An Experimental Comparison of the Voted Perceptron and Support Vector Machines in Japanese Analysis Tasks

Manabu Sassano
Yahoo Japan Corporation
6-10-1 Roppongi, Minato-ku, Tokyo 106-6182 Japan
msassano@yahoo-corp.jp

Abstract

We examine various aspects of the voted perceptron and support vector machines in classification tasks in NLP rather than ranking tasks. These aspects include training time, accuracy and learning curves. We used Japanese dependency parsing as a main task for experiments, and Japanese word segmentation and bunsetsu (base phrase in Japanese) chunking as auxiliary tasks. In our experiments we have observed that the voted perceptron is comparable to SVM in terms of accuracy and, in addition, as to learning time and prediction speed the voted perceptron is considerably better than SVM.

1 Introduction

Support vector machines (SVM) (Vapnik, 1995) have been shown to be effective for many natural language processing (NLP) tasks (e.g., (Kudo and Matsumoto, 2001; Kudo and Matsumoto, 2002)). However, there are still some practical difficulties when we apply SVM to NLP tasks. The weakness of SVM is that they are not easy to implement and their learning process is slow, especially with polynomial kernels.

(Freund and Schapire, 1999) propose the voted perceptron, which is an improved version of Perceptron (Rosenblatt, 1958), and they give theoretical analysis and have proved a good performance for the hand-written digit recognition. Although Collins and his colleagues use the voted perceptron for ranking in various NLP tasks (Collins, 2002b; Collins and Duffy, 2002; Collins, 2002a) and obtain impressive results, the use as a classifier has been not sufficiently examined. In particular, it would be an interesting question whether or not the voted perceptron is comparable to SVM for NLP tasks that are formalized as a classification one.

In this paper we focus on comparison of SVM and the voted perceptron to investigate the usefulness of the voted perceptron in NLP tasks. We would like to know the strength and weakness of the voted perceptron. We choose three tasks for this purpose. These tasks are Japanese word segmentation, *bunsetsu* (base phrase in Japanese) chunking, and dependency parsing.

Experiments indicate that SVM and the voted perceptron are equally good for the three tasks in terms of accuracy. However, the voted perceptron is superior to SVM in terms of learning time, prediction time, and memory footprint.

2 The Voted Perceptron

Following (Freund and Schapire, 1999), we show the training and prediction algorithm of the voted perceptron in Figure 1. The voted perceptron as well as SVM can use a kernel function. We show in Figure 2 the algorithm of the voted perceptron with a kernel function. This algorithm seems to require $O(k^2)$ kernel calculations. However, we can avoid them by taking advantage of the recurrence $\boldsymbol{v}_{j+1} \cdot \boldsymbol{x} = \boldsymbol{v}_j \cdot \boldsymbol{x} + y_{u_j} K(\boldsymbol{x}_{u_j}, \boldsymbol{x})$.[1]

3 Task Description

We used Japanese dependency parsing as a main task for experiments and Japanese word segmenta-

[1] Herbrich describes an optimized version of the algorithm of the kernel perceptron (Herbrich, 2002, page 322). We can use the same technique in training of the kernel version of the voted perceptron.

Training

Input: a labeled training set:
$\langle (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m) \rangle$.
number of epochs: T

Output: a list of weighted perceptrons:
$\langle (\boldsymbol{v}_1, c_1), \ldots, (\boldsymbol{v}_k, c_k) \rangle$

- Initialize: $k := 0, \boldsymbol{v}_1 := \boldsymbol{0}, c_1 := 0$.
- Repeat T times:
  - For $i := 1, \ldots, m$:
    - Compute prediction: $\hat{y} := \mathrm{sign}(\boldsymbol{v}_k \cdot \boldsymbol{x}_i)$
    - If $\hat{y} = y$ then $c_k := c_k + 1$.
      else $\boldsymbol{v}_{k+1} := \boldsymbol{v}_k + y_i \boldsymbol{x}_i$;
      $c_{k+1} := 1; k := k + 1$.

