Equation to LaTeX

Abhinav Rastogi, Sevy Harris
{arastogi,sharris5}@stanford.edu

I. Introduction

Copying equations from a pdf file to a LaTeX document can be time consuming because there is no easy way to convert the equations in pdf form back to LaTeX code. This automatic conversion requires segmentation of characters from the image, identification of these characters and conversion to LaTeX code. This project deals with the first two steps of this problem. We have proposed a novel character segmentation algorithm for printed text and have experimented with some features and classifiers which classify the segmented characters.

II. Segmentation

We have made two assumptions for printed text-

(a) All characters in the image are axis-aligned
(b) Pixels corresponding to a single character are connected

The first assumption is good because printed text is well structured. If the image is not axis-aligned, it can be corrected using convex-hull detection if the rotation is less than 180° but we have assumed that this rotation is absent from our input images. The second assumption holds true for the most of the characters in printed text. Few characters like 'i', 'j', '=' etc. violate this assumption but they can be dealt with separately.

Our segmentation algorithm is based on the intuition that if we sum along the rows of an axis-aligned image, the rows containing characters will give a lower sum than the rows containing blank pixels. Let us illustrate our algorithm. Let $\mathcal{I}$ be an $m \times n$ image (m rows, n columns) containing multiple equations.

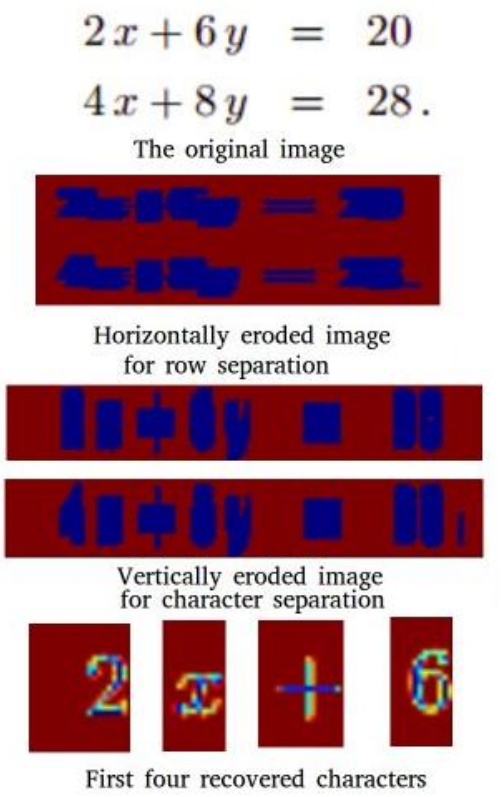

Figure 1: *The steps involved in segmentation of characters from an image of equation (order: top to bottom)*

Our segmentation algorithm proceeds in the following steps -

1. **Threshold the image** Apply a threshold of 0.9 to $\mathcal{I}$ to obtain a binary image B containing 1 in blank pixels and 0 in character pixels. Remove or add blank pixels on the border such that the minimum distance of any character pixel from the image border is 10 pixels.
2. **Perform binary erosion** We use a 1×10 horizontal mask to perform binary erosion on $\mathcal{I}$. This spreads the character pixels along the rows so that when we

sum up the rows of the image, we get a substantial difference between the sums corresponding to blank rows and rows containing characters. These images are solely generated for the purpose of obtaining row split points.

3. **Obtain blank rows** Sum the horizontally eroded image along rows to get a list of m numbers $\{r_1, r_2, \ldots r_m\}$ where $r_i = \sum_{j=1}^{n} \mathcal{I}[i,j]$. The set $\{i | r_i > 0.99 \max(r_i)\}$ corresponds to the set of blank rows.
4. **Split text lines** As expected, many blank rows occur next to each other. We take the mid-point of each contiguous stretch and split the image into many images, each containing one row of text. Let $m_1, m_2, \ldots m_p$ be the mid-points of contiguous stretches of blank columns. The images $I[m_i : m_{i+1}, :]$ are the images containing a row of text for $i = 1$ to $p - 1$.
5. **Split text vertically** We perform steps 2-4 for the image of each line using a vertical mask instead of horizontal and column sums instead of row sums to obtain column splits and divide each text row image further.
6. **Obtain segmented characters** Most of the characters are segmented after step 5 but if the text contains characters like 'i' or subscripts in $\sum$, we might not have segmented each character. So, if the image obtained after step 5 has more than 1 connected component, steps 2-5 are again repeated.
7. **Scale and center** Each obtained image is then centered by addition or removal of blank pixels on the border such that the minimum distance of any character pixel from the image border is 5 pixels.

