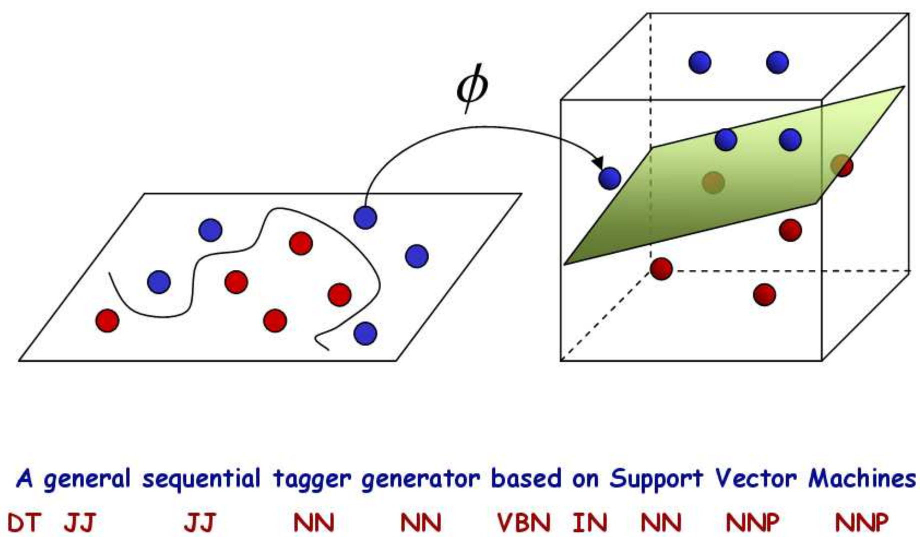
SVMTool



Technical Manual v1.4

Jesús Giménez
Lluís Màrquez

TALP Research Center, LSI Department
Universitat Politécnica de Catalunya
Jordi Girona Salgado 1–3, E-08034, Barcelona
{jgimenez,lluism}@lsi.upc.edu

Latest revision: March 2012

Contents

Abstract

This report presents the SVMTool[1], a simple, flexible, and effective generator of sequential taggers based on Support Vector Machines. We have appied the SVMTool to the problem of part-of-speech tagging. 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 97.2% for English on the Wall Street Journal corpus, which is comparable to the best taggers reported up to date. It has been also successfully applied to Spanish and Catalan exhibiting a similar performance, and to other tagging problems such as chunking. Perl and C++ versions are available[2]

[1]This work has been partially funded by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 247762 (FAUST project, FP7-ICT-2009-4-247762) and by the Spanish Government project OpenMT-2, TIN2009-14675-C03. The original versions of the tool were partially supported by the Spanish Ministry of Science and Technology (HERMES TIC2000-0335-C03-02, ALIADO TIC2002- 04447-C02) and by the European Comission (LC-STAR IST-2001-32216).

[2]The SVMTool is released under LGPL license and may be freely downloaded at `http://www.lsi.upc.es/~nlp/SVMTool/`.

1 Installation

To configure this module, cd to the directory that contains the README file and type the following:

```
perl Makefile.PL
```

Alternatively, if you plan to install SVMTool somewhere other than your system's perl library directory, you can type something like this:

```
perl Makefile.PL PREFIX=/home/me/perl
```

Then to build you run make.

```
make
```

If you have write access to the installation directories, you may then install by typing:

```
make install
```

Remember to properly set 'path' and PERL5LIB variables:

```
set path = ($path /home/me/SVMTool-1.3/bin)
setenv PERL5LIB /home/me/SVMTool-1.3/lib:$PERL5LIB
```

2 Introduction

Prior to elaborate further complex data analyses, most NLP applications demand at initial stages shallow linguistic information (e.g., part–of–speech tagging, base phrase chunking, named entity recognition). This information may be predicted fully automatically (at the cost of some errors) by means of sequential tagging over unannotated raw text.

Generally, tagging is required to be as accurate as possible, and as efficient as possible. But, certainly, there is a trade-off between this two desirable properties. Obtaining a higher accuracy requires processing more and more information, digging deeper and deeper into it. However, sometimes, depending on the kind of application, a loss in efficiency may be acceptable in order to obtain more precise results. Or the other way around, a slight loss in accuracy may be tolerated in favour of tagging speed.

Moreover, some languages have a richer morphology than others, requiring the tagger to have into acount a bigger set of feature patterns. Also the tagset size and ambiguity rate may vary from language to language and from problem to problem. Besides, if few data are available for training, the proportion of unknown words may be huge. Sometimes, morphological analyzers could be utilized to reduce the degree of ambiguity when facing unknown words. Thus, a sequential tagger should be flexible with respect to the amount of information utilized and context shape.

Another very interesting property for sequential taggers is their portability. Multilingual information is a key ingredient in NLP tasks such as Machine Translation, Information Retrieval, Information Extraction, Question Answering and Word Sense Disambiguation, just to name a few. Therefore, having a tagger that works equally well for several languages is crucial for the system robustness.

Besides, quite often for some languages, but also in general, lexical resources are hard to obtain. Therefore, ideally a tagger should be capable for learning with fewer (or even none) annotated data.

The SVMTool is intended to comply with all the requirements of modern NLP technology, by combining simplicity, flexibility, robustness, portability and efficiency with state–of–the–art accuracy. This is achieved by working in the Support Vector Machines (SVM) learning framework, and by offering NLP researchers a highly customizable sequential tagger generator. We have appied the SVMTool to the problem of part-of-speech (PoS) tagging.

2.1 Properties of the SVMTool

The properties the SVMTool is intended to exhibit are:

Simplicty The SVMTool is easy to configure and to train. The learning is controlled by means of a very simple configuration file. There are very few parameters to tune. And the tagger itself is very easy to use, accepting standard input and output pipelining. Embedded usage is also supplied by means of the SVMTool API.

Flexibility The size and shape of the feature context can be adjusted. Also, rich features can be defined, including word and PoS (tag) n-grams as well as ambiguity classes and "may_be's", apart from lexicalized features for unknown words and sentence general information. Moreover, starting from version 1.3 additional information may be provided from multiple columns other than 0 (word) and 1 (tag) in the form of n-grams (See Subsection 4.1). The behaviour at tagging time is also very flexible, allowing different strategies (See Subsection 4.2).

Robustness The overfitting problem is well addressed by tuning the C parameter in the soft margin version of the SVM learning algorithm. Also, a sentence-level analysis may be performed in order to maximize the sentence score. And, for unknown words not to punish so severely on the system effectiveness, several strategies have been implemented and tested.

Portability The SVMTool is language independent. It has been successfully applied to English and Spanish without a priori knowledge other than a supervised corpus. Moreover, thinking of languages for which labeled data is a scarce resource, the SVMTool also may learn from unsupervised[3] data based on the role of non-ambiguous words [Mih03] with the only additional help of a morpho-syntactic dictionary.

Accuracy Compared to state–of–the–art PoS taggers reported up to date, it exhibits a very competitive accuracy (97.2% for English on the WSJ corpus). Clearly, rich sets of features allow to model very precisely most of the information involved. Also the learning paradigm, SVM, is very suitable for working accurately and efficiently with high dimensionality feature spaces.

Efficiency Performance at tagging time depends on the feature set size and the tagging scheme selected. For the default (one-pass left-to-right greedy) tagging scheme, the current Perl prototype

[3]This feature remains still disabled in this version (SVMTool v1.3)

exhibits a tagging speed of 1,500 words/second whereas the C++ version achieves a tagging speed of over 10,000 words/second[4]. This has been achieved by working in the primal formulation of SVM. The use of linear kernels causes the tagger to perform more efficiently both at tagging and learning time, but forces the user to define a richer feature space. However, the learning time remains linear with respect to the number of training examples.

See Section 3 for further details on the application of the SVMTool to PoS Tagging.

[4]Tagging speed has been measured on a Pentium–IV, 2GHz, 1 GB RAM

3 Approach Description

The following sections describe our SVM–based approach to PoS–tagging. Subsection 3.1 relates our approach to previous work on the same task. In Subsection 3.2 the SVM learning paradigm is briefly presented. Subsection 3.3 describes how the problem of PoS–tagging can be seen as a multiclass classification problem for which an SVM learning framework can be set.

3.1 Related Work

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 [WSP$^+$93, Bra00], Maximum Entropy taggers [Rat96], Transformation–based learning [Bri95], Memory–based learning [DZBG96], Decision Trees [MR97], AdaBoost [ASS99], and Support Vector Machines [NKM01]. 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 [Bra00], the Transformation–based learning (TBL) tagger [Bri95], and several variants of the Maximum Entropy (ME) approach [Rat96]. 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 [TH99, LTR00], others [TM00] by enriching the feature set in a ME tagger, and others [NKM01] by using more effective learning techniques: SVM, and a Voted–Perceptron–based training of a ME model [Col02]. 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 [BW98, HZD98, MRCM99]. 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 work 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 [NKM01], 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 work 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 [NKM01] but 60 times faster in a preliminar prototype implemented in Perl and 600 faster in the C++ version.

3.2 The Theory of 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. Consider [CST00] for a good survey on SVMs.

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:

$$sgn(f(\mathbf{x}, \mathbf{w}, b)) \tag{1}$$

$$f(\mathbf{x}, \mathbf{w}, b) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b \tag{2}$$

being $\mathbf{x}$ 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):

$$y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 \tag{3}$$

See an example of a 2-dimensional SVM in Figure 1 (leftmost representation).

