UC Berkeley, Computer Science

Gender Classification of Handwritten Text

Peter Cheng, Jeff Tsui, Alice Wang

May 16, 2013

1 Introduction

For our project we designed and tested an off-line classifier for gender prediction using handwritten text. The inspiration for this project came from a Kaggle machine learning competition [1]. Kaggle provides sample training data, in the form of high-resolution (300 dpi) jpg images. Each image corresponds to a writing sample, and there are 4 writing samples for each of 475 writers. The 4 samples correspond to:

1. Arabic text, different text for each writer
2. Arabic text, same text for each writer
3. English text, different text for each writer
4. English text, same text for each writer

Section 2 provides some details that lead to preprocessing and feature extraction. Section 3 details the preprocessing tasks we performed on the Kaggle jpg images provided. Section 4 describes in detail the methodology behind feature extraction of the processed images. Section 5 demonstrates the performance of our extracted features using various canonical classifiers.

2 Background

For this particular contest, Kaggle provides 7000+ extracted features for each of the 1128 writing samples across 282 writers. In an attempt to simplify the problem, we opted to

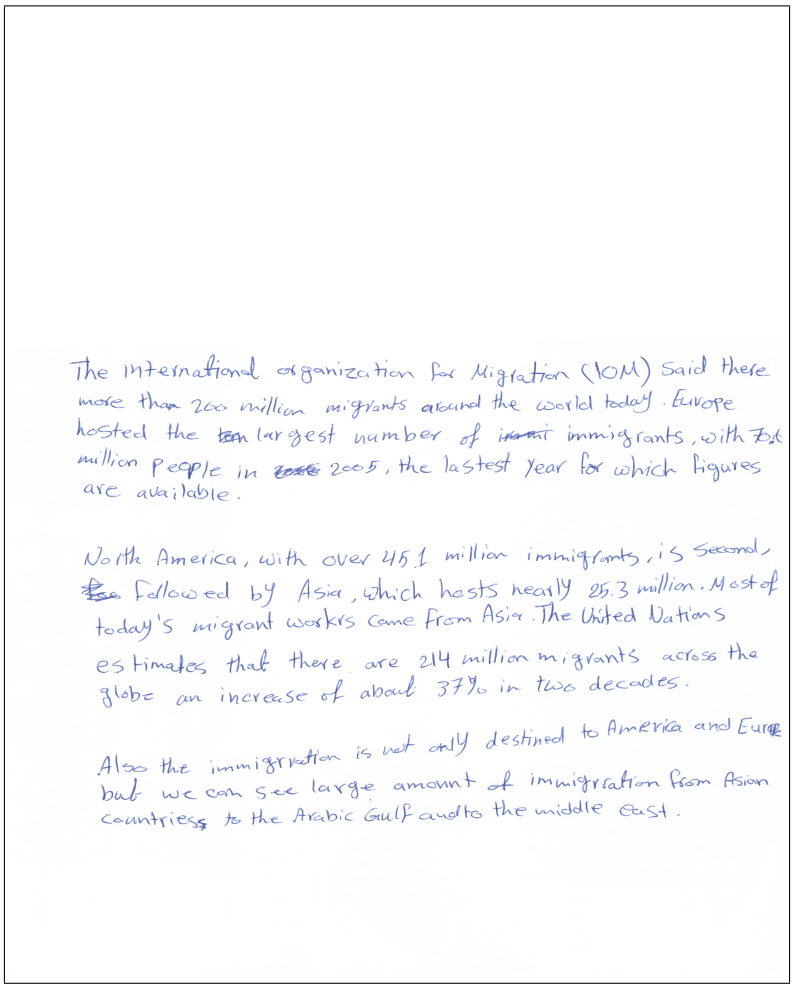

Figure 2.1: A sample input document image

consider only the fourth sample for each writer, which is the same English text written by each writer. Since the data supplied is for a competition, the test data did not have gender labels. We thus limited our dataset to the labeled training data, where label = 1 for male and label = 0 for female. Of these 282 writing samples, we reserved 25% to be our testing set.

An example of a document image from our dataset is shown in Figure 2.1. As evident in the image, the handwritten text is a color image, and the contour of each line is unconstrained. Upon relevant literature review, we discovered that properly handling such unconstrained data, and gathering useful features from them is a very interesting and active research area. We thus focused our project on performing our own text segmentation and feature extraction.

