Script Identification of a Trilingual Document

Prem kishan Edhara Kranthi Gopu Raghu Teja Alapati
Veda Bhaskar Bhamidipati

May 11, 2017

Abstract

This report summarizes our experience and results on implementing a Script Identification System based on a paper called Script identification from trilingual documents using Profile based features[1]. The basic idea is to generate a set of features from each line of text and train a KNN classifier that can be used to classify text documents as either English, Hindi or Kannada. We created our own data set using books for each language. The pre-processing step is used to clean and extract lines of text from an image of a text document. The training phase generates the features for each line of text and is used to train the KNN classifier. The test data was also chosen from various books and labelled manually.

1 Introduction

Script Identification is a necessary task for any optical character recognition system. English is mostly used in USA so the problem of Script Identification is not significant, but in some countries like India there are languages with different scripts. Officially there are 22 languages in India with 13 different scripts. So any optical character recognition system in India will have a tough time recognizing characters without a script identifier. Our goal for this project was to differentiate at least 3 different scripts and we chose English(Latin Script), Hindi(Devanagari Script) and Kannada(Kadamba and Clukya scripts). These Scripts are some of the most commonly used scripts in India and that is why we chose them. We looked at two different approaches, one using global features and one using local features. Global features are features extracted at an image level or for the entire text document, for example texture. Local features are features extracted at the line level or word level and even character level. We use features at line level i.e we extract lines of text and generate features for each line. Although line level features take significant time to compute and use, once the KNN classifier is trained we only have to compute features for the test image which doesn't take significant time. The next Section will describe the process and algorithms used to achieve script identification.

2 Technical Approach

2.1 Data Collection

We used common books written in English, Hindi and Kannada which were taken from the Internet. Using Microsoft word we extracted each page as an image file. Every image of a particular language is kept in that language's directory so instead of labelling each image manually we processed documents from each directory and use the directory label as the image label. The test data was also generated in the same way but each test document image may have multiple languages in the same image. We needed 500 lines per language to train the KNN classifier and each document gave around 20 lines on average.

2.2 Pre-processing

The main goal of pre-processing is to generate lines of text and re-size them into fixed sized blocks. Noise removal and skew correction was not necessary as these documents were electronic and were not scanned or photographed. If training or test data is scanned then it is necessary to perform noise removal and skew correction for the script identifier to work accurately. The first step in pre-processing was to calculate the horizontal project of the document image to separate out each line. The local peaks of the horizontal project indicate the presence of a text line. The next step was to remove the margins surrounding each line of text, we compute the vertical projection for each line and look for the first positive value from both left to right and right to left and crop the image accordingly. The third step is to normalize the block as text images may have different font size for that reason it was determined that each text line should have a maximum height of 40 rows. So text line image of m rows and n columns is resized to 40 rows and n/m columns. Some examples of text lines after preprocessing are shown in figure 1, 4 and 7.

2.3 Feature Generation

Features are generated for each line of text so the input to the feature generator is an image of text line which has been preprocessed using the process explained in the previous sub section. Before we generate features we calculate top-profile and bottom-profile. The top-profile (bottom-profile) for a text line is the set of black pixels obtained when we scan the text line image from top (bottom) to bottom (top) for each column. The top and bottom profiles are shown in figures 1-9. The next values we calculate are called top-max-row and bottom-max-row, the top-max-row (bottom-max-row) is a row from the top-profile (bottom-profile) with maximum density i.e. maximum number of black pixels (black pixels are with value 0 and represent the text while white pixels with value 1 are the background).
Feature 1 :

$$profile - value = \frac{density - top - max - row}{density - bottom - max - row} \tag{1}$$

Feature 2 : Bottom-max-row number.

Figure 1: Hindi Text Line

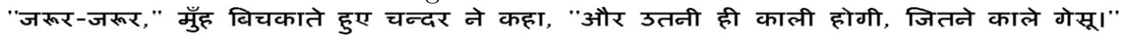

"जरूर-जरूर," मुँह बिचकाते हुए चन्दर ने कहा, "और उतनी ही काली होगी, जितने काले गेसू।"

Figure 2: Hindi top profile

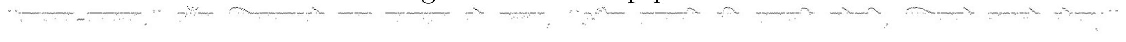


Figure 3: Hindi bottom profile

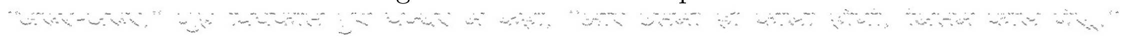


Figure 4: English Text Line

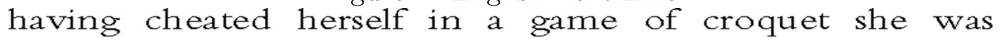

having cheated herself in a game of croquet she was

Figure 5: English top profile

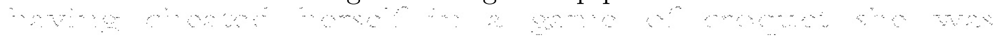


Figure 6: English bottom profile

Figure 7: Kannada Text Line

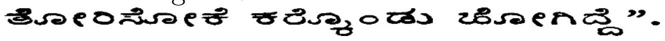

ತೋರಿಸೋಕೆ ಕರ್ಕೊಂಡು ಹೋಗಿದ್ದೆ”.