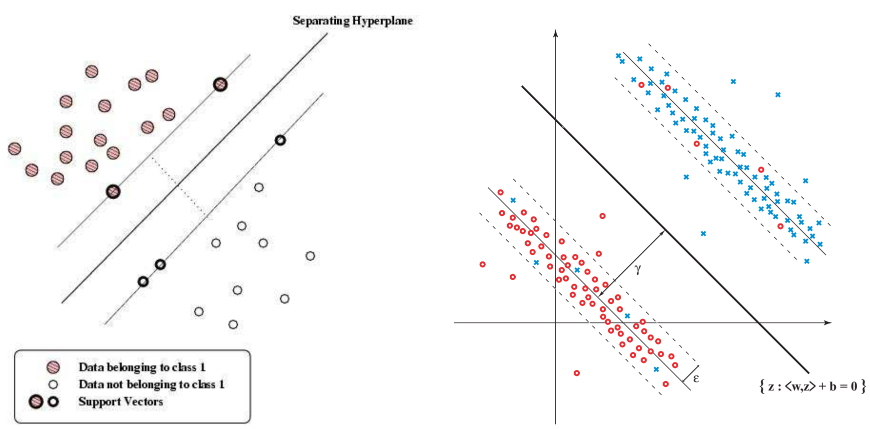


Figure 1: SVM example: hard margin (left) vs. soft margin (right) maximization in $\mathbb{R}^2$.

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:

$$f(\mathbf{x}, \boldsymbol{\alpha}, b) = \sum_{i=1}^{N} y_i \alpha_i \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b \tag{4}$$

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:

$$\mathbf{w}^* = \sum_{i=1}^{N} y_i \alpha_i^* \mathbf{x_i} \tag{5}$$

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 [CST00] 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. See Figure 1 (rightmost representation).

3.3 Problem Setting

We describe the collection of training examples and feature codification.

3.3.1 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 PoS tag 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[5].

3.3.2 Feature Codification

Each example (event) has been represented using the *local context* of the word for which the system will determine a tag (output decision). This local context and local information like capitalization and affixex of the current token will help the system make a decision even if the token hasn't been encountered during training. 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 [DZBG96] 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 pass* solution proposed in [NKM01], in which an initial first pass tagging is performed in order to have right contexts disambiguated for the second pass. Also in [NKM01], 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.

Additional features, inspired in previous work [Bri95, MRCM99, NKM01], have been used to deal with the problem of unknown words. Features appearing a number of times under a certain count cut-off might be ignored for the sake of robustness[6] (See Subsection 4.1).

[5] See [ASS99] for a discussion on the efficiency problems when learning from large PoS training sets.

[6] In our experiments we found out that discarding all features appearing only once

Table 1 shows a rich feature set used in our experiments.

word features	$w_{-3}, w_{-2}, w_{-1}, w_0, w_{+1}, w_{+2}, w_{+3}$
PoS features	$p_{-3}, p_{-2}, p_{-1}, p_0, p_{+1}, p_{+2}, p_{+3}$
ambiguity classes	a_0, a_1, a_2, a_3
may_be'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	$(w_{-2}, w_{-1}, w_0), (w_{-2}, w_{-1}, w_{+1}),$ $(w_{-1}, w_0, w_{+1}), (w_{-1}, w_{+1}, w_{+2}),$ (w_0, w_{+1}, w_{+2})
PoS trigrams	$(p_{-2}, p_{-1}, a_{+0}), (p_{-2}, p_{-1}, a_{+1}),$ $(p_{-1}, a_0, a_{+1}), (p_{-1}, a_{+1}, a_{+2})$
sentence_info	punctuation ('.', '?', '!')
prefixes	$s_1,\ s_1s_2,\ s_1s_2s_3,\ s_1s_2s_3s_4$
suffixes	$s_n,\ s_{n\text{-}1}s_n,\ s_{n\text{-}2}s_{n\text{-}1}s_n,\ s_{n\text{-}3}s_{n\text{-}2}s_{n\text{-}1}s_n$
binary word features	initial Upper Case, all Upper Case, no initial Capital Letter(s), all Lower Case, contains a (period / number / hyphen ...)
word length	integer

Table 1: Rich feature pattern set used in experiments.

See [GM03, GM04b, GM04a] for further details on experiments.

reported significant gains.

4 The SVMTool

The SVMTool software package consists of three main components, namely the model learner (SVMTlearn), the tagger (SVMTagger) and the evaluator (SVMTeval), which are described below.

Previous to the tagging, SVM models (weight vectors and biases) are learned from a training corpus using the SVMTlearn component. Different models are learned for the different strategies. Then, at tagging time, using the SVMTagger component, one may choose the tagging strategy that is most suitable for the purpose of the tagging. Finally, given a correctly annotated corpus, and the corresponding SVMTool predicted annotation, the SVMTeval component displays tagging results.

4.1 SVMTlearn

Given a training set of examples (either annotated or unannotated), it is responsible for the training of a set of SVM classifiers. So as to do that, it makes use of SVM–light[7], an implementation of Vapnik's SVMs in C, developed by Thorsten Joachims [Joa99].

The SVMlight software implementation of Vapnik's Support Vector Machine [Vapnik, 1995] by Thorsten Joachims has been used to train the models. For further information on it see:

- T. Joachims, Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning, B. Schlkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999.

4.1.1 Training Data Format

Training data must be in column format, i.e. a token per line corpus in a sentence by sentence fashion. The column separator is the blank space. The token is expected to be the first column of the line. The tag to predict takes the second column in the output. The rest of the line may contain additional information. See an example:

```
Pierre NNP B-PERSON
Vinken NNP I-PERSON
, , O
61 CD B-NUM
years NNS O
```

[7]The SVM^{light} software is freely available (for scientific use) at the following URL: `http://svmlight.joachims.org`. It is necessary to download it prior to start using the SVMTlearn component. SVM^{light} is not LGPL licensed.

```
old JJ O
, , O
will MD O
join VB O
the DT O
board NN O
as IN O
a DT O
nonexecutive JJ O
director NN O
Nov. NNP B-DATE
29 CD I-DATE
. . O
[...]
```

No special '<EOS>' mark is employed for sentence separation. Sentence punctuation is used instead, i.e. [.!?] symbols are taken as unambiguous sentence separators[8].

4.1.2 Options

SVMTlearn behaviour is easily adjusted through a configuration file.

```
Usage : SVMTlearn [options] <config-file>

options:
  - V verbose 0: none verbose
              1: low verbose [default]
              2: medium verbose
              3: high verbose

Example : SVMTlearn -V 2 config.svmt
```

These are the currently available config-file options:

- *Sliding window:* The size of the sliding window for feature extraction can be adjusted. Also, the core position in which the word to disambiguate is to be located may be selected. By default, window size is 5 and the core position is 2, starting at 0.
- *Feature set:* Three different kinds of feature types can be collected from the sliding window:
  - word features: Word form n-grams. Usually unigrams, bigrams and trigrams suffice. Also, the sentence last word,

[8]'<EOS>' tag is available at tagging time only ('<s>'. Future versions may contain a sentence separator mark at learning time as well.

which corresponds to a punctuation mark ('.', '?', '!'), is important.

  – PoS features: Annotated parts–of–speech and ambiguity classes n-grams, and "may_be's". As for words, considering unigrams, bigrams and trigrams is enough. The ambiguity class for a certain word determines which PoS are possible. A "may_be" states, for a certain word, that certain PoS may be possible, i.e. it belongs to the word ambiguity class.
  – lexicalized features: including prefixes and suffixes, capitalization, hyphenization, and similar information related to a word form.

  Default feature sets for every model are defined.

- *Feature filtering:* The feature space can be kept in a convenient size. Smaller models allow for a higher efficiency. By default, no more than 100,000 dimensions are used. Also, features appearing less than n times can be discarded, which indeed causes the system both to fight against overfitting and to exhibit a higher accuracy. By default, features appearing just once are ignored.

- *SVM model compression:* Weight vector components lower than a given threshold, in the resulting SVM models can be filtered out, thus enhancing efficiency by decreasing the model size but still preserving accuracy level. That is an interesting behaviour of SVM models being currently under study. In fact, discarding up to 70% of the weight components accuracy remains stable, and it is not until 95% of the components are discarded that accuracy falls below the current state–of–the–art (97.0% - 97.2%).

- *C parameter tuning:* In order to deal with noise and outliers in training data, the soft margin version of the SVM learning algorithm allows the misclassification of certain training examples when maximizing the margin. This balance can be automatically adjusted by optimizing the value of the C parameter of SVMs. A local maximum is found exploring accuracy on a validation set for different C values at shorter intervals.

- *Dictionary repairing:* The lexicon extracted from the training corpus can be automatically repaired either based on frequency heuristics or on a list of corrections supplied by the user. This makes the tagger robust to corpus errors. Also a heuristic threshold may be specified in order to consider as tagging errors those ($word_x tag_y$) pairs occurring less than a certain proportion of times with respect to the number of occurrences of $word_x$. For instance, a threshold of 0.001 would consider (run DT) as an error if the word run had been seen at least 1000 times and only once tagged as a 'DT'. This kind of heuristic dictionary reparing doesn't harm the tagger performance, on the contrary, it may enormously help.

Repairing list must comply with the SVMTool dictionary format, i.e. <word> <N_occurrences> <N_possible_tags> 1{<tag(i)> <N_occurrences(i)}N. See an example:

```
...
a 23673 4 DT 23647 FW 8 LS 2 SYM 11
an 3819 1 DT 3819
and 19762 1 CC 19762
are 4507 1 VBP 4507
can 1133 2 MD 1128 NN 5
did 743 1 VBD 743
do 1156 2 VB 402 VBP 754
does 601 1 VBZ 601
for 9890 2 IN 9884 RP 6
he 3181 1 PRP 3181
if 994 1 IN 994
in 18857 3 IN 18573 RB 65 RP 219
is 8499 1 VBZ 8499
it 5768 1 PRP 5768
...
```

- *Ambiguous classes:* The list of PoS presenting ambiguity is, by default, automatically extracted from the corpus but, if available, this knowledge can be made explicit. This acts in favor of the system robustness.
- *Open classes:* The list of PoS tags an unknown word may be labeled as is also, by default, automatically determined. As for ambiguous classes, if available, it is well appreciated for the same reason.
- *Backup lexicon:* A morphological lexicon containing words that are not present in the training corpus may be provided. It can be also provided at tagging time. This file must comply with the SVMTool dictionary format, described above.