Prediction

Given: the list of weighted perceptrons:
$\langle (\boldsymbol{v}_1, c_1), \cdots, (\boldsymbol{v}_k, c_k) \rangle$
an unlabeled instance: $\boldsymbol{x}$

compute a predicted label $\hat{y}$ as follows:

$$s := \sum_{i=1}^{k} c_i \, \mathrm{sign}(\boldsymbol{v}_i \cdot \boldsymbol{x}); \hat{y} := \mathrm{sign}(s).$$

Figure 1: The voted-perceptron algorithm

Training

Input: $\langle (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m) \rangle$ and T

Output: a list of mistaken examples and weights:
$\langle (u_1, c_1), \ldots, (u_k, c_k) \rangle$

- Initialize: $k := 0, \boldsymbol{v}_1 := \boldsymbol{0}, c_1 := 0$.
- Repeat T times:
  - For $i := 1, \ldots, m$:
    - Compute prediction:

$$\boldsymbol{v}_k \cdot \boldsymbol{x}_i := \sum_{j=1}^{k} y_{u_j} K(\boldsymbol{x}_{u_j}, \boldsymbol{x}_i); \hat{y} := \mathrm{sign}(\boldsymbol{v}_k \cdot \boldsymbol{x}_i)$$

    - If $\hat{y} = y$ then $c_k := c_k + 1$.
      else $u_{k+1} := i$;
      $c_{k+1} := 1; k := k + 1$.

Prediction

Given: $\langle (u_1, c_1), \cdots, (u_k, c_k) \rangle$ and $\boldsymbol{x}$

compute a predicted label $\hat{y}$ as follows:

$$\boldsymbol{v}_i \cdot \boldsymbol{x} := \sum_{j=1}^{i} y_{u_j} K(\boldsymbol{x}_{u_j}, \boldsymbol{x}); s := \sum_{i=1}^{k} c_i \, \mathrm{sign}(\boldsymbol{v}_i \cdot \boldsymbol{x});$$

$$\hat{y} := \mathrm{sign}(s).$$

Figure 2: Algorithm of the voted-perceptron with a kernel function

tion and bunsetsu chunking as auxiliary tasks.

3.1 Japanese Dependency Parsing (JDP)

Japanese dependency parsing is to determine the dependency structure of a given sentence which is represented as a sequence of *bunsetsu*s (base phrases in Japanese). We employ the Stack Dependency Analysis (SDA) algorithm (Sassano, 2004; Nivre, 2003), which is very simple and easy to implement. Sassano (2004) has proved its efficiency in terms of time complexity and reported the best accuracy on the Kyoto University Corpus Version 2 (Kurohashi and Nagao, 1998). This algorithm, which can be used with any classifier that determines whether a given bunsetsu modifies another, is suitable for our study since we intend to test both SVM and the voted perceptron.

We use a set of standard features for this task. By the "standard features" here we mean the feature set commonly used in (Uchimoto et al., 1999; Sekine et al., 2000; Kudo and Matsumoto, 2000; Kudo and Matsumoto, 2002; Sassano, 2004). We employ the features below for each bunsetsu:

1. Rightmost Content Word - major POS, minor POS, conjugation type, conjugation form, surface form (lexicalized form)
2. Rightmost Function Word - major POS, minor POS, conjugation type, conjugation form, surface form (lexicalized form)
3. Punctuation (periods, and commas)
4. Open parentheses and close parentheses
5. Location - at the beginning of the sentence or at the end of the sentence.

In addition, features as to the gap between two bunsetsus are also used. They include: distance, particles, parentheses, and punctuation.

3.2 Japanese Bunsetsu Chunking (JBC)

Following (Ramshaw and Marcus, 1995), we encode bunsetsu chunking as a tagging problem. In bunsetsu chunking, we use the chunk tag set {`B`, `I`} where `B` marks the first word of some bunsetsu and words marked `I` are inside a bunsetsu.

