Classifying Textual Components of Bilingual Documents with Decision-Tree Support Vector Machines

Xiao-Rong Lin, Chien-Yang Guo, and Fu Chang
Institute of Information Science, Academia Sinica
128 Academia Road
Taipei, Taiwan
{eclipse527, asdguo, fchang}@iis.sinica.edu.tw

Abstract—In this paper, we propose a method for classifying textual entities of bilingual documents written in Chinese and English. In contrast to earlier works that performed classification on the level of textlines or documents, we apply our method to the level of textual components, as we must first identify Chinese components before merging them into intact characters and sending the latter characters to a Chinese recognizer. To cope with a large training data set containing 365,672 samples, we employ a decision-tree support vector machine (DTSVM) method, which decomposes a given data space into small regions and trains local SVMs on those regions. By applying this method to train classifiers on various combinations of feature types, we were able to complete each training process within 3,500 seconds and achieve higher than 99.6% test accuracy in classifying a textual component into Chinese, alphanumeric, and punctuation. Moreover, the classification had no strong bias towards any of the three categories.

Keywords—bilingual document, component, decision-tree support vector machine, script and language identification

I. INTRODUCTION

The need to classify textual entities into a few categories arises from building optical character recognition systems for multilingual documents. For various characters, or character parts, and non-characters (for example, punctuation) in a multilingual document, one may classify these entities into several categories before sending them to individual recognizers.

Below, we present a brief survey of existing works on this subject, usually under the title of script and language identification. We divide the works into four approaches based on the levels of entity from which they extract features.

Character-based approaches. Some works extract features related to the shape or geometry of characters, such as white hole, centroid, sphericity, and aspect ratio features (Hochberg et al. [1]). Other works extract special features, such as “water reservoir-based” features, that are characteristic of Thai and Roman scripts (Chanda et al. [2]). Hochberg et al. [3] formed cluster-based templates as the basis for character matching.

Word-based approaches. Word shape tokens (WST) formed out of character shape codes (CSC) were proposed by Spitz [4], and have been applied extensively in classifying European languages (Spitz [5]).

Textline-based approaches. Spitz [4] employed upward concavity to classify Han-based and Latin-based textlines, and also used optical density features to classify Chinese, Japanese and Korean textlines. Other types of features in textlines include the peaks of characters (Lee et al. [6]) and the tops and bottoms of textlines (Padma and Vijaya [7]).

Image-based approaches. These approaches are based on texture features extracted from text regions (Tan [8], Busch et al. [9]).

In this paper, we propose a method for classifying textual components (in short, components) in bilingual documents comprised of Chinese and English. We classify these components into three categories: Chinese, alphanumeric, and punctuation. We consider Chinese punctuation and English punctuation as the same type at this stage. They can be differentiated easily via a post-processing step.

As mentioned earlier, there are many approaches on the script and language identification. However, most of them are applied at the level of textline or above, on the inherent assumption that a textline or a higher-level entity contains only one category of objects. This assumption does not hold for our application in which English letters, Chinese characters, and punctuation marks may appear in the same textlines, and in unpredictable locations (Figure 1).

One further complexity of our application arises from the fact that Chinese characters may consist of more than one component. So we need to identify the category of each component, rather than a complete character. At this point, we should clarify what we mean by “components.” The components dealt by us derive from the following pre-processing steps. First, we form connected components out of black pixels (Chang et al. [10]). Next, we enclose those entities in rectangles, or boxes. Then, when we deal with a horizontal (vertical) textline, we merge two boxes if their vertical (horizontal) projections overlap. For this reason, the
English letters ‘i’ and ‘j’ have a single box. On the other hand, ‘' has a single box when it appears in a vertical textline, and two boxes when it appears in a horizontal textline. In a slight abuse of the language, we refer to the derived boxes as components.

To solve the stated problem, we do not aim to invent new features, since many useful features have been proposed in the past. Rather, we apply a machine learning method, called decision-tree support vector machine (DTSVM) (Chang et al. [11]). SVM (Vapnik [12]) has proved to be a very powerful tool for pattern classification. However, the complexity of (non-linear) SVM is \( n^p \), where \( n \) is the number of training samples and \( p \geq 2 \), thus preventing a straightforward application to our problem in which there are 365,672 training samples.

