Identity Detection via Handwriting Analysis

Using the Curvelet Transform to Extract Features

Alex Thomas Parisi
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA
alex.parisi@outlook.com

Jinsol Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA
jinsollee@gatech.edu

***Abstract*—**We have analyzed handwritten letters to determine the identity of the writer. Sample handwriting is preprocessed using edge detection, and the Curvelet transform is performed to identify statistical features including the entropy, the standard deviation, the energy, the kurtosis, the skewness, the second central moment, and the interquartile range of each Curvelet wedge. The k-nearest neighbor, perceptron, and support vector machines are used to classify the features and form an identity class for each individual. With the trained classifier, a new handwriting sample can be analyzed to determine the identity class of the writer.

Keywords*—Curvelet Transform, Identity Detection, Handwriting Analysis, KNN, Perceptron, Support Vector Machine*

I. Introduction

Handwriting analysis has been performed for years, with its main application being used in criminal and civil court cases to prove that a specific person wrote a specific word, sentence, or phrase. Such forensic handwriting analysts perform their analysis by looking at past works confirmed to have been written by the suspect, and then comparing how the letters are formed to an unidentified piece of work whose original author has been called into question. As opposed to a commonly held belief, these experts aren't looking for similarities, but rather the differences between letters - if the differences are too large, then the analyst can say that the two pieces were not written by the same author. However, this process of identifying how letters are written, and the practice of spotting differences in written letters, is a very time consuming process and relies on an individual with a highly specialized skillset. Therefore, we propose a digital solution, in which the author of a sentence can be identified by analysis of both past written works and the current written work being questioned.

This paper will analyze how to perform identity detection by analyzing one's handwriting using image processing techniques. First, the profile of a user is constructed: (1) each user will have to fill out "Training Sheets", in which they print by hand the entire alphabet, upper- and lower-case, one-hundred times to form a training dataset; (2) these "Training Sheets" are then scanned and the individual letters are extracted via Photoshop; (3) seven identifying statistical features are extracted per image, and stored in an array. Now, for each user, we have 52 individual 3x100 datasets to train our classifiers with. We use this training data to test three types of classifiers: K-Nearest Neighbor (KNN), Perceptron, and Support Vector Machines (SVMs). Once all classifiers are trained, we can then begin testing them. We obtain two tests from each user, in which they write out the following sentences by hand: "The quick brown fox jumps over the lazy dog", and "My name is _____". These sentences were then scanned, the individual letters were separated and stored, and the features were extracted. We can then use this 7x1 test vector to classify it using one of the three classifiers. Each letter is analyzed individually, and the result presents what percentage the written sentence belongs to a specific user.

II. Extracting features

To create a training data set, the user provides multiple samples of handwriting for each alphabet in both uppercase and lowercase. The test data includes 100 samples of each letter from two users, resulting in 5200 samples per each individual and 10400 samples total. Each sample image is converted to a grayscale image, performed edge, and decomposed using 5-level Curvelet transform. Canny edge detection technique is used to extract structural information from each image as it increases variance between characters by enhancing features to be extracted. An example of edge detection is shown in Figure 1.

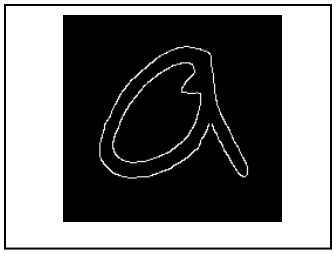

Fig. 1. Edge detection performed on a sample image

The Curvelet transform is ideal for approximating edge discontinuity since it requires less number of coefficients to represent a curve compared to other types of transforms such as wavelet. Since the majority of handwritten characters includes curves, the Curvelet transform is appropriate for extracting features from handwriting. Based on the wedge division in Figure 2, the frequency domain of each image is decomposed into 130(=1+32+32+64+1) wedges.