We initially attempted to perform Optical Character Recognition on each image, in an attempt to link handwritten text across images, though we obtained poor results, and in-

troduced more error and uncertainty than needed. Instead, we independently performed a number of preprocessing, segmentatio , and feature detection steps as will be described in the following sections. Throughout this paper, we will refer to document images as images containing a single document, while word images and line images correspond to images containing a single word or line, respectively.

3 Preprocessing and Segmentation

Typically, when performing handwriting classification and feature extraction, binary images with a number of constraints are assumed. Typical constraints seen in previous work include having images in black and white, text all equally scaled among different writers, and lines written at consistent angles [2]. Furthermore, many feature extraction approaches are tuned to work on images of individual characters, words, or lines, instead of an entire document altogether. As a result, prior to feature extraction, we perform a series of preprocessing steps to normalize each document image, followed by segmentation procedures to isolate individual lines and words.

3.1 Preprocessing

The first step in preprocessing is to translate each input color image into a binary black and white image. To accomplish this, an intensity threshold is calculated, such that pixels with intensity above that value are set to 1, while the rest are set to 0. This intensity threshold is calculated using Matlab's "graythresh" function, which performs Otsu's method for binary thresholding [3]. Following this binarization, we then trim off the margins of each image, as writing samples are centered around different locations on the page. This is accomplished by retaining the image within the minimum axis-aligned bounding box containing all text in the image.

3.2 Segmentation

While features that are calculated on handwriting samples could theoretically be extracted from entire document images all at once, many of the features we use are specific to words or lines, so it makes sense to first break down the input space into line images and word images, so as to potentially reduce error when extracting these features.

3.2.1 Line Segmentation

To perform line segmentation, the Hough-transform-based algorithm proposed by Louloudis, G., et al, [4] is loosely followed. First, we use Matlab's probabilistic Hough line segment detector (HoughlinesP) implementation to detect lines on the document images, preprocessed after the previous section. We restricted Hough peak detection to only occur in the angle domain such that detected lines deviate at most 5 degrees from horizontal. We binned angles at a resolution of 1 degree, and the rho parameter at a resolution of 1 pixel. Since the parameters for minimum line length and maximum gap within a line

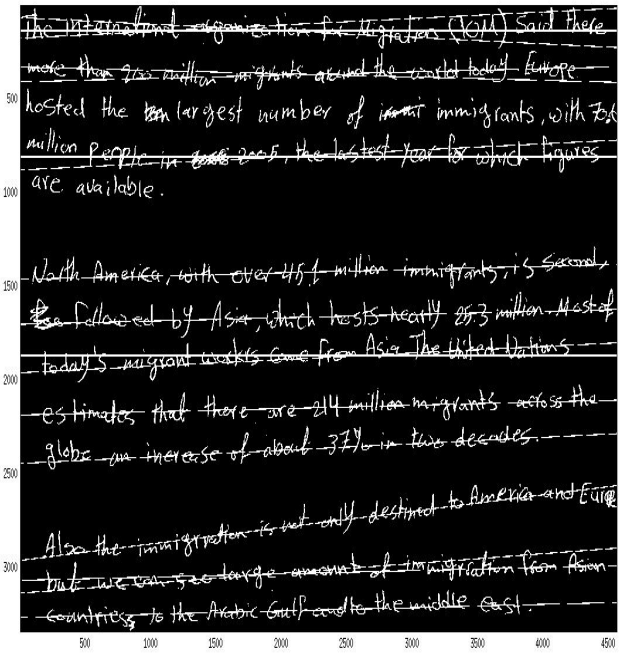

Figure 3.1: Line segments detected on a document image roughly correspond to lines of text

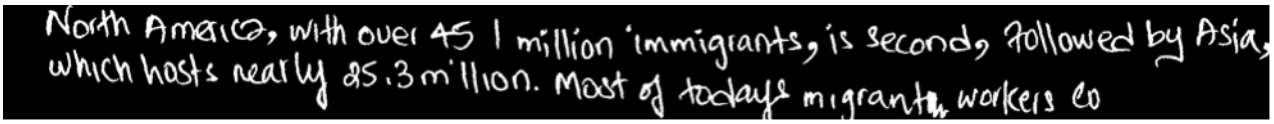

Figure 3.2: In this case, line segmentation failed to perform properly because two lines were very close together and a single detected line segment spanned both lines.

