Greedy Point Match Handwriting Recognition

Steven Stanek Woodley Packard

May 16, 2005

In which we propose a new algorithm to assist in handwriting recognition.

1 Assumptions And Overview

For this paper, we assume that a handwriting recognizer has access to a handwriting profile based on a large number of samples of a user's handwriting. A user might be asked to write each letter, number and punctuation mark that is to be recognized 20 times. In the case of letters, upper and lower case must be written separately and in all cases the glyphs must be formed in the same manner. For example, it is inappropriate for a user to sometimes draw the body of a five and then the hat and other times the hat first and then the body.

The training data is scaled to fit within a rectangular (not square) bounding box with the length of one dimension set to a constant. Since we are dealing with pixels of finite size, the variable sized dimension of the bounding box never takes on a zero value.

For recognition, the user writes an unknown symbol which is then scaled to the same bounding box as the training data. The recognizer has access to all of the training data and precomputed statistics about this data. By comparing aggregate statistical information from the training data with statistics from the unknown symbol, and by directly engaging in a pointwise comparison between the unknown symbol and all known symbols in the training set, the recognizer identifies what it believes is the best match.

2 Terminology

For the purposes of this paper, we will use the following lexicon.

1. Stroke: A series of points which represent one period of time during which the pen did not leave the writing tablet. The points are ordered with the first point representing the pen down location and the last representing the pen up location. [ex: The "hat" on a '5' is a stroke, the "body" is another stroke.]

2. Glyph: A single handwritten figure, composed of one or more strokes. The temporal order of the strokes is known. Recognition translates a glyph into a symbol. [ex: A handwritten '5', with the first stroke for the "hat" and the second stroke for the "body". The strokes together are the glyph.]

3. Sample: Generally used to mean a Glyph.

4. Symbol: A recognizer takes handwriting (a glyph) and determines which symbol it represents. Symbols generally are the upper and lower case letters, numbers and some punctuation marks. [ex: The letter 'E', the number '2' and an exclamation mark are all symbols.]

5. Training Data: Training data consists of glyphs whose symbols are already known. [ex: We may have 20 glyphs for the letter 'E'.]

6. Handwriting Profile: A Handwriting Profile encompasses training data and perhaps some additional data from preprocessing steps.

3 Feature Vector Recognizers

3.1 A Layman's View

Though not exactly correct, from a layman's perspective, one can think of a feature vector recognizer (henceforth called a vector recognizer) as a game of 20-questions with imperfect questions. The designer of the handwriting recognizer chooses certain properties (questions in our analogy) which she wants the recognizer to examine. Unlike a standard game of 20-questions, the properties do not have "yes/no" responses but rather numerical ones. The recognizer looks at its training data and determines how the various data for each symbol responds to the "questions". When an unknown glyph is provided, the recognizer asks the same "questions" of it. To find a best match, the recognizer picks the symbol whose training data responses are closest to those of the unknown glyph. As in 20-questions, the individual questions are usually insufficient to determine which character was written; it is instead the combination of questions that provides an accurate result.

3.2 Detailed Explanation

A feature vector recognizer, such as the one used in CIT [2], is a common and fast tool used for recognizing handwriting. Vector recognizers generally take ordered points from a handwriting sample and extract statistical properties based on features. A recognizer might, for example consider the ratio of width to height, number of strokes, locations of start and end points, angular properties of strokes and center of mass. The recognizer examines all training samples for a each symbol and computes the average and standard deviation for each of the features. These values are computed and stored as part of a preprocessing step, which takes place before recognition begins.

In the recognition step the recognizer takes a unknown sample of handwriting and computes the same statistical values that we computed for the training data. These values are then compared with the aggregate values for the symbols in the training set. We compute the difference between the aggregate value from the profile and features of the unknown glyph, divide by standard deviation and use the normal curve to find the likelihood that the feature is the same for both the unknown glyph and the profile symbol. This process is repeated for every symbol in the training set and every feature the recognizer uses. The likelihood are then combined using statistical techniques to weigh their relative importance and the symbol with the highest combined likelihood is identified as the best match.

3.3 Limitations of the Vector Approach

The vector recognizer is a surprisingly strong recognizer provided it has a large set of training data. Our tests show that with a well groomed handwriting profile, these recognizers can attain over 99% accuracy when fed samples from their own training data. However, these results are not always mirrored in real life.

Vector Recognizers only look at general characteristics of handwriting, not shape. As a result, they occasionally will mistake two characters which, though quite different in appearance, have similar characteristics. Our experience shows that such problems can only be rectified by changing the way the user draws letters; simple retraining or providing additional samples to the recognizer does not suffice.

Second, we have found that the CIT recognizer performs poorly when its training data is moved between computers with different tablet types. We believe this is because different tablet models sample at different rates and many of the vector features are indirectly dependent on the number of points sampled. Further investigation in this area could include the creation of "filler" points and training the recognizer from several tablets with varying sampling rates.

4 The Greedy Point Match Technique

We endeavored to create a simple yet reasonably accurate shape based recognizer to complement a vector recognizer. Our solution is a technique we refer to as Greedy Point Match (GPM).

4.1 The Algorithm

Greedy Point Match takes every sample in the training profile and places it on a three dimensional grid. The X and Y coordinates directly correspond to the X and Y coordinates of the points within the normalized bounding box while the Z coordinate is the distance along the arc from the pen down point. If graphed in three dimensions, our representation should appear to be growing along the +Z axis from the start point of the stroke to the finish.

