Lightweight Client-Side Chinese/Japanese Morphological Analyzer Based on Online Learning

Masato Hagiwara **Satoshi Sekine**
Rakuten Institute of Technology, New York
215 Park Avenue South, New York, NY
`{masato.hagiwara, satoshi.b.sekine}@mail.rakuten.com`

Abstract

As mobile devices and Web applications become popular, lightweight, client-side language analysis is more important than ever. We propose Rakuten MA, a Chinese/Japanese morphological analyzer written in JavaScript. It employs an online learning algorithm SCW, which enables client-side model update and domain adaptation. We have achieved a compact model size (5MB) while maintaining the state-of-the-art performance, via techniques such as feature hashing, FOBOS, and feature quantization.

1 Introduction

Word segmentation (WS) and part-of-speech (PoS) tagging, often jointly called morphological analysis (MA), are the essential component for processing Chinese and Japanese, where words are not explicitly separated by whitespaces. There have been many word segmentater and PoS taggers proposed in both Chinese and Japanese, such as Stanford Segmenter (Tseng et al., 2005), zpar (Zhang and Clark, 2011), MeCab (Kudo et al., 2004), JUMAN (Kurohashi and Nagao, 1994), to name a few. Most of them are intended for server-side use and provide limited capability to extend or re-train models. However, as mobile devices such as smartphones and tablets become popular, there is a growing need for client based, lightweight language analysis, and a growing number of applications are built upon lightweight languages such as HTML, CSS, and JavaScript. Techniques such as domain adaptation and model extension are also becoming more important than ever.

In this paper, we present Rakuten MA, a morphological analyzer entirely written in JavaScript based on online learning. We will be releasing the software as open source before the COLING 2014 conference at `https://github.com/rakuten-nlp/rakutenma`, under Apache License, version 2.0. It relies on general, character-based sequential tagging, which is applicable to any languages and tasks which can be processed in a character-by-character basis, including WS and PoS tagging for Chinese and Japanese. Notable features include:

1. JavaScript based — Rakuten MA works as a JavaScript library, the de facto "lingua franca" of the Web. It works on popular Web browsers as well as node.js, which enables a wide range of adoption such as smartphones and Web browser extensions. Note that TinySegmenter[1] is also entirely written in JavaScript, but it does not support model re-training or any languages other than Japanese. It doesn't output PoS tags, either.
2. Compact — JavaScript-based analyzers pose a difficult technical challenge, that is, the compactness of the model. Modern language analyzers often rely on a large number of features and/or dictionary entries, which is impractical on JavaScript runtime environments. In order to address this issue, Rakuten MA implements some notable techniques. First, the features are character-based and don't rely on dictionaries. Therefore, while it is inherently incapable of dealing with words which are longer than features can capture, it may be robust

[1]`http://chasen.org/~taku/software/TinySegmenter/`

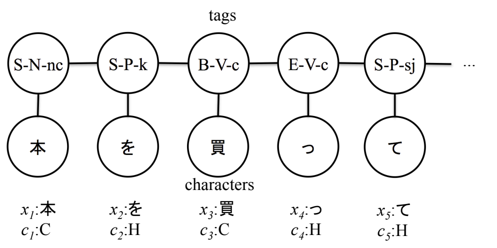

Figure 1: Character-based tagging model

Feature	Description
$x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}$	char. unigrams
$x_{i-2}x_{i-1}, x_{i-1}x_i, x_ix_{i+1}, x_{i+1}x_{i+2}$	char. bigrams
$c_{i-2}, c_{i-1}, c_i, c_{i+1}, c_{i+2}$	type unigrams
$c_{i-2}c_{i-1}, c_{i-1}c_i, c_ic_{i+1}, c_{i+1}c_{i+2}$	type bigrams

Table 1: Feature Templates Used for Tagging x_i and c_i are the character and the character type at i[a].

