Signature Recognition on Multitouch Computers

Chris Moore

10 December 2010

1 Introduction

Mobile computing devices sporting multitouch displays for user input such as Apple's iPhone and Google's Nexus S have exploded in popularity in the last few years. On such devices, performing a common function such as initiating a phone call often requires the user to look at the screen and tap several elements. For instance, to call someone using an iPhone requires looking at the screen and tapping at least twice: once to select the icon for the phone application, and again to select the person to call. Such devices, however, are often used in situations such as driving in which looking at the screen is dangerous or impossible, so it is useful to have a way to trigger specific, common actions without looking at the screen. One way to do so is to allow the user to assign specific actions (ex: "call Mom") to specific "signatures" - continuous shapes traced out on the multitouch display.

Here we explore an algorithm for classifying such signatures using training data provided by the user. The algorithm is encapsulated in the provided iPad app. The user is able to define as many signature classes as he/she wishes and provide multiple training examples for each class. When finished, the user can enter test mode and the app will classify each new signature the user then enters.

2 Algorithm

2.1 Goals

At minimum, the algorithm should satisfy the following goals:

1. Recognize signatures invariant to rotation and scale.
2. Detect the presence of outliers in test data since there is no guarantee that a user will enter a signature that "reasonably" corresponds to one of the defined signature classes.
3. Ensure that all examples in one signature class are sufficiently distinct from all examples in other signature classes.

2.2 Scale and Rotation Invariance

Raw input to the algorithm consists of a sequence of screen coordinates along the path of the user's signature provided by the OS. For instance, the sequence of coordinates produced when drawing the letter 'M' is shown below.

To ensure that the algorithm recognizes signatures independent of rotation, we transform the provided screen coordinates to a feature vector that is rotation-invariant. Specifically, we track the direction of the signature, relative

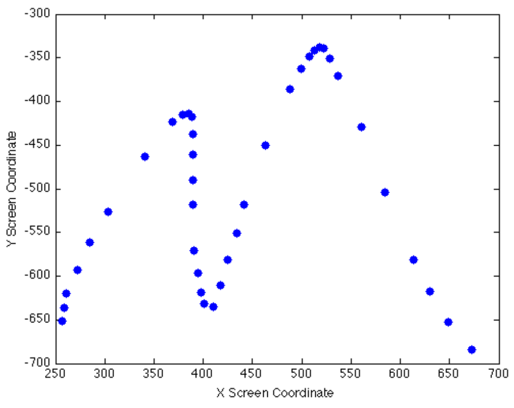

Figure 1: Screen Coordinates while Drawing the Letter 'M'

to the initial direction. Furthermore, to ensure that the algorithm recognizes signatures independent of scale, the feature vectors list this direction as a function of the fraction of total distance travelled.

Specifically, let $z_i, i \in \{1, 2, ...m\}$ be the ith screen coordinate in the signature. The total distance travelled is computed as:

$$D = \sum_{i=1}^{m-1} ||z_{i+1} - z_i||$$

We then normalize the distance between each point by the total distance travelled:

$$d_i = \frac{||z_{i+1} - z_i||}{D}, i \in \{1, 2, ...m-1\}$$

and compute the direction between sequential points:

$$u_i = \frac{z_{i+1} - z_i}{||z_{i+1} - z_i||}, i \in \{1, 2, ...m-1\}$$

Then, given a set of normalized distances $d_j^{interp}, j \in \{1, 2, ...n\}$[1] at which we'd like to have direction information $u_j^{interp}, j \in 1, 2, ...n$, we linearly interpolate the d_i's and u_i's to calculate the u_j^{interp}'s.

Finally, we calculate the elements of our feature vector, x as:

$$x_j = cos^{-1}(u_1^T u_j^{interp})$$

The feature vector extracted from the signature plotted in Figure 1 is shown in Figure 2

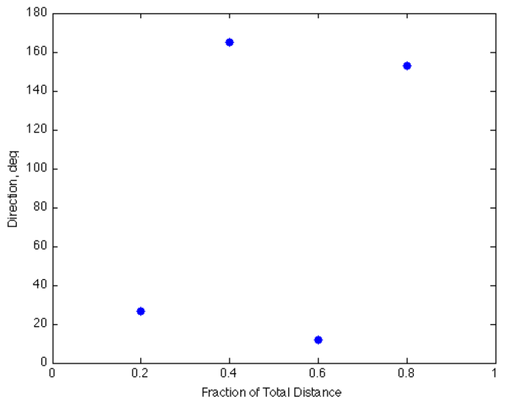

Figure 2: Features for One Example of Letter 'M'

2.3 Outlier Detection

When users enter a signature, there is no guarantee that they will enter a valid signature. They could enter an *outlier*, a signature that does not match any of the defined classes. Thus, a generative model was selected so that $P(x)$, the likelihood of seeing a given feature vector could be computed and used to detect

[1]For this project,we use $d^{interp} = [0.2, 0.4, 0.6, 0.8]^T$

outliers. Specifically, the Gaussian Discriminant Analysis model was used:

$$x|y_i \sim N(\mu_i, \Sigma)$$
$$y \sim Multinomial(\phi)$$

Here y is our vector of signature classes.

