# Fast and Accurate Part–of–Speech Tagging: The SVM Approach Revisited

**Jesús Giménez** and **Lluís Màrquez**
TALP Research Center, LSI Department
Universitat Politècnica de Catalunya
Jordi Girona Salgado 1–3, E-08034, Barcelona
{jgimenez,lluism}@lsi.upc.es

## Abstract

In this paper we present a very simple and effective part–of–speech tagger based on Support Vector Machines (SVM). Simplicity and efficiency are achieved by working with linear separators in the primal formulation of SVM, and by using a greedy left-to-right tagging scheme. By means of a rigorous experimental evaluation, we conclude that the proposed SVM–based tagger is robust and flexible for feature modelling (including lexicalization), trains efficiently with almost no parameters to tune, and is able to tag thousands of words per second, which makes it really practical for real NLP applications. Regarding accuracy, the SVM–based tagger significantly outperforms the TnT tagger exactly under the same conditions, and achieves a very competitive accuracy of 97.0% on the WSJ corpus, which is comparable to the best taggers reported up to date.

## 1 Introduction

Automatic part–of–speech (POS) tagging is the task of determining the morphosyntactic category of each word in a given sentence. It is a very well–known problem that has been addressed by many researchers at least for the two last decades. It is a *fundamental* problem in the sense that almost all NLP applications need some kind of POS tagging previous to construct more complex analysis and it is permanently *on–fashion* since current applications demand an efficient treatment of more and more quantities of (possibly multilingual) text.

In the recent literature, we can find several approaches to POS tagging based on statistical and machine learning techniques, including among many others: Hidden Markov Models (Weischedel *et al.* 93; Brants 00), Maximum Entropy taggers (Ratnaparkhi 96), Transformation–based learning (Brill 95), Memory–based learning (Daelemans *et al.* 96), Decision Trees (Màrquez & Rodríguez 97), AdaBoost (Abney *et al.* 99), and Support Vector Machines (Nakagawa *et al.* 01). Most of the previous taggers have been evaluated on the English WSJ corpus, using the Penn Treebank set of POS categories and a lexicon constructed directly from the annotated corpus. Although the evaluations were performed with slight variations, there was a wide consensous in the late 90's that the state–of-the–art accuracy for English POS tagging was between 96.4% and 96.7%.

In the recent years, the most succesful and popular taggers in the NLP community have been the HMM–based TnT tagger (Brants 00), the Transformation–based learning (TBL) tagger (Brill 95), and several variants of the Maximum Entropy (ME) approach (Ratnaparkhi 96). In our opinion, TnT is an example of a really practical tagger for NLP applications. It is available to anybody, simple and easy to use, considerably accurate, and extremely efficient, allowing a training from 1 millon word corpora in just a few seconds and tagging thousands of words per second. In the case of TBL and ME approaches, the great success has been due to the flexibility they offer in modelling contextual information, being ME slightly more accurate than TBL.

Far from being considered a closed problem, several researchers tried to improve results on the POS tagging task during last years. Some of them by allowing richer and more complex HMM models (Thede & Harper 99; Lee *et al.* 00), others by enriching the feature set in a ME tagger (Toutanova & Manning 00), and others by using more effective learning techniques: SVM (Nakagawa *et al.* 01), and a Voted–Perceptron–based training of a ME model (Collins 02). In these more complex taggers the state–of–the–art accuracy was raised up to 96.9%–97.1% on the same WSJ corpus. In a complementary direction, other researchers suggested the combination of several pre-existing taggers under several alternative voting schemes (Brill & Wu 98; Halteren *et al.* 98; Màrquez *et al.* 99). Although the accuracy of these taggers is even better (around 97.2%) the ensembles of POS taggers are undeniably more complex and less efficient.

In this paper we suggest to go back to the *TnT*

*philosophy* (i.e., simplicity and efficiency with state–of–the–art accuracy) but within the SVM learning framework. We claim that the SVM–based tagger introduced in this work fulfills the requirements for being a practical tagger and offers a very good balance of the following properties. (1) *Simplicity*: the tagger is easy to use and has few parameters to tune; (2) *Flexibility* and *robustness*: rich context features can be efficiently handled without overfitting problems, allowing lexicalization; (3) *High accuracy*: the SVM–based tagger performs significantly better than TnT and achieves an accuracy competitive to the best current taggers; (4) *Efficiency*: training on the WSJ is performed in around one CPU hour and the tagging speed allows a massive processing of texts.

It is worth noting that the Support Vector Machines (SVM) paradigm has been already applied to tagging in a previous paper (Nakagawa *et al.* 01), with the focus on the guessing of unknown word categories. The final tagger constructed in that paper gave a clear evidence that the SVM approach is specially appropriate for the second and third of the previous points, the main drawback being a low efficiency (in that paper a running speed of around 20 words per second is reported). In the present paper we overcome this limitation by working with linear kernels in the primal setting of the SVM framework taking advantage of the extremely sparsity of example vectors. The resulting tagger is almost as accurate as that of (Nakagawa *et al.* 01) but 60 times faster in a preliminar prototype implemented in Perl.

The rest of the paper is organized as follows: In section 2, the formal SVM learning setting is presented. Section 3 is devoted to explain the details of our approach to tagging. Section 4 describes the experimental work carried out in order to validate the presented SVM tagger. Section 5 includes some discussion on the presented approach in comparison to other related work, and, finally, section 6 concludes and outlines the directions of the future research.

## 2 Support Vector Machines

SVM is a machine learning algorithm for binary classification, which has been successfully applied to a number of practical problems, including NLP (Cristianini & Shawe-Taylor 00).

Let $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be the set of $N$ training examples, where each instance $\mathbf{x}_i$ is a vector in $\mathbb{R}^N$ and $y_i \in \{-1, +1\}$ is the class label. In their basic form, a SVM learns a linear hyperplane that separates the set of positive examples from the set of negative examples with *maximal margin* (the margin is defined as the distance of the hyperplane to the nearest of the positive and negative examples). This learning bias has proved to have good properties in terms of generalization bounds for the induced classifiers.

The linear separator is defined by two elements: a weight vector $\mathbf{w}$ (with one component for each feature), and a bias $b$ which stands for the distance of the hyperplane to the origin. The classification rule of a SVM is $sgn(f(\mathbf{x}, \mathbf{w}, b))$, where $f(\mathbf{x}, \mathbf{w}, b) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$ and $\mathbf{x}$ is the example to be classified. In the linearly separable case, learning the maximal margin hyperplane $(\mathbf{w}, b)$ can be stated as a convex quadratic optimization problem with a unique solution: *minimize* $||\mathbf{w}||$, *subject to the constraints* (one for each training example): $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1$.

The SVM model has an equivalent dual formulation, characterized by a weight vector $\boldsymbol{\alpha}$ and a bias $b$. In this case, $\boldsymbol{\alpha}$ contains one weight for each training vector, indicating the importance of this vector in the solution. Vectors with non null weights are called *support vectors*. The dual classification rule is: $f(\mathbf{x}, \boldsymbol{\alpha}, b) = \sum_{i=1}^{N} y_i \alpha_i \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b$, and the $\boldsymbol{\alpha}$ vector can be calculated also as a quadratic optimization problem. Given the optimal $\boldsymbol{\alpha}^*$ vector of the dual quadratic optimization problem, the weight vector $\mathbf{w}^*$ that realizes the maximal margin hyperplane is calculated as:

$$\mathbf{w}^* = \sum_{i=1}^{N} y_i \alpha_i^* \mathbf{x_i} \qquad (1)$$

The $b^*$ has also a simple expression in terms of $\mathbf{w}^*$ and the training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. See (Cristianini & Shawe-Taylor 00) for details.

The advantage of the dual formulation is that permits an efficient learning of non–linear SVM separators, by introducing *kernel functions*. Technically, a kernel function calculates a dot product between two vectors that have been (non linearly) mapped into a high dimensional feature space. Since there is no need to perform this mapping explicitly, the training is still feasible although the dimension of the real feature space can be very high or even infinite.

In the presence of outliers and wrongly classified training examples it may be useful to allow

some training errors in order to avoid overfitting. This is achieved by a variant of the optimization problem, referred to as *soft margin*, in which the contribution to the objective function of margin maximization and training errors can be balanced through the use of a parameter called $C$.

## 3 Problem Setting

In this section the details about our approach to POS tagging, regarding the collection and feature codification of training examples, are presented.

### 3.1 Binarizing the Classification Problem

Tagging a word in context is a multi-class classification problem. Since SVMs are binary classifiers, a binarization of the problem must be performed before applying them. We have applied a simple *one-per-class* binarization, i.e., a SVM is trained for every part-of-speech in order to distinguish between examples of this class and all the rest. When tagging a word, the most confident tag according to the predictions of all binary SVMs is selected.

However, not all training examples have been considered for all classes. Instead, a dictionary is extracted from the training corpus with all possible tags for each word, and when considering the occurrence of a training word $w$ tagged as $t_i$, this example is used as a positive example for class $t_i$ and a negative example for all other $t_j$ classes appearing as possible tags for $w$ in the dictionary. In this way, we avoid the generation of excessive (and irrelevant) negative examples, and we make the training step faster[1]. In the following sections we will see how a 635,000 word corpus generates training sets of about 20,000 examples on average, instead of 635,000.

### 3.2 Feature Codification

Each example has been codified on the basis of the *local context* of the word to be disambiguated. We have considered a centered window of seven tokens, in which some basic and $n$–gram patterns are evaluated to form binary features such as: "previous_word_is_`the`", "two_preceeding_tags_are_`DT_NN`", etc. Table 1 contains the list of all patterns considered.

As it can be seen, the tagger is lexicalized and all word forms appearing in window are taken into account. Since a very simple left–to–right tagging scheme will be used, the tags of the following words are not known at running time. Following the approach of (Daelemans *et al.* 96) we use the more general *ambiguity–class* tag for the right context words, which is a label composed by the concatenation of all possible tags for the word (e.g., `IN-RB`, `JJ-NN`, etc.). Each of the individual tags of an ambiguity class is also taken as a binary feature of the form "following_word_**may_be**_a_VBZ". Therefore, with ambiguity classes and "maybe's", we avoid the *two passes* solution proposed in (Nakagawa *et al.* 01), in which a first tagging is performed in order to have right contexts disambiguated for the second pass. Also in (Nakagawa *et al.* 01), it is suggested that explicit $n$–gram features are not necessary in the SVM approach, because polynomial kernels acoount for the combination of features. However, since we are interested in working with a linear kernel, we have included them in the feature set. In section 4.1 we will evaluate the importance of this kind of features.

| | |
|---|---|
| word features | $w_{-3}, w_{-2}, w_{-1}, w_0, w_{+1}, w_{+2}, w_{+3}$ |
| POS features | $p_{-3}, p_{-2}, p_{-1}$ |
| ambiguity classes | $a_0, a_1, a_2, a_3$ |
| maybe's | $m_0, m_1, m_2, m_3$ |
| word bigrams | $(w_{-2}, w_{-1}), (w_{-1}, w_{+1}), (w_{-1}, w_0)$ $(w_0, w_{+1}), (w_{+1}, w_{+2})$ |
| POS bigrams | $(p_{-2}, p_{-1}), (p_{-1}, a_{+1}), (a_{+1}, a_{+2})$ |
| word trigrams | $(w_{-3}, w_{-2}, w_{-1}), (w_{-2}, w_{-1}, w_0),$ $(w_{-2}, w_{-1}, w_{+1}), (w_{-1}, w_0, w_{+1}),$ $(w_{-1}, w_{+1}, w_{+2}), (w_0, w_{+1}, w_{+2})$ |
| POS trigrams | $(p_{-3}, p_{-2}, p_{-1}), (p_{-2}, p_{-1}, a_{+1}),$ $(p_{-1}, a_{+1}, a_{+2})$ |

Table 1: Feature patterns used to codify examples.

## 4 Experiments

This section presents the experiments carried out in order to evaluate the SVM approach to POS tagging. As in many other works, the Wall Street Journal data from the Penn Treebank III has been used as the benchmark corpus. We have randomly divided (at a sentence level) this 1,17 million word corpus into three subsets: training (60%, 635,138 words), validation (20%, 271,933 words, and test (20%, 266,695 words). All the tagging experiments reported are evaluated on the complete test set. The validation set has been used to optimize parameters. The Penn Treebank tagset contains 48 tags. However, after compiling training examples in the way explained in section 3.1, only 34 of them receive positive and negative examples.

[1] See (Abney *et al.* 99) for a discussion on the efficiency problems when learning from large POS training sets.

| model | acc. | #sv | l_time |
|---|---|---|---|
| $d = 1$, $fs_1$ | 93.50% | 3,472.47 | 44':48" |
| $d = 2$, $fs_1$ | 93.91% | 4,385.56 | 4h:10':19" |
| $d = 3$, $fs_1$ | 93.47% | 6,040.59 | 5h:35':30" |
| $d = 4$, $fs_1$ | 92.78% | 8,196.74 | 7h:23':07" |
| $d = 1$, $fs_2$ | 93.84% | 3,532.44 | 1h:21':14" |
| $d = 2$, $fs_2$ | 93.67% | 4,935.74 | 5h:38':48" |

Table 2: Accuracy results (on ambiguous words) of alternative SVM models varying kernel degree and feature set. '#sv' stands for the average number of support vectors per tag and 'l_time' is the CPU time needed for the whole training.

Thus, only 34 SVM classifiers have to be trained in the binarized setting. The 14 *unambiguous* tags correspond to punctuation marks, symbols, and the categories `TO` and `WP$`.

### 4.1 Linear vs. Polynomial Kernels

The first experiment explores the effect of the kernel in the training process and in the generalization accuracy. So as to do that, we have trained several SVM classification models with the whole 635k word training set by varying the degree ($d$) of the polynomial kernel. The software package used in all the experiments reported was SVM$^{light}$.[2] A very simple frequency threshold (common to all the experiments) was used to filter out unfrequent features. In particular, we have discarded features that occur less than $n$ times, where $n \geq 2$ is the minimum number so that the total amount of features is not greater than 100,000.[3] When training, the setting of the $C$ parameter has been left to its default value. In the following subsections we will see how the optimization of this parameter leads to a small improvement in the final tagger, being the SVM algorithm quite robust with respect to parameterization.

Results obtained when classifying the ambiguous words in the test set are presented in Table 2. This test has been performed in a *batch* mode, that is, examples of ambiguous words are taken separately, and the features involving the POS tags of the left contexts are calculated using the *correct* POS tag assignment of the corpus.

[2] The SVM$^{light}$ software is freely available at the following URL: `http://svmlight.joachims.org`.

[3] More complex methods can be used to filter out irrelevant features. However, the feature selection problem is beyond the scope of this work, in which the simplest alternatives are preferred.

Regarding accuracy, similar results to those of (Nakagawa *et al.* 01) can be drawn. When using the set of atomic features ($fs_1$), the best results are obtained with a degree 2 polynomial kernel (93.91%). Greater degrees produce overfitting to the training data, since the number of support vectors highly increases and the accuracy decreases. When using the $n$-gram extended set of features ($fs_2$), the linear kernel becomes very competitive (93.84%) compared to the degree 2 polynomial kernel, being clearly preferable regarding the sparsity of the solution and the learning time (3.1 times faster). Interestingly, the advantage of the extended set of features is only noticeable in the case of the linear kernel, since accuracy decreases when it is used with the polynomial kernels.

As a conclusion, we can state that a linear kernel, with an $n$–gram based set of basic features, suffices to obtain highly accurate SVM models being relatively fast to train. In the next section we will see how the linear solution has the additional advantage of allowing to work in the primal setting with a very sparse vector of weights. This fact is crucial to obtain a fast POS tagger.

### 4.2 Evaluating the SVM Tagger

Hereafter we will focus only on the linear SVM model. In this experiment, the POS tagger is tested in a more realistic situation, that is, performing a left–to–right tagging of the sequence of words, with an *on-line* calculation of features making use of the already assigned left-context POS tags. Following the simplicity and efficiency principles, a greedy left–to–right tagging scheme is applied, in which no optimization of the tag sequence is performed at the sentence level. We have implemented this first POS tagger prototype in Perl-5.0, which will be referred to as SVM*tagger*. The SVM dual representation based on a set of support vectors output by SVM$^{light}$ is converted into the primal form, $(\mathbf{w}, b)$, using equation 1 explained in section 2.

The tagger has been tested under the *closed vocabulary assumption*, in which no unknown words are allowed. This is simulated by directly including in the dictionary the words of the test set that do not occur in the training set. The results obtained by increasing sizes of the training set are presented in table 3. Learning and tagging times are also graphically presented in Figure 1. All the experiments were performed (under Linux) with