are more important, they are set to be 75% of the image's width and 10% of the image's width. An example of such detected lines is shown in Figure 3.1. Each detected line is then "drawn" onto the word image, such that pixel values corresponding to the line are set to 1. Next, connected components are detected on the resulting image. The effect of drawing each detected line upon the image is such that each character and word in a line of text should now all be in one contiguous connected component held together by the lines drawn over them. Connected components with fewer than 10000 pixels were filtered as they often correspond to punctuation or characters that were not properly grouped. Connected components with a width less than 8 times their height are also discarded, since they generally correspond to cases where multiple lines of text are detected together. Each connected component is now output at as a separate line of text.
In some cases, our line segmentation technique is unable to correctly separate adjacent lines. An example of this is seen in Figure 3.2, where the two lines present are both angled and very close together.

3.2.2 Word Segmentation

For word segmentation, we use is a method similar to the line segmentation method described in the previous section. Beginning with line images, shorter lines are detected and drawn, such that each word is its own connected component. We use the same angle and offset thresholds and resolutions as before, but set the minimum line length to be 2% of the line image's width, and the maximum gap in a line is 1.5% of the line image's width. After filtering out connected components with fewer than 500 pixels, we can then extract word images via the connected components. As seen in Figure 3.3, this works well for a high percentage of words but does not in certain cases. The writer used cursive writing with little spacing between consecutive words, which resulted in the three words "there", "are", and "21" being are segmented as one word.
It is useful to have words that are rotated such that their baseline is horizontal during feature extraction. Caesar, T., et al [2] cite a number of approaches to accomplish this, but we use a fairly simple approach. For each word, we compute its (rotated) bounding box, and rotate the image such that the bounding box is axis-aligned.

Figure 3.3: A case where word segmentation does not perform properly, as adjacent words are connected or close together.

4 Feature Extraction

In this section, we describe the features recovered from line images and word images, after preprocessing and segmentation, as described in the previous two sections. A listing of each feature is shown in Figure 4.1

4.1 Word Features

The first group of features are generated based on word images, referencing the work of Marti, U.-V., et al [5] where a feature vector is extracted from each word. These feature vectors are averaged by line to combine them with line-specific feature vectors. The word features are comprised of the width, slant, and height of the three main writing zones, described below.

4.1.1 Word Height

The three writing zones are separated by the upper and lower baselines. The upper and lower baselines are defined via analysis via the histogram of number of dark pixels in every line, where 15% of dark pixels exist above the upper baseline and 90% of dark pixels exist above the lower baseline. An example is shown in Figure 4.2. The baselines are used to calculate features 1 through 6, where the top line is first row of the image and bottom line is the bottom row. These features encode the height of words. The ratios of 1, 2, and 3 are taken to adjust for the variation in word size between writing samples.

4.1.2 Word Width

The row with the most black and white transitions defines the width of words. To represent the width of the writing, the number of white gaps between every group of black-white-black pixels is calculated and the median of these values is taken. Again, the ratio is taken with feature 2, the vertical height of the middle portion of the writing to account for the word size variation between writers. An example of this is shown in Figure 4.3

1. Upper baseline to top line distance
2. Lower baseline to upper baseline distance
3. Bottom line to lower baseline distance
4. 1 / 2
5. 1 / 3
6. 2 / 3
7. Median of the gap lengths
8. 2 / 7
9. Average slant angle
10. Std dev of slant angles
11. Line angle
12. Slant of lower contour
13. Mean squared error of lower contour
14. Freq of local max for lower contour
15. Freq of local min for lower contour
16. Avg left slope of local max for lower contour
17. Avg right slope of local max for lower contour
18. Avg left slope of local min for lower contour
19. Avg right slope of local min for lower contour
20. 12 for upper contour
21. 13 for upper contour
22. 14 for upper contour
23. 15 for upper contour
24. 16 for upper contour
25. 17 for upper contour
26. 18 for upper contour
27. 19 for upper contour
28. Avg width of connected components
29. Avg height of connected components
30. Std dev of width of CCs
31. Std dev of height of CCs
32. Avg dist between adjacent CCs
33. Std dev of dist between adjacent CCs
34. Avg area of enclosed regions
35. Avg length of major axis of ERs
36. Avg length of minor axis of ERs
37. Avg orientation of ERs
38. Avg eccentricity of ERs
39. Avg equiv diameter squared of ERs
40. Avg extent of ERs
41. Avg perimeter of ERs
42. Avg form factor of ERs
43. Avg roundness of ERs
44. Std dev of 34
45. Std dev of 35
46. Std dev of 36
47. Std dev of 37
48. Std dev of 38
49. Std dev of 39
50. Std dev of 40
51. Std dev of 41
52. Std dev of 42
53. std dev of 43
54. Fractal dimension slope 1
55. Fractal dimension slope 2
56. Fractal dimension slope 3