Our recognition algorithm simply takes points from an unknown glyph and finds their distances (errors) to the closest points on every glyph in the training data. The process is then reversed and training glyphs are matched against the unknown glyph. The total distances (errors) are summed and the symbol represented by the training glyph with the least total error is chosen as the best match.

4.2 Performance

This algorithm is very costly as it must iterate through all of the samples of a training set and compare every point in the unknown glyph to all of those in the sample. Formally, the running time of the algorithm is: O($NumTrainingSamples \times AvgPointsPerTrainingSample \times PointsInUnknownSample$). This raw computation is extremely expensive and should be avoided when possible. Additionally, the price of the algorithm grows as the set of training data (and thus accuracy) increases. Modest speed increases are made possible through certain spatial subdivision algorithms[1], but these increases are not sufficient to offset long run times on aggressive tests.

4.3 Testing and Accuracy

We tested the Greedy Point Match recognizer using a large training data set of 2589 samples. An individual symbol was removed from the training set and compared against all of symbols still in the training set using our GPM algorithm. We repeated this process for every sample in the entire training set. As mentioned above, this process was quite time consuming and we multithreaded our code to run on both processors of a dual processor computer to speed up testing. The overall result, was a mere 80% accuracy which does not compete well with feature vector handwriting recognizers.

4.4 Variant: Matching To Line

We examined the most obvious variant on the Greedy Point Match algorithm: matching points from an unknown sample to the lines that connect the points in the training data (instead of the points themselves). Our testing showed that this variant resulted in a slight decrease in recognition accuracy but the difference was so slight it is probably not statistically significant.

It should be noted that the our data consisted exclusively of samples taken on a single machine, and all samples had the same sampling rate. We have yet to investigate the implications of using this technique on samples from different sources with different sampling rates. One would hypothesize that under those cases it might perform better than standard GPM.

5 Combining Vector Recognizers and GPM

As stated above, the GPM algorithm is not very accurate but does examine the shape of handwriting. We show that through a combination of a vector recognizer and GPM a very high recognition accuracy can be achieved.

5.1 Implementation

In order to test our hypothesis, we constructed a relatively simply vector recognizer which considered properties such as: aspect ratio of the letter, center of mass, moments of inertia, location of the first point, ratio of points above and below the center of the glyph, ratio of points on the left and right sides of the glyph and

[1]We would like to thank Greg Burns for his help with spatial subdivision and Line Matching

number of strokes in the glyph. Our vector recognizer was not as sophisticated as the CIT recognizer so its accuracy alone was only 90.5%.

Coupling our GPM algorithm to the vector recognizer was tricky as it produces distances (net errors), not probabilities, as its outputs. We used a binary search to find the appropriate weighting constant to assign to the GPM values. Our search results fortunately converged to a single constant which we used. When we coupled the GPM algorithm to our vector recognizer using this weighting constant, we were able to achieve a 98.6% accuracy. We believe that this value could be increased if our vector recognizer is improved to examine more features.

The vector recognizer and GPM seem to stack very well. The accuracy of the GPM was approximately 80% and the accuracy of the vector recognizer was approximately 90%. Another way of looking at this is that 20% of symbols are misrecognized by the GPM and 10% by the vector recognizer. If we assume the two recognizers are completely independent and we could apply GPM to only values which we somehow know the vector recognizer will get wrong. We would expect that GPM would correctly recognize an additional 8% of the total samples giving a total accuracy of 98% which is exactly what we get!

5.2 Pruning for Speed

As we mentioned before, the GPM algorithm is extraordinarily slow. Our original attempts to speed up the combined algorithm dealt with speeding up the GPM itself through various subdivision and caching techniques. These approaches did improve performance but only by relatively small percentage.

Eventually, we realized that we could use the vector recognizer to help us prune the search space. The vector recognizer's running time is a linear $O(NumPointsInUnknown + NumTotalSymbols)$, so it very fast. We used the vector recognizer to find the most promising symbol from the profile and then calculated the GPM values for that symbol only. The vector recognizer allows us to reject an extremely large number of possible symbols because their vector recognizer values alone make it mathematically impossible for them to perform better than the best sample we have found so far. For values which are not automatically disqualified by their vector recognizer values, we begin the GPM algorithm but terminate it before completion if large distances (errors) make it impossible for the current glyph to beat our best sample so far. If a figure does provide a better overall value than the previous "best so far", the old "best so far" is replaced with the new figure and the loop continues until all symbols and all necessary training glyphs have been examined.

The pruning algorithm resulted in a radical speedup. In testing each sample from our 2589 sample training set on a 1.25 Ghz PowerPC G4 we found that the combined algorithm runs in only 19s with pruning as opposed to 658 s without. This results in a total speedup of 34.6 times.

6 Conclusions and Further Work

The GPM algorithm appears to be a promising complement to a vector handwriting recognizer as shown by our results. By exploiting the fast vector recognizer, we can prune the search space and run the slower GPM algorithm only as much as necessary. Logical next steps include improving our vector recognizer and incorporating our recognizer into a program which accepts live handwriting input for real world evaluation.

References

[1] Steven Smithies, Kevin Novins and Jim Arvo. *Behaviour and Information Technology*, 20(1), pp. 84-91, January 2001.

[2] J. Arvo. Caltech interface tools (cit). www.cs.caltech.edu/~arvo/software.html