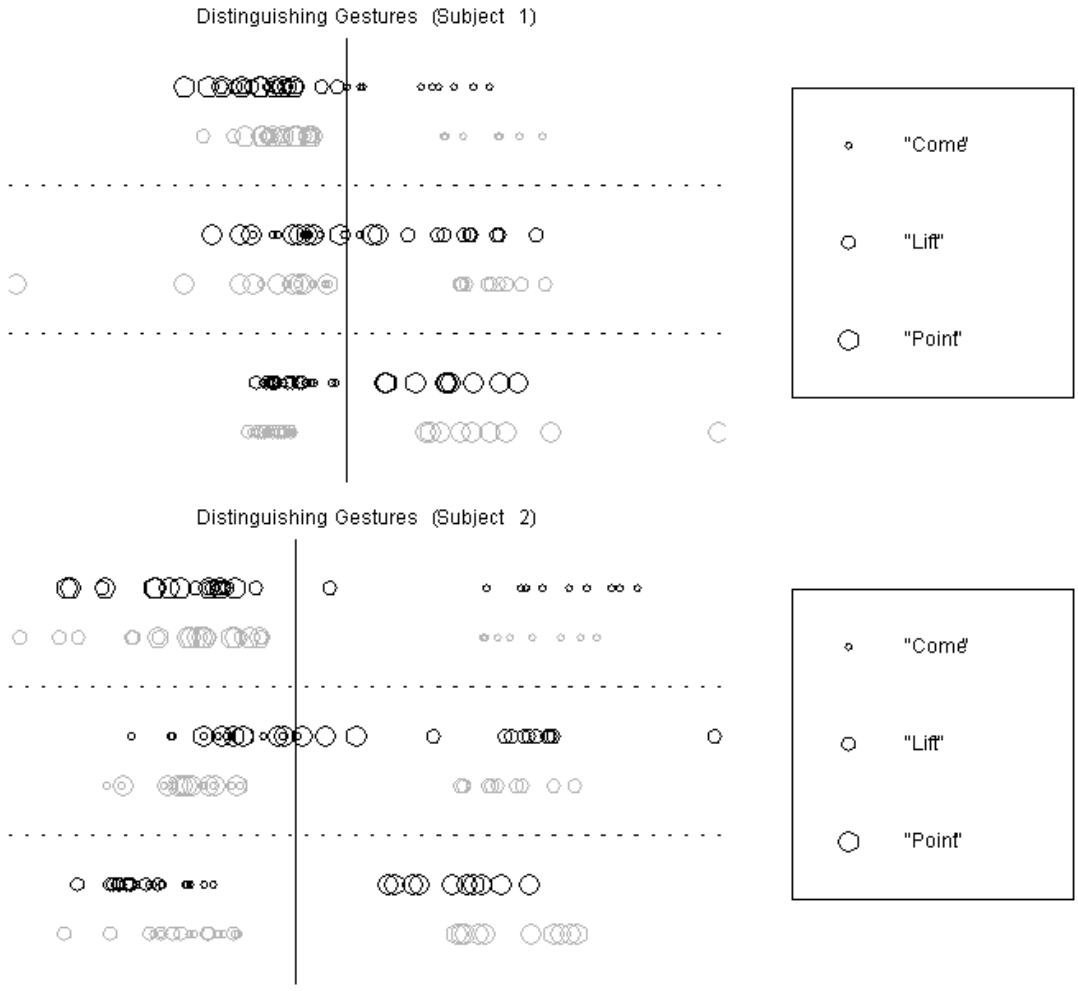

Figure 26: Hyperplane distance distributions for distinct gesture classes (zero and midpoint bias results identical). Note that variability is high enough with respect to the margins that some test patterns spill over the separation boundary provided by the classifier.

Zero and midpoint offset results are also identical for distinguishing subjects making the same gesture. The margins are wide enough for the point gesture to achieve perfect accuracy on test patterns as well as the corpus, and nearly so for the other gestures (all have perfect, relatively even corpi separation).