Figure 4.1: The features we extracted from each input document image

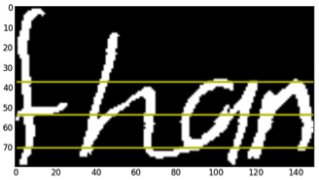

Figure 4.2: The upper and lower baselines are shown in this example. The middle baseline halfway between the upper and lower baselines is also shown.

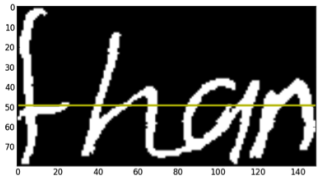

Figure 4.3: The line drawn in this image contains the most black/white transitions

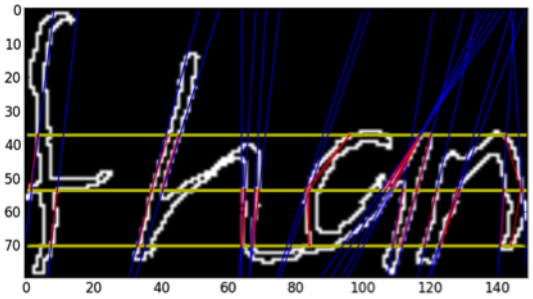

Figure 4.4: The slant of various parts of letters, as detected via adjacent pixels

4.1.3 Word Slant

We created a histogram of angles for each word in order to generate the slant of writing. First we converted the image to an outline such that only the perimeter pixels at the edge of each word remain. For each black pixel in the middle row of the middle zone (found between upper and lower baselines), we calculated the angle from the pixel to a connected pixel intersecting with the upper and lower baselines. To encode the slant for the word, the angle between pairs of pixels and their connected intersecting pixels are averaged and the standard deviations used. An example of this is shown in Figure 4.4

4.2 Line Features

The next group of features we generated were extracted from line images. This includes the angle of each line, the characteristics of its contour, various calculations performed on connected components and enclosed regions, as well as fractal dimension. We referenced work from Bouletreau, V., et al [6], Hertel, C. and Bunke, H., [7], and Vincent, N. [8] to create these features.

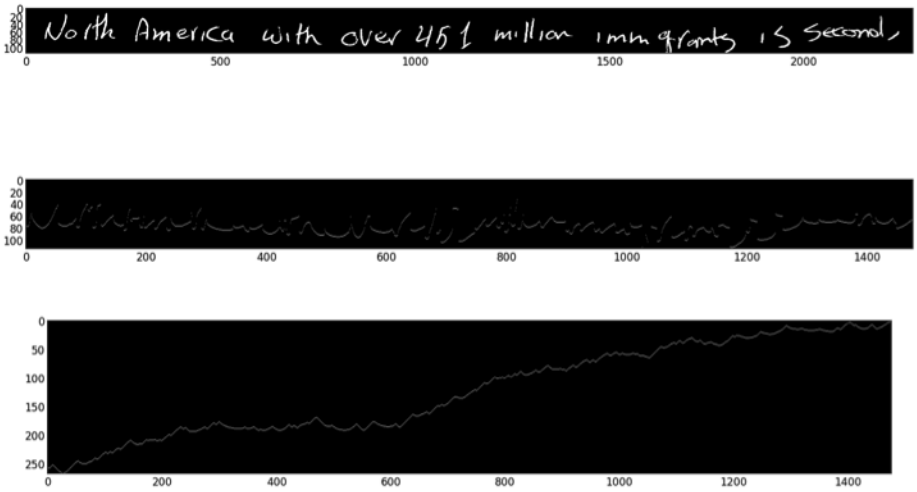

Figure 4.5: From top to bottom: the rotated line image, the lower contour of the line, and the lower characteristic contour.

