REPUBLIKA SLOVENIJA
MINISTRSTVO ZA VISOKO ŠOLSTVO,
ZNANOST IN TEHNOLOGIJO

BMT: razvoj glasovnega bralnika besedil za mobilne telefone za slepe in slabovidne uporabnike

Odprtokodni oblikoskladenjski označevalnik

Open-source SVM-based Part-of-speech Tagger

Version 1.0

Technical Report

Operacijo sta delno financirala Evropska unija iz Evropskega sklada za regionalni razvoj ter Ministrstvo za visoko šolstvo, znanost in tehnologijo v okviru razvojne prioritete Gospodarsko-razvojna infrastruktura in prednostne usmeritve Informacijska družba v okviru Operativnega programa krepitve regionalnih razvojnih potencialov 2007-2013.

Miha Grčar,
research and development

Nejc Grčar,
development

Ljubljana, 2008

Content

1 Introduction

This report presents an open-source implementation of a part-of-speech (POS) tagger based on Support Vector Machines (SVMs)[1]. We discuss the tagger from several different perspectives as follows:

- In Section 2, the implemented algorithms, both for training and tagging, are presented.
- Several implementation details are covered in Section 3 where we discuss software libraries on which the tagger is based.
- Section 4 discusses licensing of the software components presented in the preceding section.
- Last but not least, in Section 5, we give instructions on how to use the developed command-line utilities.

2 Algorithms

The presented POS tagger needs to be trained on a manually tagged training corpus prior to being employed for tagging. We therefore discuss two separate algorithms in this section: the algorithm for training and the algorithm for tagging.

2.1 Algorithm for Training

In short, the training algorithm loads a tagged corpus (i.e. training corpus) and, by analyzing it, builds the following structures:

- **The dictionary** which contains information about words observed in the training corpus (see Section 2.1.1).
- **The model for predicting tags of known words** (known words are those words in the test corpus that were also observed in the training corpus). This model consists of **several disjunctive multi-class SVM models** being accessed through an index. The details are discussed in Section 2.1.2.
- **The model for predicting tags of unknown words** (unknown words are those words in the test corpus that were **not** observed in the training corpus). In contrast to the model for known words, this model is a single multi-class SVM model, further discussed in Section 2.1.3.

The training algorithm works as follows (illustrated in Figure 1):

1. The dictionary is built. Some words in the dictionary are marked as “hidden”. This process is discussed in Section 2.1.1.
2. The dictionary supports the construction of feature vectors (hence the dashed arrows in Figure 1).
3. The feature vectors are used to train the two models. This is discussed in Sections 2.1.2 and 2.1.3.

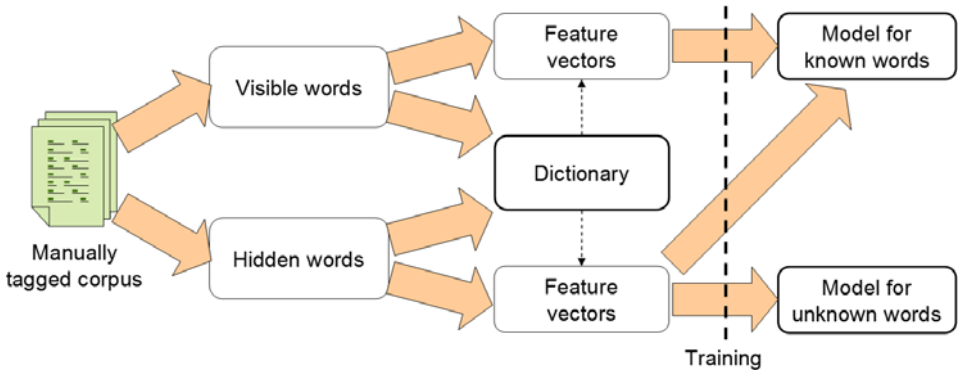

Figure 1: Algorithm for training: the workflow.