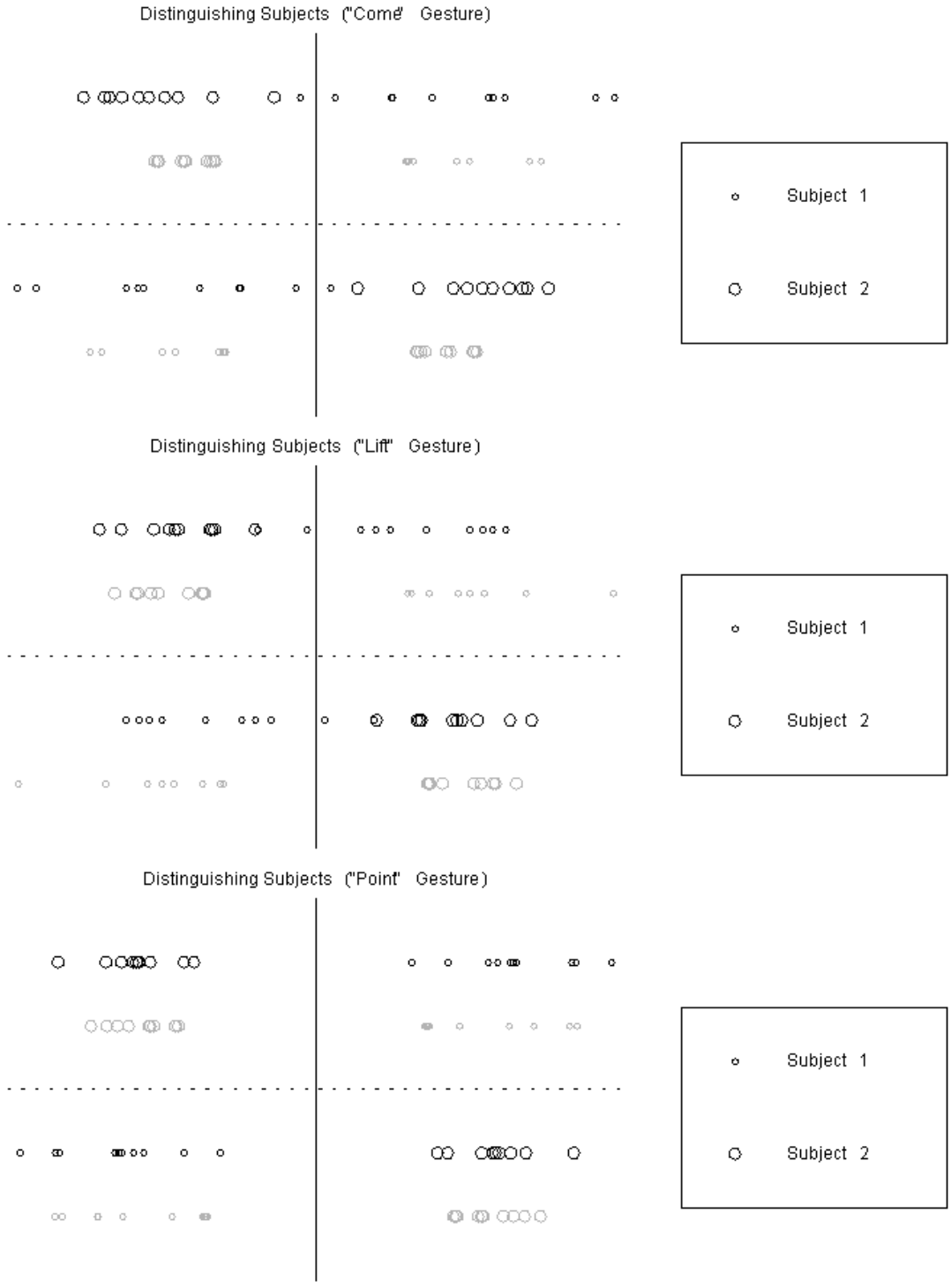

Figure 27: Hyperplane distance distributions for distinguishing subjects making identical gestures.

Unlike in the case of handwriting recognition, classification of distinct gestures made by the same subject is no longer perfect, but shows strong improvement with corpus size, particularly above a threshold of ~3 training patterns each. Only the “Come” gesture for Subject 1 seems unusually problematic. (Figure 28)

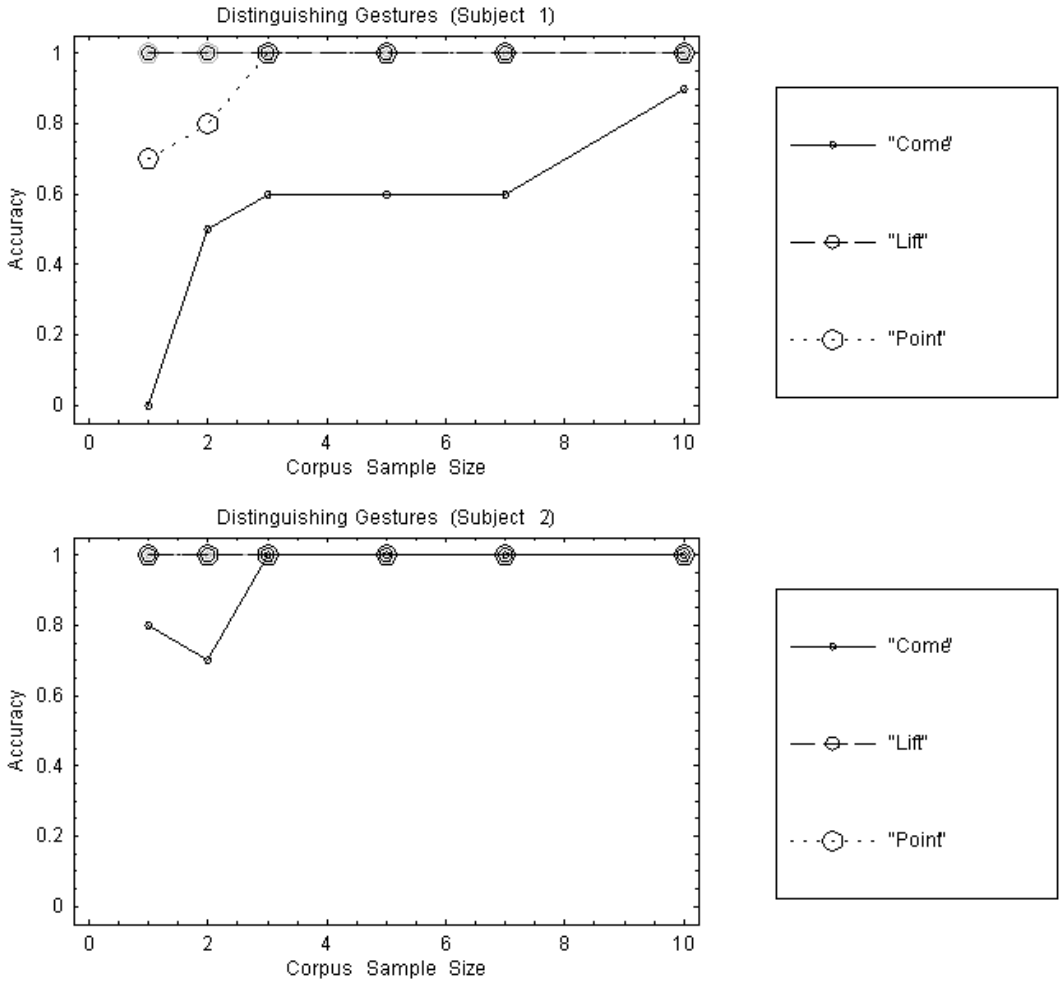

Figure 28: Accuracy variation with corpus size for distinct gestures shows strong improvement to near perfection above a critical threshold of ~3 training patterns each.