4.2.1 Line Angle

The first feature comes from the angle of the line. This is simply the angles of the lines detected in Section 3 when performing line segmentation. If multiple line segments were detected for a single line, the average of their angles is used. For subsequent features, we use the rotated line so that writing is parallel with the x-axis.

4.2.2 Contour

The upper and lower contours of a line are the sequence of uppermost and lowermost pixels in each column of a line. Gaps in which a column of the image has no black pixels are removed. By eliminating discontinuities in the upper and lower contours, the characteristic contours are generated via shifts of y-coordinates of consecutive points such that they are at most 1 pixel apart on the y-axis. Several features are extracted from the characteristic lower and upper contours, such as 1) the slope of the least squared regression line 2) the mean squared error for the line 3) the frequency of local minima and maxima, found by dividing the number of local extrema by the length of the contour, 4) the local extrema, determined by comparing the the neighboring three points on either side. The average slopes of the left three points and the right three points for each local maxima are computed, as well. All contour features we generated are summarized below. Figure 4.5 demonstrates an example of an extracted contour.

4.2.3 Connected Components and Enclosed Regions

The next two sets of features correspond to characteristics of connected components and enclosed regions within a line of text. Connected components will generally be larger for writers whose handwriting is larger. Writers whose letters overwrite in handwriting will often have very wide connected components relative to their heights. The spacing between connected components also provides additional information about character spacing as well. Enclosed regions, on the other hand are the opposite of connected components. In a line of handwriting, they correspond to areas within loops and other enclosed areas. The roundness or eccentricity of each enclosed region can reveal information about how slanted a text sample is as well as the curvature of handwriting. Perimeter size contains the size of loops in text, and numerous other calculations provide similar characterizations of text. The features derived from connected components and enclosed regions correspond to features 28-53 from the table in Figure 4.1.

4.2.4 Fractal Dimensions

Fractal dimensions encode the degree of irregularity and fragmentation of the handwriting, from which our final three features are derived[8]. At a high level, they measure the area (measured in number of pixels) a handwritten text grows as a dilation operation is applied onto a line image. Given X as the contour of the handwriting sample, its fractal behavior is generated via the evolution of the areas of successive dilation sets of boxes on its contour. Xn is defined as the n-sized structuring square element, and A(X) denotes the area of set X. The x values are log(n), and corresponding y values are log[A(Xn)] - log(n).
Using the Minkowski-Bouligand dimension, the fractal behavior of the X set is expressed by the linear relationship between log[A(Xn)] and log(n) [6]. In plotting the x's and y's, the fractal features are the slopes of the three-part linear regression line that fits all possible points on the x-axis and and minimizes the mean squared error between the original points of the graph and the line segments[9]. The three regressions correspond to three zones of the image: zone 0 characterizes the line thickness, which is omitted since it varies based on resolution and image quality; zone 1 characterizes the writing shape; and zone 2 matches the dilations from which the writing is hidden. An example plot of 3 detected lines correspond to the 3 fractal dimension features is shown in Figure 4.6

5 Results

In this section, we apply 4 classification techniques to 4 different feature sets, and compare their results. Because our features are independently acquired on a per-line basis, and the end goal is to classify each document, we decided upon three ways to aggregate our features and determine their performance. The first is simply to classify each line separately, and consider our accuracy to be the percentage of lines accurately classified. The second method is to average the feature vectors for all lines within a document, and end with an "average" line feature vector for each document. The third method is to

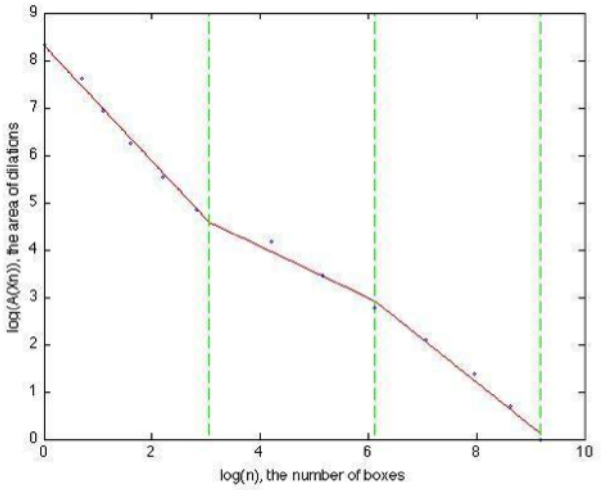

Figure 4.6: The slopes of the 3 straight lines that produce the best least squares fit constitute 3 features.