4.1.3 Configuration File

Several arguments are mandatory (See Table 2). The rest are optional (See Table 4). Lists of features are defined in the SVMTool feature language (SVMTfl). See Table 5. Lines beginning with '#' are ignored.

The list of action items for the learner must be declared (see Table 3):

NAME	name of the model to create (a log of the experiment is generated onto the file "NAME.EXP")
TRAINSET	location of the training set
SVMDIR	location of the Joachims SVM^{light}software

Table 2: SVMTlearn config-file mandatory arguments.

```
Syntax: do <MODEL> <DIRECTION> [<CK>] [<CU>] [<T>]

where MODEL     = [M0|M1|M2|M3|M4]
      DIRECTION = [LR|RL|LRL]
      CK        = [CK:<<range1>:<range2>:<#iterations>:
                        <#segments_per_iteration>:
                         <log>|<nolog>> | <CK-value>]
      CU        = [CU:<<range1>:<range2>:<#iterations>:
                        <#segments_per_iteration>:
                         <log>|<nolog>> | <CU-value>]
      T         = [T[:<Nfolders>]]
```

MODEL	model type
DIRECTION	model direction
CK	known word C parameter tuning options (optional)
CU	unknown word C parameter tuning options (optional)
T	test options (optional)

Table 3: SVMTlearn config-file action arguments.

Here is an example of a valid config-file:

```
# -------------------------------------------------------------
#SVMTool configuration file for English on the whole WSJ corpus
# -------------------------------------------------------------
#prefix of the model files which will be created
NAME = WSJTP
# -------------------------------------------------------------
#location of the training set
TRAINSET = /home/me/SVMT/corpora/WSJTP/WSJTP.TRAIN
# -------------------------------------------------------------
#location of the Joachims svmlight software
SVMDIR = /home/me/SVMT/soft/
# -------------------------------------------------------------
#action items
```

SET	location of the whole set
VALSET	location of the validation set
TESTSET	location of the test set
TRAINP	proportion of sentences belonging to the provided whole SET which will be used for training
VALP	proportion of sentences belonging to the provided whole SET which will be used for validation
TESTP	proportion of sentences belonging to the provided whole SET which will be used for test
REMOVE_FILES	remove intermediate files?
REMAKE_FOLDERS	remake cross-validation folders?
Kfilter	Weight Filtering for known word models
Ufilter	Weight Filtering for unknown word models
R	dictionary repairing list (heuristically repaired by default)
D	dictionary repairing heuristic threshold (0.001 by default)
BLEX	backup lexicon
LEX	lexicon for unsupervised learning (Model 3)
W	window definition (size, core_position)
F	feature filtering (count_cut_off, max_mapping_size)
CK	C parameter for known words -all models- (0 by default)
CU	C parameter for unknown words -all models- (0 by default)
X	percentage of unknown words expected (3 by default)
AP	list of parts-of-speech presenting ambiguity (automatically created by default)
UP	list of open-classes (automatically created by default)
A0k..A4k	known word feature definition for models 0..4
A0u..A4u	unknown word feature definition for models 0..4

Table 4: SVMTlearn config-file optional arguments.

COLUMN n-grams	$C(col_{id}; n_1, ...n_i...n_m)$
WORD n-grams	$w(n_1, ...n_i...n_m)$ (equivalent to $C(0; n_1, ...n_i...n_m)$)
TAG n-grams	$p(n_1, ...n_i...n_m)$ (equivalent to $C(1; n_1, ...n_i...n_m)$)
AMBIGUITY CLASSES	$k(n)$
MAYBE's	$m(n)$
	where n_i is the relative position with respect to the element to disambiguate
CHARACTER_A(i)	$ca(i)$
	where i is the relative position of the character with respect to the beginning of the word
CHARACTER_Z(i)	$cz(i)$
	where i is the relative position of the character with respect to the end of the word
PREFIXES	$a(i) = s_1 s_2 ... s_i$
SUFFIXES	$z(i) = s_{n\text{-}i} ... s_{n\text{-}1} s_n$
sa	does the word start with lower case?
SA	does the word start with upper case?
CA	does the word contain any capital letter?
CAA	does the word contain several capital letters?
aa	are all letters in the word in lower case?
AA	are all letters in the word in upper case?
SN	does the word start with number?
CP	does the word contain a period?
CN	does the word contain a number?
CC	does the word contain a comma?
MW	does the word contain a hyphen?
L	word length
sentence_info	punctuation ('.', '?', '!')

Table 5: SVMTool feature language.

```
# ---------------------------------------------------------------------
do M0 LR
# ---------------------------------------------------------------------
```

Unspecified parameters take default values. See below an enriched version of the previous config-file:

```
# -------------------------------------------------------------
#SVMTool configuration file for English on the whole WSJ corpus
# -------------------------------------------------------------
#prefix of the model files which will be created
NAME = WSJTP
# -------------------------------------------------------------
#location of the training set
TRAINSET = /home/me/SVMT/corpora/WSJTP/WSJTP.TRAIN
# -------------------------------------------------------------
#location of the Joachims svmlight software
SVMDIR = /home/me/SVMT/soft/
# -------------------------------------------------------------
#dictionary repairing list
R = /home/me/SVMT/corpora/WSJT/WSJ.200
# -------------------------------------------------------------
#window definition (size, core_position)
W = 5 2
# -------------------------------------------------------------
#feature filtering (count_cut_off, max_mapping_size)
F = 5 200000
# -------------------------------------------------------------
#% of unknown words expected  (3 by default)
X = 10
# -------------------------------------------------------------
#remove intermediate files
REMOVE_FILES = 1
# -------------------------------------------------------------
#list of classes (automatically determined by default)
#list of parts-of-speech presenting ambiguity
AP = '' CC CD DT EX FW IN JJ JJR JJS LS MD NN NNS NNP NNPS PDT
      POS PRP PRP$ RB RBR RBS RP SYM UH VB VBD VBG VBN VBP VBZ
      WDT WP WRB
#list of open-classes
UP = FW JJ JJR JJS NN NNS NNP NNPS RB RBR RBS VB VBD VBG VBN
      VBP VBZ
# -------------------------------------------------------------
#action items
# -------------------------------------------------------------
do M0 LR
# -------------------------------------------------------------
#feature definition for model 0
#ambiguous-right [default]
```

```
A0k = w(-2) w(-1) w(0) w(1) w(2) w(-2,-1) w(-1,0) w(0,1)
     w(-1,1) w(1,2) w(-2,-1,0) w(-2,-1,1) w(-1,0,1)
     w(-1,1,2) w(0,1,2) p(-2) p(-1) p(-2,-1) p(-1,1) p(1,2)
     p(-2,-1,1) p(-1,1,2) a(0) a(1) a(2) m(0) m(1) m(2)
A0u = w(-2) w(-1) w(0) w(1) w(2) w(-2,-1) w(-1,0) w(0,1)
     w(-1,1) w(1,2) w(-2,-1,0) w(-2,-1,1) w(-1,0,1)
     w(-1,1,2) w(0,1,2) p(-2) p(-1) p(-2,-1) p(-1,1) p(1,2)
     p(-2,-1,1) p(-1,1,2) a(0) a(1) a(2) m(0) m(1) m(2)
     a(2) a(3) a(4) z(2) z(3) z(4) ca(1) cz(1)
     L SA AA SN CA CAA CP CC CN MW
# -------------------------------------------------------------
```

In this case model NAME is 'WSJTP', so model files will begin with this prefix. Only the training set is specified. A list of dictionary repairings is provided. A window of 5 elements, being the core in the third position, is defined for feature extraction. The expected proportion of unknown words is 10%. Intermediate files will be removed. The list of parts-of-speech presenting ambiguity is supplied. Also, a list of open-classes is provided.

This config-file is designed to learn Model 0 on the Wall Street Journal English Corpus. That would allow for the use of tagging strategies 0 and 5, only left-to-right though. See how, instead of using default feature sets, two feature sets are defined for Model 0 (for the two distinct problems of known word and unknown word tag guessing).

4.1.4 C Parameter Tuning

C parameter tuning is optional. Either you specify no C parameter (C = 0 by default), or you can specify a fixed value (e.g., *CK:0.1 CU:0.01*), or you can perform an automatic tuning by greedy exploration. If you decide to do so then you must provide a validation set and specify the interval to be explored and how to do it (i.e. number of iterations and number of segments per iteration). Moreover, the first iteration can take place in a logarithmic fashion or not. For instance, *CK:0.01:10:3:10:log* would try these values for C: 0.01, 0.1, 1, 10 at the first iteration. For the next iteration, the algorithm explores on both sides of the point where the maximal accuracy was obtained, half to the next and previous points. For instance, suppose the maximal accuracy was obtained for C = 1, then it would explore the range form 0.1 / 2 = 0.05 to 0.1 / 2 = 0.5. The segmentation ratio would be 0.045 so the algorithm would go for values 0.05, 0.095, 0.14, 0.185, 0.23, 0.275, 0.32, 0.365, 0.41, 0.455, and 0.5. And so on for the following iteration. See further detail in Subsection 6.3.