In our experiments on bunsetsu chunking, we estimated the chunk tag of each word using five

Table 1: The Size of the Training Data

	JWS	JBC	JDP
# of features	11916	121081	40842
# of examples	350584	198514	98689

words and their derived attributes. These five words are the word to be estimated and its two preceding/following words. Features are extracted from the followings for each word: word (token) itself, major POS, minor POS, conjugation type, conjugation form, the leftmost character, the character type of the leftmost character, the rightmost character, and the character type of the rightmost character. A character type has a value which indicates a script. This value can be one of the following: kanji (Chinese character), hiragana (Japanese syllabic character), katakana (another syllabic character), number, Latin letter, or symbol.

3.3 Japanese Word Segmentation (JWS)

Japanese word segmentation can be formulated as a classification task (Shinnou, 2000). Let a Japanese character sequence be $s = c_1 c_2 \cdots c_m$ and a boundary b_i exist between c_{i-1} and c_i. The b_i is either $+1$ (word boundary) or -1 (non-boundary). The word segmentation task can be defined as determining the class of the b_i.

We assume that each character c_i has two attributes. The first attribute is a character type (t_i). The second one is a character code (k_i). We use here five characters to decide a word boundary. A set of the attributes of $c_{i-2}, c_{i-1}, c_i, c_{i+1}$, and c_{i+2} is used to predict the label of the b_i.

4 Experimental Results and Discussion

4.1 Corpus

We used the Kyoto University Corpus Version 2 (Kurohashi and Nagao, 1998). Analysis systems used in any of our experiments were trained on the articles on January 1st through 8th (7,958 sentences) and tested on the articles on January 9th (1,246 sentences). The articles on January 10th were used for development. The usage of these articles is the same as in (Uchimoto et al., 1999; Sekine et al., 2000; Kudo and Matsumoto, 2002; Sassano, 2004). The size of the training data set is given in Table 1.

4.2 Parameters

We selected the best value of the misclassification cost C of SVM by using the development test set. We carried out training SVM with 0.0001, 0.001, 0.01, 0.1, and 1 as a value of C and measured accuracy on the development test set. We then used models with these best values of C on the test set. Similarly, as to the best value of the number of epoch T of the voted perceptron, we applied the same procedure and found the best value of T for the development test set.

We use polynomial kernels with the degree of 3 for all the experiments. The main reason for this is as follows. Polynomial kernels with the degree of 3 have been widely used for Japanese analysis tasks and they have reported better performance than that of other kernels. A cubic kernel would be a good first choice. In particular, cubic kernels are used for Japanese dependency parsing in many papers (e.g., (Kudo and Matsumoto, 2000; Kudo and Matsumoto, 2002; Sassano, 2004)). Thus there are additional benefits that we can compare our results with others.

4.3 Accuracy

We show in Table 2 the summary of performance for the three tasks with p-values of McNemar's test (Gillick and Cox, 1989) at .05 significance level. SVM outperforms the voted perceptron[2] in all the three tasks. In the cases of both JWS and JBC, the differences between SVM and the voted perceptron are statistically significant. However, in the case of JDP the difference is not significant. Since absolute differences are little in any of the cases, there would be no serious impact in many practical applications whichever you may choose.

Figures 3, 4 and 5 show how accuracy changes at each epoch. In the cases of JWS and JBC, the accuracy peaks at the epoch of around 15. On the other hand, in the case of JDP, the accuracy peaks at the epoch of 2 and then it fluctuates a little and declines gradually.

[2]We actually used "averaging" instead of voting for simplicity. That is, we used in Figure 2 $s = \sum_{i=1}^{k} c_i \, (\boldsymbol{v}_i \cdot \boldsymbol{x})$ instead of $s = \sum_{i=1}^{k} c_i \, \mathrm{sign}(\boldsymbol{v}_i \cdot \boldsymbol{x})$.