The features chosen to be extracted from each wedge are descriptive statistics to represent the quantitative analysis of data, including the following: entropy, standard deviation, energy, kurtosis, skewness, variance (second central moment), and IQR. The description of the features is in Table I.

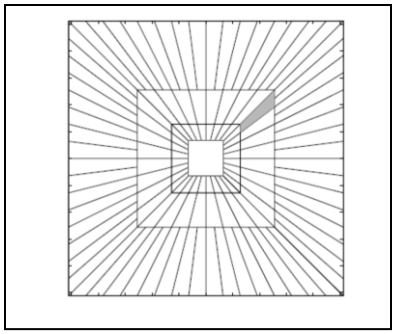

Fig. 2. 5-Level-Cuvelet Transform wedge division

Once the features of each wedge are determined, the average of each feature from all 130 wedges in an image is computed to serve as a feature to represent the certain sample image. The test data set consists of 7 features per image, 700 per character. The extracted features will be classified to create an identity class to compare to the features of a test handwriting image and determine the identity of the writer. The overall process of feature extraction is shown in Figure 3.

TABLE I. DESCRIPTION OF EXTRACTED FEATURES

Features	Description
Entropy	The average amount of information
Standard Deviation	The amount of variation or dispersion of a set of data values
Energy	The sum of squared data values
Kurtosis	The sharpness of the peak of a frequency-distribution curve
Skewness	The asymmetry of the probability distribution of data
Variance	The measure of variation in a set of data values
IQR (Interquartile range)	The amount of variability based on dividing a data set into quartiles

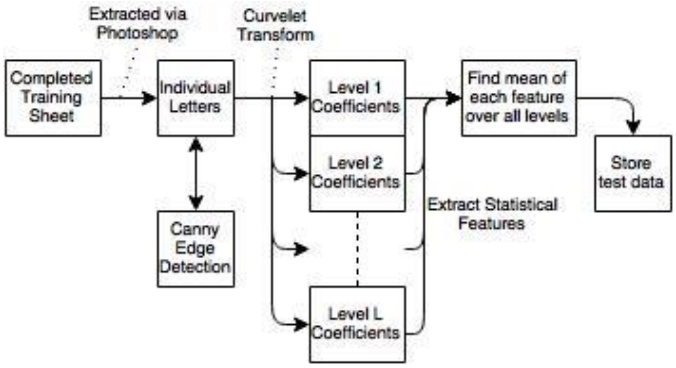

Fig. 3. Feature-Extraction Process

III. CLASSIFIERS

There are three classifiers we will be using during this experiment: the KNN algorithm, the Perceptron algorithm, and the SVM algorithm. We wanted to test a multiple of classifiers to see which one would perform best, as our dataset is only marginally separable. In the end, we believe the KNN classifier is the optimal choice since it performed the best.

The K-Nearest-Neighbor algorithm is a linear classifier that performs its classification in a "majority vote", by analyzing the classes of the k nearest neighbors and determining which class is most represented. For example, with k = 1, the algorithm will look for the closest neighbor and use its class to represent the current point. If k = 3, the algorithm will look for the three closest neighbors. While considered the simplest of machine learning algorithms, we believe that it can provide us with some important information regarding how well it can classify our data, while producing moderately acceptable results. In this implementation, we used the KNN algorithm using k = 1, meaning we were taking the class of the closest neighboring data point.

The Perceptron algorithm is a linear classifier, like the k-nearest-neighbor algorithm, that perform supervised learning of a binary classifier, in which it decides if a point belongs to a specific class or not. This algorithm is tried and true, with the first implementation being used in the 1950s to perform image recognition. The algorithm is performed in three steps:

(1) Initialize. We set the weights to zero or a small random value, and set the threshold γ to a small error limit.