4.1.5 Test

After training a model it can be evaluated against a test set. To indicate so the T option must be activated in the corresponding do-action, e.g., *"do M0 LR CK:0.01:10:3:10:log CU:0.07 T"*. By default it expects a test set definition in the config file. But training/test can also be performed through a cross-validation. The number of folders must be provided, e.g., *"do M0 LR CK:0.01:10:3:10:log CU:0.07 T:10"*. 10 is a good number.

Furthermore, if training/test goes in cross-validation then the C Parameter tuning goes too, even if a validation set has been provided.

4.1.6 Models

Five different kinds of models have been implemented by now. Models 0, 1, and 2 differ only in the features they consider. Model 3 and Model 4 are just like Model 0 with respect to feature extraction but examples are selected in a different manner. Model 3 is for unsupervised learning so, given an unlabeled corpus and a dictionary, at learning time it can only count on knowing the ambiguity class, and the PoS information only for unambiguous words. Model 4 achieves robustness by simulating unknown words in the learning context at training time.

Model 0 This is the default model. The unseen context remains ambiguous. It was thought having in mind the one-pass on-line tagging scheme, i.e. the tagger goes either left-to-right or right-to-left making decisions. So, past decisions feed future ones in the form of PoS features. At tagging time only the parts-of-speech of already disambiguated tokens are considered. For the unseen context, ambiguity classes are considered instead. See features in Table 6.

Model 1 This model considers the unseen context already disambiguated in a previous step. So it is thought for working at a second pass, revisiting and correcting already tagged text. See features in Table 7.

Model 2 This model does not consider pos features at all for the unseen context. It is designed to work at a first pass, requiring Model 1 to review the tagging results at a second pass. See features in Table 8.

Model 3 The approach is similar to that of [Mih03]. The training is based on the role of unambiguous words. Linear classifiers are trained with examples of unambiguous words extracted from an unannotated corpus. So, fewer PoS information is available.

ambiguity classes	a_0, a_1, a_2
may_be's	m_0, m_1, m_2
PoS features	p_{-2}, p_{-1}
PoS bigrams	$(p_{-2}, p_{-1}), (p_{-1}, a_{+1}), (a_{+1}, a_{+2})$
PoS trigrams	$(p_{-2}, p_{-1}, a_{+0}), (p_{-2}, p_{-1}, a_{+1}),$ $(p_{-1}, a_0, a_{+1}), (p_{-1}, a_{+1}, a_{+2})$
single characters	$ca(1), cz(1)$
prefixes	$a(2), a(3), a(4)$
suffixes	$z(2), z(3), z(4)$
lexicalized features	SA, CAA, AA, SN, CP, CN, CC, MW, L
sentence_info	punctuation ('.', '?', '!')

Table 6: Model 0. Example of suitable PoS features.

ambiguity classes	a_0, a_1, a_2
may_be's	m_0, m_1, m_2
PoS features	$p_{-2}, p_{-1}, p_{+1}, p_{+2}$
PoS bigrams	$(p_{-2}, p_{-1}), (p_{-1}, p_{+1}), (p_{+1}, p_{+2})$
PoS trigrams	$(p_{-2}, p_{-1}, a_0), (p_{-2}, p_{-1}, p_{+1}),$ $(p_{-1}, a_0, p_{+1}), (p_{-1}, p_{+1}, p_{+2})$
single characters	$ca(1), cz(1)$
prefixes	$a(2), a(3), a(4)$
suffixes	$z(2), z(3), z(4)$
lexicalized features	SA, CAA, AA, SN, CP, CN, CC, MW, L
sentence_info	punctuation ('.', '?', '!')

Table 7: Model 1. Example of suitable PoS features.

ambiguity classes	a_0
may_be's	m_0
PoS features	p_{-2}, p_{-1}
PoS bigrams	(p_{-2}, p_{-1})
PoS trigrams	(p_{-2}, p_{-1}, a_0)
single characters	$ca(1), cz(1)$
prefixes	$a(2), a(3), a(4)$
suffixes	$z(2), z(3), z(4)$
lexicalized features	SA, CAA, AA, SN, CP, CN, CC, MW, L
sentence_info	punctuation ('.', '?', '!')

Table 8: Model 2. Example of suitable PoS features.

The only additional information required is a morpho-syntactic dictionary.

Model 4 The errors caused by unknown words at tagging time punish severly the system. So as to alleviate this problem, during the learning some words are artificially marked as unknown in order to learn a more realistic model. The process is very simple. The corpus is divided in a number of folders. Before starting to extract samples from each of the folders, a dictionary is generated out from the rest of folders. So, the words appearing in a folder but not in the rest are unknown words to the learner.

4.2 SVMTagger

Given a text corpus (one token per line) and the path to a previously learned SVM model (including the automatically generated dictionary), it performs the PoS tagging of a sequence of words. The tagging goes on-line based on a sliding window which gives a view of the feature context to be considered at every decision (see Figure 2). Calculated part–of–speech tags feed directly forward next tagging decisions as context features.

The SVMtagger component works on standard input/output. It processes a token per line corpus in a sentence by sentence fashion. The token is expected to be the first column of the line. The predicted tag will take the second column in the output. The rest of the line remains unchanged. Lines beginning with '## ' are ignored by the tagger. See an example of input to the SVMTagger:

```
## this is just an example
We
could
be
heroes
## you can place a comment anywhere
,
just ## hey! only the first field is processed
for
one
day
.
```

This could be the expected output:

```
## this is just an example
We PRP
```

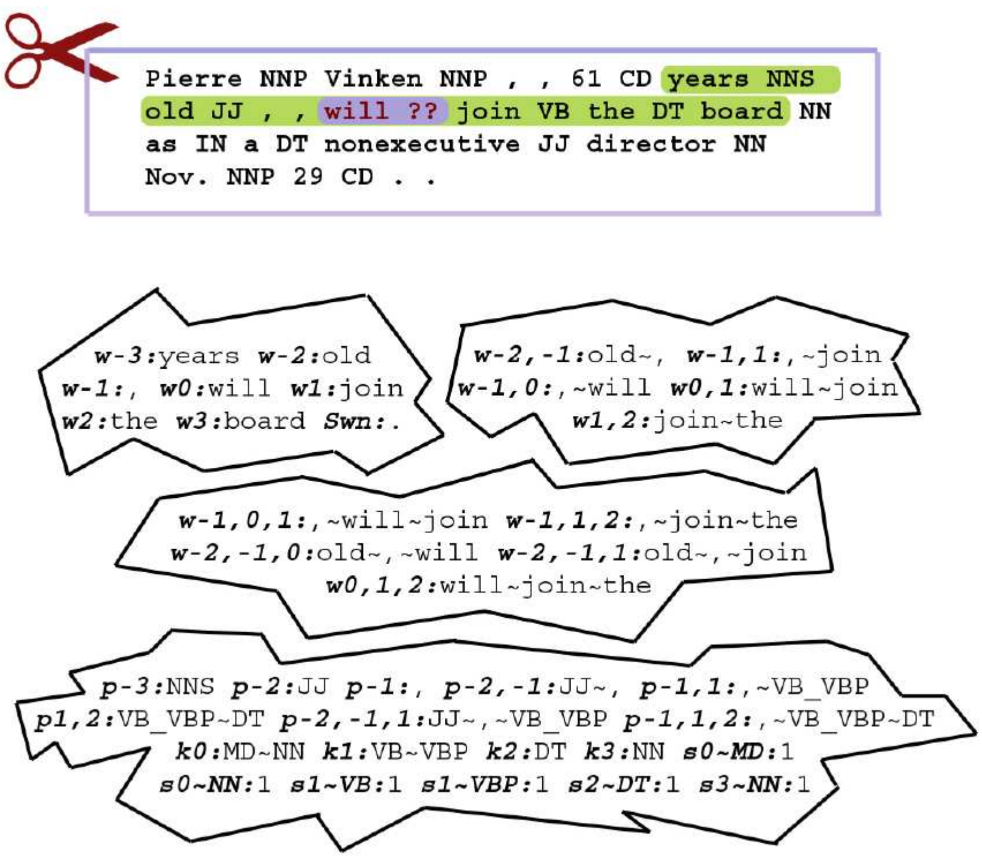


Figure 2: SVMTagger. Feature extraction

```
could MD
be VB
heroes NNS
## you can place a comment anywhere
, ,
just RB ## hey! only the first field is processed
for IN
one CD
day NN
. .
```

4.2.1 Options

SVMTagger is very flexible, and adapts very well to the needs of the user. Thus you may find the several options currently available:

- *Tagging scheme:* Two different tagging schemes may be used.
  - Greedy: Each tagging decision is made based on a reduced context. Later on, decisions are not further reconsidered, except in the case of tagging at two steps or tagging in two directions.
  - Sentence-level: By means of dynamic programming techniques (Viterbi algorithm), the global sentence sum of SVM tagging scores is the function to maximize. See equation 6. Given a sentence $S = w_1..w_n$ as a word sequence and the set $T(S) = \{t_i : 1 \leq i \leq |S|) \wedge t_i \in ambiguity\_class(w_i)\}$ of all possible sequences of PoS tags associated to S.

    $$t(S) = argmax_{s \in T(S)} \sum_{i=1}^{|s|} score_s(i) \tag{6}$$

    A softmax function is used by default so as to transform this sum of scores into a product of probabilities.
    Because sentence-level tagging is expensive, two pruning methods are provided. First, the maximum number of beams may be defined. Alternatively, a threshold may be specified so that solutions scoring under a certain value (with respect to the best solution at that point) are discarded. Both pruning techiques have proved effective and efficient in our experiments.
- *Tagging direction:* The tagging direction can be either "left-to-right", "right-to-left", or a combination of both. The tagging direction varies results yielding a significant improvement when

both are combined. For every token, every direction assigns a tag with a certain score. The highest-scoring tag is selected. This makes the tagger very robust. In the case of sentence-level tagging there's an additional way to combine left-to-right and right-to-left. "GLRL" direction makes a global decision, i.e. considering the sentence as a whole. For every sentence, every direction assigns a sequence of tags with an associated score that corresponds to the sum of scores (or the product of probabilities when using a softmax function, the default option). The highest-scoring sequence of tags is selected.