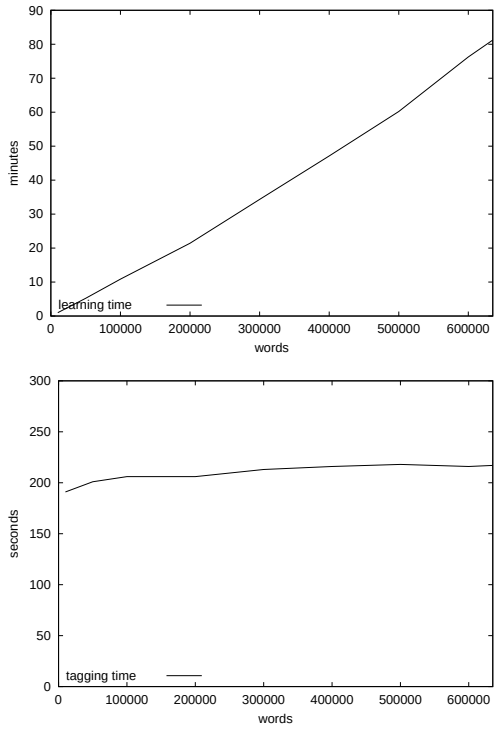


Figure 1: Learning and tagging time plots of SVM*tagger* by increasing sizes of the training set.

a 2Ghz Pentium-IV processor with 1Gb of RAM. The time figures have been calculated using the Benchmark package of Perl, and reflect CPU time.

As it could be expected, the accuracy of the tagger grows with the size of the training set, presenting a logarithmic behaviour. Regarding efficiency, it can be observed that training time is almost linear with respect to the number of examples of the training set. Besides, the compression of the SVM model with respect to the training set also increases with the training set size, and goes from 68.17% (from 1572.1 examples to 500.4 support vectors) to 82.18% (from 19,951.6 examples to 3,556.1 support vectors). However, this compression level would not permit an efficient tagger in the dual form, since thousands of dot products would be needed to classify each word. More interestingly, the model in primal form is quite compact since the weight vectors, resulting from compacting all support vectors, contains no more than 11,000 features (among the 90,000 possible features). Provided that the test examples are very sparse (they contain about 38 features in average, irrespective of the training set size), the classification rule is very efficient, since a single dot product with a sparse vector is needed to classify each word. Basing this dot product on the non null dimensions of the example to classify, the tagging time can be mantained almost invariant. In particular, the tagging speed corresponding to the 635kw is about 1,230 words per second.

By optimizing the $C$ parameter of the SVM algorithm, i.e., the tradeoff between training error and margin maximization, slightly better results can be obtained. We have optimized the $C$ parameter on the validation set (by maximizing accuracy) [4]. With this value properly set the accuracy results obtained by SVM*tagger* (using the whole training set) were 94.35% on ambiguous words and 97.74% overall (an increase of 0.18 points). To have an idea of the quality of these values, we also run TnT exactly on the same conditions (including unknown words in the backup lexicon) and the results obtained were significantly lower: 92.64% for ambiguous words and 97.27% overall.

### 4.3 Including unknown words

The tagger results presented in the previous section are still not realistic, since we can not assume a closed vocabulary. In order to deal with this problem we have developed an SVM–based model to recognize unknown words. Unknown words are treated as ambiguous words with all possible POS tags corresponding to open–class words (18 in the Penn Treebank tagset), but specialized SVMs are learned with particular features to disambiguate among the POS tags of unknown words. The approach is similar to that of (Nakagawa *et al.* 01) and the features used, which are presented in table 4, are taken from the following works (Brill 95; Màrquez *et al.* 99; Nakagawa *et al.* 01).

Training examples for unknown words have been collected from the training set in the following way: First, the training corpus is randomly divided into twenty parts of equal size. Then, the first part is used to extract the examples which do not occur in the remaining nineteen parts, that is, taking the 95% of the corpus as known and the re-

[4] The tunning of the $C$ parameter is also automatically done by iteratively exploring shorter and shorter intervals of values in which the best accuracies are observed in the validation set. The setting of our algorithm involves five parameters: $minC, maxC, log, n\_iters, n\_segments$. The first two ones, $minC$ and $maxC$, determine the overall interval to examine. If $log$ is true the first iteration is approached logarithmically. The last two arguments, $n\_iters$ and $n\_segments$, stand for the total number of iterations and the number of intervals that must be explored at each iteration, respectively. In our experiments the combination (0.01, 1, true, 3, 5) was used.

| #words | amb. | all | #exs | #feat. | #sv | #$\mathbf{x}_i$ | #$\mathbf{w}$ | l_time | t_time |
|---|---|---|---|---|---|---|---|---|---|
| 50k | 90.53% | 96.27% | 1,572.1 | 38,613 | 500.4 | 38.00 | 3,339.4 | 05':12" | 3':21" |
| 100k | 91.90% | 96.81% | 3,141.8 | 72,030 | 879.8 | 38.03 | 5,552.2 | 10':52" | 3':26" |
| 200k | 92.71% | 97.13% | 6,237.8 | 50,699 | 1,424.4 | 38.02 | 5,911.0 | 21':26" | 3':26" |
| 300k | 93.15% | 97.29% | 9,371.4 | 72,988 | 1,959.0 | 38.01 | 7,833.8 | 34':16" | 3':33" |
| 400k | 93.45% | 97.40% | 12,501.9 | 93,660 | 2,490.1 | 38.02 | 9,669.2 | 47':07" | 3':36" |
| 500k | 93.67% | 97.48% | 15,672.4 | 89,179 | 2,935.4 | 38.04 | 10,102.8 | 1h:00':12" | 3':38" |
| 600k | 93.74% | 97.52% | 17,875.2 | 86,273 | 3,218.9 | 38.04 | 10,297.5 | 1h:16':15" | 3':36" |
| 635k | 93.84% | 97.56% | 19,951.6 | 90,517 | 3,556.1 | 38.04 | 11,041.6 | 1h:21':14" | 3':37" |

Table 3: Accuracy results of SVM*tagger* under the closed vocabulary assumption by increasing sizes of the training corpus. 'amb.' and 'all' columns contains accuracy achieved on ambiguous words and overall, respectively. '#exs' and '#sv' stand for the average number of examples and support vectors per POS tag, '#feat.' for the total number of binary features (after filtering), '#$\mathbf{x}_i$' for the average number of active features in the training examples, and '#$\mathbf{w}$' for the average number of dimensions of the weight vectors. 'l_time' and 't_time' refer to learning and tagging time.

| | |
|---|---|
| All features for known words | (see table 1) |
| prefixes | $s_1$, $s_1s_2$, $s_1s_2s_3$, $s_1s_2s_3s_4$ |
| suffixes | $s_n$, $s_{n-1}s_n$, $s_{n-2}s_{n-1}s_n$, $s_{n-3}s_{n-2}s_{n-1}s_n$ |
| begins with Upper Case | yes/no |
| all Upper Case | yes/no |
| all Lower Case | yes/no |
| contains a Capital Letter not at the beginning | yes/no |
| contains more than one Capital Letter not at the beginning | yes/no |
| contains a period | yes/no |
| contains a number | yes/no |
| contains a hyphen | yes/no |
| word length | integer |

Table 4: Feature templates for unknown words

| | amb. | known | unk. | all |
|---|---|---|---|---|
| TnT | 92.2% | 96.9% | 84.6% | 96.5% |
| SVM*tagger* | 93.6% | 97.2% | 83.5% | 96.9% |
| SVM*tagger*+ | 94.1% | 97.3% | 83.6% | 97.0% |

Table 5: Accuracy results of SVM*tagger* compared to TnT, under the open vocabulary assumption. 'known' and 'unk.' refer to the subsets of known and unknown words, respectively, 'amb' to the subset of ambiguous known words, and 'all' to the overall accuracy.

maining 5% to extract the examples. This procedure is repeated with each of the twenty parts, obtaining approximately 20,700 examples from the whole corpus. The choice of dividing by twenty is not arbitrary. 95%–5% is the proportion that results in a percentage of unknown words very similar to the test set (2.91%).

Results obtained are presented in table 5. The label SVM*tagger*+ corresponds to the case in which the $C$ parameter has been optimized [5].

Similar to the previous section the results obtained by the SVM*tagger* clearly outperform the results of TnT tagger and they are comparable to the accuracy of the best current taggers (which range from 96.9% to 97.1%). Again, the tunning of the $C$ parameter provides a small increment of performance, but at a cost of increasing the training time to almost 20 CPU hours.

Following a suggestion by one of the referees, experiments were replicated on a different partition of the Wall Street Journal corpus in order to compare our work to some other related previous ones. Sections 0-18 were used for training, 19-21 for validation, and 22-24 for test, respectively. The tunning of the C parameter lead the system to achieve a token accuracy of 97.05%, significantly outperforming TnT (96,48%). Result is competitive to the one reported in (Collins 02) (97.11%), although still a little lower than the one found in (Toutanova *et al.* 03) (97.24%), see further details in Section 5.

The Perl implementation of SVM*tagger*+ model achieves a tagging speed of 1,335 words per second. Given the type of operations computed by the tagging algorithm we fairly believe that a re-implementation in C++ could speed up the tagger, making the efficiency valid for massive text processing. Of course, the TnT tagger is still much more efficient, achieving a tagging speed of more than 50,000 words per second on the same conditions.

[5] The C parameter value is 0.109 for known words and 0.096 for unknown words.

## 5 Discussion

The work presented in this paper is closely related to (Nakagawa *et al.* 01). In that paper, an SVM–based tagger is presented and compared to TnT, obtaining a best accuracy of 97.1% (when training from a 1 million word corpus). Apart from the training set size, the main differences of both approaches are explained below.

Nakagawa's work is focused on tagging unknown words. A certainly *ad–hoc* procedure is performed to tag the word sequence in two passes. In the first pass, the tagger disambiguates the whole sentence and in the second pass, the previously assigned tags are assumed correct in order to extract the right–context tag features. This tagging overhead is also projected into training, since two versions, with and without right–context tag features, must be trained. Additionally, it is argued that one advantage of using SVM is that it is not necessary to codify complex $n$–gram features, since polynomial kernels themselves succeed at doing this task. Thus, they base their best tagger in a dual solution with degree 2 polynomial kernels, instead of a linear separator in the primal setting.

Since they are using kernels and SVM classification in the dual setting, the tagger simply can not be fast at running time. In particular, a tagging speed of 19.80 words per second (4 hours needed to tag a 285k word test set) is reported. Training is also much more slower than ours (the time required for training from a 100k word set is about 16.5 hours), probably due to the way in which they select training examples for each POS.

By the time we were preparing this document another paper was brought to our attention, which reports the best results on the WSJ corpus to date with a single tagger. We refer to (Toutanova *et al.* 03), in which a tagger based on a cyclic dependency network is presented. This tagger achieves accuracy values between 97.15% and 97.24% in the WSJ corpus (with a training set of 912k words), allows to explicitly model left and right context features in the sequence tagging scheme, and it is lexicalized. Overfitting in the extremely large feature spaces induced is avoided by model regularization.

The good results of the tagger are partly due to the outstanding recognition of unknown words (accuracy values between 88.61% and 89.04%). However, the treatment of unknown words is a bit tricky, including the use of a company named entity detector and several ad-hoc feature patterns based on the observation of the errors commited by the tagger. This fact may cause the POS tagger to be highly WSJ–dependent. Regarding efficiency, few comments are included in the paper. The training time for the best model is said to be about 56 hours on a 2GHz processor (134 iterations at 25 minutes per iteration), while tagging times are not reported.

## 6 Conclusions and Future Work

In this work we have presented a SVM–based POS tagger suitable for real applications, since it provides a very good balance of several good properties for NLP tools: simplicity, flexibility, high performance, and efficiency. The next step we plan to do is to re-implement the tagger in C++ to significantly increase efficiency and to provide a software package for public use[6].

Regarding the study of the SVM–approach to POS tagging some issues deserve further investigation. First, the learning model for unknown words experimented in this paper is preliminar and we think that can be clearly improved. Second, we have applied only the simplest greedy left–to–right tagging scheme. Since SVM predictions can be converted into probabilities, a natural extension would be to consider a sentence–level tagging model in which the probability of the whole sentence assignment is maximized.

Finally, we are exploring the possibility of simplifying the models by a posteriori feature filtering on the weight vector ($\mathbf{w}$). That simplification incides collaterally on the tagging speed and accuracy. First experiments on the models trained on 912k words indicate that eliminating the features with lower weights does not hurt the performance very much, see Figure 2. Indeed, a very little improvement is obtained discarding between 40% and 50% of the $\mathbf{w}$ dimensions. But the size of the models may be still further reduced. In particular, a very competitive overall accuracy of 97.01% can be still obtained discarding up to 70% of the $\mathbf{w}$ dimensions. And, very interestingly, it is not until we discard over 99% of the $\mathbf{w}$ dimensions that accuracy falls down 96%. These observations hold for both the test and validation sets,

[6] By now, the prototype version of the tagger is public for demostration at the following Web address: `www.lsi.upc.es/~nlp/SVMtagger.html`.

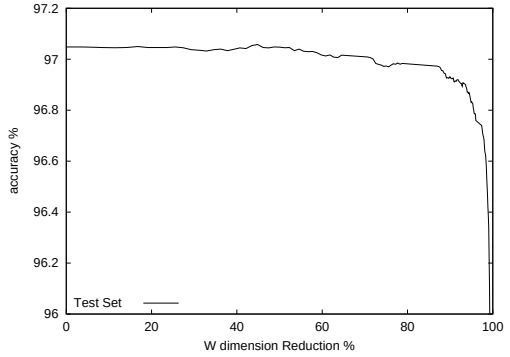


Figure 2: Token Accuracy behaviour for the test set when reducing the size of the weight vector ($\mathbf{w}$) dimensions by discarding those weights closer to zero.

and surely are opening the avenue for a further increasing on the tagging speed.

## Acknowledgements

The authors want to thank the anonymous reviewers for their valuable comments and suggestions in order to prepare the final version of the paper.

This research has been partially funded by the Spanish Ministry of Science and Technology (MCyT's projects: HERMES TIC2000-0335-C03-02, ALIADO TIC2002-04447-C02) and by the European Comission (LC-STAR IST-2001-32216), and by the Catalan Research Department (CIRIT's consolidated research group 2001SGR-00254).

## References

(Abney *et al.* 99) S. Abney, R. E. Schapire, and Y. Singer. Boosting applied to tagging and pp–attachment. In *Proceedings of EMNLP/VLC'99*, 1999.

(Brants 00) T. Brants. TnT - A Statistical Part-of-Speech Tagger. In *Proceedings of the Sixth ANLP*, 2000.

(Brill & Wu 98) E. Brill and J. Wu. Classifier Combination for Improved Lexical Disambiguation. In *Proceedings of COLING-ACL'98*, 1998.

(Brill 95) E. Brill. Transformation–based Error–driven Learning and Natural Language Processing: A Case Study in Part–of–speech Tagging. *Computational Linguistics*, 21(4), 1995.

(Collins 02) M. Collins. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the 7th EMNLP Conference*, 2002.

(Cristianini & Shawe-Taylor 00) N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

(Daelemans *et al.* 96) W. Daelemans, J. Zavrel, P. Berck, and S. Gillis. MBT: A Memory–Based Part–of–speech Tagger Generator. In *Proceedings of the 4th Workshop on Very Large Corpora*, 1996.

(Halteren *et al.* 98) H. van Halteren, J. Zavrel, and W. Daelemans. Improving Data Driven Wordclass Tagging by System Combination. In *Proceedings of COLING-ACL'98*, 1998.

(Lee *et al.* 00) S. Lee, J. Tsujii, and H. Rim. Part-of-Speech Tagging Based on Hidden Markov Model Assuming Joint Independence. In *Proceedings of the 38th Annual Meeting of the ACL*, 2000.

(Màrquez & Rodríguez 97) L. Màrquez and H. Rodríguez. Automatically Acquiring a Language Model for POS Tagging Using Decision Trees. In *Proceedings of the Second RANLP Conference*, 1997.

(Màrquez *et al.* 99) L. Màrquez, H. Rodríguez, J. Carmona, and J. Montolio. Improving POS Tagging Using Machine-Learning Techniques. In *Proceedings of EMNLP/VLC'99*, 1999.

(Nakagawa *et al.* 01) T. Nakagawa, T. Kudoh, and Y. Matsumoto. Unknown word guessing and part-of-speech tagging using support vector machines. In *Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium*, 2001.

(Ratnaparkhi 96) A. Ratnaparkhi. A Maximum Entropy Part–of–speech Tagger. In *Proceedings of the 1st EMNLP Conference*, 1996.

(Thede & Harper 99) S. M. Thede and M. P. Harper. A Second-Order Hidden Markov Model for Part-of-Speech Tagging. In *Proceedings of the 37th Annual Meeting of the ACL*, 1999.

(Toutanova & Manning 00) K. Toutanova and C. D. Manning. Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. In *Proceedings of EMNLP/VLC'00*, 2000.

(Toutanova *et al.* 03) K. Toutanova, D. Klein, and C. D. Manning. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL'03*, 2003.

(Weischedel *et al.* 93) R. Weischedel, R. Schwartz, J. Palmucci, M. Meteer, and L. Ramshaw. Coping with Ambiguity and Unknown Words through Probabilistic Models. *Computational Linguistics*, 19(2), 1993.