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
{eclipsex527, 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).

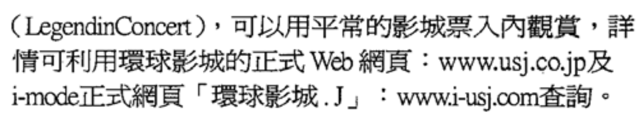
（LegendinConcert），可以用平常的影城票入內觀賞，詳
情可利用環球影城的正式 Web 網頁：www.usj.co.jp及
i-mode正式網頁「環球影城．J」：www.i-usj.com查詢。

Figure 1. Three categories of textual components (Chinese, alphanumeric, and punctuations) appear in the same textlines.

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 preprocessing steps. First, we form connected components out of black pixels (Chang et al. [10]). Next, we enclose those entities in rectangles, or boxes. Then, we merge two boxes into one if they overlap. For this reason, '玉' is enclosed in a single box, instead of two. Finally, 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, ‘汽’ 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 σ reduces the complexity from n^p to $(n/\sigma)\times\sigma^p = n\sigma^{p-1}$. 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 around 1,000 per second. All the computations were performed on an IBM XSERIES-3550 Intel Xeon CPU 2.49 GHz with 8GB RAM. The above results show that the bottleneck in our application lies in feature extraction, rather than in classification.

The remainder of the paper is organized as follows. In Section II, we present the DTSVM method. Section III describes our experiments, including the features extracted from components and the classifiers built on multiple feature types, as well as single types, and a sensitivity analysis. Section IV contains some concluding remarks.

II. The DTSVM Method

In this section, we give a brief description of the DTSVM method. A more detailed description can be found in Chang et al. [11]. The implementation of this method is available at

http://ocrwks11.iis.sinica.edu.tw/dar/Download/WebPages/DTSVM.htm,

along with the source code, the execution file, and a few exemplars.

We assume that all samples are represented as a d-dimensional feature vector whose class type is specified by a label y. For the decomposition scheme, we adopt the CART method (Breiman [12]) or binary C4.5 (Quinlan [13]). To train a CART decision tree, we start with the root, which takes all the training samples as input. We then decide whether to send each sample to the left-hand or the right-hand child node. The same procedure is repeated for each child node in a recursive manner.

At a given node E, we pick a feature f_E and a split point v_E so that all elements of E with $f_E < v_E$ are sent to the left-hand child node, and the remaining elements are sent to the right-hand child node. The values of f and v that maximize the information gain are taken as the values of f_E and v_E respectively, i.e.,

$$(f_E, v_E) = \arg\max_{(f,v)} IG_E(f, v).$$

The information gain $IG_E(f, v)$ is defined as follows.

$$IG_E(f,v) = U(E) - \frac{|E_L|}{|E|}U(E_L) - \frac{|E_R|}{|E|}U(E_R),$$

where E_L consists of the elements of E with $f < v$, $E_R = E \backslash E_L$, and |S| is the size of any set S. Furthermore,

$$U(S) = -\sum_y p(S_y)\log p(S_y),$$

where $p(S_y)$ is the proportion of S's samples that are labeled y.

We stop splitting a node E when one of following conditions is satisfied: (1) the number of samples that flow to E is lower than a *ceiling size* σ; or (2) when $IG_E(f, v) = 0$ for all f and v at E. The value of σ in the first condition is determined in a data-driven fashion, which we describe below. The second condition occurs most often when all the samples that flow to E are homogeneous.

After growing a tree, we train a local SVM on each of its leaves, using samples that flow to each leaf as training data. A tree and all local SVMs associated with its leaves constitute a DTSVM classifier.

The parameters associated with a DTSVM classifier are: (i) σ, the ceiling size of the decision tree; and (ii) the SVM-parameters. All the local SVMs in a DTSVM classifier take the same SVM-parameter values.

Given a training data set and a validation data set, we build DTSVM classifiers on the training data set and determine the optimal parameter values with the help of the validation data set. The training process proceeds as follows.

In the first stage, we train a tree with an *initial* ceiling size σ_0, and then train all the local SVMs with the same SVM-parameters $\boldsymbol{\theta}$. Note that $\boldsymbol{\theta}$ is expressed in boldface to indicate that it may consist of more than one parameter. Let

$v(\sigma_0, \boldsymbol{\theta})$ be the accuracy rate of the resultant DTSVM classifier, measured on the validation data set. In our experiments, we set $\sigma_0 = 1{,}500$.