[1] http://en.wikipedia.org/wiki/Support_vector_machine

2.1.1 Dictionary

The dictionary contains all words found in the training corpus. Each word is assigned a set of tags; each tag in the set was observed to be assigned to the corresponding word at least once in the training corpus. On the other hand, if a tag is not in the set, it was never assigned to the corresponding word. In addition, several words in the dictionary are marked as "hidden".

To determine hidden words, we partition the training corpus into 10 equal folds. A word is a hidden word if (and only if) it only appears in one of the folds. The same procedure to determine hidden words was applied by Giménez and Márquez [1] in their POS tagger implementation. Hidden words play the role of unknown words and thus simulate situations that occur when processing the test corpus. With this we avoid overfitting the models to the training data.

2.1.2 Model for Predicting Tags of Known Words

As already mentioned, we train several disjunctive multi-class SVM models to predict tags of known words. The process is as follows:

1. By exploring the dictionary, disjunctive tag classes are computed with respect to the following requirements:
   - If any two tags are associated with the same word in the dictionary, they belong to the same disjunctive class.
   - Disjunctive tag classes do not overlap (hence the name).
   - The number of disjunctive classes is maximized.
2. One model is initialized for each disjunctive tag class (illustrated in Figure 2). The model is indexed so that it can be retrieved by using any tag from the disjunctive class as the key.

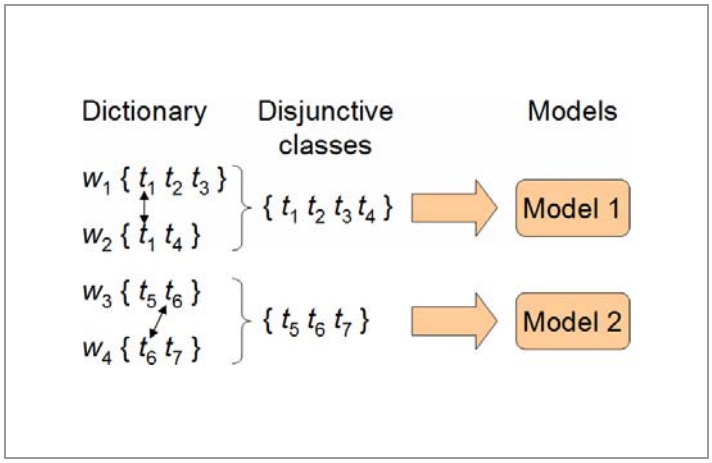

Figure 2: Disjunctive tag classes and the corresponding multi-class SVM models.

3. Each word in the training set is converted into a labeled feature vector. The label is the word's tag and the features are computed from the word's context (surrounding words and tags) and some additional properties (prefixes and suffixes). The feature set that we use is almost identical to the one used in [1] and is summarized in Table 1. The computation of *ambiguity classes* and *maybe* features for right-hand side context words and the current word itself (see [1] for more details) is supported by the dictionary. If the word is hidden, then the set of tags assigned to hidden words is considered as the ambiguity class. If the word is not hidden, the set of tags assigned to that particular word is considered instead.
4. Each feature vector is assigned to the appropriate disjunctive model with respect to its label (i.e. tag). In this manner a subset of the training set is associated with each model.
5. Each training subset undergoes a simple feature selection process. If a certain feature appears in the training subset less than *m* times (*m* is a parameter of the training module), it is removed.
6. Disjunctive models are trained with respect to the associated training subsets. The algorithm used for training is a multi-class SVM classifier with a linear kernel, specifically SVM^{multiclass}, developed by

Thorsten Joachims [2]. When running SVM^{multiclass}, we only set the tradeoff between training error and margin (parameter C) and the termination criterion (parameter Eps); all other settings are left at their default values (see SVM^{multiclass} Web page[2] for details about the settings and their default values). By tuning parameters C and Eps, it is possible to find a suitable balance between training time and model accuracy.