Table 2: Performance Summary

	JWS	JBC	JDP
SVM	**98.48%**	**99.69%**	**88.72%**
	$(C = 0.01)$	$(C = 1)$	$(C = 0.001)$
VP	98.32%	99.63%	88.48%
	$(T = 30)$	$(T = 6)$	$(T = 12)$
p-value	1.49e-06	0.021	0.12

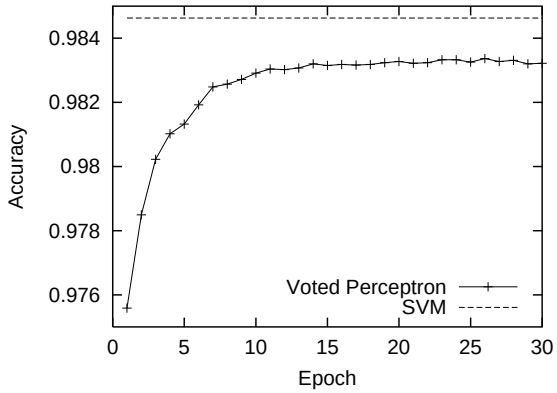

Figure 3: Accuracy of Japanese Word Segmentation

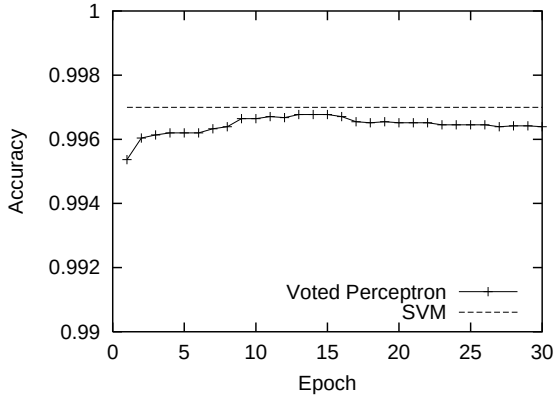

Figure 4: Accuracy of Bunsetsu Chunking

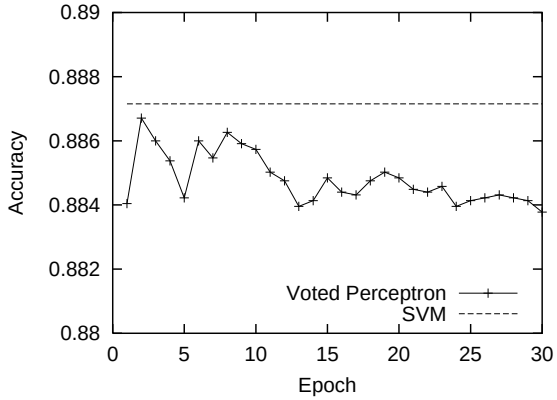

Figure 5: Accuracy of Dependency Analysis

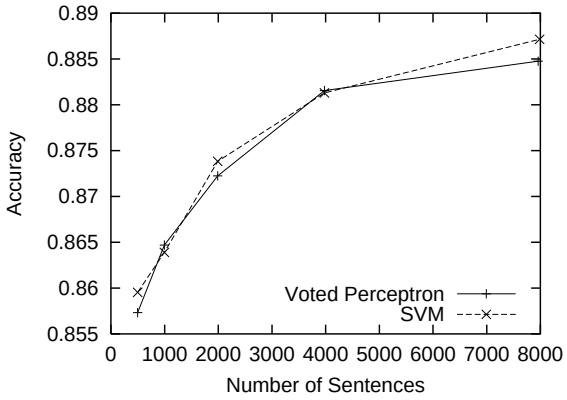

Figure 6: Learning Curves of Dependency Analysis

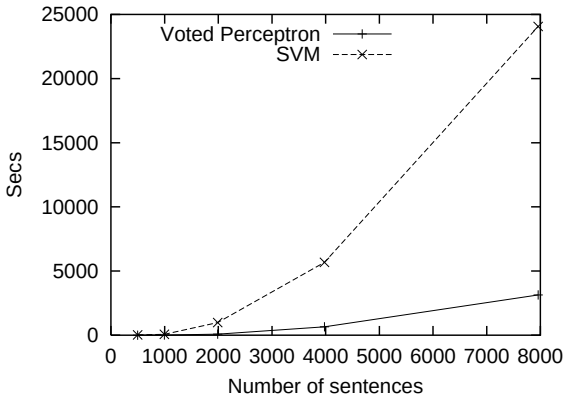

Figure 7: Learning Time

4.4 Learning Curves

We here show the learning curves (Figure 6) of SVM and the voted perceptron for the JDP task. Both curves exhibit a similar shape.

4.5 Learning Time

Now let us see the learning time of SVM and the voted perceptron. We examined the learning time[3] (Figure 7) of the JDP task.

We used LIBSVM (Chang and Lin, 2001) for SVM and an original tool written in C++ for the voted perceptron. LIBSVM used 300MB memory for kernel caching, while our tool for the voted perceptron used no extra memory. The learning time of the voted perceptron is more than five times faster than that of SVM although the tool for the voted perceptron requires less memory.