The variation of accuracy with sample points per gesture again shows a strong threshold effect, reaching perfection at only 5 points per motion with the exception of Subject 1’s problematic “Come” gesture.

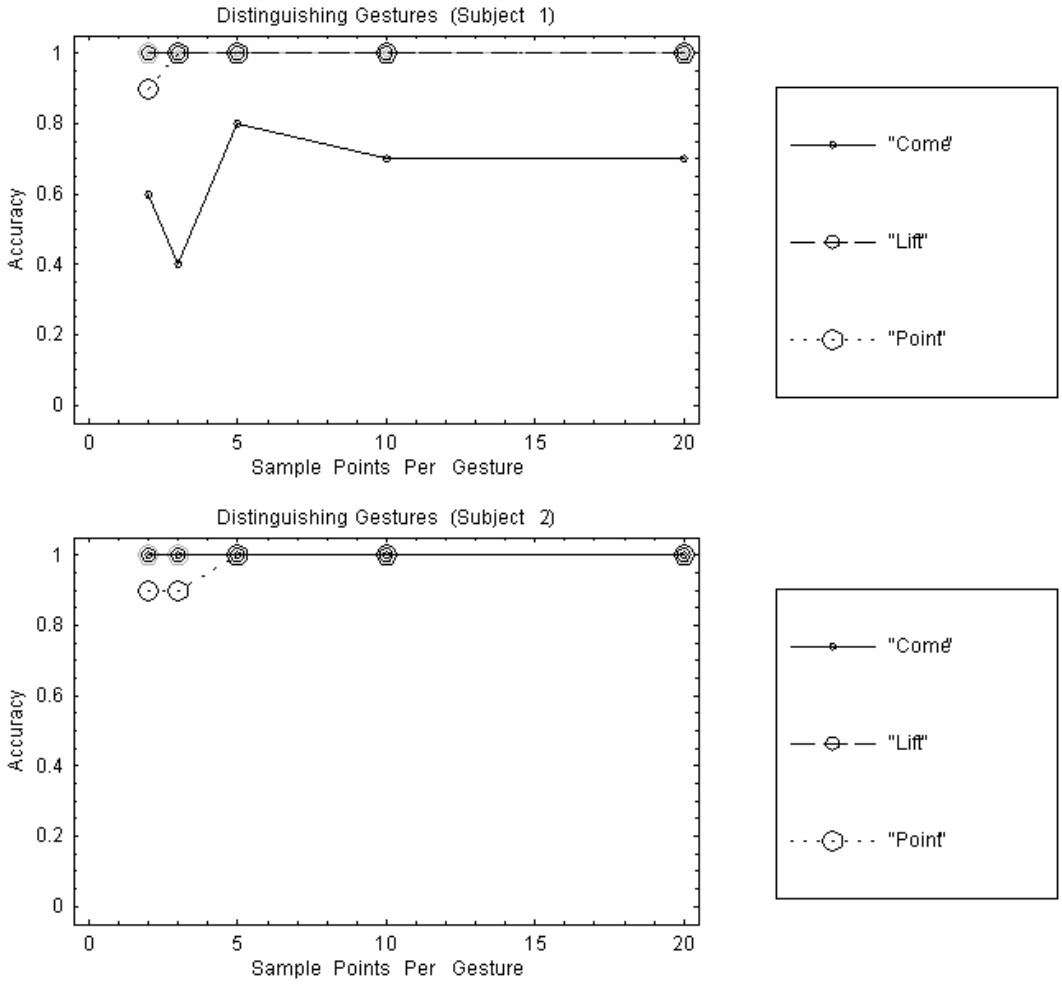

Figure 29: Accuracy variation with sample resolution for distinct gestures shows strong improvement to near perfection above a critical threshold of ~5 sample points per gesture.

Looking at accuracy for distinguishing subjects making identical gestures, we again see promising results with increasing corpus size, though it is possible one or two gestures may level off slightly below perfect recognition. (Figure 30)

The threshold of limiting accuracy for sample resolution appears slightly higher for gesture recognition (~10 points per gesture), which makes sense given that the pattern space dimensionality is three times higher, so that the threshold of full determination of hypersurfaces by support patterns is likewise tripled. (Figure 31)

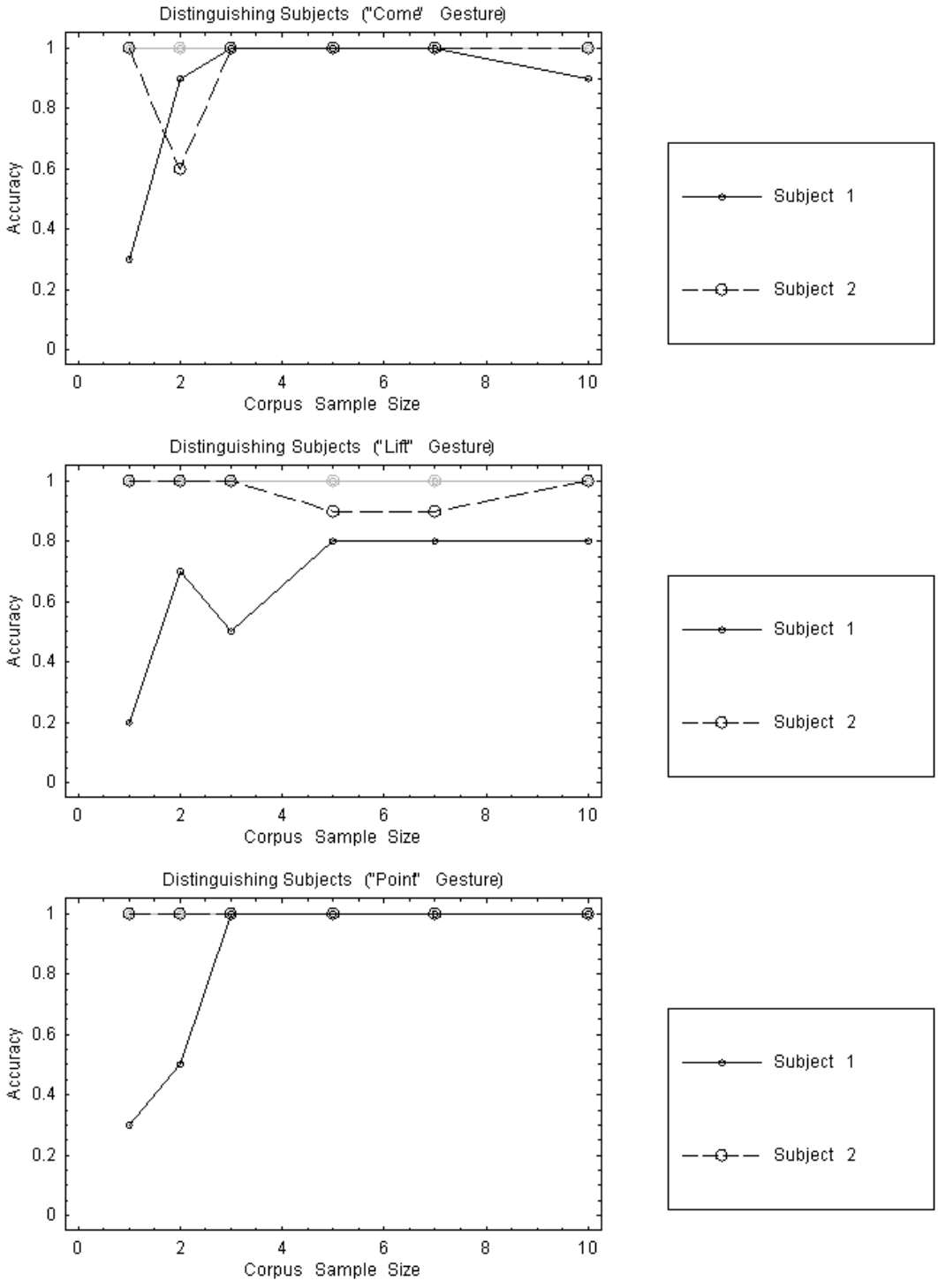

Figure 30: Accuracy variation with corpus size to distinguish different subjects shows strong improvement to the 80-100% range above a critical threshold of ~5 training patterns each.

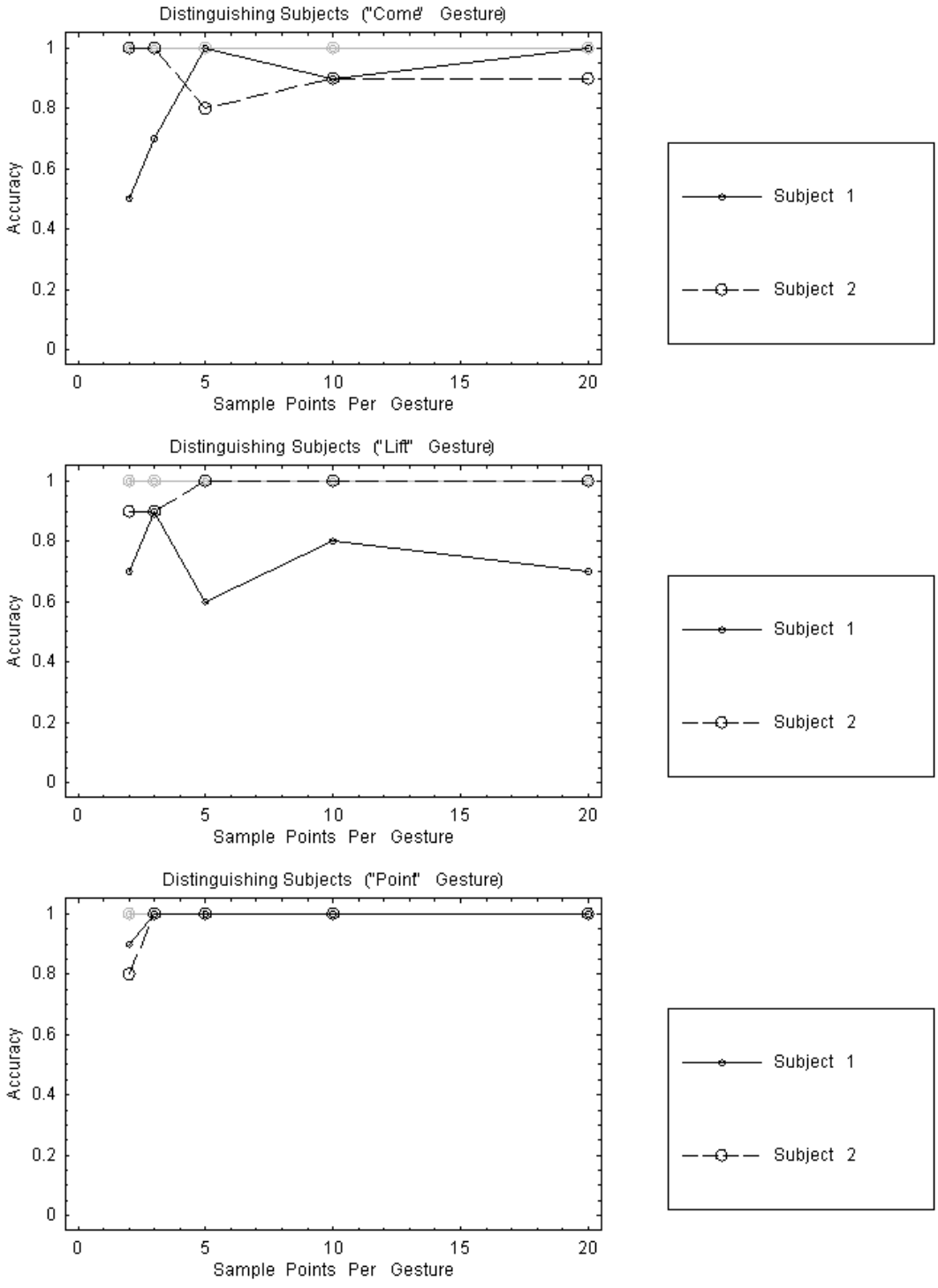

Figure 31: Accuracy variation with sample resolution to distinguish different subjects shows strong improvement to the 80-100% range above a critical threshold of ~10 sample points per gesture.

The trends for computational effort with respect to corpus size and sample resolution are virtually identical to those for handwriting recognition. This is not surprising given that the primary distinction between the problems is an increase in dimensionality of the pattern space, which as we have seen has minimal effect on run time. The higher variation seen in the gesture recognition domain does not appear to significantly affect the performance relationships of the algorithm (Figure 32). The resulting recommendations for best performance are thus unchanged.

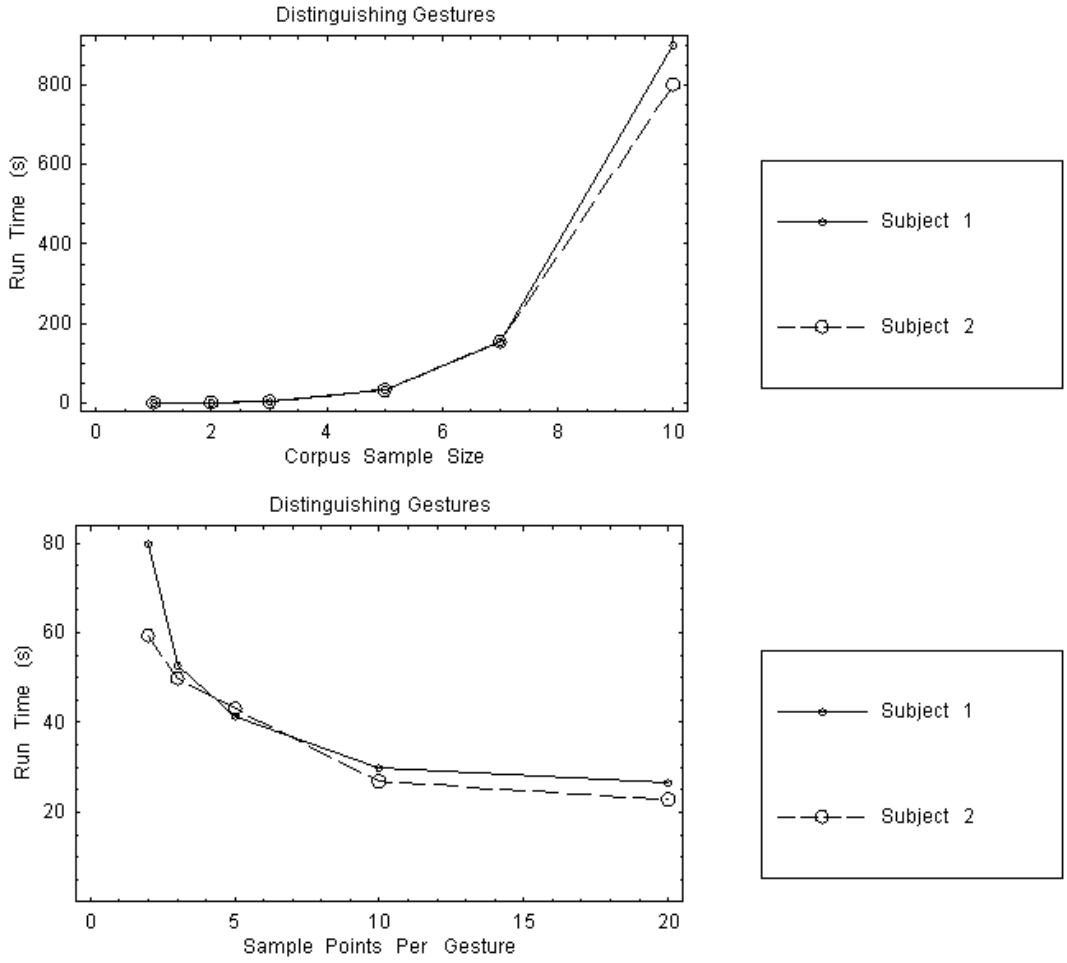

Figure 32: The computational effort required to compute the optimal margin classifiers for varying corpus size and sample resolution closely parallels results observed in handwriting recognition.

4. Possible Extensions

This section covers some extensions to our project that we thought about or partly implemented but seemed out of the scope of our project to complete.

4.1 Pattern Recognition Given Continuous Input

At the moment, all our classifiers are tested using only discrete test inputs. This means that the program must know exactly when an input will arrive in order to perform the required pattern recognition. In the astronaut gesture recognition scenario, this could correspond to a switch or button of some sort that the astronaut must turn on before making a meaningful gesture. While this switch and discrete gesture method could certainly be used in real world applications, it may prove unnatural and unnecessarily increase the user's workload. A desirable alternative would be a program capable of recognizing a meaningful gesture among a continuous stream of data containing all user actions, meaningful or otherwise.

In our system, we were able to adjust our hardware interface program to export two simultaneous outputs. One continuously records user motions as a list of six dimensional vectors while the other selectively records part of the user motion. For the latter output, recording can be turned on or off by way of a single keyboard stroke, and discrete segments are separated by a divider symbol. This interface system allows us to use the same gesture sequence for both discrete and continuous gesture recognition; it not only cuts down the testing data generation time, but also provides the means for direct comparison between discrete and continuous recognition using the exact same gesture. Unfortunately, we were unable to complete the code required to analyze continuous data; however, we were able to identify some important features needed in order to extend our program to handle such cases.

Since we do not know when a meaningful gesture will appear among motions made by the user, we must continuously run the 6 dimensional vector sequences through the classifiers. To accomplish this, we would like to implement a buffer that moves along the data stream at some predetermined step size. A buffer in this case is an imaginary slide of a fixed length. There should be a different buffer for every classifier. At each step, the data contained within a buffer will be tested against the associated classifier to look for a match, as visually depicted in Figure 33.

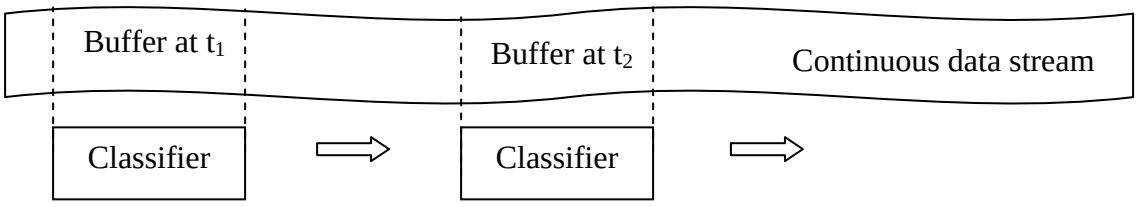

Figure 33: Buffer along continuous data stream. To perform dynamic classification over a continuous data stream, we step through the data stream with a classifier as comparison at each time step

One problem with using a fix buffer is that the length of a gesture is not fixed. As an approximation, we can use the average corpus gesture length as the buffer length; however, we risk missing gestures due to length variations. Therefore it may be necessary to add to the training data set meaningful gestures with the ends chopped off or extended.

The choice of step size for the buffer can also affect the performance of the program. We could shift the buffer by one data point per step, which gives us the best chance of finding a meaningful gesture if one exists. However, this may be too computationally demanding. Increasing the step size would improve the computational performance, but could also decrease the accuracy of the algorithm. The optimal step size can only be determined by amble testing and may vary from gesture to gesture.

4.2 Distinguish Pattern from Non-pattern

All of the testing presented in this report consists of classifiers separating different classes of gestures from one another or separating the same gesture made by different test subjects. We have not constructed or tested a classifier dividing a class from all data patterns not contained in that class. The key reason for not performing this test is that for discrete data recognition, we know that each input should match one of the predetermined patterns: if an astronaut turns on the recognition switch and then makes a gesture then the gesture should not be random. However, this is not true for continuous data recognition. In this case, meaningless gestures are made between meaningful gestures, and the classifiers must be able to distinguish one from the other in order to successfully filter out and recognize the astronaut's intentions. This type of data classification presents some issues.

First of all, generating the complete set of meaningless data is not easy because it consists of all possible patterns aside from the one meaningful pattern that we aim to recognize (see Figure 34). But as an approximation, we could record a set of typical motions made by a test subject within a certain work environment and use that as the opposing class. However, the effectiveness of this data generation method cannot be determined without thorough testing.

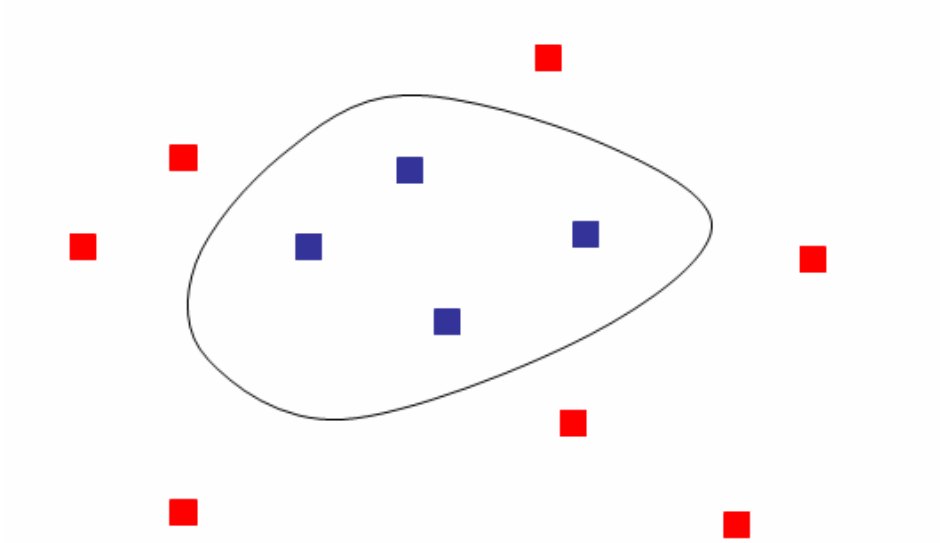

Figure 34: Pattern against non-pattern. The blue points represent a cluster of four data points for a class of patterns, the red points represent other pattern that does not belong to that class, and the solid curve around the blue points represents the classifier between pattern and non-pattern points.