III. Dataset

We wrote a python script which takes a txt file as input and and substitutes its contents in a tex file. A bash script then compiles the tex file to generate the pdf document, converts the pdf to an image, runs our segmentation algorithm on the obtained image to segment the different characters and labels each image according to the its character name obtained from the txt file. The above procedure can be used to obtain as much data as we need with very little effort. For the purpose of this project, we have focused on the identification of upper and lower case roman alphabets and digits. Our dataset consists of 8 images of each of these characters and contains 472 images in total.

IV. Features

Feature selection becomes a critical part of this problem since it involves a large number of classes and the size of each class is much smaller than the number of classes. The following feature mappings map the image of a character to a vector in a multidimensional euclidean space. The learning algorithms are then trained and tested on those vectors. We tested different models on these three sets of features-

(i) **Average Pixel Intensity features (A)** The whole image is divided into a 16x16 grid and the average pixel intensity in each grid is calculated. The feature vector is of size 256x1 and is obtained by concatenating the columns of the 16x16 image.

(ii) **Neighborhood pattern features (B)** The pattern formed by the adjacent pixels seems to be an important cue in the identification of a digit. However, average pixel intensity features do not take the pattern formed by neighboring pixels into account. If we consider a 3×3 neighborhood of a pixel, there are $2^9 = 512$ possible patterns that can be formed by the pixels in this neighborhood. We assign a number to each such pattern. Under this feature mapping, we first convert the image into a 16×16 image as described above. Then each non-border pixel, which are $14 \times 14 = 196$ in number are assigned the number corresponding to the pattern formed by its 3×3 neighborhood. The feature vector of size 196×1 is obtained by concatenating the numbers assigned to each of the non-border pixels in the

16×16 image.

(iii) **Discrete Fourier Transform features (C)** The fourier transform captures the trends in relative patterns and is not as susceptible to noise as the neighborhood pattern features. The feature vector is obtained by taking the DFT of the 16x16 average pixel intensity image. The real and imaginary parts of the DFT coefficients are then concatenated to form a 512×1 feature vector.

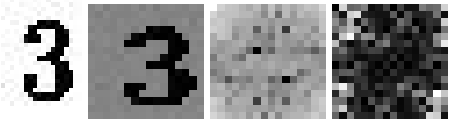

Figure 2: *(i) Original image extracted from text (ii)* 16×16 *image with average pixel intensities (iii) Absolute value of real part DFT coefficients and (iv) Absolute value of imaginary part DFT coefficients (order: left to right)*

V. Models

We employed the multiclass version of the following models to classify the characters. The training and test error results for these models are shown in the results section.

(i) **Naive Bayes** We chose Naive Bayes as a baseline model so that we could get a rough idea of the difficulty of the task. We trained a Naive Bayes model using the average pixel intensity features and neighborhood pattern features.

(ii) **Linear Discriminant Analysis** This model assumes that the conditional distribution of the features given the class is gaussian. This assumption is not explicitly true for our dataset and all classes having the same covariance matrix does not seem to be a good assumption. Nevertheless, we expected a fairly good result because much variability is not present in our class. For the multiclass formulation of LDA, we used a one-vs-one approach. This involves training a classifier for each pairwise combination of classes. So, if the number of classes is K then we train $K(K-1)/2$ classifiers. During prediction, we classify the input using all of these classifiers and the class which gets the majority vote is picked to be our predicted class.

(iii) **Support Vector Machine** SVM is a fairly good choice if we assume that the feature space is linearly separable. This is not intuitively clear as our feature space has a high dimension. Nevertheless, as previously mentioned, our classes do not have a large variability and hence we expect classes to be linearly separable pairwise. However, another issue is that our feature space has far more dimensions than the number of data points and our data points are close to each other. SVM (or any linear classifier) may pose the danger of overfitting. For this reason, we did not use polynomial or RBF kernels for SVM because it further increases the effective dimensionality for the feature space. The objective function of or algorithm is

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$

$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i$$

$$\text{where } \zeta_i \geq 0 \text{ for } i = 1, \ldots n$$

Our model uses a one-vs-one approach for the multiclass formulation of SVM as described previously.