The threshold $P(x)$, below which the software considers a test example to be an outlier, is set to the minimum $P(x^{(i)})$ out of all the $x^{(i)}$'s in the training set.

2.4 Distinctness Detection

One of the first training sets collected consisted of approximately 20 examples each of the letters 'M' and 'N'. Running leave-one-out cross-validation (LOOCV) on this dataset with an early implementation of the algorithm gave a test error of 34%, unacceptably high. Projecting the data into a 2-dimensional subspace selected by PCA and plotting in Figure 3 clearly shows the problem.

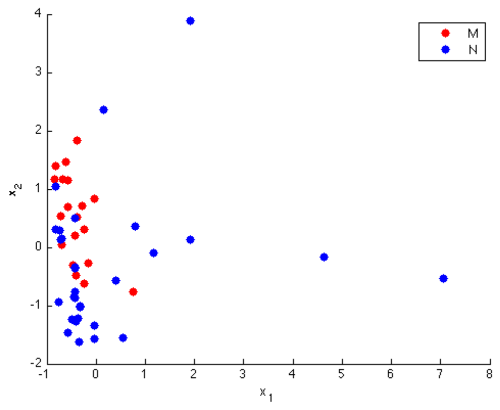

Figure 3: Feature Vectors Projected into 2D Subspace

There is significant overlap in training examples from each class using the feature vectors described above. There are two ways to resolve this problem:

1. Modify the feature vectors so that 'M' and 'N' are more distinct.

2. Add a check to ensure that examples in different classes are sufficiently distinct from one another that they will not easily be confused.

While the first modification may help with these examples, there is no guarantee that it would help with all examples that users could enter. It would still be possible, for instance, for users to enter identical examples (in feature space) for two different signature classes. Thus, the second option was chosen. After each training example is entered, we run LOOCV on the training set and ensure that the test error is zero.

3 Implementation

The app was implemented using a combination of C++ and Objective-C. It requires the iOS 4.2 SDK. Feature extraction is handled in the C++ class *DirectionVsDistanceMovedSignature*, while training and testing are implemented in *SignatureRecognizerGda*. The app will run in the iPad simulator included in the iOS SDK, but it is much easier to enter consistent training data using one's finger on an iPad than using a mouse in the iPad simulator. The interface was designed to be used in landscape mode.

Evaluating a multivariate Gaussian parameterized by covariance matrix Σ and mean μ at a point x requires computing $\Sigma^{-1}(x-\mu)$ and $|\Sigma|$. The LAPACK function *dgetrf()* was used to perform LU decomposition of Σ, and then *dgetrs()* was used to solve the system of equations required to compute $\Sigma^{-1}(x-\mu)$. The determinant is easily computed from the product of the diagonals of the U factor, but in reality it is not needed. When computing $P(y|x) = \frac{P(x|y)P(y)}{P(x)}$ for classification, the normalization constant for the Gaussians cancel. For outlier detection, all of the $P(x^{(i)})$'s for the training examples $x^{(i)}$'s and $P(x)$ for test example x will be scaled by the same normalization constant $((2\pi)^{n/2}|\Sigma|^{1/2})^{-1}$ so it is ignored (i.e. we actually threshold on "unnormalized $P(x)$").

4 Future Work

There is ample room for future work. Several avenues include:

1. Evaluating other models such as hidden Markov models that might be better suited to sequential data.

2. Improving how we handle failures in the LOOCV-based distinctness test. Currently, if after the user enters a new training example, the new training set fails LOOCV, we simply block the user from entering test mode. We make no effort to either automatically or with user input reject the new training example. A better approach to distinctness failures might be to run a clustering algorithm on the training set to try and detect mis-labeled examples. The user might then be given the option of re-labeling or rejecting an example.

3. Instead of using a constant $d^{interp} = [0.2, 0.4, 0.6, 0.8]^T$ to compute feature vectors, applying a feature selection algorithm to find the most relevant subsets of values from, for instance, $d^{interp} = [.01, 0.02, ...0.99]^T$. Care must be taken, however, to include enough features that outliers are not easily confused with valid examples.