Also, data corresponding to a specific pattern follows a certain trend: for example, they could form a cluster within the space of all possible patterns. And all of our test cases involve distinguishing a specific pattern from other distinct patterns. Since the classifier only needs to provide a barrier between distinct groups of patterns, one can see how a linear decision function would work well in this case (refer to Figure 2). However if we extend our program to handle the case of recognizing pattern from non-pattern then a linear classifier may or may not be acceptable. In Figure 34 we see that if the blue points represent a pattern cluster then the red point must cover all space around the blue points. In this case, a linear classifier cannot possibly distinguish one class from the other. Therefore, a program that processes continuous data may also require the ability to determine the optimal or at least an acceptable classifier, linear or otherwise.

4.3 Alternative Classification for Multiple Classes

All classifiers discussed so far only deals with binary classification between two classes. Therefore, when testing with n classes (n > 2), we calculated n classifiers, each distinguishing a specific class from the rest (see Figure 35). The subsections below discuss some possible alternatives to this approach for handling classification of multiple classes. However, these are only ideas, and we do not have sufficient data to show whether they work in practice.

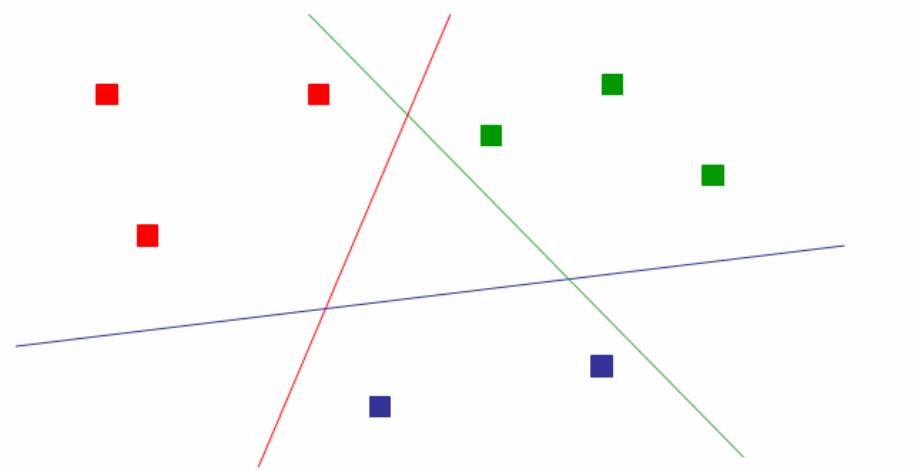

Figure 35: One classifier per class for multiple class classification. The red, green, and blue points represent different classes. The 3 lines represent the 3 classifiers used to separate these classes. The red classifier separates the red pattern from the blue and green, the blue classifier separates the blue pattern from the red and green, and the green classifier separates the green class from the red and blue.

4.3.1 Single classifier for Multiple Classes

The idea here is to use one classifier to distinguish more than two classes of patterns. Here we once again visit the notion that data for a single pattern lies in a cluster, away from other patterns. In fact, during testing, we noticed that data for a specific pattern tend to lie a similar distance away from the decision boundary. This opens the possibility of using one classifier to distinguish multiple classes by assigning a range of decision values for each class. Instead of having $D > 0$ correspond to class A and $D < 0$ correspond to class B, we could have classes A, B, C, etc., where $(a < D < b) \rightarrow$ class A, $(b < D < c) \rightarrow$ class B, $(c < D < d) \rightarrow$ class C, etc.

Figure 36 shows a single classifier used to distinguish 4 classes. Assume the red and green classes rests on the positive side of the classifier and the black and blue on the negative, then the distances *a*, *b*, *c*, and *d* should have the property $b < d < c < a$. If *a*, *b*, *c*, and *d* are average distances from a class to the classifier, then we could use the following rule or a similar one to distinguish the classes: for pattern **x**, if $D(\mathbf{x}) < \frac{b+d}{2}$, then **x** belongs to the black class, if $\frac{b+d}{2} < D(\mathbf{x}) < 0$, then **x** belongs to the blue class, etc. Intuitively, this approach should work best when dealing with patterns that are dissimilar from one another.