[a]Each feature template is instantiated and concatenated with possible tags. Character type bigram features were only used for JA. In CN, we built a character type dictionary, where character types are simply all the possible tags in the training corpus for a particular character.

to unknown words compared with purely dictionary-based systems. Second, it employs techniques such as feature hashing (Weinberger et al., 2009), FOBOS (Duchi and Singer, 2009), and feature quantization, to make the model compact while maintaining the same level of analysis performance.

3. Online Learning — it employs a modern online learning algorithm called SCW (Wang et al., 2012), and the model can be incrementally updated by feeding new instances. This enables users to update the model if errors are found, without even leaving the Web browser or node.js. Domain adaptation is also straightforward. Note that MeCab (Kudo et al., 2004), also supports model re-training using a small re-training corpus. However, the training is inherently a batch, iterative algorithm (CRF) thus it is hard to predict when it finishes.

2 Analysis Model and Compact Model Representation

Base Model Rakuten MA employs the standard character-based sequential tagging model. It assigns combination of position tags[2] and PoS tags to each character (Figure 1). The optimal tag sequence $\boldsymbol{y}^*$ for an input string x is inferred based on the features $\phi(y)$ and the weight vector $\boldsymbol{w}$ as $\boldsymbol{y}^* = \arg\max_{y \in Y(x)} \boldsymbol{w} \cdot \phi(y)$, where $Y(x)$ denotes all the possible tag sequences for x, via standard Viterbi decoding. Table 1 shows the feature template sets.

For training, we used soft confidence weighted (SCW) (Wang et al., 2012). SCW is an online learning scheme based on Confidence Weighted (CW), which maintains "confidence" of each parameter as variance Σ in order to better control the updates. Since SCW itself is a general classification model, we employed the structured prediction model (Collins, 2002) for WS.

The code snippet in Figure 2 shows typical usage of Rakuten MA in an interactive way. Lines starting with "//" and ">" are comments and user input, and the next lines are returned results. Notice that the analysis of バラクオバマ大統領 "President Barak Obama" get better as the model observes more instances. The analyzer can only segment it into individual characters when the model is empty ((1) in the code), whereas WS is partially correct after observing the first 10 sentences of the corpus ((2) in the code). After directly providing the gold standard, the result (3) becomes perfect.

We used and compared the following three techniques for compact model representations:

Feature Hashing (Weinberger et al., 2009) applies hashing functions h which turn an arbitrary feature $\phi_i(y)$ into a bounded integer value v, i.e., $v = h(\phi_i(y)) \in R$, where $0 \leq v < 2^N$, (N = hash size). This technique is especially useful for online learning, where a large, growing number of features such as character/word n-grams could be observed on the fly, which the model would otherwise need to keep track of using flexible data structures such as trie, which could make training slower as the model observes more training instances. The negative effect of hash collisions to the performance is negligible because most collisions are between rare features.

[2]As for the position tags, we employed the SBIEO scheme, where S stands for a single character word, BIE for beginning, middle, and end of a word, respectively, and O for other positions.

```
// initialize with empty model
> var r = new RakutenMA({});

// (1) first attempt, failed with separate chars.
> r.tokenize("バラクオバマ大統領").toString()
"バ,,ラ,,ク,,オ,,バ,,マ,,大,,統,,領,"

// train with first 10 sentences in a corpus
> for (var i = 0; i < 10; i ++)
>   r.train_one( rcorpus[i] );

// the model is no longer empty
> r.model
Object {mu: Object, sigma: Object}
```

```
// (2) second attempt -> getting closer
> r.tokenize("バラクオバマ大統領").toString()
"バラク,N-nc,オバマ,N-pn,大,,統,,領,Q-n"

// retrain with an answer
// return object suggests there was an update
> r.train_one([["バラク","N-np"],
...    ["オバマ","N-np"],["大統領","N-nc"]]);
Object {ans: Array[3], sys: Array[5], updated: true}

// (3) third attempt
> r.tokenize("バラクオバマ大統領").toString()
"バラク,N-np,オバマ,N-np,大統領,N-nc"
```

Figure 2: Rakuten MA usage example

Chinese	Prec.	Rec.	F	Japanese	Prec.	Rec.	F
Stanford Parser	97.37	93.54	95.42	MeCab+UniDic	99.15	99.61	99.38
zpar	91.18	92.36	91.77	JUMAN	**88.55	**83.06	**85.72
Rakuten MA	92.61	92.64	92.62	KyTea	*80.57	*85.02	*82.73
				TinySegmenter	*86.93	*85.19	*86.05
				Rakuten MA	96.76	97.30	97.03

Table 2: Segmentation Performance Comparison with Different Systems
* These systems use different WS criteria and their performance is shown simply for reference. ** We postprocessed JUMAN's WS result so that the WS criteria are closer to Rakuten MA's.

Forward-Backward Splitting (FOBOS) (Duchi and Singer, 2009) is a framework to introduce regularization to online learning algorithms. For each training instance, it runs unconstrained parameter update of the original algorithm as the first phase, then solves an instantaneous optimization problem to minimize a regularization term while keeping the parameter close to the first phrase. Specifically, letting $w_{t+\frac{1}{2},j}$ the j-th parameter after the first phrase of iteration t and λ the regularization coefficient, parameter update of FOBOS with L1 regularization is done by: $w_{t+1,j} = \text{sign}\left(w_{t+\frac{1}{2},j}\right)\left[\left|w_{t+\frac{1}{2},j}\right| - \lambda\right]_+$. The strength of regularization can be adjusted by the coefficient λ. In combining SCW and FOBOS, we retained the confidence value Σ of SCW unchanged.

Feature Quantization simply multiplies float numbers (e.g., 0.0165725659236262) by M (e.g., 1,000) and round it to obtain a short integer (e.g., 16). The multiple M determines the strength of quantization, i.e., the larger the finer grained, but the larger model size.

3 Experiments

We used CTB 7.0 (Xue et al., 2005) for Chinese (CN), and BCCWJ (Maekawa, 2008) for Japanese (JA), with 50,805 and 60,374 sentences, respectively. We used the top two levels of BCCWJ's PoS tag hierarchy (38 unique tags) and all the CTB PoS tags (38 unique tags). The average decoding time was 250 millisecond per sentence on Intel Xeon 2.13GHz, measured on node.js. We used precision (Prec.), recall (Rec.), and F-value (F) of WS as the evaluation metrics, averaged over 5-fold cross validation. We ignored the PoS tags in the evaluation because they are especially difficult to compare across different systems with different PoS tag hierarchies.

Comparison with Other Analyzers First, we compare the performance of Rakuten MA with other word segmenters. In CN, we compared with Stanford Segmenter (Tseng et al., 2005) and zpar (Zhang and Clark, 2011). In JA, we compared with MeCab (Kudo et al., 2004), JUMAN (Kurohashi and Nagao, 1994), KyTea (Neubig et al., 2011), and TinySegmenter.

Table 2 shows the result. Note that, some of the systems (e.g., Stanford Parser for CN and MeCab+UniDic for JA) use the same corpus as the training data and their performance is unfairly high. Also, other systems such as JUMAN and KyTea employ different WS criteria, and their performance is unfairly low, although JUMAN's WS result was postprocessed so that

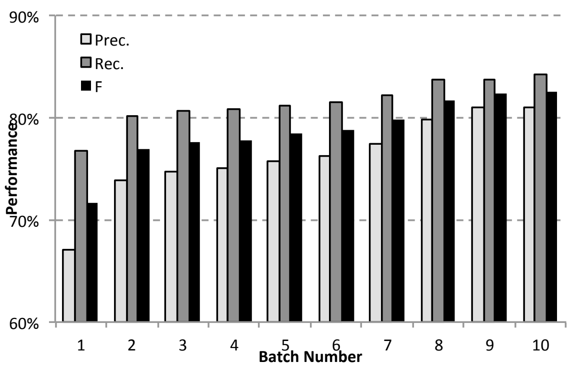

Figure 3: Domain Adaptation Result for CN

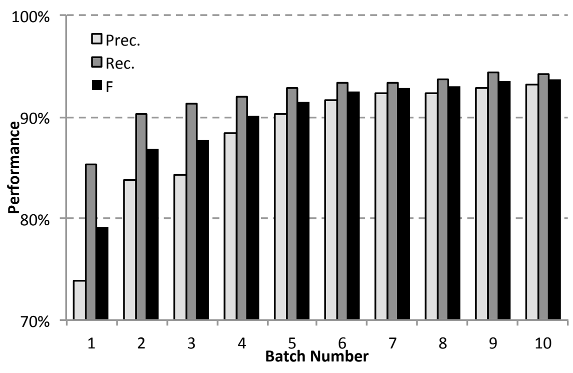

Figure 4: Domain Adaptation Result for JA

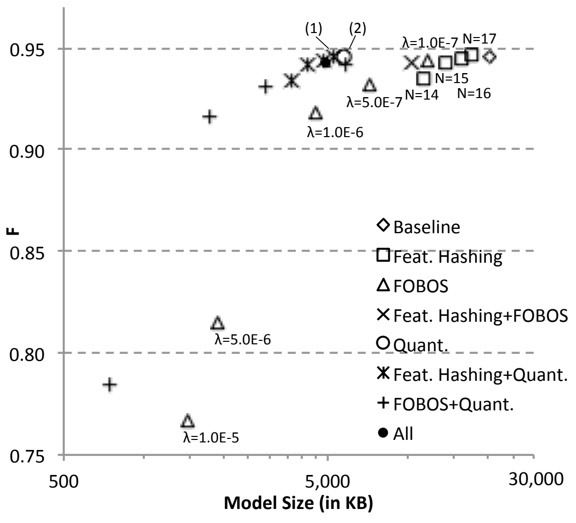

Figure 5: Model Comparison

it gives a better idea how it compares with Rakuten MA. We can see that Rakuten MA can achieve WS performance comparable with the state-of-the-art even without using dictionaries.

Domain Adaptation Second, we tested Rakuten MA's domain adaptation ability. We chose e-commerce as the target domain, since it is a rich source of out-of-vocabulary words and poses a challenge to analyzers trained on newswire text. We sampled product titles and descriptions from Rakuten Taiwan[3] (for CN, 2,786 sentences) and Rakuten Ichiba[4] (for JA, 13,268 sentences). These collections were then annotated by human native speakers in each language, following the tagging guidelines of CTB (for CN) and BCCWJ (for JA).

We divided the corpus into 5 folds, then used four of them for re-training and one for testing. The re-training data is divided into "batches," consisting of mutually exclusive 50 sentences, which were fed to the pre-trained model on CTB (for CN) and BCCWJ (for JA) one by one.

Figure 3 and 4 show the results. The performance quickly levels off after five batches for JA, which gives an approximated number of re-training instances needed (200 to 300) for adaptation. Note that adaptation took longer on CN, which may be attributed to the fact that Chinese WS itself is a harder problem, and to the disparity between CTB (mainly news articles in mainland China) and the adaptation corpus (e-commerce text in Taiwan).

[3]`http://www.rakuten.com.tw/`. Note that Rakuten Taiwan is written in Taiwanese traditional Chinese. It was converted to simplified Chinese by using Wikipedia's traditional-simplified conversion table `http://svn.wikimedia.org/viewvc/mediawiki/trunk/phase3/includes/ZhConversion.php`. Still, having large Taiwan specific vocabulary poses additional challenges for domain adaptation.