- *One pass / Two passes:* Another way of achieving robustness is by tagging in two passes. At the first pass only PoS features related to already disambiguated words are considered. At a second pass disambiguated PoS features are available for every word in the feature context, so when revisiting a word tagging errors may be alleviated.
- *SVM Model Compression:* Just as for the learning, weight vector components lower than a certain threshold, can be ignored.
- *All scores:* Because sometimes not only the tag is relevant, but also its score and scores of all competing tags, as a measure of confidence, this information is available.
- *Backup lexicon:* Again, a morphological lexicon containing new words that were not available in the training corpus may be provided.
- *Lemmae lexicon:* Given a lemmae lexicon containing `<word form, tag, lemma>` entries, output may be lemmatized. See an example of such a lexicon:

```
...
playing JJ playing
playing NN playing
playing VBG play
...
```

- `<EOS>` *tag:* The '`<s>`' tag may be employed for sentence separation. Otherwise, sentence punctuation is used instead, i.e. [.!?] symbols are taken as unambiguous sentence separators.
- *Guessing module:* If we have noisy input text, we can assume that many unknown words come from user mistakes or misspellings. In those cases, we should be able to guess the set of possible POS tags for those unknown words, by hypothesizing

the correct words in the current dictionary that are more similar to each of the erroneous input words and transferring correct words' information into the unknown word entry (by simply merging all the possible POS tags). To do that, the SVMTagger calls a script which implements a Levenstein distance algorithm to enrich the dictionary of the model to be used for tagging. This script (*dist_edit_dicc.py*) can be used independently from the SVMTagger. Apart from speeding up execution, with the separate execution of the *guessing module*, the user can create multiple variants of the word lists to be included in the model dictionary file (e.g., by varying parameter values) and play with them to estimate the best parameterization for the robust tagger. The following is an example command to execute the guessing module.

```
python dist_edit_dicc.py input_file DICT_file [-d 3] > output_file
```

Following the previous practice, the user can generate several alternative dictionaries for the same model depending on which input he/she used. The user can also indicate a threshold on the maximum distance allowed among words using the *-d* option. Otherwise, the POS attached to the unknown words will be those of the best candidates, that is, the words in the dictionary with the lowest distance to the input words.

In order to call the script inside the SVMTagger, the user should indicate the options to run the script as follows:

```
SVMTagger.pl -V 2 -d "input_file dictfile -d 1" WSJTP
```

Furthermore, if the user wants to keep the dictionary created by the script, then he/she can use the *saving* option as follows:

```
SVMTagger.pl -V 2 -d "input_file dictfile -ds 1" WSJTP
```

In that way, the program will keep the new extended dictionary in a file called *dictfile_File*.

```
Usage : SVMTagger [options] <model>

options:
  - T <strategy>
```

```
           0: one-pass   (default - requires model 0)
           1: two-passes [revisiting results and relabeling
                          - requires model 2 and model 1]
           2: one-pass   [robust against unknown words
                          - requires model 0 and model 2]
           3: one-pass   [unsupervised learning models
                          - requires model 3]
           4: one-pass   [very robust against unknown words
                          - requires model 4]
           5: one-pass   [sentence-level likelihood
                          - requires model 0]
           6: one-pass   [robust sentence-level likelihood
                          - requires model 4]
  - S <direction>
           LR: left-to-right (default)
           RL: right-to-left
          LRL: both left-to-right and right-to-left
         GLRL: both left-to-right and right-to-left
              (global assignment, only applicable under a
               sentence level tagging strategy)
  - K <n> weight filtering threshold for known words (default is 0)
  - U <n> weight filtering threshold for unknown words (default is 0)
  - Z <n> number of beams in beam search, only applicable under
          sentence-level strategies (default is disabled)
  - R <n> dynamic beam search ratio, only applicable under
          sentence-level strategies (default is disabled)
  - F <n> softmax function to transform SVM scores into
          probabilities (default is 1)
              0: do_nothing
              1: ln(e^score(i) / [sum:1<=j<=N:[e^score(j)]])
  - A     predicitons for all possible parts-of-speech are returned
  - B     <backup_lexicon>
  - L     <lemmae_lexicon>
  - EOS   enable usage of end_of_sentence '<s>' string
          (disabled by default, [!.?] used instead)
  - V     <verbose> -> 0: none verbose
                       1: low verbose
                       2: medium verbose
                       3: high verbose
                       4: very high verbose
  - D     <distance script options> -> options for the distance module (without saving the new dictionary)
  - DS     <distance script options> -> options for the distance module (saving the new dictionary)

model: model location (path/name)
             (name as declared in the config-file NAME)

Example : SVMTagger -V 2 -S LRL -T 0 WSJTP < wsj.test > wsj.test.out
```

4.2.2 Strategies

Seven different tagging strategies have been implemented so far:

Strategy 0 It is the default one. It makes use of Model 0 in a greedy on-line fashion, one-pass.

Strategy 1 As a first attempt to achieve robustness in front of error propagation, it works in two passes, in an on-line greedy way. It uses Model 2 in the first pass, and Model 1 in the second. In other words, in the first pass the unseen morphosyntactic context remains ambiguous while in the second pass the tag predicted in the first pass is available also for unseen tokens and used as a feature.

Strategy 2 This strategy tries to achieve robustness by using two models at tagging time, namely Model 0 and Model 2. When all the words in the unseen context are known it uses Model 0. Otherwise it makes use of Model 2.

Strategy 3 It uses Model 3, again in a greedy and on-line manner. This unsupervised learning strategy is still under experimentation. It's not yet ready for release.

Strategy 4 It simply uses Model 4 as is, in an on-line greedy fashion.

Strategy 5 Still working on a more robust scheme, this strategy performs a sentence-level tagging by means of dynamic programming techniques (Viterbi algorithm). It uses Model 0.

Strategy 6 Same as strategy 5, this strategy performs a sentence-level tagging, this time applying Model 4.

4.3 SVMTeval

Given a SVMTool predicted tagging output and the corresponding gold-standard, SVMTeval evaluates the performance in terms of accuracy. It is a very useful component for the tuning of the system parameters, such as the C parameter, the feature patterns and filtering, the model compression et cetera.

Moreover, based on a given morphological dictionary (e.g., the automatically generated at training time) results may be presented also for different sets of words (known words vs unknown words, ambiguous words vs unambiguous words). A different view of these same results can be seen from the class of ambiguity perspective, too, i.e., words sharing the same kind of ambiguity may be considered together. Also words sharing the same degree of disambiguation complexity, determined by the size of their ambiguity classes, can be grouped.

```
Usage : SVMTeval [mode] <model> <gold> <pred>

        - mode:  0 - complete report (everything)
                 1 - overall accuracy only [default]
                 2 - accuracy of known vs unknown words
                 3 - accuracy per level of ambiguity
                 4 - accuracy per kind of ambiguity
                 5 - accuracy per class

        - model: model name

        - gold:  correct tagging file

        - pred:  predicted tagging file

Example : SVMTeval WSJTP WSJTP.IN WSJTP.OUT
```

4.3.1 Reports

brief report By default, a brief report mainly returning the overall accuracy is ellaborated. It also provides information about the number of tokens processed, and how much were known/unknown and ambiguous/unambiguous according to the model dictionary. Results are always compared to the most-frequent-tag (MFT) baseline.

```
* ======================== SVMTeval report ==================
* model   = [W52/WSJTP.TRAIN.W5]
* testset = [/home/me/WSJTP/WSJTP.TEST]
* ==========================================================
* ================= TAGGING SUMMARY =========================
#WORDS  = 129654
#KNOWN  = 126005 / 129654   --> (97.1856 %)
#UNKNOWN  = 3649 / 129654   --> (2.8144 %)
#AMBIGUOUS  = 45779 / 129654   --> (35.3086 %)
MFT baseline  = 118376 / 129654   --> (91.3015 %)
* ================= OVERALL ACCURACY ========================
HITS       TRIALS             ACCURACY   MFT-baseline
* ----------------------------------------------------------
125966  /  129654     =      97.1555 %      91.3015 %
* ==========================================================
```

known vs unknown tokens Accuracy for four different sets of words is returned. The first set is that of all known tokens, tokens which were seen during the training. The second and third sets contain respectively all ambiguous and all unambiguous tokens among these known tokens. Finally, there is the set of unknown tokens, which were not seen during the training.

```
* ========================= SVMTeval report ==================
* model   = [W52/WSJTP.TRAIN.W5]
* testset = [/home/me/WSJTP/WSJTP.TEST]
* ============================================================
* ================= TAGGING SUMMARY ==========================
#WORDS  = 129654
#KNOWN  = 126005 / 129654   --> (97.1856 %)
#UNKNOWN  = 3649 / 129654   --> (2.8144 %)
#AMBIGUOUS  = 45779 / 129654   --> (35.3086 %)
MFT baseline  = 118376 / 129654   --> (91.3015 %)
* ================= KNOWN vs UNKNOWN WORDS ===================
HITS     TRIALS   ACCURACY
* ------------------------------------------------------------
* ======= known ==============================================
125966 / 129654 = 97.393 %
-------- known unambiguous words -----------------------------
125966 / 129654 = 99.3817 %
-------- known ambiguous words -------------------------------
125966 / 129654 = 93.9077 %
* ======= unknown ============================================
125966 / 129654 = 88.9559 %
* ============================================================
* ================= OVERALL ACCURACY =========================
HITS       TRIALS             ACCURACY   MFT-baseline
* ------------------------------------------------------------
125966  /  129654     =      97.1555 %      91.3015 %
* ============================================================
```