DTSVM is a new method that speeds up the training of SVMs while maintaining comparable test accuracy. The method first trains a decision tree to decompose a given data space into small regions, and then trains local SVMs on the decomposed regions. DTSVM is an effective and efficient method for two reasons. First, training SVMs on decomposed regions of size \( \sigma \) reduces the complexity from \( n^p \) to \( (n/\sigma) \times (\sigma^p) = nd^p \). Second, the decision tree may decompose the data space so that certain decomposed regions become homogeneous (i.e., they contain samples of the same label), thereby reducing the cost of SVM training applied to the remaining samples. Each factor plays an important role in our application. In the experiments, we can build highly effective DTSVM classifiers on a tree whose leaf size falls below 1,500 training samples. Moreover, over 80% of the training samples flow to homogeneous leaves. The advantage of having a fast learning machine is that we can experiment with various combinations of feature types so as to find the best classifier to satisfy our requirements.

Applying the top-10 DTSVM classifiers to an independent data set comprised of 91,418 test samples, we obtained above 99.6% test accuracy (the best of them achieved 99.8%), without a strong bias toward any category. Moreover, the DTSVM classifiers classified textual components at an average rate of approximately 18,000 per second, while the average rate of extracting features from components was above 99.6% test accuracy (the best of them achieved 99.8%), without a strong bias toward any category. Moreo-
v(σ₀, θ) be the accuracy rate of the resultant DTSVM classifier, measured on the validation data set. In our experiments, we set σ₀ = 1,500.

In the subsequent stages, we construct DTSVM classifiers with larger ceiling sizes; however, we only train their local SVMs with the top-ranked θ, obtained by ranking θ in descending order of v(σ₀, θ). Let Θ[k] be a set containing k top-ranked θ. In our experiments, we set k at 5.

More specifically, we implement the following sub-process, denoted as SubProcess(θ), for each θ in Θ[k].

1. Set t = 0 and get the binary tree with the ceiling size σ₀.
2. Increase t by 1 and set σₜ = 4σₜ₋₁. Modify the tree with ceiling size σₜ to obtain a tree with ceiling size σ₀. Then, train local SVMs on the leaves with SVM-parameters θ. Let v(σ₀, θ) be the validation accuracy of the resultant DTSVM classifier.
3. If v(σ₀, θ) - v(σₜ₋₁, θ) ≥ 0.5% and σₜ is less than the size of the training component, proceed to step 2.
4. If v(σ₀, θ) - v(σₜ₋₁, θ) < 0.5%, then σ(θ) = σₜ; otherwise, σ(θ) = σ₀.

When we have conducted SubProcess(θ) for each θ in Θ[k], we define Θ[opt] to be the θ such that v(σ(θ), θ₀) ≥ v(σ(θ), θ) for all θ in Θ[k]. We also define σ[opt] to be σ(Θ[opt]). We then output the DTSVM classifier with the SVM-parameter Θ[opt] and the ceiling size σ[opt].

III. EXPERIMENTAL RESULTS

This section is divided into a few subsections. They address (A) the data set used in the experiments, (B) the types of features based on which DTSVM classifiers are built, (C) the results of training DTSVM classifiers on multiple feature types, (D) the results of training them on a single feature type, and (E) a sensitivity analysis.

A. The Data Set

For our experiments, we collected 1,517 images from bilingual newspapers and magazines. The images were comprised of 29,907 textlines and 548,508 components. All the components were labeled with their types. Finally, we normalize these components to the size of 64×64. Table I shows the information about them.

To conduct the experiments, we randomly divided our data set into three subsets: training, validation, and test subsets, in a ratio of 4:1:1. We built DTSVM classifiers on the training subset, consisting of 365,672 samples. Following the standard procedure, we normalized each feature vector to a vector of values between 0 and 1. The local SVMs in the DTSVM classifiers were RBF-based SVMs, whose parameter values are specified as in [11]. To save both training and testing times, we adopted a one-against-one training mode (Knerr et al. [15]). We then used the validation subset, consisting of 91,418 samples, to find the optimal parameter values. Finally, we applied the DTSVM classifier trained with the optimal parameter values to the test subset, consisting of 91,418 samples, to obtain the test accuracy rate.

<table>
<thead>
<tr>
<th>Textual Entity</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>1,517</td>
</tr>
<tr>
<td>Textline</td>
<td>29,907</td>
</tr>
<tr>
<td>Component (Total)</td>
<td>548,508</td>
</tr>
<tr>
<td>Chinese Component</td>
<td>173,038</td>
</tr>
<tr>
<td>Alphanumeric Component</td>
<td>343,313</td>
</tr>
<tr>
<td>Punctuation Component</td>
<td>32,157</td>
</tr>
</tbody>
</table>

B. The Types of Features

Six types of features are used to describe the properties of a component and the relation between the component and the textline that contains it. We briefly describe the features below. At the end of each description, we specify the ID of the feature type and the number of features (Dim) in that type.

Density. A 64×64 bitmap image is divided into 8×8 regions, each comprising 64 pixels. For each region, the counts of black pixels are used as a density feature. ID = 1, Dim = 64.

Cross Count. A cross count is the average number of black intervals that lie within eight consecutive scan lines that run through a bitmap in either a horizontal or vertical direction. ID = II, Dim = 16.

Aspect Ratio. For a component C that appears in a horizontal textline H, we obtain the following features: 1) bit ‘1’ for the slot indicating that H is a horizontal textline; 2) ‘0’ for the slot indicating that H is a vertical textline; 3) the ratio between C’s height and H’s height; 4) the ratio between C’s width and C’s height; 5) the ratio between C’s top gap and H’s height; and 6) the difference between C’s bottom gap and H’s height. We follow the same procedure for a component that appears in a vertical textline. ID = III, Dim = 6.

White Hole and Sphericity. The number of white components and the number of black pixels (cf. [1]). ID = IV, Dim = 2.

Upward Concavity. An upward concavity appears in ‘Y’ or ‘J’ when a fork is formed (cf. [4]). ID = V, Dim = 64.

Centroid. A centroid is either the center of a horizontal mass (black pixels) or the center of a vertical mass (cf. [1]). ID = VI, Dim = 2.

C. DTSVM classifiers Built on multiple Feature Types

We have 6 types of features, so there are 64 combinations of them. Since the DTSVM training and testing process is fast, we endeavored to study all DTSVM classifiers, each of which was built on one of the combinations. Because of space limitations, we only consider the 10 combinations associated with the top-10 accuracy rates. Table II shows the statistics of the 10 combinations, including the features in each combination (e.g., the first combination includes types I, II, III, and V), the dimension (Dim) of the resultant feature vectors, the average speed of extracting one feature vector, and the training and testing results of the DTSVMs for the 10 combinations. The H-rate is the percentage of training samples that flow to homogeneous leaves. The training time of DTSVM includes the time required to build DTSVM classifiers and the time taken to find optimal parameter values. The testing speed is the average time of testing one sample; and the online speed is the average speed of extracting features from one sample and then testing it.
In Table III, we further show the test accuracy rates obtained by the top-10 classifiers for the three categories. From the results, we conclude that there is no strong favor for a particular category at the expense of other categories.

**D. DTSVM Classifiers Built on a Single Feature Type**

The top-10 DTSVM classifiers, as shown in Tables II and III, were built on multiple feature types. It would be interesting to know how a DTSVM classifier based on a single feature type would perform. Table IV shows the results. The best classifier was built on feature type I, achieving a test accuracy of 99.51%, which is lower than the test accuracy of all top-10 classifiers built on multiple feature types. It is clear then that multiple feature types give rise to better-performing classifiers.

**E. Sensitive Analysis**

To build global SVMs is not possible for our training data subset, due to its gigantic size. Instead, we trained a CART on the training part and tested it on the test part, so that we could have something to compare with DTSVM. To train CART, we stop splitting a node when \( IG(f, v) = 0 \) for all \( f \) and \( v \) at \( E \). In the testing process, we assign each test sample \( x \) the label that is shared by the majority members of the leaf to which \( x \) flows. This implies that when this leaf has only one member, we assign its label to \( x \). We compare CART with DTSVM for two purposes. First, we would like to see if DTSVM performs any better than CART. For if it does not, there is no point of adopting DTSVM as a solution. Second, we would like to see how DTSVM and CART perform when a certain percentage of noise are added to training samples, in the sense that the labels of these samples are altered. Table V shows DTSVM’s and CART’s test accuracy rates and how much (A) DTSVM’s test accuracy rates exceed CART’s, when \( p\% \) of noise is added to the training data. Note that all the numbers were derived from the DTSVM and CART classifiers built on the first combination of feature types, including density, cross count, aspect ratio, and upward concavity.