(iv) **Random forest** This model can perform well given large feature sets because it combines the predictions of various decision trees to build a more robust classifier. While constructing new decision trees, this method uses a random subset of features. This helps us to get rid of spurious features and might improve the robustness of our estimate. Our random forest model makes use of ID3 algorithm for training decision trees and uses the *Gini* measure for calculating the goodness of a split criteria. The Gini impurity measure is defined as

$$\text{Gini}(X_m) = \sum_k p_{mk}(1 - p_{mk})$$

where p_{mk} is the fraction of times an element of class k occurs in a split X_m of set X. *Gini* approximates the entropy function and hence our decision tree approximates a sequence of split criteria that decrease the entropy of the final partition. After trainig numerous decision trees, our model combines them using the AdaBoost algorithm.

VI. Results

We experimented with various combinations of features and models described above. The training and test results are as follows. The abbreviations for feature names have been given in the section on features. The errors are reported as the percentage of data misclassified. The training data consists for 354 images and the test data consists of 118 images.

	Feature		
Model	A	B	C
NB	0.1638	0.3842	-
LDA	0.1299	0.3729	0.0
SVM	0.0734	0.3672	0.0
RF	0.1582	0.3616	0.0

Table 1: *Training Error of feature model combination*

	Feature		
Model	A	B	C
NB	0.0339	0.3729	-
LDA	0.0339	0.3559	0.0678
SVM	0.0169	0.3559	0.0508
RF	0.1017	0.3559	0.0847

Table 2: *Test Error of feature model combination*

VII. Discussion

As expected, the average pixel intensities fail to give a good result because a slight shift in the image might lead to a different feature vector altogether. We expected neighborhood pixel pattern features to work better than average pixel features but it was surprising to find that they perform worse. On further deliberation, we feel that these features fail due to a similar reason. A slight shift in the image might severely affect the feature vector. Moreover, the effect of noise is more pronounced in this feature because a noisy pixel will considerably change up to 9 components of the feature vector. These components correspond to the noisy pixel and its neighbors.

It is tough to visualize the exact correspondence between a DFT feature and the image. Hence, we had a neutral expectation from this feature mapping before we began working on it. As can be seen, these features give a considerably better result than other features. Our explanation for this observation is that DFT features capture the spatial periodicities of various pixels. DFT can be thought of as averaging after multiplication by sinusoids of different angular frequencies which are integral fractions of 2π. Due to the averaging operation, the DFT features are robust to noise. Further, a shift in the image corresponds to the change in phase of the DFT coefficients and hence the feature vectors are not affected much.

As far as models are concerned, we didn't expect a good result from our baseline model, which was a multiclass Naive Bayes classifier. This is because NB assumes that the pixel intensities are independent, which doesn't seem to be a good assumption. The LDA classifier works for our data because we have very little variability within each class and hence the assumption that all classes share the same covariance matrix doesn't have much effect on the accuracy of our algorithm if the classes are far apart in the feature space, which is true for most of the classes.

The random forest model is more robust and helps in decreasing the variance of the classifier because it uses a set of decision trees to make its decision. Since we pick up a random subset of features in the process of training the decision trees, we automatically filter out good features which affect our decision. Our intuition is validated by the fact that RF does better than SVM when we use an inferior set of features (features A and B) because of its

ability to discard irrelevant features. However, when we have a better set of features (feature C), its performance is close to that of SVM.

SVM and DFT features was the best model-feature combination we found. On test data, our model misclassifies 6 out of 118 images. The errors involve misclassification of '0', '1' and '2' as '*o*', '*t*' and '*a*' respectively. The confused characters are structurally similar and hence the confusion is plausible.

VIII. Future Work

Classification of printed characters is a tough task. Close to perfect accuracy is desirable in order to build a system for converting equations to LaTeX commands. We have only worked on a subset of characters in this project. If one were to include greek alphabets too, the error would increase because of presence of similar characters like '*a*' and 'α', '*B*' and 'β' etc.

While doing a literature survey, we found that neural networks give a good result for handwritten digits. It would be interesting to explore how they work on printed text when lots of different classes are present.

Another interesting but non-machine learning aspect of this problem which we have not considered in this project is the generation of LaTeX commands after segmentation and classification of text.

References

[1] T. Mitchell, Machine Learning: A Guide to Current Research: Springer, 1986. pp52-78

[2] A. Ng, "CS 229 Lecture Notes: Support Vector Machines," [online] cs229.stanford.edu/notes

[3] E. Brown, "Character Recognition by Feature Point Extraction," [online], www.ccs.neu.edu/home/feneric

[4] Mahmoud, S.A.; Mahmoud, A.S., "Arabic Character Recognition using Modified Fourier Spectrum (MFS)," Geometric Modeling and Imaging - New Trends, 2006 , vol., no., pp.155,159, 16-18 Aug. 1993

[5] Scikit Learn- A machine learning library for Python [online] scikitlearn.org