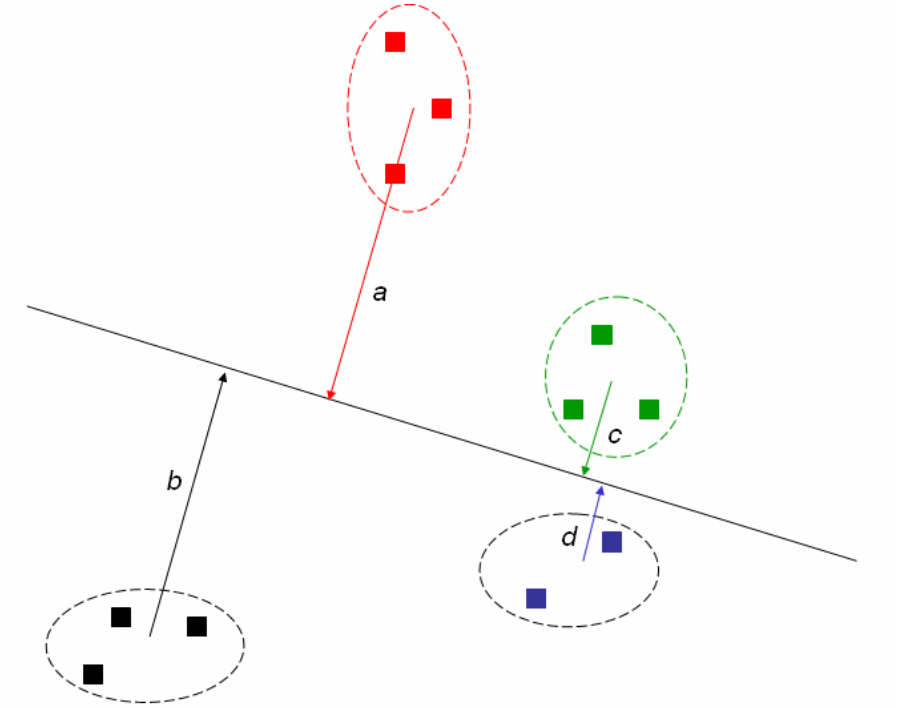

Figure 36: One classifier for multiple classes. The red, green, black, and blue points represent 4 distinct classes. The black line is the single classifier used to distinguish these classes. The red, green, black, and blue arrows labeled *a*, *b*, *c*, and *d* are representative distances from their respective classes to the classifier.

4.3.2 Binary Tree-Structured Classifiers

Also, instead of calculating a classifier for each class we could separate all classes into two groups with a classifier dividing the groups, and then further separate and divide each group. For example, in Figure 37, assume the red and green classes rest on the positive side of the black classifier, and the green class rests on the negative side of the red classifier. In this case if $D(\mathbf{x}) > 0$ for the black classifier and $D(\mathbf{x}) < 0$ for the red classifier, then pattern **x** belongs to the green class. However, with this method also comes the question of how best to divide the classes into groups in order to obtain maximum separation between groups.

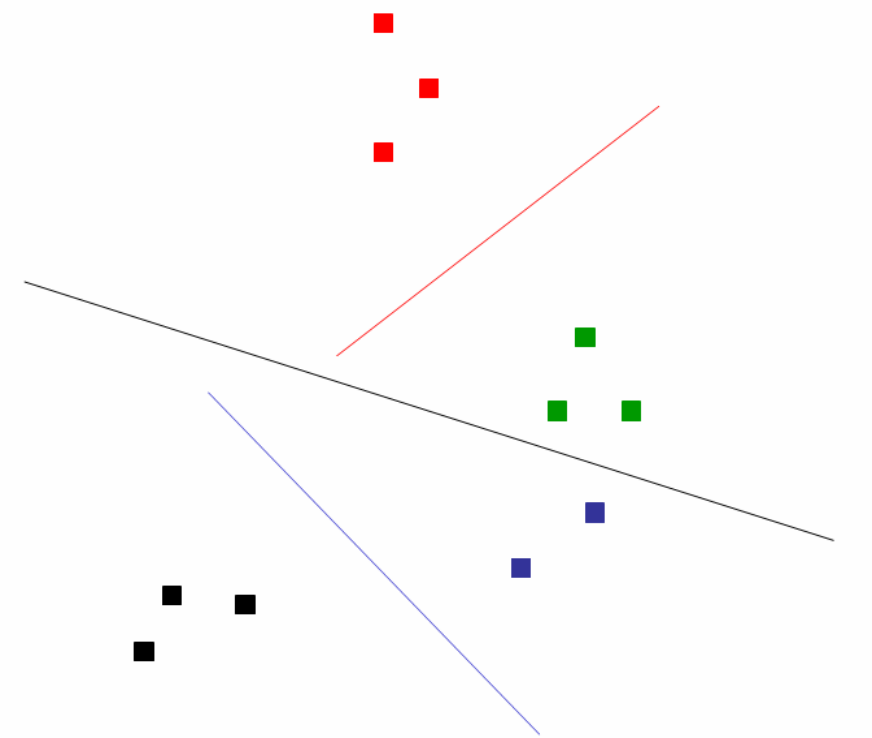

Figure 37: Binary, tree-structured classifiers. The red, green, black, and blue points represent 4 distinct classes. The black, blue, and red lines represent classifiers. The black classifier separates the red and green class from the black and blue class. The blue classifier further separates the black class from the blue class, and the red classifier separates the red class from the green class.

Conclusion

In this paper, we have described the optimal margin classifier algorithm in detail. We also described the implementation of this algorithm with respect to our test environment, and conducted an analysis of the algorithm using our test results. Our implementation may be useful for future space flight activity because we can conceivably integrate the small hardware components into an astronaut's spacesuit so that a robot in collaboration with the astronaut would be able to detect the astronaut's arm and hand gestures. Further work is required to fully develop this capability.

Acknowledgements

The authors of this paper would like to acknowledge the MIT Man-Vehicle Laboratory for providing us with the InterSense tracking device which was used in the gesture recognition portion of our project implementation. Specifically, we would like to thank Dr. Andrew Liu for his support of the project by aiding us with the hardware equipment setup.

References

[1] B. E. Boser, I. Guyon, and V. N. Vapnik, “A Training Algorithm for Optimal Margin Classifiers”, *In the Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pps144-152, Pittsburgh 1992.

[2] Vladimir Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer Verlag, New York, 1982.

[3] InterSense. *IS-600 Precision Motion Tracker User Manual*.