level of ambiguity This view of the results groups together all words having the same degree of PoS–ambiguity.

```
* ========================= SVMTeval report ==================
* model   = [W52/WSJTP.TRAIN.W5]
* testset = [/home/me/WSJTP/WSJTP.TEST]
* ============================================================
* ================= TAGGING SUMMARY ==========================
#WORDS  = 129654
#KNOWN  = 126005 / 129654   --> (97.1856 %)
#UNKNOWN  = 3649 / 129654   --> (2.8144 %)
#AMBIGUOUS  = 45779 / 129654   --> (35.3086 %)
MFT baseline  = 118376 / 129654   --> (91.3015 %)
* ================= ACCURACY PER LEVEL OF AMBIGUITY =========
#CLASSES = 7
* ============================================================
LEVEL HITS TRIALS ACCURACY MFT-ACCURACY
* ------------------------------------------------------------
1     79738  /  80234 = 99.3818 %      99.3831 %
2     24332  /  25625 = 94.9541 %      86.7746 %
3     12892  /  13974 = 92.257  %      82.4388 %
```

```
4      5142  /   5398 = 95.2575 %      82.4935 %
5       414  /    544 = 76.1029 %      56.4338 %
6       202  /    230 = 87.8261 %      52.6087 %
17     3246  /   3649 = 88.9559 %            0 %
* ================ OVERALL ACCURACY ========================
HITS       TRIALS            ACCURACY   MFT-baseline
* ----------------------------------------------------------
125966  /  129654     =      97.1555 %      91.3015 %
* ==========================================================
```

kind of ambiguity This view is much finer. Every class of ambiguity is studied separately.

```
* ======================== SVMTeval report ==================
* model   = [W52/WSJTP.TRAIN.W5]
* testset = [/home/me/WSJTP/WSJTP.TEST]
* ==========================================================
* ================ TAGGING SUMMARY =========================
#WORDS  = 129654
#KNOWN  = 126005 / 129654   --> (97.1856 %)
#UNKNOWN  = 3649 / 129654   --> (2.8144 %)
#AMBIGUOUS  = 45779 / 129654   --> (35.3086 %)
MFT baseline  = 118376 / 129654   --> (91.3015 %)
* ================ ACCURACY PER CLASS OF AMBIGUITY =========
#CLASSES = 256
* ==========================================================
CLASS
HITS TRIALS ACCURACY MFT-baseline
* ----------------------------------------------------------
[...]
DT_IN_RB_WDT
     1000 / 1061 = 94.2507 %      59.4722 %
FW_JJ_JJR_JJS_NN_NNS_NNP_NNPS_RB_RBR_RBS_VB_VBD_VBG_VBN_VBP_VBZ
     3246  /  3649 = 88.9559 %            0 %
JJ_NNP
     1157  /  1204 = 96.0963 %      88.6213 %
JJ_NN_VB_VBP
     562   /   592 = 94.9324 %      74.1554 %
JJ_VBD_VBN
     1256  /  1439 = 87.2828 %      66.3655 %
NNS_VBZ
     2084  /  2124 = 98.1168 %      90.0659 %
NN_VB
     2133  /  2163 =  98.613 %      94.2672 %
NN_VBG
     835   /   925 = 90.2703 %      78.4865 %
NN_VB_VBP
     3691  /  3772 = 97.8526 %      81.9194 %
POS_VBZ
```

```
     1331  /  1365 = 97.5092 %      86.8864 %
VBD_VBN
     3187  /  3356 = 94.9642 %      80.8701 %
VB_VBP
     1876  /  1918 = 97.8102 %      75.3389 %
[...]
* ================= OVERALL ACCURACY =========================
HITS       TRIALS             ACCURACY   MFT-baseline
* ------------------------------------------------------------
125966  /  129654     =      97.1555 %      91.3015 %
* ============================================================
```

class Every class is studied individually.

```
* ========================= SVMTeval report ==================
* model   = [W52/WSJTP.TRAIN.W5]
* testset = [/home/me/WSJTP/WSJTP.TEST]
* ============================================================
* ================= TAGGING SUMMARY ==========================
#WORDS  = 129654
#KNOWN  = 126005 / 129654   --> (97.1856 %)
#UNKNOWN  = 3649 / 129654   --> (2.8144 %)
#AMBIGUOUS  = 45779 / 129654   --> (35.3086 %)
MFT baseline  = 118376 / 129654   --> (91.3015 %)
* ================= ACCURACY PER LEVEL OF AMBIGUITY =========
POS HITS TRIALS ACCURACY MFT-ACCURACY
* ------------------------------------------------------------
#       15  /     15  =      100 %           100 %
$      943  /    943  =      100 %           100 %
''    1044  /   1045  =  99.9043 %       99.0431 %
(      186  /    186  =      100 %           100 %
)      187  /    187  =      100 %           100 %
,     6876  /   6876  =      100 %           100 %
.     5381  /   5381  =      100 %           100 %
:      752  /    752  =      100 %           100 %
CC    3237  /   3250  =  99.6    %       99.5692 %
CD    4789  /   4823  =  99.295  %       90.6075 %
DT   11117  /  11183  =  99.4098 %       98.453  %
EX     126  /    126  =      100 %           100 %
FW       7  /     30  =  23.3333 %            20 %
IN   13322  /  13492  =  98.74   %       98.3398 %
JJ    7617  /   8215  =  92.7206 %       85.2708 %
JJR    388  /    423  =  91.7258 %       95.9811 %
JJS    262  /    267  =  98.1273 %       95.5056 %
LS      10  /     15  =  66.6667 %             0 %
MD    1264  /   1267  =  99.7632 %       99.8421 %
NN   17257  /  17834  =  96.7646 %       91.9143 %
NNP  12717  /  13177  =  96.5091 %       85.0118 %
NNPS    98  /    170  =  57.6471 %       49.4118 %
```

```
NNS   7948  /   8061  =  98.5982 %       93.971  %
PDT     27  /     44  =  61.3636 %            0 %
POS   1266  /   1276  =  99.2163 %          100 %
PRP   2194  /   2205  =  99.5011 %       99.093  %
PRP$  1067  /   1068  =  99.9064 %          100 %
RB    4017  /   4405  =  91.1918 %       83.0874 %
RBR    189  /    271  =  69.7417 %       21.7712 %
RBS     61  /     69  =  88.4058 %        2.8986 %
RP     312  /    397  =  78.5894 %       65.995  %
SYM     10  /     11  =  90.9091 %       72.7273 %
TO    2913  /   2913  =      100 %          100 %
UH       7  /     17  =  41.1765 %       58.8235 %
VB    3394  /   3573  =  94.9902 %       69.6054 %
VBD   4361  /   4561  =  95.615  %       84.6525 %
VBG   1816  /   1933  =  93.9472 %       85.1526 %
VBN   2476  /   2707  =  91.4666 %       72.0355 %
VBP   1476  /   1565  =  94.3131 %       71.9489 %
VBZ   2571  /   2639  =  97.4233 %       86.5479 %
WDT    571  /    584  =  97.774  %       52.911  %
WP     282  /    283  =  99.6466 %       99.6466 %
WP$     37  /     37  =      100 %          100 %
WRB    302  /    304  =  99.3421 %       99.3421 %
``    1074  /   1074  =      100 %          100 %
* ================= OVERALL ACCURACY =========================
HITS       TRIALS            ACCURACY   MFT-baseline
* ------------------------------------------------------------
125966  /  129654     =      97.1555 %      91.3015 %
* ============================================================
```

4.4 SVMTool API

Embedded usage of the SVMTool is also possible. It is based on the SVMTool API which offers all the capabilities of the SVMTool in an ellegant manner. The user must follow four simple steps:

1. *Loading of SVMTool models* according to the settings selected.
2. *Preparation* of input tokens for SVMTool processing.
3. *PoS–tagging* of input tokens.
4. *Collection* of tagging results. Not only the winner PoS is available but the losers as well, and the SVM score for all of them. This information could be really helpful in the case of hard decisions, when two or more PoS were nearly tied.

See an example of use in Table 9.

5 Evaluation

In this section we consider the results of applying SVMTool to PoS Tagging for different languages and different corpora. In Subsection4.3 we discussed how the user can evaluate the performance of a model they have created using the SVMTeval utility. SVMTeval provides token accuracy from different viewpoints.

5.1 Accuracy

The SVMTool has been already successfully applied to English and Spanish PoS Tagging, exhibiting state–of–the–art performance (97.16% and 96.89%, respectively). In both cases results clearly outperform the HMM–based TnT part–of–speech tagger [Bra00], compared exactly under the same conditions. In our opinion, TnT is an example of a really practical tagger for NLP applications.

5.2 Efficiency

At tagging time, a speed of 1,500 words per second is achieved by the Perl prototype on a Pentium–IV, 2GHz, 1 GB RAM. C++ tagger achieves a speed of 10,000 words per second under the same conditions. Regarding learning time, it strongly depends on the training set size, tagset, feature set, learning options, et cetera. The upper bound for the experiments reported below are about 24 CPU hours machine (Wall Street Journal corpus, 912K words for training, full set of attributes and fine adjusting of the C parameter). See [GM03] for further details. Below you may find a summary of the results obtained by the SVMTool.