Figure 8: Kannada top profile

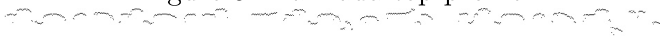
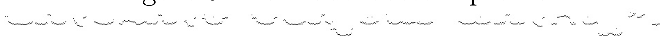


Figure 9: Kannada bottom profile



For the next feature we need to calculate top-vector and bottom-vector. The top (bottom) vector is basically a 1-d vector that contains the position value of the spatial occurrence of black pixels in the top (bottom) profile. Then we compute Coeff-top and Coeff-bottom using:

$$Coeff-top = \frac{Stdev(top-vector)}{mean(top-vector)} \tag{2}$$

$$Coeff-bottom = \frac{Stdev(bottom-vector)}{mean(bottom-vector)} \tag{3}$$

Feature 3 :

$$Coeff-profile = \frac{Coeff-top}{Coeff-bottom} \tag{4}$$

Feature 4 : top component density or density of top-max-row.

Table 1: Features

Label	Feature 1	Feature 2	Feature 3	Feature 4
Hindi	2.85007828	10.78	0.94469757	0.76818515
Hindi	2.85081517	11.25	0.95394078	0.76003251
Hindi	2.8673329	11.17	1.0032672	0.7612368
Hindi	2.82929308	11.23	0.97883488	0.76846823
Hindi	2.89211153	10.86	0.99999336	0.75447767
English	1.1485654	24.34	2.66160546	0.28976421
English	1.05158889	25.02	2.71952475	0.26918239
English	1.12939796	24.55	2.31757158	0.29000777
English	1.14686082	24.55	2.37852444	0.28632709
English	1.12316958	24.56	2.43969337	0.28776502
Kannada	0.77805514	28.21	1.35667298	0.22803911
Kannada	0.7051425	28.52	1.5236272	0.21803037
Kannada	0.78782433	28.81	1.40172316	0.22565663
Kannada	0.74869213	29.16	1.49666745	0.22819362
Kannada	0.66767657	29.2	1.56925854	0.21670745

2.4 Training

In this phase we train the KNN Classifier using the features discussed in the previous subsection and given in table 1. The collected data is loaded one directory at a time. For each directory we assign a label to it and compute features for all text images in that directory. For every 100 lines we average the feature values, so basically for every language after computing features for 500 lines of that language we end up with 5 feature sets for that language. So after features are computed for all 3 languages we end up with 15 feature sets which is used to train the KNN classifier.

2.5 Testing

In testing phase each of the test documents is preprocessed in the same as the training data was preprocessed. The features for the resulting preprocessed text lines are computed and that feature set is used as test input to the KNN classifier. The labels for each of the languages are Hindi (0), Kannada (1) and English (2).

Table 2: Results

	Total Lines	(3nn) Correctly predicted	(5nn) Correctly predicted	(7nn) Correctly predicted
Hindi	607	607 (100%)	607 (100%)	607 (100%)
Kannada	510	401 (78.62%) (4 as Hindi, 105 as English)	402 (78.82%) (4 as Hindi, 104 as English)	407 (79.80%) (4 as Hindi, 99 as English)
English	775	751 (96.90%) (2 as Hindi, 22 as Kannada)	751 (96.90%) (2 as Hindi, 22 as Kannada)	751 (96.90%) (2 as Hindi, 22 as Kannada)
	1892	92.97%	93.02%	93.28%

3 Results and Discussions

The results are given in table 2. We classified a total of 1892 lines and we got 92.97% accuracy for 3nn classifier, 93.02% for 5nn and 93.28% for 7nn. The classifier was least accurate for Kannada and mostly misclassified lines in Kannada as English. Some lines in English were mostly misclassified as Kannada. The features were good enough to classify between Hindi and the other two languages but it seems that Kannada and English have some overlap in the features.

4 Conclusion

We successfully developed and tested a Script Identification System based on a paper called Script identification from trilingual documents using Profile based features[1]. The results of 90% and above suggests an above average system using simple features and methods. Some modifications will be helpful to improve the performance of the classifier when classifying Kannada and English languages. Future work can include improving the accuracy and expanding the scope to include other languages.

References

[1] Script Identification from Trilingual documents using Profile based features. International Journal of Computer Science and Applications, @Technomathematics Research Foundation Vol. 7 No. 4, pp. 16 - 33 , 2010