On the Recognition of Handwritten Math Equations

Quan Nguyen, Maximillian Wang, Le Cheng Fan

December 12, 2014

1 Introduction

1.1 Background

Handwriting recognition is a classic machine learning problem that has led to extensive research and a number of highly accurate solutions. Some handwriting recognition algorithms are achieving near-human level performance.

We employ tools used in handwriting recognition to handwritten math equations. While typing is generally much faster than writing by hand, math equations have the opposite property: writing equations by hand is more efficient than typesetting them. Furthermore, whereas handwritten equations are human-readable, the "code" of typesetting languages are often highly nested and difficult to edit. These problems of typesetting present an opportunity for improving the workflow of writing math equations digitally. In addition, recognizing mathematical equations is important for digitalizing past scientific works.

1.2 Goals and Outline

Though typesetting equations is difficult, storing equations in typeset form allows for concise storage that is relatively flexible for future editing. Since LaTeXis the most popular typesetting tool, our team's long-term goal is to produce a system to recognize images of handwritten equations and output the corresponding characters in LaTeX. Under the given time constraints, building the full system is infeasible, so our team chose a more reasonable goal of constructing a system for recognizing individual math symbols, with the possibility of building some functionality necessary to recognize multi-symbol expression. In the next section, we discuss the data collection process to obtain a wide array of characters. Afterward, we discuss preprocessing and algorithmic methods used to implement the system. Finally, we discuss our results and compare each of the different approaches we took.

2 Methods

2.1 Process

We run all of our images for both training and testing through a sequence of steps in order to improve the effectiveness of the system as a whole.

We begin by reading in each image and preprocessing it as outlined below in order to reduce the variation between different examples. This allows the system to create a much clearer picture of the distinctions between different labels. The next step is to detect features in the images and place them in a feature vector. Finally, we train and test several different machine learning models on our generated features.

We use a "pipeline analysis" approach in our research. Since our system consists of a series of separate steps, the overall performance is affected by several factors. At each step of the process, we compare different methods used for preprocessing, feature detection, and learning. In doing so, we can achieve a more complete understanding of how each step affects the system. (See Figure 1)

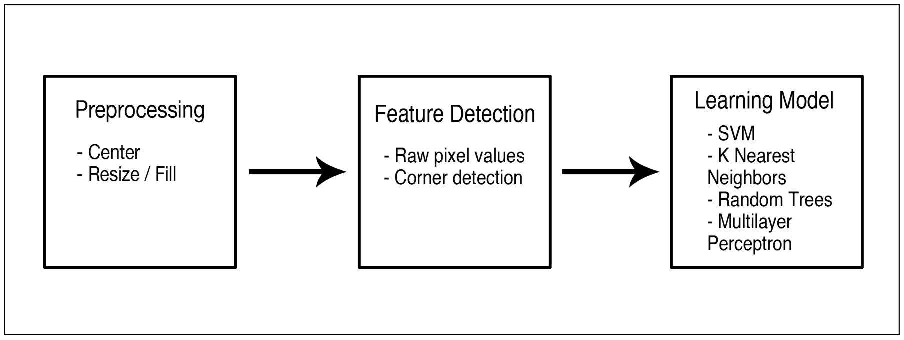


Figure 1: Pipeline steps

2.2 Software

Our main tool for performing all preprocessing, feature detection, and learning algorithms is the OpenCV library for C++. This software package includes utilities for every computer vision and Machine learning functionality that we require for this project.

2.3 Data Set

All the training and testing data we use are handwritten Greek letters produced in Photoshop with varied brush sizes and hardness values. The dataset consists of 128x128 grayscale images of the following letters: α, β, λ, μ, ϕ, π, and ζ. There are 100 images of each Greek letter, and hold-out cross validation is used to separate the data set into training/testing sets and to compare training and generalization error.

In order to reduce variations between images, we implement two different preprocessing techniques. In the first method, we calculate the centroid, $(\bar{x}, \bar{y})$, of the image via moments. These are given by $\bar{x} = \dfrac{M_{10}}{M_{00}}$ and $\bar{y} = \dfrac{M_{01}}{M_{00}}$, where

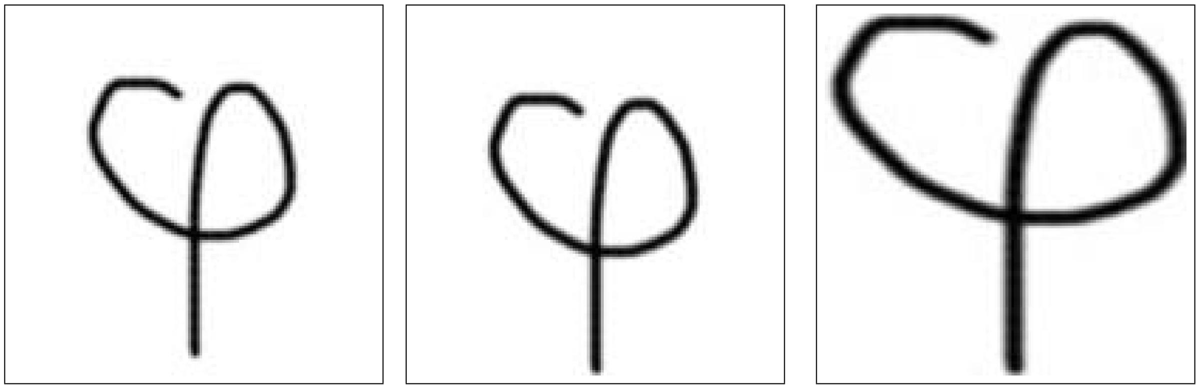
$$M_{pq} = \sum_{x=1}^{128} \sum_{y=1}^{128} x^p \cdot y^q \cdot ImageIntensity(x, y).$$

Once we find the centroid, we then translate the entire image inside the 128x128 square such that the centroid lies on the center of the image, namely at $(64, 64)$. This corrects for differences in symbol positioning in the hand-drawn images.

Our second technique is to eliminate as much background "white space" in the image as possible. This is done by drawing a bounding box around the actual symbol, and then expanding that section of the image to fill the entire square. The technique corrects for size differences in each image. It also adjusts for positioning since the end result is tightly snug in the 128x128 square. Since the centroid of the original image is generally not the same as the centroid of the blown-up image, we only perform one of these preprocessing techniques at a given time. We also run the system without preprocessing to see how the techniques affect the performance. (See Figure 2)

2.4 Feature Detection

We compare two different types of feature vectors for our learning models. The first is treating the grayscale image as a matrix of pixel values, and then reformatting this matrix into a single row vector. This vector thus encodes the each of the 128×128 pixel values of the image. The second type of feature detection method we use is the Harris operator for detecting corners in an image. We use an abstracted OpenCV implementation which assigns a "confidence level" for each of the detected corners, and then returns a vector of the coordinates of the k top scores. In our research, we compare the performance when using 5 corners, 7 corners, and raw pixel values.



(a) Original (b) Centered (c) Resized

Figure 2: Data set with preprocessing

3 Models

We use three different learning algorithms in our project: Support Vector Machines, K-Nearest-Neighbors, and Random Forest. Here we outline each of these algorithms.

3.1 Support Vector Machine

A support vector machine attempts to find a hyperplane that divides training examples of two different types, such that the distance between the plane and each of the training examples is maximized. When training data is linearly separable, this hyperplane will perfectly divide the points into groups by label. However, in most cases, data is not perfectly separable, so L_2 regularization is used. More precisely, find

$$max_\alpha \sum_{i=1}^{m} a_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)} x^{(j)} \rangle$$

such that $0 \leq \alpha_i \leq C, i = 1, ..., m$ and $\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$. And then calculating w as $\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$.

For our implementation, we tested a few different kernels for differences in performance. Since the SVM is only used for binary classifications, the OpenCV implementation creates one SVM for each pair of labels in order to generalize the algorithm to multiclass situations (like our own research).

3.2 K-Nearest-Neighbors

The K-Nearest-Neighbors is a non-parametric method which classifies a test data point by a ”majority vote” of the closest training example (by Euclidean distance of feature vectors). Given a test point, the algorithm finds the k nearest neighbors to that point, and predicts the new point’s label as the mode of the labels of the neighboring points. The algorithm breaks ties arbitrarily.

3.3 Random Forest

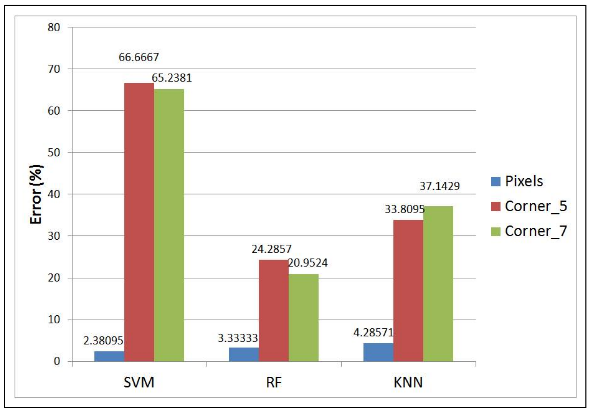
The random forest model bootstraps by selecting, with replacement, a random subset of the data for training each new decision tree (bagging). Furthermore, a random subset of the data is selected at each decision point. In total, 21 decision trees were constructed for our forest.

4 Results and Analysis

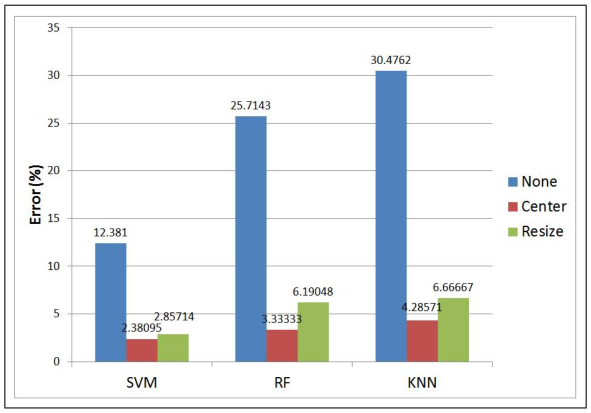
4.1 Results

We found that the best performance came from a combination of either centering or resizing, raw pixel value features, and a support vector machine model. Figure 3 below shows our results when using a linear kernel.

We were later able to achieve slightly better results using a degree-3 polynomial kernel, yielding approximately 1.5% generalization error rate when centering the images and using raw pixel values. Other kernel types, such as radial basis function and higher-order polynomials, performed significantly worse due to overfitting, as shown by a training error of 0% and generalization error of greater than 80%. We also found that the use of centering and resizing did not yield significantly different results, although they both were significantly more effective than no preprocessing. In addition, the corner detection method produced very high error rates, proving infeasible for our desired application.



(a) Comparison of feature detection (with centering)



(b) Comparison of preprocessing (with pixel detection

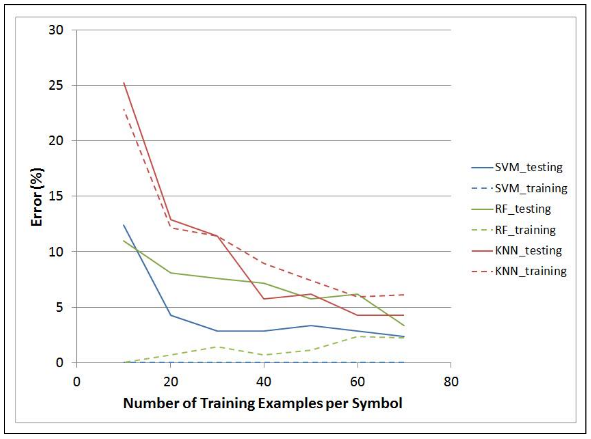
Figure 3: Comparison of results

4.2 Analysis

Our results were somewhat unexpected. We had hypothesized that the corner detection method would yield features that were more unique to each symbol, and would therefore be more effective at classifying them. Our results disagreed with this hypothesis. We believe that the high error rate associated with the corner detection method is due to the relatively low dimension of the feature space, meaning that less information is encoded in the feature vectors. In addition, the corner detection technique is sensitive to individual handwriting habits. While the dark pixels will generally occur in a similar pattern when preprocessed, the corners detected depend strongly on where the writer draws a rounded edge or a pointed one.

While initial efforts were funneled toward testing different types of learning algorithms, the most significant results came from preprocessing (Figure 3). Using the centered or resized images increased all algorithms to at least acceptable performance. It is interesting to observe that the relative difference between each the algorithms remained approximately constant: KNN had about twice the error rate for un-processed, resized, and centered images. Preprocessing decreases noise in the data due to variations in different positions and symbol sizes.

Lastly, the training and testing error curves for the different models (Figure 4) show that our choice of 70 training examples is sufficient, and that adding more training examples will not decrease generalization error significantly. The curve shows that training and testing error converge past 60 training examples for both SVM and RF. The small difference between training and testing error ($< 5\%$), and the low training error for SVM, also implies that both bias and variance are low. We cannot do much better by generating a larger training set, so we must look elsewhere for improvements.



(a) Testing and training error

Figure 4: Comparison of results

5 Future Work

We will extend our framework to work for full handwritten math equations. The primary addition will be a segmentation algorithm for dividing a full expression into individual symbols, which may be identified using our current work. In addition, we hope to test our system on photographs taken of real handwriting rather than Photoshop drawings. We predict that this use case will require some additional pre-processing techniques, such as contrast boosting, to achieve current performance levels.

Finally, we will further study neural networks as an option to outperform our other learning models. We tested neural networks with all preprocessing and feature selection permutations during our research. However, due to problems with overfitting, we did not achieve useful results. Some successful preliminary results have shown an error rate of 2% for 2 classes with a training set of 20 examples.

6 References

A. Graves, et. al. A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 5, 2009.

Ciresan, Dan, Ueli Meier, and Jurgen Schmidhuber. ”Multi-column Deep Learning Neural Networks for Image Classification.”. IEEE Conference on Computer Vision and Pattern Recognition. (2012): 3642-649. IEEE. Web.

Hsu, Chih-Wei, and Chih-Jen Lin. ”A Comparison of Methods for Multiclass Support Vector Machines.”. IEEE Transactions on Neural Networks. 13.2 (2002): 415-25. Web.

Knerr, S., L. Personnaz, and G. Dreyfus. ”Single-layer Learning Revisited: A Stepwise Procedure for Building and Training a Neural Network.”Springer. 68 (1990): 41-50. Web.

Liu, Cheng-Lin, Kazuki Nakashima, Hiroshi Sako, and Hiromichi Fujisawa. ”Handwritten Digit Recognition: Investigation of Normalization and Feature Extraction Techniques.”. Pattern Recognition. 37.2 (2004): 265-79. Web.