Perceptron Learning for Chinese Word Segmentation

Yaoyong Li[†], **Chuanjiang Miao**[‡], **Kalina Bontcheva**[†], **Hamish Cunningham**[†]
[†]Department of Computer Science, The University of Sheffield, Sheffield, S1 4DP, UK
{yaoyong,kalina,hamish}@dcs.shef.ac.uk
[‡]Institute of Chinese Information Processing, Beijing Normal University, Beijing, 100875, China
miaochj@bnu.edu.cn

Abstract

We explored a simple, fast and effective learning algorithm, the uneven margins Perceptron, for Chinese word segmentation. We adopted the character-based classification framework and transformed the task into several binary classification problems. We participated the close and open tests for all the four corpora. For the open test we only used the utf-8 code knowledge for discrimination among Latin characters, Arabic numbers and all other characters. Our system performed well on the as, cityu and msr corpora but was clearly worse than the best result on the pku corpus.

1 Introduction

We participated in the closed and open tests for all the four corpora, referred to as, cityu, msr and pku, respectively. We adopted the character-based methodology for Chinese word segmentation, that processed text character by character. We explored a simple and effective learning algorithm, the Perceptron with Uneven Margins (PAUM) for Chinese word segmentation task.

For the open task, we only used the minimal external information – the utf-8 code knowledge to distinguish Latin characters and Arabic numbers from other characters, justified by the fact that the English text requires no segmentation since they has been segmented already, and another fact that any Arabic number in one particular context should have the same segmentation.

2 Character Based Chinese Word Segmentation

We adopted the character based methodology for Chinese word segmentation, in which every character in a sentence was checked one by one to see if it was a word on its own or it was beginning, middle, or end character of a multi-character word. In contrast, another commonly used strategy, the word based methodology segments a Chinese sentence into the words in a pre-defined word list possibly with probability information about each word, according to some maximum probability criteria (see e.g. Chen (2003)). The performance of word based segmentation is dependent upon the quality of word list used, while the character based method does not need any word list – it segments a sentence only based on the characters in the sentence.

Using character based methodology, we transform the word segmentation problem into four binary classification problems, corresponding to single-character word, the beginning, middle and end character of multi-character word, respectively. For each of the four classes a classifier was learnt from training set using the one vs. all others paradigm, in which every character in the training data belonging to the class considered was regarded as positive example and all other characters were negative examples.

After learning, we applied the four classifiers to each character in test text and assigned the character the class which classifier had the maximal output among the four. This kind of strategy has been widely used in the applications of machine learning to named entity recognition and has also been used in Chinese word segmentation (Xue and Shen, 2003). Finally a word delimiter (often a blank space, depending on particular corpus) was added to the right of one character if it was not the last character of a sentence and it was predicted as end character of word or as a single character word.

3 Learning Algorithm

Perceptron is a simple and effective learning algorithm. For a binary classification problem, it checks the training examples one by one by predicting their labels. If the prediction is correct, the example is passed; otherwise, the example is used to correct the model. The algorithm stops when the model classifies all training examples correctly. The margin Perceptron not only classifies every training example correctly but also outputs for every training example a value (before thresholding) larger than a predefined parameter (margin). The margin Perceptron has better generalisation capability than the standard Perceptron. Li et al. (2002) proposed the Perceptron algorithm with uneven margins (PAUM) by introducing two margin parameters τ_+ and τ_- into the update rules for the positive and negative examples, respectively. Two margin parameters allow the PAUM to handle imbalanced datasets better than both the standard Perceptron and the margin Perceptron. PAUM has been successfully used for document classification and information extraction (Li et al., 2005).

We used the PAUM algorithm to train a classifier for each of four classes for Chinese word segmentation. For one test example, the output of the Perceptron classifier before thresholding was used for comparison among the four classifiers. The important parameters of the learning algorithm are the uneven margins parameters τ_+ and τ_-. In all our experiments $\tau_+ = 20$ and $\tau_- = 1$ were used.

Table 1 presents the results for each of the four classification problems, obtained from 4-fold cross-validation on training set. Not surprisingly, the classification for middle character of multi-character word was much harder than other three classification problems, since middle character of Chinese word is less characteristic than beginning or end character or single-character word. On the other hand, improvement on the classification for middle character, while keeping the performances of other classification, would improve the overall performance of segmentation.