	SVM-Linear	SVM-RBF	Random Forest	kNN
Kaggle Features (per document)	75.2%	74.8%	73.8%	70.2%
Our Features-per line	65.5%	69.4%	66.1%	57.9%
Our Features-averaged	75.0%	82.4%	71.0%	73.5%
Our Features-voting	65.5%	70.6%	69.1%	64.7%

Figure 5.1: Accuracy of each classification method on each feature set

classify each line separately, but classify each document based on the majority vote of its line classifications. As a benchmark, we also acquired Kaggle's 7067-length feature vectors for each document. Because the per-line classification method works with a feature vector for each line, it has roughly 10x amount training and test samples, which impacts the parameters chosen for classification, as described later in this section. Each of these 4 feature sets was run through the following classification techniques: linear SVM, SVM with an RBF kernel, random forest, and k nearest neighbors. Accuracies of each can be seen in Figure 5.1.

For SVMs, we found that the C (cost) value strongly impacted accuracy. Upon cross-validating across multiple values, we ended with a C value of 10 for the per-document feature sets, while a C value of 1 performed better for the per-line feature set. The RBF SVM also has a σ parameter, which strongly affects accuracy as well; we performed cross-validation to obtain σ of 6 and 10 for the line and document feature sets respectively. The RBF SVM performed better than the linear one, and in fact better than all other classifiers. Our averaged feature vector also greatly outperformed the other feature sets here, including Kaggle's set.

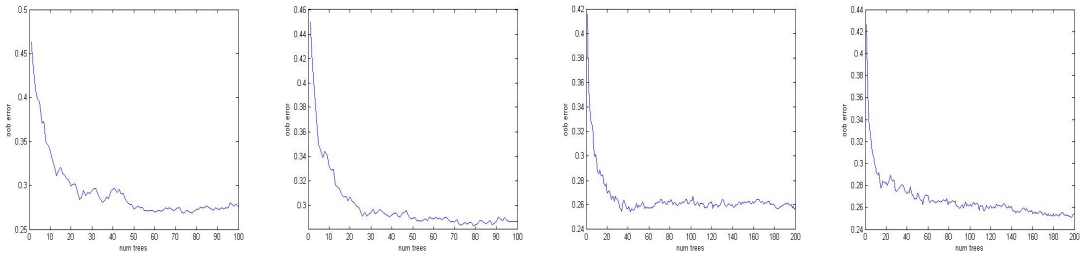

Figure 5.2: Out of bag error plots for the Kaggle, per-line, averaged, and voting datasets from left to right.

With the random forest method, we found no improvements in out-of-bag error past 100 trees, for each feature set. The minimum leaf size seemed to impact overfitting, and cross-validation resulted in a minimum leaf size of 50 for line features, and 10 for document features. Plots of the out-of-bag errors versus number of trees for each feature set is shown in Figure 5.2, and their asymptotic minimum values match the accuracies obtained.

For K nearest neighbors, we simply used euclidean distance and cross-validated on values of k. The best values were 46, 18, 4, and 7, for the Kaggle, per-line, averaged, and voting datasets respectively. These numbers were not very sensitive however, in that changing K generally had minimal effect on accuracy. Here, the averaged dataset actually outperforms Kaggle's feature set again.

Across all classifiers, the averaging method outperformed the voting method, followed by the per-line method. The per-line method performs the worst, as it is the most sensitive to outliers or errors in either segmentation or feature detection. The voting method performs better for all classifiers though only minimally, or identically, in the linear and rbf SVM. Plots of the majority classification votes received from each line across each document are shown in Figure 5.3. As evident, the linear SVM and rbf SVM plots have many documents where all lines receive the same classification. This is a positive result to see, as it implies that there is some amount of clustering of each line's feature vector within a document.

Universally however, the averaging method is superior for our dataset on each of our classification methods, even beating Kaggle's on two of them. This suggests that there is likely a high amount of error and variance in our features, as the voting method should perform similarly to the averaging method, if provided with consistent line vectors. Averaging likely improves accuracy by working with an averaged feature vector, which is less prone to error.