5.3 Results for English

5.3.1 WSJ

Experiments for English used the Wall Street Journal corpus (1,173 Kwords). Sections 0-18 were used for training (912 Kwords), 19-21 for validation (131 Kwords), and 22-24 for test (129 Kwords), respectively. 2.81% of the words in the test set are unknown to the training set. Best other results so far reported on this same test set are [Col02] (97.11%) and [TKM03] (97.24%). See results in Table 10.

```
my @tokens = ('The SVMTool was presented to NLP researchers
                     at LREC-2004 in Lisbon.');
my $svmt = SVMTAGGER::SVMT_load(...);
my $input = SVMTAGGER::SVMT_prepare_input(@tokens);
my $output = SVMTAGGER::SVMT_tag($input, $svmt...);
for my $elem (@{$output}) {
    print $elem->get_word."_".$elem->get_pos;
}
```

Table 9: SVMTool API. Example of use.

	known	amb.	unk.	all.
TnT	96.76%	92.16%	85.86%	96.46%
SVMTool	97.39%	93.91%	89.01%	97.16%

Table 10: Accuracy results of the SVMTool (on a one-pass, left-to-right and right-to-left combined, greedy tagging scheme) compared to TnT for English on the WSJ corpus test set. 'known' and 'unk.' refer to the subsets of known and unknown words, respectively, 'amb' to the set of ambiguous known words and 'all' to the overall accuracy.

5.4 Results for Spanish

5.4.1 LEXESP

Experiments used the LEXESP corpus (106 Kwords). It was randomly divided into training set (86 Kwords) and test set (20 Kwords). 12.21% of the words in the test set are unknown to the training set. See results in Table 11.

	known	amb.	unk.	all.
TnT	97.73%	93.70%	87.66%	96.50%
SVMTool	98.08%	95.04%	88.28%	96.89%

Table 11: Accuracy results of the SVMTool (on a one-pass, left-to-right and right-to-left combined, greedy tagging scheme) compared to TnT for Spanish on the LEXESP corpus, 'known' and 'unk.' refer to the subsets of known and unknown words, respectively, 'amb' to the set of ambiguous known words and 'all' to the overall accuracy.

Using additional morpho-syntactic information provided by a morphological analyzer [XCPP04] in the form of a backup lexicon both tools improve very considerably their performance. Sure it is due to the fact that now there are no unknown words. But notice these words have not been seen among the training data. See results in Table 12.

	amb.	all.
TnT	94.05%	98.41%
SVMTool	95.43%	98.86%

Table 12: Accuracy results of the SVMTool (on a one-pass right-to-left greedy tagging scheme) compared to TnT for Spanish on the LEXESP corpus with the aid of a backup morphological lexicon, 'amb' refers to the set of ambiguous known words and and 'all' to the overall accuracy.

5.4.2 3LB

Experiments used the 3LB[9] corpus (75 Kwords). It was randomly divided into training set (59 Kwords) and test set (16 Kwords). 13.65%

[9]The 3LB project is funded by the Spanish Ministry of Science and Technology (FIT-15050-2002-244), visit the project website at http://www.dlsi.ua.es/projectes/3lb/

of the words in the test set are unknown to the training set. See results in Table 13.

	known	amb.	unk.	all.
TnT	97.73%	93.70%	87.66%	96.50%
SVMTool	98.08%	95.04%	88.28%	96.89%

Table 13: Accuracy results of the SVMTool (on a one-pass, left-to-right and right-to-left combined, greedy tagging scheme) compared to TnT for Spanish on the LEXESP corpus, 'known' and 'unk.' refer to the subsets of known and unknown words, respectively, 'amb' to the set of ambiguous known words and 'all' to the overall accuracy.

5.5 Results for Catalan

5.5.1 3LB

Experiments used the 3LB corpus (104 Kwords). It was randomly divided into training set (85 Kwords) and test set (19 Kwords). 8.1989 % of the words in the test set are unknown to the training set. See results in Table 14.

	known	amb.	unk.	all.
TnT	98.10%	96.12%	90.10%	97.44%
SVMTool	98.33%	96.23%	91.49%	97.77%

Table 14: Accuracy results of the SVMTool (on a one-pass, left-to-right and right-to-left combined, greedy tagging scheme) compared to TnT for Catalan on the 3LB corpus, 'known' and 'unk.' refer to the subsets of known and unknown words, respectively, 'amb' to the set of ambiguous known words and 'all' to the overall accuracy.

6 Tutorial

This tutorial is intended to guide you through the process of configurating and tuning the SVMTool to suit the requirements of your system. In Subsection 6.1 a simple case of use involving no learning is presented. Moreover, because languages may differ in such a number of aspects regarding morphology and syntax, it may be very useful and sometimes crucial to devote some initial time to configure the SVMTool. Subsection 6.2 discusses the most relevant aspects to take into account when training the SVMTool. Finally, Subsection 6.3 provides some useful advices regarding the adjustment of the SVMTool parameters.

6.1 Running the SVMTool

Suppose you have an NLP system that requires some PoS tagging at a first stage as a pre-processing step. Let's say your system works for English. Now suppose, too, you heard late news about the SVMTool telling there's a new C++ version that's much faster than the Perl prototype. So you decided you wanted to give it a try and see if you can make your system better with a more accurate tagger. In that case all you have to do is to download the SVMTool++ and the models for English (Wall Street Journal) from the SVMTool website[10]. Fast, you're done. You can start using the SVMTagger component. See Table 15.

6.2 Training the SVMTool

Now either suppose that you're dealing with a problem other than PoS tagging for English, Spanish or Catalan [11]. Therefore you're going to work first with the SVMTlearn component.

6.2.1 Learning Time

At this point, several decisions must be made:

data sets In principle, only the training set is required. However, if you have enough data it is a good practice to split it into three working sets (i.e. training, validation and test). That will allow you to train, tune and evaluate your system before you start using it. If you don't have much data there's no need to

[10] `http://www.lsi.upc.es/~nlp/SVMTool/`

[11] Models for English (based on the Wall Street Journal Corpus), Spanish and Catalan (based on the 3LB Corpus) are available at the SVMTool website.

```
echo 'I am very happy with the SVMTool .' | tr -s ' ' '\012' |
SVMTagger -V 1 -T 0 -S LR /SVMT/EXP/ENG/WSJTP
```

SVMTool v1.3. Copyright (C) 2004-2006 Jesus Gimenez and Lluis Marquez.
MODEL = /home/me/SVMT/EXP/ENG/S/912k/WSJTP.912k T = 0 :: S = LR :: K = 0 :: U = 0
TAGGING < DIRECTION = left-to-right > I PRP am VBP very RB happy JJ with IN the DT SVMTool NNP . . .1 sentences [DONE]
BENCHMARK TIME: 0 wallclock secs (0.02 usr + 0.00 sys = 0.02 CPU)
START-UP: 6.79 secs
TAGGING: 0.02 secs
F.EXTRACTION: 0.01 secs SVM: 0 secs PROCESS: 0.01 secs
OVERALL_TIME = START-UP + TAGGING = 6.79 + 0.02 = 6.81 secs

Table 15: Running the SVMTagger.

worry, you can still train, tune and test your system through cross-validation.

tagging strategy It is also important to think of which tagging strategies you're going to use. This may depend for instance on efficiency requirements. If the tagging must be as fast as possible then you should forget about strategies 1, 5, and 6, because strategy 1 goes in two passes and strategies 5 and 6 perform a sentence-level tagging. Strategy 3 is only for unsupervised learning (no hand-annotated data is available). Now, to choose among strategies 0, 2 and 4 the best solution is try them all. However, if you have an idea of the proportion of unknown words you're going to find at tagging time, strategies 2 and 4 are more robust than strategy 0 in the case of unknown words. Finally, if you don't have any speed requirement nor information about future data, in our experiments tagging strategies 4 and 6 systematically obtained best results. In all cases, combining left–to–right and right–to–left tagging improved results. As a remark, the choice for the tagging strategy determines which models must be learned.

feature set First, you must decide how much context you're going to take into account, i.e. the sliding window size. The more context you take the more information the more accurate, the more costly and slow though. Empirically, we've shown that a window size of 5 elements suffices to achieve state–of–the–art results. Second, you must define the feature set, i.e. which features the system is going to consider. The feature set may consist of any word (column 0) and parts–of–speech n-grams (column 1) as well as ambiguity classes and maybe's. Furthermore, lexicalized features may be included (e.g. prefixes, suffixes, capitalization, etc.). Additional information may be provided in further columns, as well.

C parameter This parameter belongs to the soft margin version of the algorithm for training SVMS. It balances the trade-off between the number of misclassified samples and the margin size. A good tuning of this parameter highly increases the sequential tagger performance, so our recommendation is obvious, always adjust the C parameter. See Subsection 6.3 for details.

6.2.2 Tagging Time

Default tagging is fine, although the tuning of some parameters may be helpful:

tagging direction Because languages are different, also the difference in performance between left–to–right and right–to–left directions may be significantly different. In our experiments, a combination of both was always better than any of them in isolation, but, of course, this combination causes the tagger to be twice slower.

weight filtering Very interestingly, we shown that discarding up to 90% of the weights wasn't harmful at all. In fact, discarding about 40-60% of the weight actually results improved, not very significantly though. Also, because tagging speed depends linearly on the sample size (i.e. number of features per sample) the model compression doesn't allow for a higher efficiency.

beam search pruning This applies only for strategies 5 and 6. Because sentence-level tagging intends to maximize the sum of scores (or a product of proabilities through the use of a softmax function, default option) opposite to a greedy on-line tagging, the search space is bigger and so is the computational cost.

backup lexicon Additional morphological information may be provided. This way you can specify better information for words that had been seen at training time, and, specially, information for words that hadn't been seen before. That causes the system to be more robust in front of unknown words.

Table 16 shows a valid configuration file, for training Model 0 on the Wall Street Journal (left-to-right and right-to-left):

Again, efficiency requirements may force you to choose a left-to-right/right-to-left tagging direction instead of a combination of both. A similar reasoning may apply to beam search pruning. In any case, to select among different alternatives our recommendation is that you apply common sense and try them all before discarding any.