[3]Executed on FreeBSD with Pentium III 1.4GHz and 3GB memory.

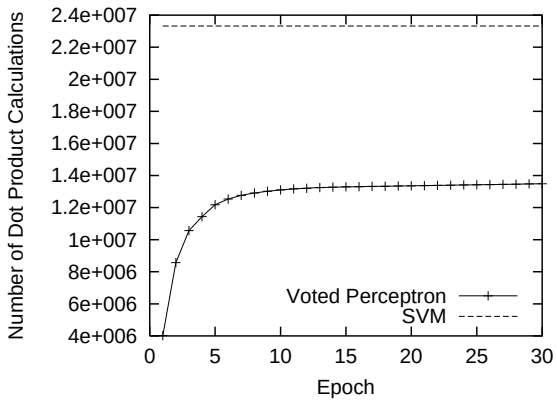

Figure 8: Number of dot product calculations in the training phase of JDP with 498 sentences.

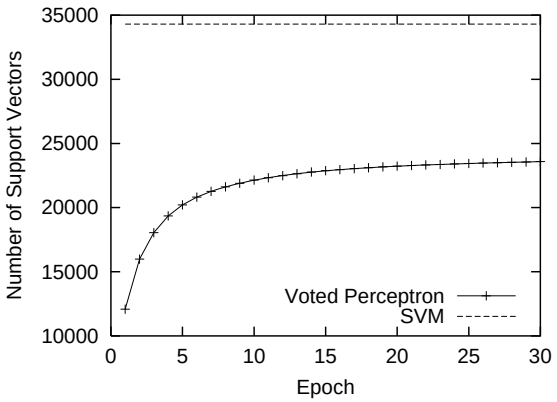

Figure 9: Number of Support Vectors

4.6 Number of Dot Product Calculations

Learning time somewhat may be affected by the implementation details of these tools. Therefore, we counted the number of dot product computation, which directly indicates the learning cost. Figure 8 shows the number of dot product calculations in the case of training of JDP with 498 sentences. The voted perceptron requires considerably fewer calculations of dot products than SVM does. This means the learning of the voted perceptron can be much faster than that of SVM.

4.7 Number of Support Vectors

We also measured the number of support vectors in models of both SVM and the voted perceptron[4]. Figure 9 shows the change of support vectors of the voted perceptron in the case of JDP with the full training data depending on the number of epochs. As (Freund and Schapire, 1999) pointed out, the number of support vectors of the voted perceptron is significantly fewer than that of SVM. This leads to faster prediction of the voted perceptron.