6 Conclusion

In our study of gender classification based on handwriting, we extracted 56 features based on word and line properties and applied several classification models. We combined these features in 3 different ways, and classified them with 4 different classifiers. To benchmark

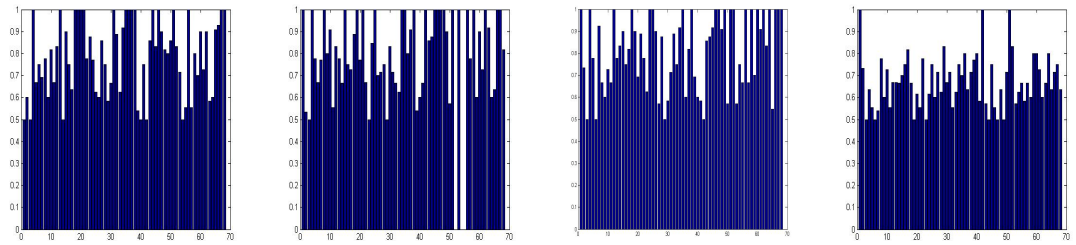

Figure 5.3: Each bar represents the number of majority votes received on each document. Bars with values near 1 represent a high amount of certainty and agreement among lines in a document, while values near 0.5 indicate disagreement among lines. The plots are for the linear SVM, RBF SVM, random forest, and k nearest neighbors results from left to right.

our results, we also used the features provided in the Kaggle competition. In general, our features resulted in slightly higher error rates than the Kaggle feature set, though our averaging approach resulted in comparable results. Overall, the RBF SVM performed best, both in terms of single best accuracy, as well as accuracy over each dataset.
While our results are somewhat lackluster for some of the classification schemes, our focus on feature extraction was rewarded in that some of our results are comparable to results obtained using Kaggle's provided featureset, obtained from a number of methods coming from recent state-of-the-art reseach. Furthermore, the dataset we worked with was not very large in size, and was intended for usage in a highly popular competition, so our results with fairly standard classifiers are not too surprising.
Nevertheless, there are several clear ways to improve our classification system. An improvement to our preprocessing steps that segment documents into words and lines would result in higher accuracy in what each feature is measuring, by reducing input error to our feature extraction methods. Furthermore, we noticed that dividing highly variable handwritten documents with skewed lines and different sizes is not a trivial task. Much of our work consisted of improving the accuracy of segmentation and reducing false positives that would skew extracted feature vectors. This resulted in many false negatives that greatly reduced the amount of useful input data. To further decrease the error rate, we could continue to generate additional features as well. The Kaggle dataset has over 7000 features, while we achieved similar results at 56 features with rudimentary classifiers. Finally, we extracted most of our features based on methods developed for writer identification, since gender classification based on handwriting is similar to writer identification. However, it is very likely that many of the features we implemented, designed for writer identification, may not be best suited for the task of gender classification. Performing more literature research to search for more features that measure different properties of handwriting and better separate our data would improve the accuracy of our classifier.

References

[1] "Icdar2013 - gender prediction from handwriting." `http://www.kaggle.com/c/icdar2013-gender-prediction-from-handwriting`.

[2] T. Caesar, J. Gloger, and E. Mandler, "Preprocessing and feature extraction for a handwriting recognition system," in *IEEE 1993*, 1993.

[3] N. Otsu, "A threshold selection method from gray-level histograms," *Systems, Man and Cybernetics, IEEE Transactions on* **9**(1), pp. 62–66, 1979.

[4] G. Louloudis, B. Gatos, I. Pratikakis, and K. Halatsis, "A block-based hough transform mapping for text line detection in handwritten documents."

[5] U.-V. Marti, R. Messerli, and H. Bunke, "Writer identification using text line based features," in *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pp. 101–105, 2001.

[6] V. Bouletreau, Vincent, R. N., Sabourin, and H. Emptoz, "Parameters for handwriting classification," in *IEEE 1997*, 1997.

[7] C. Hertel and H. Bunke, "A set of novel features for writer identification," in *AVBPA*, 2003.

[8] N. Vincent, V. Bouletreau, H. Emptoz, and R. Sabourin, "How to use fractal dimensions to qualify writings and writers," in *Fractals*, 2000.

[9] A. Hassaine, A. Somaya, and A. Bouridane, "A set of geometrical features for writer identification, neural information processing," in *Neural Information Processing*, 2012.