Word features	w_{-3}, w_{-2}, w_{-1}, w_0, w_{+1}, w_{+2}, w_{+3}
Tag features	t_{-3}, t_{-2}, t_{-1}
Ambiguity classes	a_0, a_{+1}, a_{+2}, a_{+3}
Maybe features	M_0, M_{+1}, M_{+2}, M_{+3} (each is a set of features)
Word bigrams	(w_{-2}, w_{-1}), (w_{-1}, w_0), (w_0, w_1), (w_1, w_2)
Tag bigrams	(t_{-2}, t_{-1}), (t_{-1}, a_0), (a_0, a_1), (a_1, a_2)
Word trigrams	(w_{-2}, w_{-1}, w_0), (w_{-1}, w_0, w_1), (w_0, w_1, w_2)
Tag trigrams	(t_{-2}, t_{-1}, a_0), (t_{-1}, a_0, a_1), (a_0, a_1, a_2)
Prefixes	$w_0[1]$, $w_0[1..2]$, $w_0[1..3]$, $w_0[1..4]$
Suffixes	$w_0[n_0]$, $w_0[n_0-1..n_0]$, $w_0[n_0-2..n_0]$, $w_0[n_0-3..n_0]$
Prefix bigrams	($w_{-2}[1]$, $w_{-1}[1]$), ($w_{-1}[1]$, $w_0[1]$), ($w_0[1]$, $w_1[1]$), ($w_1[1]$, $w_2[1]$), ($w_{-2}[1..2]$, $w_{-1}[1..2]$), ($w_{-1}[1..2]$, $w_0[1..2]$), ($w_0[1..2]$, $w_1[1..2]$), ($w_1[1..2]$, $w_2[1..2]$), …, ($w_{-2}[1..4]$, $w_{-1}[1..4]$), ($w_{-1}[1..4]$, $w_0[1..4]$), ($w_0[1..4]$, $w_1[1..4]$), ($w_1[1..4]$, $w_2[1..4]$)
Suffix bigrams	($w_{-2}[n_{-2}]$, $w_{-1}[n_{-1}]$), ($w_{-1}[n_{-1}]$, $w_0[n_0]$), ($w_0[n_0]$, $w_1[n_1]$), ($w_1[n_1]$, $w_2[n_2]$), ($w_{-2}[n_{-2}-1..n_{-2}]$, $w_{-1}[n_{-1}-1..n_{-1}]$), ($w_{-1}[n_{-1}-1..n_0]$, $w_0[n_0-1..n_0]$), ($w_0[n_0-1..n_0]$, $w_1[n_1-1..n_1]$), ($w_1[n_1-1..n_1]$, $w_2[n_2-1..n_2]$), …, ($w_{-2}[n_{-2}-3..n_{-2}]$, $w_{-1}[n_{-1}-3..n_{-1}]$), ($w_{-1}[n_{-1}-3..n_{-1}]$, $w_0[n_0-3..n_0]$), ($w_0[n_0-3..n_0]$, $w_1[n_1-3..n_1]$), ($w_1[n_1-3..n_1]$, $w_2[n_2-3..n_2]$)
Prefix trigrams	analogous to prefix bigrams
Suffix trigrams	analogous to suffix bigrams

Table 1: The feature set.

2.1.3 Model for Predicting Tags of Unknown Words

The model for predicting tags of unknown words is simpler than that for predicting tags of known words in the sense that it is in fact one single multi-class SVM model. The training process is as follows:

1. Only hidden training words are considered (in the training corpus, they play the role of unknown words). Each hidden word is converted into a labeled feature vector by using the exact same procedure as when training the model for known words (see Section 2.1.2, item 3).
2. The training set undergoes a feature selection process. If a certain feature appears in the training set less than *m* times (*m* is a parameter of the training module), it is removed.
3. The model is trained with respect to the training set. The algorithm used for training is a multi-class SVM classifier with a linear kernel. See Section 2.1.2, item 6, for more details.