(2) For each observation in the training set, use input x_j and the desired output d_j:

(i) $y_j(t) = f\left[w(t) \cdot x_j\right]$

$y_j(t) = f[w_0(t) \cdot x_{j,0} + \cdots + w_n(t) \cdot x_{j,n}]$

Where $f[x] = \begin{cases} 1 & if\ w \cdot x + b > 0 \\ 0 & otherwise \end{cases}$

(ii) $w_i(t+1) = w_i(t) + \left(d_j - y_j(t)\right) \cdot x_{j,i}$

(3) Repeat step (2) until $\frac{1}{s}\left[\sum_{j=1}^{s} \left|d_j - y_j(t)\right| < \gamma\right.$

The SVM algorithm is another linear classifier, but has the ability to use non-linear kernels, which allows for non-linear classification. Because we are training the SVM using labeled data, we are accomplishing supervised learning. The SVM algorithm works by minimizing the following equation:

$$\left[\frac{1}{n}\sum_{i=1}^{n} \max\left(0, 1 - y_i(w \cdot x_i + b)\right)\right] + \lambda \parallel w \parallel^2 \qquad (1)$$

We can transform the SVM classifier function into a non-linear space by using a kernel $k\left(x_i, x_j\right) = \varphi(x_i) \cdot \varphi(x_j)$. The weights are now defined as:

$$w = \sum_{i=1}^{n} c_i \cdot y_i \cdot \varphi(x_i) \qquad (2)$$

The c_i are obtained by maximizing the following function:

$$\sum_{i=1}^{n} c_i - \sum_{i=1}^{n}\sum_{j=1}^{n} y_i \cdot c_i \left(\varphi(x_i) \cdot \varphi\left(x_j\right)\right) \cdot y_j \cdot c_j \qquad (3)$$

$$subject\ to\ \sum_{i=1}^{n} c_i \cdot y_i = 0\ and\ 0 \le c_i \le \frac{1}{2n\lambda}\ \forall\ i$$

In our implementation, we used SVM parameters that were not the set defaults in MATLAB. To change the SVM algorithm from linear to non-linear, we implement a non-linear kernel function. In this case, the 'rbf', or Gaussian, kernel gave us the best results.

Each of the three classifiers are trained with the features extracted from the test dataset. Using the seven extracted

statistical features per test image, we train a separate classifier for each written letter type, so as to reduce the probability of overlap between statistical information in letter types and to more accurately classify on a letter-by-letter basis. Each individual is passed into its respective classifier, and its predicted class is assigned to an array. Then, we assign a percent chance for each test sentence based on how many letters are assigned to each class.

Because we must train a total of 52 classifiers for each classifier type, the training process takes a significant amount of time. Performance-wise, the KNN trains and predicts our data the fastest whereas the Perceptron trains and predicts our data the slowest – we had initially thought the SVM algorithm would take the longest since it is the most complex classifier we are implementing, but the Perceptron proved to be the slowest to train, taking about two hours of computational time.

IV. Results

After running our analysis, we obtained some interesting results. The information shown below in Table I detail our results. Each test sentence was segmented into its individual letters, and the seven statistical features were extracted in the same way as detailed in Figure 1 under section II.

TABLE II. TEST RESULTS

		Classifier	(%) Chance written by user 0	(%) Chance written by user 1	Verified
USER 0	**TEST 1**	*KNN*	54.2857	45.7143	✓
		Perceptron	42.8571	57.1429	X
		SVM	28.5714	71.4286	X
	TEST 2	*KNN*	50.0000	50.0000	X
		Perceptron	50.0000	50.0000	X
		SVM	50.0000	50.0000	X
USER 1	**TEST 1**	*KNN*	40.0000	60.0000	✓
		Perceptron	51.4286	48.5714	X
		SVM	22.8571	77.1429	✓
	TEST 2	*KNN*	41.6667	58.3333	✓
		Perceptron	41.6667	58.3333	✓
		SVM	41.6667	58.3333	✓

The KNN algorithm performed quite well for being a linear classifier, especially considering our data is 7-dimensional and therefore rather complex. With such high dimensionality data, there is a very small chance that the data will be linearly separable; yet, the KNN correctly classified our test data 75% of the time, with the only failure being a tie in its analysis. These results are surprisingly accurate for one of the simplest binary classifiers.

The Perceptron algorithm performed quite poorly, as it only correctly classified our test data 25% of the time. This may be due to how our features are extracted - linear dependence between features, scalability, and other factors may have an impact on the classification results. If we had more time to tweak our classifiers, perhaps we could have changed the default hard-limit transfer function from the unit step to a different discontinuous function, such as a ramp function or a suitable variant. We also could change the perceptron learning function to see if any of our optimizations will produce better results. Unfortunately, since the perceptron algorithm takes about 3 hours to train our dataset, our tinkering had to be stifled to save time.

The SVM algorithm performed rather poorly, with it correctly classifying our test data only 50% of the time. As with the perceptron algorithm, we would have liked to tweak the SVM parameters in order to obtain better results, but were hindered again by the run-time. Training the SVM algorithm with our test data took about 2 hours to complete, which limited how many times we could edit it and test the results. We produced better results when we standardized our training data, in which MATLAB centers and scales each column of the test data by the weighted column mean and standard deviation. We also initialized the Box Constraint at 100, which controls the maximum penalty imposed on margin-violating observations, and encourages regularization of the test data. This value will change as we train the SVM because we decided to optimized the Box Constraint size and the Kernel Scale size by using MATLAB's built-in auto hyper-parameter optimization to increase the accuracy of the classifier. If we had more time to optimize the SVM, we would have tweaked the parameters more than we already have. Perhaps a variant of the polynomial kernel would have performed better; perhaps we could have used a different hyper-parameter optimization function. In the end, we believe there are faults in both the classifiers and the features extracted from the training/test data.

After tweaking our classifiers extensively without producing optimal results, we believe most problems are associated with the features we extracted. Perhaps the statistical information we extracted from the Curvelet coefficients was not sufficient by itself to produce separability between the two classes. If this is the case, we can most likely reduce the dimensionality of statistical information extracted and add additional non-statistic features to the dataset. The second central moment is equal to the variance, which is the standard deviation squared – this

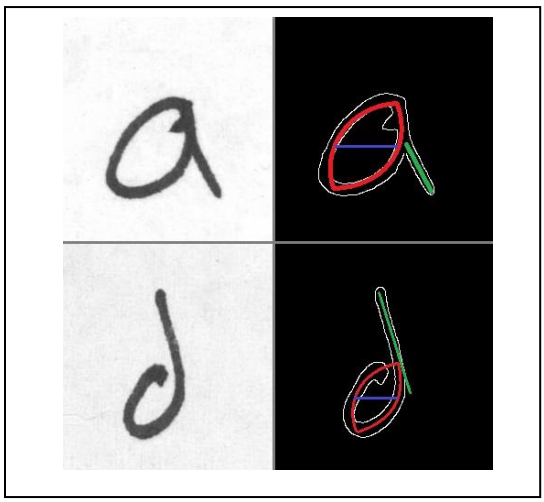

Fig. 4. Example of Geometric Feature Extraction

means one of these features can be eliminated since it directly depends on another feature. Perhaps we could limit the amount of statistical features to 4 dimensions or so, and add an additional 4 dimensions of non-statistical features or so.

Some examples of non-statistical features we could have extracted include slant, diameter, and other geometrical features. These can be extracted from the preprocessed images, before they are decomposed by the Curvelet transform. Figure 4 above details three geometric features we may extract: slant of loop (red), slant of stem (green), and diameter of loop (purple).

Such features would be especially useful in this implementation because we are focusing on *how* people write their letters. After performing our analysis, we believe that the statistical features extracted from the Curvelet coefficients of our test images aren't enough information to make the proper identity classification. If we add additional information, such as geometrically extracted features, we may be able to significantly improve our classification performance.