6.3 Tuning the SVMTool

It is assumed that you either have a training-validation-test partition of your data, or that you're going to run a cross-validation.

Because you don't have still a precise idea about which tagging strategy / direction will work best, the most sensible choice is to train all models and try all strategies and directions. In general, there are four important adjustments you should have into account:

Feature Engineering Selecting the appropriate feature set to represent the knowledge you want to capture is crucial. If you have some knowledge of the target language and the problem, this may help you to select a

```
TRAINSET = /home/me/WSJ/WSJTP.TRAIN
SVMDIR = /home/me/soft/svmlight/
NAME = WSJTP.912k
R = /home/me/WSJ/WSJ.repair.DICT
W = 5 2
F = 2 100000
X = 3
Dratio = 0.001
REMOVE_FILES = 1

do M0 LRL
```

```
# M0 ambiguous-right [default]

# known word feature set definition
A0k = w(-2) w(-1) w(0) w(1) w(2) w(-2,-1) w(-1,0) w(0,1) w(-1,1) w(1,2)
w(-2,-1,0) w(-2,-1,1) w(-1,0,1) w(-1,1,2) w(0,1,2) p(-2) p(-1) p(-2,-1)
p(-1,1) p(1,2) p(-2,-1,1) p(-1,1,2) k(0) k(1) k(2) m(0) m(1) m(2)

# unknown word feature set definition
A0u = w(-2) w(-1) w(0) w(1) w(2) w(-2,-1) w(-1,0) w(0,1) w(-1,1) w(1,2)
w(-2,-1,0) w(-2,-1,1) w(-1,0,1) w(-1,1,2) w(0,1,2) p(-2) p(-1) p(-2,-1)
p(-1,1) p(1,2) p(-2,-1,1) p(-1,1,2) k(0) k(1) k(2) m(0) m(1) m(2) a(2)
a(3) a(4) z(2) z(3) z(4) ca(1) cz(1) L SA AA SN CA CAA CP CC CN MW
```

Table 16: SVMTlearn configuration file.

good feature set for every model. If you don't have any intuition our recommendation is that you use the default configuration, i.e. don't specify any feature set. Let SVMs do the work for you.

Mapping Filtering	Discarding features appearing only once is always a good choice. Also, keeping the mapping under a convenient size can be very useful. Empirically we found that mappings under a size of 100,000 features succeeded obtaining the best results.
Dictionary Repairing	If you can work on a list of repairings for, let's say, the top-100 most frequent words, that would be excellent. However, in our experiments results with a heuristic ratio around 0.001 were equivalent to results obtained using a manually built list of repairings.
C Parameter	The tuning of this parameter improves dramatically the performance of your system.

The C parameter is by large the most important. Properly tuning it will help us fighting against overfitting by controlling the proportion of misclassified samples allowed. The tuning process is totally greedy. That is, several values of the C parameter are tried and the one yielding the highest accuracy is selected. There will be a C value for known words, and another for unknown words. This may be automatically controlled by means of the configuration file.

If you have enough data you can split it into training, validation (development), and test. In this case the training data will be used to train the models with different values of the C parameter. For every C value models will be tested against the validation file. In the config file in Table 17 there are 4 action items (lines beginning with *'do '*:

- The command *'do M0 LRL CK:0.01:1:3:10:log CU:0.01:1:3:10:log'* performs a tuning of the model 0, both directions LR and RL. The C value for known words (CK) is explored in the range [0.01, 1] in three iterations, exploring 10 values per iteration. The first iteration is logarithmical (it explores only values 0.01, 0.1 and 1). Similarly for the case of unknown words (CU), although the exploration range is different [0.001, 0.1].
- The command *'do M1 RL CK:0.01:1:3:10:***nolog** *CU:0.01:1:3:10:log* **T***'* trains model 1, RL, and differs in two other aspects. First, now the first iteration for known words is not logarithmical. Second, models be tested against the test set after tuning.
- The command *'do M2 LRL CK:0.1 CU:0.08'* does not perform any tuning in the training of model 2, LRL. It directly tries C values 0.1 and 0.08 for known and unknown words, respectively.

```
TRAINSET = /home/me/WSJ/WSJTP.TRAIN
VALSET = /home/me/WSJ/WSJTP.VAL
TESTSET = /home/me/WSJ/WSJTP.TEST
SVMDIR = /home/me/soft/svmlight/
NAME = WSJTP.912k
R = /home/me/WSJ/WSJ.repair.DICT
W = 5 2
F = 2 100000
X = 3
Dratio = 0.001
REMOVE_FILES = 1

# do M0 LRL
do M0 LRL CK:0.01:1:3:10:log CU:0.001:0.1:3:10:log
do M1 RL CK:0.01:1:3:10:nolog CU:0.01:1:3:10:log T
do M2 LRL CK:0.1 CU:0.08
do M4 LR CK:0.1 CU:0.01 T
```

Table 17: SVMTlearn configuration file.

Similarly the command *'do M4 LR CK:0.1 CU:0.01 T'* trains model 4, LR. Additionally, it tests models against the test set.

If you do not have enough data, still tuning may be performed in a cross-validation fashion. See Table 18.

- The command *'do M0 LRL CK:0.01:1:3:10:log CU:0.01:1:3:10:log T:10'* performs a cross-validation style tuning by splitting the training file in 10 folders, using each time 9 different folders to train and the remaining one to test.

As a final remark, the process of tuning a system is very important but it can be extremely time consuming if you're seeking for a masterpiece perfection. Our advice, devoting a week or two to the tuning of the SVMTool is surely necessary and rewarding. Spending more than that time may be possibly exciting but probably wasteful.

Ongoing Work

New strategies to further increase the system effectiveness while guaranteeing robustness and efficiency, are being studied. Unsupervised models are also currently under study.

```
TRAINSET = /home/me/WSJ/WSJTP.TRAIN
SVMDIR = /home/me/soft/svmlight/
NAME = WSJTP.912k
R = /home/me/WSJ/WSJ.repair.DICT
W = 5 2
F = 2 100000
X = 3
Dratio = 0.001
REMOVE_FILES = 1

# do M0 LRL
# do M0 LRL CK:0.01:1:3:10:log CU:0.01:1:3:10:log
do M0 LRL CK:0.01:1:3:10:log CU:0.01:1:3:10:log T:10
```

Table 18: SVMTlearn configuration file.

Another focus of attention is the development of models and strategies to deal with noisy unstructured text. For the moment, we have adapted the tagger to the datasets of the FAUST project[12], which consist of real Internet translation requests, automatic translation candidates and correction feedback provided by the users.

Feedback

Discussion on features and bugs of this software as well as information about oncoming updates takes place on the SVMTool group, to which you can subscribe at `http://groups-beta.google.com/group/SVMTool` and post messages at `SVMTool@googlegroups.com`.

[12] `http://www.faust-fp7.eu`

References

[ASS99] S. Abney, R. E. Schapire, and Y. Singer. Boosting applied to tagging and pp–attachment. In *Proceedings of EMNLP/VLC'99*, 1999.

[Bra00] T. Brants. TnT - A Statistical Part-of-Speech Tagger. In *Proceedings of the Sixth ANLP*, 2000.

[Bri95] E. Brill. Transformation–based Error–driven Learning and Natural Language Processing: A Case Study in Part–of–speech Tagging. *Computational Linguistics*, 21(4), 1995.

[BW98] E. Brill and J. Wu. Classifier Combination for Improved Lexical Disambiguation. In *Proceedings of COLING-ACL'98*, 1998.

[Col02] M. Collins. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the 7th EMNLP Conference*, 2002.

[CST00] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[DZBG96] 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.

[GM03] J. Gimnez and L. Mrquez. Fast and Accurate Part-of-Speech Tagging: The SVM Approach Revisited. In *Proceedings of the Fourth RANLP*, 2003.

[GM04a] Jesus Giménez and Lluis Màrquez. *Recent Advances in Natural Language Processing III: Selected Papers from RANLP 2003*, volume 260 of *Current Issues in Linguistic Theory (CILT)*, chapter Fast and accurate part–of–speech tagging: The SVM approach revisited, pages 153–162. John Benjamins, Amsterdam/Philadelphia, 2004.

[GM04b] Jess Gimnez and Llus Mrquez. Svmtool: A general pos tagger generator based on support vector machines. In *Proceedings of the 4th LREC Conference*, 2004.

[HZD98] H. van Halteren, J. Zavrel, and W. Daelemans. Improving Data Driven Wordclass Tagging by System Combination. In *Proceedings of COLING-ACL'98*, 1998.

[Joa99] T. Joachims. *Making large-Scale SVM Learning Practical.* MIT-Press, 1999.

[LTR00] 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.

[Mih03] Rada Mihalcea. The Role of Non-Ambiguous Words in Natural Language Disambiguation. In *Proceedings of the Fourth RANLP*, 2003.

[MR97] L. Màrquez and H. Rodríguez. Automatically Acquiring a Language Model for POS Tagging Using Decision Trees. In *Proceedings of the Second RANLP Conference*, 1997.

[MRCM99] L. Màrquez, H. Rodríguez, J. Carmona, and J. Montolio. Improving POS Tagging Using Machine-Learning Techniques. In *Proceedings of EMNLP/VLC'99*, 1999.

[NKM01] 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.

[Rat96] A. Ratnaparkhi. A Maximum Entropy Part–of–speech Tagger. In *Proceedings of the 1st EMNLP Conference*, 1996.

[TH99] 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.

[TKM03] 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.

[TM00] 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.

[WSP+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.

[XCPP04] X.Carreras, I. Chao, L. Padr, and M. Padr. Freeling: An open-source suite of language analyzers. In *Proceedings of the 4th LREC Conference*, 2004.