5 Related Work

Carreras et al. (2003) uses a voted perceptron for named entity recognition (NER). However, they have not compared their results with systems using SVMs. Therefore, it is not clear that the NER system with the voted perceptron has any advantages over NER systems with SVM.

Collins' work (Collins, 2002b; Collins and Duffy, 2002; Collins, 2002a) on the voted perceptron focuses mainly on ranking tasks in various problems. It does not treat classification tasks directly and there is no comparison with SVM.

6 Conclusion

In this paper we have compared SVM with the voted perceptron in three tasks of Japanese analysis. In our experiments we have observed that the voted perceptron is comparable to SVM in terms of accuracy and, in addition, as to learning time and prediction speed the voted perceptron is considerably better than SVM. These observations are consistent with the theoretical analysis and experimental results in (Freund and Schapire, 1999).

The voted perceptron is found to be a strong alternative to SVM in classification tasks in NLP as well as ranking tasks. If you choose SVM eventually, the voted perceptron would be very useful when designing a kernel because the same kernel function can be used in both SVM and the voted perceptron and you can obtain benefits from the easiness of implementation and the learning speed of the voted perceptron.

We have a plan to apply the voted perceptron to text classification and other diverse tasks in NLP. We would like to report experimental results and clear the effectiveness and the weakness of the voted perceptron.

[4]We use the term "support vectors" for the voted perceptron as well as SVM. "Support vectors" of the voted perceptron means vectors which are selected in the training phase and contribute to the prediction. Note that the number of support vectors in Figure 9 is fewer than the number of prediction mistakes in training of the voted perceptron.

References

Xavier Carreras, Lluís Màrquez, and Lluís Padró. 2003. Learning a perceptron-based named entity chunker via online recognition feedback. In *Proc. of the CoNLL 2003*, pages 156 – 159.

Chih-Chung Chang and Chih-Jen Lin, 2001. *LIBSVM: a library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proc. of ACL-2002*, pages 263–270.

Michael Collins. 2002a. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP-2002*, pages 1–8.

Michael Collins. 2002b. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *Proc. of ACL-2002*, pages 489–496.

Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277 – 296.

L. Gillick and Stephen Cox. 1989. Some statistical issues in the comparison of speech recognition algorithms. In *Proc. of ICASSP-89*, volume 1, pages 532 – 535.

Ralf Herbrich. 2002. *Learning Kernel Classifiers*. The MIT Press.

Taku Kudo and Yuji Matsumoto. 2000. Japanese dependency structure analysis based on support vector machines. In *Proc. of EMNLP/VLC 2000*, pages 18–25.

Taku Kudo and Yuji Matsumoto. 2001. Chunking with support vector machines. In *Proc. of NAACL 2001*, pages 192–199.

Taku Kudo and Yuji Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In *Proc. of CoNLL-2002*, pages 63–69.

Sadao Kurohashi and Makoto Nagao. 1998. Building a Japanese parsed corpus while improving the parsing system. In *Proc. of the 1st LREC*, pages 719–724.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proc. of IWPT-03*, pages 149–160.

Lance A. Ramshaw and Mitchell P. Marcus. 1995. Text chunking using transformation-based learning. In *Proc. of VLC 1995*, pages 82–94.

Frank Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407.

Manabu Sassano. 2004. Linear-time dependency analysis for Japanese. In *Proc. of COLING 2004*, pages 8–14.

Satoshi Sekine, Kiyotaka Uchimoto, and Hitoshi Isahara. 2000. Backward beam search algorithm for dependency analysis of Japanese. In *Proc. of COLING-00*, pages 754–760.

Hiroyuki Shinnou. 2000. Deterministic Japanese word segmentation by decision list method. In *Proc. of PRICAI-2000*, page 822.

Kiyotaka Uchimoto, Satoshi Sekine, and Hitoshi Isahara. 1999. Japanese dependency structure analysis based on maximum entropy models. In *Proc. of EACL-99*, pages 196–203.

Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag.