Optical Linear Equation Analysis Using Support Vector Machines

Drew Schmitt, Nicholas McCoy

June 5, 2012

This paper presents a method for implementing an optical linear equation analyzer. Recognition of this type shows potential for becoming a promising field within computer vision with applications in many areas of the sciences and engineering. This technique works by performing basic image processing techniques and matching these to a database of trained data samples to perform a classification. Construction of the classified equation is done using a recursive structure. Results are displayed using the public API's provided by Wolfram Alpha.

1 Introduction

There are numerous applications for the optical recognition and analysis of linear equations. Rapid and repeatable equation analysis could benefit the scientific community much as optical character recognition (OCR) has revolutionized the business world [1]. Potential applications include a tool for students to visualize the solution of blackboard equations in the classroom, which we believe would enrich the learning experience.

Our method first trains a set of SVM's, where each SVM represents a comparison between one class and another in the class set. The class set used in this paper is a small subset of all characters that could appear in a linear equation. However, the framework is constructed such that the subset of recognizable characters is easily extensible to large scale class sets. Section 3 details the SVM training and classification procedures used in this paper.

After the SVM's have been trained, our method performs image processing techniques to extract all pertinent characters in the given image. The specific processing techniques used include locally-adaptive binarization, region labeling, and filtering based on characteristics of regions. Section 2 discusses the image processing techniques used in this paper.

Once all characters are extracted from the image, the underlying equation is constructed using the centroid and bounding box locations of each extracted character. This information is passed into a recursive equation construction technique. Section 4 elaborates on the details of this procedure.

Finally, the constructed equation is sent to WolframAlpha for analysis and display. Section 5 outlines the process used to communicate with WolframAlpha.

Section 6 illustrates the results of the algorithm on test equations and comments on the running time and efficiency of our proposed procedure. In this section, we also offer recommendations for future research in this area.

2 Image Processing

The input image is assumed to be dark writing or text on a lighter background. Characters from the equation should be the only foreground objects in the image. This framework can be extended to compensate for clutter around the edges of the image by performing filtering based on region centroid locations.

The first step is to convert the incoming RGB image to a binary image, in which the foreground equation is represented by white (1) and the background is represented by black (0). Working in a binary domain allows for easy manipulation and analysis of the foreground characters of interest. To do this, the image is first converted to grayscale and then adaptive thresholding based on Otsu's method is used to convert all pixels to the binary domain. Adaptive thresholding is used to ensure that images with shadows, or uneven background intensity are binarized correctly.

Once the pixels are in the binary domain, connected regions are located and labeled. Each connected region represents a character in the equation. The regions are analyzed and properties such as the region's area, centroid location, and bounding box are extracted. The connected region is then resized to be a 30x30 patch of pixels.

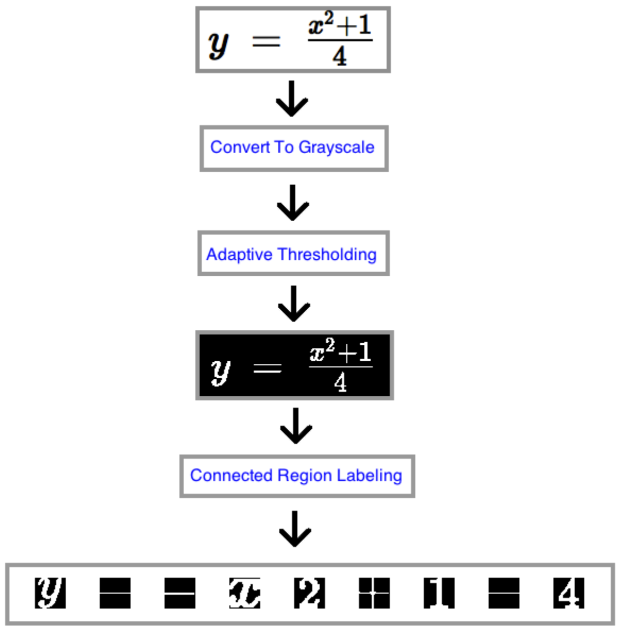


Figure 1: Process flow of extractPatches function.

The region's centroid and bounding box are used in classifying the character as part of a fraction entity or as an exponent, as discussed in Section 4. Figure 1 provides a visual representation of the general flow of the image processing steps used to extract the character patches from the input image containing the equation $y = \frac{x^2+1}{4}$.

A support vector machine (SVM) is used to classify the preprocessed patch into one of the classes in the designed class set. The next section details the implementation of the SVM.

3 Learning Algorithm

An SVM supervised learning algorithm is investigated and utilized in this paper. The learning algorithm is used to train and classify patches of characters [2].

An SVM with a linear kernel is described by the kernel function

$$K(x, y) = x^T y + c, \tag{1}$$

where x and y are inputs to the kernel and c is a constant offset term. A linear kernel is more desirable than its nonlinear counterparts due to its simplicity and computational efficiency in training and classification. While in some applications, a linear kernel is unable to capture subtle correlations between classes, the linear kernel performs exceedingly well in this particular application.

Given that an SVM is a binary classifier, and it is often desirable to classify an image between more than two distinct groups, multiple SVM's must be used in conjunction to produce a multiclass classification. A one-vs-one scheme can be used in which a different SVM is trained for each combination of individual classes. An incoming image must be classified using each of these different SVM's. The resulting classification of the image is the class that tallies the most "wins" [3].

The one-vs-one scheme involves making $\binom{N}{2}$ different classifications for N classes, which grows factorially with the number of classes. The class set used in this paper only contains the set of digits '0'-'9', Roman characters 'x', 'y', 'z', Greek characters 'π', 'λ', 'τ', and basic mathematical symbols including '+', '-', '(', and ')'. Thus, $N = 20$ in our experiments and so 190 SVM's are trained. This class set can easily be expanded to include all uppercase and lowercase Roman and Greek characters and more advanced mathematical symbols, such as '$\int$' and '$\leq$', at the expense of training considerably more SVM's.

For the training data $\{(x_i, y_i)\}_{i=1}^m$, $y_i \in 1, ..., N$, a multiclass SVM scheme aims to train $\binom{N}{2}$ separate SVM's that optimize the dual optimization problem

$$\max_a W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j K(x^{(i)}, x^{(j)}), \tag{2}$$

using John Platt's SMO algorithm [4]. In Equation 2, $K(x, z)$ corresponds to the linear kernel function discussed above.

Upon classification, a query image is then counted towards the votes of classes p and q, respectively, using the following scheme

$$\begin{aligned} \text{votes}_p{+}{=} & \left(1 - sgn\left\{\sum_{i=1}^m \alpha_i y^{(i)} K(x^{(i)}, z)\right\}\right) \\ \text{votes}_q{+}{=} & sgn\left\{\sum_{i=1}^m \alpha_i y^{(i)} K(x^{(i)}, z)\right\} \end{aligned} \tag{3}$$

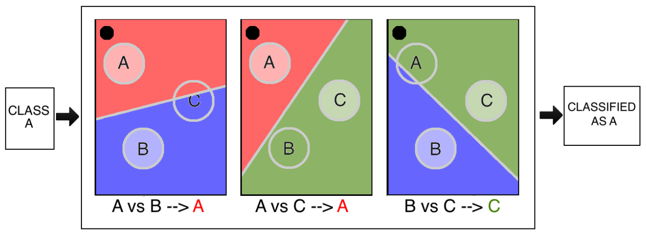


Figure 2: Portrayal of one-vs-one SVM's when query image is of type A. The black dot is the query image.

where $sgn(x)$ is an operator that returns the sign of its argument and z is the query image reshaped into a vector of length N^2 by 1. The query image, z, is then classified as the class that obtains the most votes after passing it through all $\binom{N}{2}$ SVM's.

When the query image is of class A, the A-vs-B and A-vs-C SVM's will be queried with the input patch. If the query patch truly contains an object of type A, then A will win in both of its contests. On the other hand, either B or C will erroneously win in the B-vs-C contest. However, since A has won two votes overall - compared to B or C having at most one vote, A will be declared the overall winner of this scheme. Figure 2 represents this concept visually where the black dot is the query image.

When the query patch is of a different class, D, which is not already existent in the class structure, there will not be a dominant winner in this round-robin contest. Hence, the query will not be falsely categorized into any class. A system designed in this way does not suffer from many false positives [5].

The specific multiclass SVM implementation used in this paper is MATLAB's built-in version as described by Kecman [6].

4 Equation Reconstruction

At this point, all the characters of the equation have been extracted, along with their corresponding area, centroid location, and bounding box. From these properties, equations containing addition, subtraction, multiplication, division, fractions, and exponents can be built.

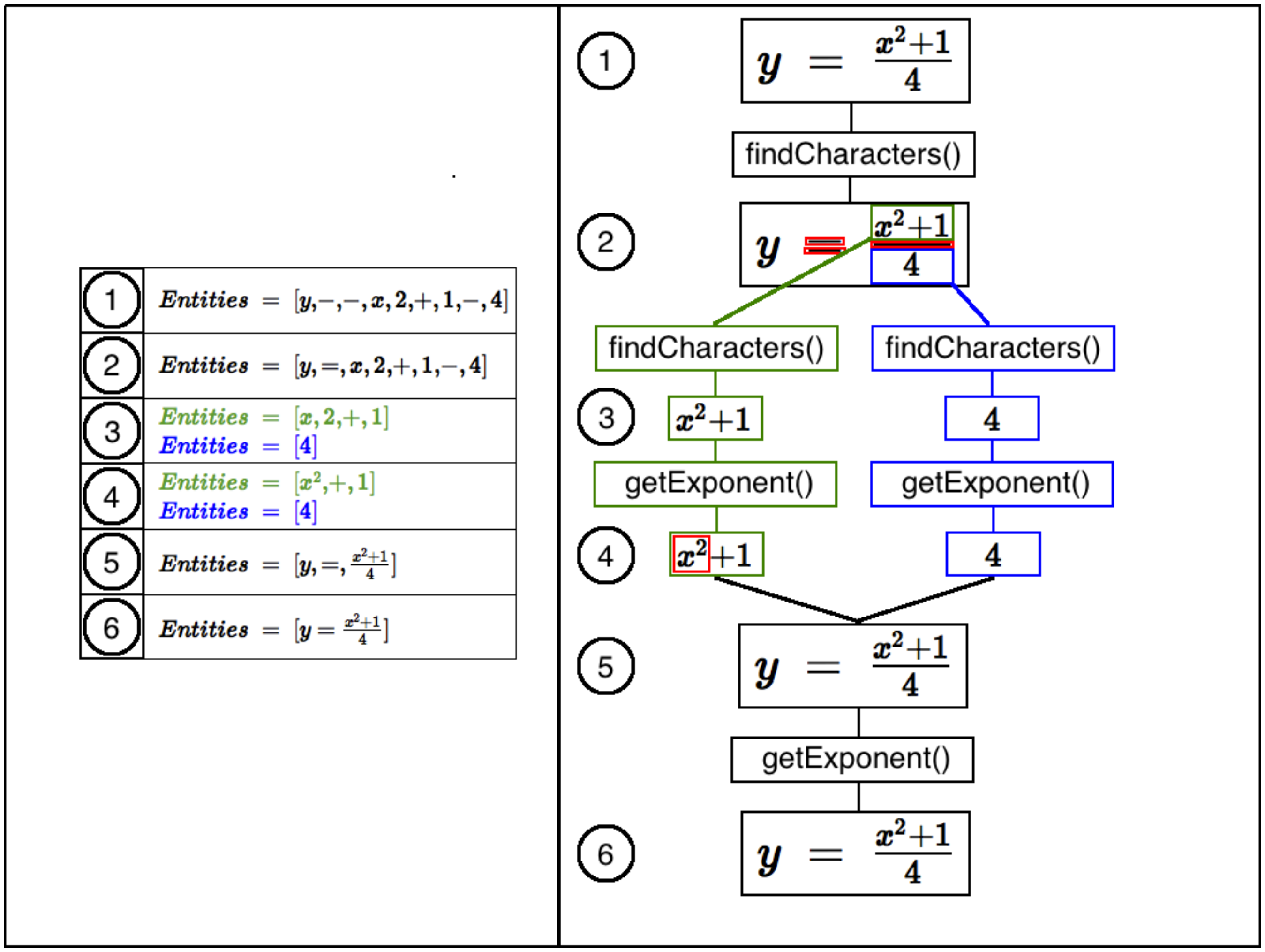
The general goal is to group characters together into entities. For example, the numerator, denominator, and dividing line of a fraction should be grouped as a single entity. If exponents are present, the base and exponent should be grouped as a single entity. Each entity has its own properties, including a string that represents that entity as a string equation, an overall entity centroid location, and bounding box. Once all characters have been grouped appropriately into entities, the equation strings for each entity can be concatenated left to right according to centroid location.

The first step is to recognize special characters in the equation. For the equations of interest in this paper, the only special character to find is the '$-$' character. This horizontal line indicates either a subtraction symbol, an equals sign, or the dividing line of a fraction. If there is horizontal line with no characters above or below it, the line is classified as a minus sign. Should there be another horizontal line above or below, the two lines are grouped together as a single equals sign entity. Otherwise, this symbol is the dividing line of a fraction.

The entire equation is built using a single call to the recursive function findCharacters(). This function searches the input set of characters for the longest horizontal line. If there are characters above and below, indicating a fraction, the numerator and denominator are passed recursively to findCharacters().

Once no more dividing lines have been found, the findCharacters calls getExponent() on each entity. This function looks to the entity immediately to the right, and if that entity is smaller in area, has a heightened centroid location, and its top bounding line isn't climaxed by its left neighbor's top bounding line, the two entities are combined into a single entity given by x^y where x is the base and y is the exponent.

After the recursion returns to the initial findCharacters() call, the final equation string is built containing all division entities, exponents, etc. Figure 3 details the flow of this recursion for the input image containing the equation $y = \frac{x^2+1}{4}$.



1	$Entities = [y, -, -, x, 2, +, 1, -, 4]$
2	$Entities = [y, =, x, 2, +, 1, -, 4]$
3	$Entities = [x, 2, +, 1]$ $Entities = [4]$
4	$Entities = [x^2, +, 1]$ $Entities = [4]$
5	$Entities = [y, =, \frac{x^2+1}{4}]$
6	$Entities = [y = \frac{x^2+1}{4}]$

Figure 3: Recursive flow of findCharacters function.

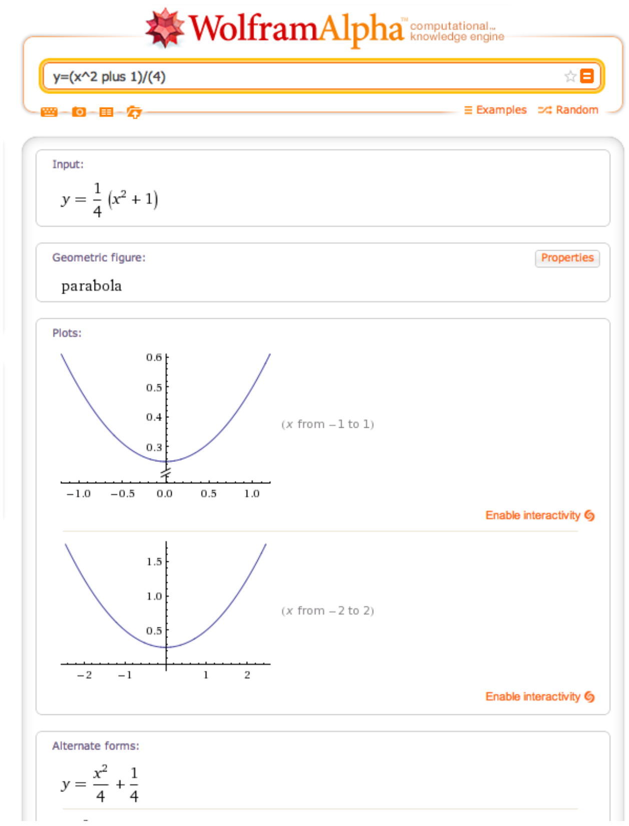


Figure 4: Sample screenshot of a WolframAlpha query result.

5 Graphical Representation

Once all the characters are correctly extracted, classified, and the underlying equation is constructed, we display the results in a variety of formats, both textual and graphical. Because the image processing algorithms are the focal point of this paper, the actual equation analysis techniques are outsourced to WolframAlpha [7]. WolframAlpha is an online service that quickly displays both textual and graphical results of equations and calculations.

Figure 4 shows a sample screenshot of the results obtained by querying $y = \frac{x^2+1}{4}$ directly on WolframAlpha's website.

To access this service directly from MATLAB, the native website access functionality is used. By issuing the following commands, MATLAB will launch the default browser and open the provided website where masterString is the constructed equation in string format from Section 3.

```
% Display the results of the equation using
%     WolframAlpha
baseString = ...
    'http://www.wolframalpha.com/input/?i=';
inputURL = [baseString masterString];
web(inputURL,'-browser')
```

It should be noted that certain symbols in mathematical formulas, such as '+' and 'π' either have special meanings in the URL or cannot be represented as is. Therefore, it is necessary to assign an alternate word representation for the symbol when writing it to the masterString, which will be used as the URL. Figure 4 demonstrates the appropriate handling of the '+' symbol in the given equation.

6 Results and Future Work

Our proposed algorithm demonstrates accurate classification results on input images containing characters in the trained class set. Figure 4 shows a successful classification output for the input image containing the equation $y = \frac{x^2+1}{4}$. Misclassification commonly occurs when characters are present in the query image that were not trained in the SVM's.

Furthermore, the precision of the classification decreases for images taken under noisy conditions, such as poor lighting or blurriness. Figure 5 shows the success rate versus variance of additive Gaussian noise added to the input image. Ten input images representing different phenomena in typical linear equations are used as the test group. Moving along the x-axis, the input images are corrupted with additive Gaussian noise with a larger variance.

Notice that the success rate is 100% when no Gaussian noise is added and then decreases approximately linearly in the dB realm as the noise variance increases. This result agrees with the intuition that as the input image becomes more corrupted, the accuracy of the region labeling and SVM classification stages of the algorithm will suffer.

Future work for this research should focus on making the algorithm more robust to clutter and interference. For example, images taken with a camera phone proved difficult to region label correctly due to artifacts in the

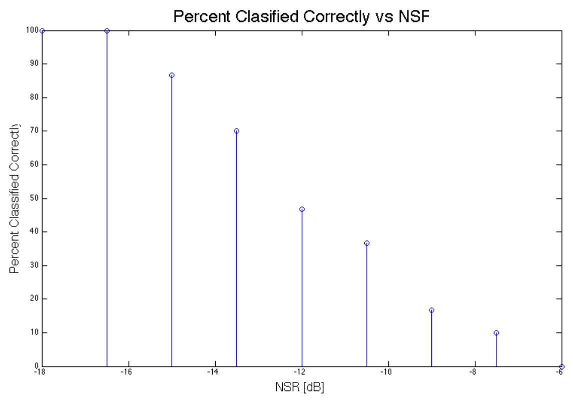


Figure 5: Success rate of classification versus variance of additive Gaussian noise.

input image. As mentioned in Section 2, clutter around the edges of the input image can be dismissed by filtering regions based on centroid locations and total area.

Furthermore, there is still open-ended work to be done on the classification of handwritten equations. A certain amount of classification success is achieved for using the algorithm described above on a handwritten equation. However, the clarity of the handwritten equation plays a large role in the success of the classification. The SVM stage of the algorithm can misclassify a character patch because it does not appear similar enough to the trained versions of the character, which are in a typed format.

The MATLAB code used to produce the results described in this paper is available upon request.

7 Appendix A: Division of Labor

Drew Schmitt: Image processing, SVM training, WolframAlpha visualization.
Nicholas McCoy: Image processing, Equation reconstruction, Plots and figures.

References

[1] D. Chen, J.M. Odobez, H. Bourlard. Text detection and recognition in images and video frames. *Pattern Recognition* 37:595608, 2004.

[2] T. Joachims. Text categorization with support vector machines. *European Conference on Machine Learning*, 1998.

[3] D. Schmitt, N. McCoy. Object classification and localization using SURF descriptors. *CS 229 Final Project*, 2011.

[4] J. Platt. Sequential minimal optimization: a fast algorithm for training support vector machines, 1998.

[5] C. Hsu, C. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13:415-425, 2002.

[6] V. Kecman. Learning and soft computing, *MIT Press*, Cambridge, MA. 2001.

[7] Wolfram Alpha LLC. 2012. WolframAlpha. http://www.wolframalpha.com (access June 1, 2012).