In the subsequent stages, we construct DTSVM classifiers with larger ceiling sizes; however, we only train their local SVMs with the top-ranked $\boldsymbol{\theta}$, obtained by ranking $\boldsymbol{\theta}$ in descending order of $v(\sigma_0, \boldsymbol{\theta})$. Let $\Theta_{[k]}$ be a set containing k top-ranked $\boldsymbol{\theta}$. In our experiments, we set k at 5.

More specifically, we implement the following subprocess, denoted as *SubProcess*($\boldsymbol{\theta}$), for each $\boldsymbol{\theta}$ in $\Theta_{[k]}$.

1. Set $t = 0$ and get the binary tree with the ceiling size σ_0.
2. Increase t by 1 and set $\sigma_t = 4\sigma_{t-1}$. Modify the tree with ceiling size σ_{t-1} to obtain a tree with ceiling size σ_t. Then, train local SVMs on the leaves with SVM-parameters $\boldsymbol{\theta}$. Let $v(\sigma_t, \boldsymbol{\theta})$ be the validation accuracy of the resultant DTSVM classifier.
3. If $v(\sigma_t, \boldsymbol{\theta})$-$v(\sigma_{t-1}, \boldsymbol{\theta}) \geq 0.5\%$ and σ_t is less than the size of the training component, proceed to step 2.
4. If $v(\sigma_t, \boldsymbol{\theta})$-$v(\sigma_{t-1}, \boldsymbol{\theta}) < 0.5\%$, then $\sigma(\boldsymbol{\theta}) = \sigma_{t-1}$; otherwise, $\sigma(\boldsymbol{\theta}) = \sigma_t$.

When we have conducted *SubProcess*($\boldsymbol{\theta}$) for each $\boldsymbol{\theta}$ in $\Theta_{[k]}$, we define $\boldsymbol{\theta}_{opt}$ to be the $\boldsymbol{\theta}_0$ such that $v(\sigma(\boldsymbol{\theta}_0), \boldsymbol{\theta}_0) \geq v(\sigma(\boldsymbol{\theta}), \boldsymbol{\theta})$ for all $\boldsymbol{\theta}$ in $\Theta_{[k]}$. We also define σ_{opt} to be $\sigma(\boldsymbol{\theta}_{opt})$. We then output the DTSVM classifier with the SVM-parameter $\boldsymbol{\theta}_{opt}$ 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 I. THE TEXT CONTENT OF OUR DATA BASE USED IN THE EXPERIMENTS.

Textual Entity	**Size**
Document	1,517
Textline	29,907
Component (Total)	548,508
Chinese Component	173,038
Alphanumeric Component	343,313
Punctuation Component	32,157

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 = **I**, 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 height and C's width; 5) the ratio between C's top gap and H's height; and 6) the ratio 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 '丁' 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 features 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 in 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 E when $IG_E(f, v) = 0$ for all f and v at E. In the testing process, we assign each test sample $\mathbf{x}$ the label that is shared by the majority members of the leaf to which $\mathbf{x}$ flows. This implies that when this leaf has only one member, we assign its label to $\mathbf{x}$. 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 (Δ) 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 II. STATISTICS OF THE 10 COMBINATIONS OF FEATURE TYPES: THE TRAINING TIME IS EXPRESSED IN *SECONDS*; AND THE EXTRACTION, TESTING, AND ONLINE SPEEDS ARE EXPRESSED IN *COMPONENTS PER SECOND*.

Top	Feature								Training			Testing		
	I	***II***	***III***	***IV***	***V***	***VI***	***Dim***	***Extraction Speed***	***Ceiling Size***	***H-Rate***	***Time***	***Testing Speed***	***Online Speed***	***Test Accuracy***
1	Y	Y	Y		Y		150	820	1,500	85.00%	3,145	18,114	785	99.80%
2	Y	Y	Y		Y	Y	152	633	1,500	83.81%	3,254	17,838	612	99.79%
3	Y	Y	Y				86	1,186	1,500	84.42%	2,705	18,932	1,116	99.79%
4	Y		Y				70	2,033	1,500	81.03%	2,854	19,501	1,841	99.78%
5	Y		Y		Y		134	1,152	1,500	83.14%	3,035	19,312	1,087	99.78%
6	Y		Y			Y	72	1,174	1,500	81.12%	2,740	19,373	1,107	99.78%
7	Y	Y	Y			Y	88	831	1,500	83.81%	2,769	18,114	795	99.78%
8	Y		Y		Y	Y	136	514	1,500	83.30%	3,050	17,838	781	99.77%
9		Y	Y			Y	24	1,220	1,500	77.37%	1,363	18,932	1,165	99.64%
10	Y	Y			Y		144	901	1,500	75.44%	3,923	19,501	850	99.60%

TABLE III. THE TEST ACCURACY RATES FOR EACH CATEGORY OBTAINED BY THE TOP-10 DTSVM CLASSIFIERS.