V. Conclusion

Although our final results were not very good, we believe we have made good progress in the right direction. We will continue to work on our code and test data until we can produce correct classification more than 90% of the time. With further tweaking of the classifiers and their parameters, our classification scheme could see an improvement in classification; however, if tweaking the classifiers continues to produce unacceptable results, the error must lie in the features we are extracting. We believe we can either add additional features or tweak the current features to produce a more independent dataset, resulting in better classification.

Using a couple different classifiers allowed us to analyze how each of them perform on the same training and test data, and provided us with important information regarding the structure of our data. Because we did not obtain optimal results from our classifiers even after extensive tweaking of the parameters, we can make an educated guess that our extracted features were not separable enough. The KNN, while being the simplest of the classifiers used, produced the best results, while the perceptron, the first machine learning algorithm developed, performed the worst. However, if we chose different, more optimal features to extract, we believe that each of the classifiers would perform significantly better. The SVM, while being the most complex machine learning algorithm implemented in this project, gave us mediocre results – again, we believe the error lies in the features extracted from our dataset.

The Curvelet Transform is incredibly useful in the realm of image processing. Because it allows for multi-scale representation of an object, we can extract important information from the image by extracting statistical information from each transform wedge. The wedges in each level contain varying amounts of information about the original image, with the majority of the information being stored in the lower levels – this is not to say that the higher levels are not important, because they can provide statistical information about the specifics in the image, and result in more accurate statistical information being extracted from the image.

Handwriting analysis can play an important part of civil and criminal court cases, especially when identifying signatures in legal documents in civil cases, or handwriting in threatening letters, ransom letters, and other documents in a criminal case. Such cases require a handwriting expert to spend hours analyzing writing samples confirmed to have been written by the defendant/plaintiff/complainant as well as the current writing sample whose author is unidentified, but suspected to be the defendant/plaintiff/complainant. In order to avoid such time consuming and expensive procedures, our proposed digital solution would allow courts to correctly identify whether or not a specified person wrote a specific article or sentence while saving them time and money.

VI. References

[1] Cristianini, N., and Shawe-Taylor, J. (2000). An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, First Edition (Cambridge: Cambridge University Press).

[2] Canny, John, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*,Vol. PAMI-8, No. 6, 1986, pp. 679-698.

[3] E.J. Candes, L. Demanet, D.L. Donoho, L. Ying, "Fast Discrete Curvelet Transforms" Technical Report, Cal Tech, 2005.

[4] Friedman, J. H., Bentely, J., and Finkel, R. A. (1977) An Algorithm for Finding Best Matches in Logarithmic Expected Time, ACM Transactions on Mathematical Software 3, 209.

[5] Guesmi, Hanen, Hanene Trichili, Adel M. Alimi, and Basel Solaiman. "Curvelet Transform-Based Features Extraction For Fingerprint Identification." REGIM: Research Group on Intelligent Machines (n.d.): n. pag. Web.

[6] J. L. Starck, E. J. Cand`es, and D. L. Donoho. The curvelet transform for image denoising. IEEE Trans. Im. Proc., 11-6 (2002), 670–684

[7] Kecman, V., Learning and Soft Computing, MIT Press, Cambridge, MA. 2001.

[8] Lim, Jae S., *Two-Dimensional Signal and Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1990, pp. 478-488.

[9] Ma, Jianwei, and Gerlind Plonka. "The Curvelet Transform." IEEE Signal Processing Magazine Mar. 2010: 118-33. Print.

[10] Scholkopf, B., and Smola, A.J., Learning with Kernels, MIT Press, Cambridge, MA. 2002.

[11] Suykens, J.A.K., Van Gestel, T., De Brabanter, J., De Moor, B., and Vandewalle, J., Least Squares Support Vector Machines, World Scientific, Singapore, 2