Table 1: Results for each of the four classifiers: F1 (%) averaged over 4-fold cross-validation on training sets of the four corpora. C1, C2 and C3 refer to the classifier for beginning, middle and end character of multi-character word, respectively, and C4 refers to the classifier for single character word.

	C1	C2	C3	C4
as	95.64	90.07	95.47	95.27
cityu	96.64	90.06	96.43	95.14
msr	96.36	89.79	96.00	94.99
pku	96.09	89.99	96.18	94.12

Support vector machines (SVM) is a popular learning algorithm, which has been successfully applied to many classification problems in natural language processing. Similar to the PAUM, SVM is a maximal margin algorithm. Table 2 presents a comparison of performances and computation times between the PAUM and the SVM with linear kernel[1] on three subsets of cityu corpora with different sizes. The performance of SVM was better than the PAUM. However, the larger the training data was, the closer the performance of PAUM to that of SVM. On the other hand, SVM took much longer computation time than PAUM. As a matter of fact, we have run the SVM with linear kernel on the whole cityu training corpus using 4-fold cross-validation for one month and it has not finished yet. In contrast, PAUM just took about one hour to run the same experiment.

4 Features for Each Character

In our system every character was regarded as one instance for classification. The features for one character were the character form itself and the character forms of the two preceding and the two following characters of the current one. In other word, the features for one character c_0 were the character forms from a context win-

[1] The SVM^{light} package version 5.0, available from http://svmlight.joachims.org/, was used to learn the SVM classifiers in our experiments.

Table 2: Comparison of the Perceptron with SVM for Chinese word segmentation: averaged F1 (%) over the 4-fold cross-validation on three subsets of cityu corpus and the computation time (in second) for each experiment. The three subsets have 100, 1000 and 5000 sentences, respectively.

	100	1000	5000
PAUM	73.55	78.00	88.08
	4s	14s	92s
SVM	75.50	79.15	88.78
	227s	3977s	49353s

dow centering at c_0 and containing five characters $\{c_{-2}, c_{-1}, c_0, c_1, c_2\}$ in a sentence. Our experiments on training data showed that co-occurrences of characters in the context window were helpful. Taking account of all co-occurrences of characters in context window is equivalent to using a quadratic kernel in Perceptron, while not using any co-occurrence amounts to a linear kernel. Actually we can only use part of co-occurrences as features, which can be regarded as some kind of semi-quadratic kernel.

Table 3 compares the three types of kernel for Perceptron, where for the semi-quadratic kernel we used the co-occurrences of characters in context window as those used in (Xue and Shen, 2003), namely $\{c_{-2}c_{-1}, c_{-1}c_0, c_0c_1, c_1c_2, c_{-1}c_1\}$. It was shown that the quadratic kernel gave much better results than linear kernel and the semi-quadratic kernel was slightly better than fully quadratic kernel. Semi-quadratic kernel also led to less feature and less computation time than fully quadratic kernel. Therefore, this kind of semi-quadratic kernel was used in our submissions.

Table 3: Comparisons between different kernels for Perceptron: F1 (%) averaged over 4-fold cross-validation on three training sets.

	linear	quadratic	semi-quadratic
cityu	81.30	94.78	95.13
msr	79.80	94.78	94.93
pku	82.33	94.80	95.05

Actually it has been noted that quadratic kernel for Perceptron, as well as for SVM, performed better than linear kernel for information extraction and other NLP tasks (see e.g. Carreras et al. (2003)). However, quadratic kernel was usually implemented in dual form for Perceptron and it took very long time for training. We implemented the quadratic kernel for Perceptron in primal form by encoding the linear and quadratic features into feature vector explicitly. Actually our implementation performed even slightly better than the Perceptron with quadratic kernel as we used only part of quadratic features, and it was still as efficient as the Perceptron with linear kernel.

5 Open Test

While closed test required the participants only to use the information presented in training material, open test allowed to use any external information or resources besides the training data. In our submissions for the open test we just used the minimal external information, namely the utf-8 code knowledge for identifying a piece of English text or an Arabic number. and What we did by using this kind of knowledge was to pre-process the text by replacing each piece of English text with a symbol "E" and replacing every Arabic number with another symbol "N". This kind of pre-processing resulted in a smaller training data and less computation time and yet slightly better performance on training data, as shown in Table 4 which compares the results of collapsing the English text only and collapsing both the English text and Arabic number with those for closed test. Table 4 also presents the 95% confidence intervals for the F-measures.

6 Results on Test Data

Table 5 presents our official results on test corpora for both close and open tests. First, comparing with the results in Table 4, the results on test set are significantly different from the result using 4-fold cross validation on training set for all the four corpora. The test result was better than the results on training set for the msr corpus but was worse for other three corpora, especially for the pku corpora. We suspected that this may be caused by difference between training and test data, which needs further investigation.

Table 4: Comparisons between the results for close and open tests: averaged F1 (%) and the 95% confidence interval on the 4-fold cross-validation on the training sets of four corpora and the computation time (in hour) for each experiment. "English" means only collapsing English texts and "E & N" means collapsing both English texts and Arabic numbers.

	close test	English	E & N
as	95.53±0.46	95.65±0.47	95.78±0.46
	8.88h	7.66h	7.07h
cityu	95.13 ±1.49	95.25 ±1.48	95.25 ±1.48
	1.03h	0.86h	0.82h
msr	94.92 ±0.36	94.98 ±0.40	95.00 ±0.39
	2.62h	1.69h	1.62h
pku	95.05 ±0.43	95.08 ±0.36	95.15 ±0.46
	0.70h	0.63h	0.60h

Secondly, the test results for close and open tests are close to each other on other three corpora except the pku corpora, for which the result for open test is clearly better than that for close test. This was mainly because of different encoding of Arabic number in training and test sets of the pku corpus. Since Arabic number was encoded in three bytes in training set but was encoded in one byte in test set for the pku corpora, for close test the trained model for Arabic number was not applicable to the Arabic numbers in test set. However, for open test, as we replaced Arabic number with one symbol in both training and test sets, the different encoding of Arabic number in training and test sets could not cause any problem at all, which led to better result. On the other hand, our pre-processing with respect to the English text and Arabic numbers seemed have slightly effect on the F-measure for other three corpora.

Finally, comparing with the results of closed test from other participants, our F1 figures were no more than 0.008 lower than the best ones on the as, cityu and msr corpora, but was 0.023 lower than the best one on the pku corpus.

7 Conclusion

Table 5: The official results on test set: F-measure (%) for close and open tests, respectively.

	as	cityu	msr	pku
close	94.4	93.6	95.6	92.7
open	94.8	93.6	95.4	93.8

We applied the uneven margins Perceptron to Chinese word segmentation. The learning algorithm is simple, fast and effective. The results obtained are encouraging.

The performance of Perceptron was close to that of the SVM on Chinese word segmentation for large training data. On the other hand, the Perceptron took much less computation time than SVM. We implemented the Perceptron with semi-quadratic kernel in primal form. Our implementation was both effective and efficient.

Our system performed well for the three of four corpora, as, cityu and msr corpora. But it was significantly worse than the best result on the pku corpora, which needs further investigation.

Acknowledgements

This work is supported by the EU-funded SEKT project (http://www.sekt-project.org).

References

X. Carreras, L. Màrquez, and L. Padró. 2003. Learning a perceptron-based named entity chunker via online recognition feedback. In *Proceedings of CoNLL-2003*, pages 156–159. Edmonton, Canada.

A. Chen. 2003. Chinese Word Segmentation Using Minimal Linguistic Knowledge. In *Proceedings of the 2nd SIGHAN Workshop on Chinese Language Processing*.

Y. Li, H. Zaragoza, R. Herbrich, J. Shawe-Taylor, and J. Kandola. 2002. The Perceptron Algorithm with Uneven Margins. In *Proceedings of the 9th International Conference on Machine Learning (ICML-2002)*, pages 379–386.

Y. Li, K. Bontcheva, and H. Cunningham. 2005. Using Uneven Margins SVM and Perceptron for Information Extraction. In *Proceedings of Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*.

N. Xue and L. Shen. 2003. Chinese Word Segmentation as LMR Tagging. In *Proceedings of the 2nd SIGHAN Workshop on Chinese Language Processing*.