Top	**Chinese**	**Alphanumeric**	**Punctuation**
1	99.70%	99.90%	99.29%
2	99.68%	99.90%	99.29%
3	99.71%	99.88%	99.29%
4	99.67%	99.88%	99.37%
5	99.68%	99.87%	99.38%
6	99.66%	99.87%	99.40%
7	99.68%	99.87%	99.29%
8	99.63%	99.87%	99.40%
9	99.50%	99.81%	98.53%
10	99.47%	99.74%	98.81%

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*.

ID	Training			Testing		
	Ceiling Size	***H-Rate***	***Training Time***	***Testing Speed***	***Online Speed***	***Test Accuracy***
I	1,500	67.52%	4,115	14,700	2,217	99.51%
II	1,500	55.56%	1,976	15,770	2,413	98.33%
III	1,500	56.18%	4,265	15,395	5,749	96.26%
IV	1,500	5.17%	35,021	5,862	1,688	87.65%
V	1,500	13.44%	11,982	4,971	1,733	87.26%
VI	1,500	0.44%	65,440	10,194	2,183	73.25%

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.

Percentage of Noise	**DTSVM**	**CART**	**Δ**
0.0%	99.80%	99.45%	0.35%
0.2%	99.26%	98.97%	0.29%
0.4%	98.83%	98.44%	0.38%
0.6%	98.48%	97.92%	0.56%
0.8%	98.21%	97.45%	0.76%
1.0%	97.99%	97.02%	0.97%
2.0%	96.52%	94.96%	1.56%
3.0%	94.94%	92.69%	2.25%
4.0%	93.84%	90.80%	3.04%
5.0%	93.11%	88.65%	4.46%

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

[1] J. Hochberg, K. Bowers, M. Cannon, and P. Kelly. Script and language identification for handwritten document images. *International Journal of Document Analysis and Recognition*, 2(2-3):45-52, 1999.

[2] S. Chanda, U. Pal, and O. Terrades. Word-Wise Thai and Roman Script Identification. *ACM Transactions on Asian Language Information Processing*, 8(3):1-21, 2009.

[3] J. Hochberg, P. Kelly, T. Thomas, and L. Kerns. Automatic script identification from document images using cluster-based templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):176-181, February 1997.

[4] A. L. Spitz. Using character shape codes for word spotting in document images. *Shape, Structure and Pattern Recognition*, 382-389, 1995.

[5] A. L. Spitz. Determination of the script and language content of document images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):235-245, March 1997.

[6] D. S. Lee, C. R. Nohl, and H. S. Baird. Language identification in complex, un-oriented and degraded document images. *Proceedings of the IAPR Workshop on Document Analysis Systems*, 17-39, 1996.

[7] M. C. Padma and P. A. Vijaya. Identification of Telugu, Devanagari and English script using discriminating features. *International Journal of Computer science & Information Technology*, 64-78, November 2009.

[8] T. N. Tan. Rotation invariant texture features and their use in automatic script identification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 751-756, July 1998.

[9] A. Busch, W. W. Sridharan, and S. Sridharan. Texture for script identification. *IEEE Transcriptions on Pattern Analysis and Machine Intelligence*, 27(11):1720-1731, November 2005.

[10] F. Chang, C.-J. Chen, and C.-J. Lu. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93(2):206-220, 2004.

[11] F. Chang, C.-Y. Guo, X.-R. Lin, and C.-J. Lu, Tree Decomposition for Large-Scale SVM Problems, Journal of Machine Learning Research, 11(Oct):2935−2972, 2010.

[12] V. Vapnik. The Nature of Statistical Learning Theory. Springer-Verlag, 1999.

[13] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman and Hall, 1984.

[14] J. R. Quinlan. Induction of Decision Tree. In *Machine Learning*, 1(1): 81-106, 1986.

[15] S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited: A stepwise procedure for building and training a neural network. In J. Fogelman, editor, *Neurocomputing: Algorithms, Architectures and Applications*. Springer-Verlag, 1990.