[4]`http://www.rakuten.co.jp/`

Compact Model Representation Third, we consider the three techniques, and investigate how these techniques affect the trade-off between WS performance and the model size, which is measured by the feature trie byte size in raw JSON format[5].

Figure 5 shows the scatter plot of F-value vs model size in KB, since we are rather interested in the trade-off between the model size and the performance. The baseline is the raw model without any techniques mentioned above. Notice that the figure's x-axis is in log scale, and the upper left corner in the figure corresponds to better trade-off (higher performance with smaller model size). We can observe that all the three techniques can reduce the model size to some extent while keeping the performance at the same level. In fact, these three techniques are independent from each other and can be freely combined to achieve better trade-off. If we limit strictly the same level of performance compared to the baseline, feature hashing (17bit hash size) with quantization (Point (1) in the figure) seems to achieve the best trade-off, slightly *outperforming* the baseline (F = 0.9457 vs F = 0.9455 of the baseline) with the model size of as little as one fourth (5.2MB vs 20.6MB of the baseline). It is somewhat surprising to see that feature quantization, which is a very simple method, achieves relatively good performance-size trade-off (Point (2) in the figure).

4 Conclusion

In this paper, we proposed Rakuten MA, a lightweight, client-side morphological analyzer entirely written in JavaScript. It supports online learning based on the SCW algorithm, which enables quick domain adaptation, as shown in the experiments. We successfully achieved a compact model size of as little as 5MB while maintaining the state-of-the-art performance, using feature hashing, FOBOS, and feature quantization. We are planning to achieve even smaller model size by adopting succinct data structure such as wavelet tree (Grossi et al., 2003).

Acknowledgements

The authors thank Satoko Marumoto, Keiji Shinzato, Keita Yaegashi, and Soh Masuko for their contribution to this project.

References

Michael Collins. 2002. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proc. of the EMNLP 2002*, pages 1–8.

John Duchi and Yoram Singer. 2009. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2899–2934.

Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. 2003. High-order entropy-compressed text indexes. In *Prof. of SODA 2003*, pages 841–850.

Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. 2004. Applying conditional random fields to Japanese morphological analysis. In *Proceedings of EMNLP*, pages 230–237.

Sadao Kurohashi and Makoto Nagao. 1994. Improvements of Japanese morphological analyzer JUMAN. In *Proceedings of the International Workshop on Sharable Natural Language Resources*, pages 22–38.

Kikuo Maekawa. 2008. Compilation of the Kotonoha-BCCWJ corpus (in Japanese). *Nihongo no kenkyu (Studies in Japanese)*, 4(1):82–95.

Graham Neubig, Yosuke Nakata, and Shinsuke Mori. 2011. Pointwise prediction for robust, adaptable japanese morphological analysis. In *Proceedings of ACL-HLT*, pages 529–533.

Huihsin Tseng, Pichuan Chang, Galen Andrew, Daniel Jurafsky, and Christopher Manning. 2005. A conditional random field word segmenter. In *Fourth SIGHAN Workshop on Chinese Language Processing*.

Jialei Wang, Peilin Zhao, and Steven C. Hoi. 2012. Exact soft confidence-weighted learning. In *Proc. of ICML 2012*, pages 121–128.

Kilian Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford, and Alex Smola. 2009. Feature hashing for large scale multitask learning. In *Proc. of ICML 2009*, pages 1113–1120.

Naiwen Xue, Fei Xia, Fu-dong Chiou, and Marta Palmer. 2005. The penn Chinese treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

[5]We used one fifth of the Japanese corpus BCCWJ for this experiment. Parameter λ of FOBOS was varied over $\{1.0 \times 10^{-7}, 5.0 \times 10^{-7}, 1.0 \times 10^{-6}, 5.0 \times 10^{-6}, 1.0 \times 10^{-5}\}$. The hash size of feature hashing was varied over $14, 15, 16, 17$[6]. The multiple of feature quantization M is set to $M = 1000$.