From this table, we observe the following facts. (1) DTSVM outperforms CART in test accuracy rate. (2) Both classifiers’ test accuracy rates deteriorate as the noise level increases. (3) CART’s test accuracy rates deteriorate much faster than DTSVM’s.

### IV. Conclusion

In this paper, we have considered the problem of classifying each textual component into Chinese, alphanumeric, and punctuation. Because of the large size of our training data, we employed the DTSVM method to train classifiers. One advantage of using a fast method for training and testing was that we could experiment with various combinations of feature types. As a result, we were able to find 10 classifiers that achieved higher than 99.6% test accuracy and manifested no strong bias towards any particular category. Comparing DTSVM with CART, we also found that DTSVM is able to produce higher test accuracy rates and has better resistance to noises. Thus, we recommend DTSVM not only for its efficiency, but also for its robustness.
TABLE IV.  The training and testing results derived from DTSVM on the 6 single feature types. The training time is expressed in seconds. The testing and online speeds are expressed in components per second.

<table>
<thead>
<tr>
<th>ID</th>
<th>Ceiling Size</th>
<th>H-Rate</th>
<th>Training Time</th>
<th>Testing Speed</th>
<th>Online Speed</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1,500</td>
<td>67.52%</td>
<td>4,115</td>
<td>14,700</td>
<td>2,217</td>
<td>99.51%</td>
</tr>
<tr>
<td>II</td>
<td>1,500</td>
<td>55.56%</td>
<td>1,976</td>
<td>15,770</td>
<td>2,413</td>
<td>98.33%</td>
</tr>
<tr>
<td>III</td>
<td>1,500</td>
<td>56.18%</td>
<td>4,265</td>
<td>5,862</td>
<td>1,688</td>
<td>96.26%</td>
</tr>
<tr>
<td>IV</td>
<td>1,500</td>
<td>5.17%</td>
<td>11,982</td>
<td>5,862</td>
<td>1,688</td>
<td>87.65%</td>
</tr>
<tr>
<td>V</td>
<td>1,500</td>
<td>0.44%</td>
<td>65,440</td>
<td>10,194</td>
<td>2,183</td>
<td>73.25%</td>
</tr>
<tr>
<td>VI</td>
<td>1,500</td>
<td>0.0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE V.  The test accuracy rates of DTSVM and CART on the first combination of feature types after certain percentages of noise are added to the training data.

<table>
<thead>
<tr>
<th>Percentage of Noise</th>
<th>DTSVM</th>
<th>CART</th>
<th>∆</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0%</td>
<td>99.80%</td>
<td>99.45%</td>
<td>0.35%</td>
</tr>
<tr>
<td>0.2%</td>
<td>99.62%</td>
<td>98.97%</td>
<td>0.65%</td>
</tr>
<tr>
<td>0.4%</td>
<td>98.92%</td>
<td>98.65%</td>
<td>0.27%</td>
</tr>
<tr>
<td>0.6%</td>
<td>98.21%</td>
<td>97.92%</td>
<td>0.29%</td>
</tr>
<tr>
<td>0.8%</td>
<td>97.99%</td>
<td>97.02%</td>
<td>0.97%</td>
</tr>
<tr>
<td>1.0%</td>
<td>96.52%</td>
<td>94.96%</td>
<td>1.56%</td>
</tr>
<tr>
<td>2.0%</td>
<td>94.94%</td>
<td>92.69%</td>
<td>2.25%</td>
</tr>
<tr>
<td>4.0%</td>
<td>93.84%</td>
<td>90.80%</td>
<td>3.04%</td>
</tr>
<tr>
<td>5.0%</td>
<td>93.11%</td>
<td>88.65%</td>
<td>4.46%</td>
</tr>
</tbody>
</table>

ACKNOWLEDGMENT

This work was supported in part by the National Science Council, Taiwan, under Grant 100-2631-H-001-013 and 99-2221-E-001-017.

REFERENCES