2.2 Algorithm for Tagging

The tagging algorithm loads the two models, the dictionary, and an untagged corpus. It predicts a tag for each

[2] http://svmlight.joachims.org/svm_multiclass.html

word in the untagged corpus. The tagging process involves the following steps (illustrated in Figure 3):

1. The word in question is converted into a feature vector. The dictionary supports this task by providing information required to determine word features. The feature vector generation process is the same as in the training phase. The only difference is that unknown words (those not found in the dictionary) are treated in the same way as hidden words: the set of tags assigned to hidden words is considered as the ambiguity class.

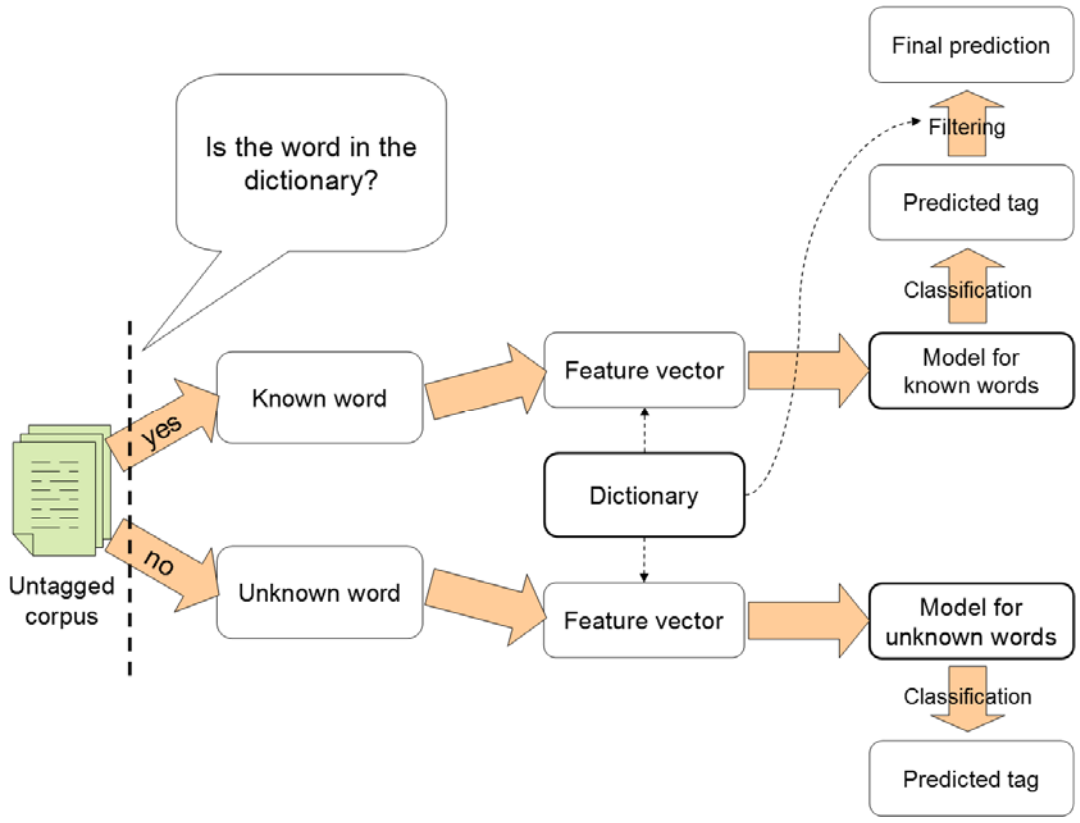

Figure 3: Algorithm for tagging: the workflow.

2. The appropriate model is employed for prediction:
   - If the word exists in the dictionary, the model for known words is employed for prediction. The initial prediction is a list of tags ordered from the most to the least probable tag (according to the model). This vector is filtered with respect to the dictionary entry for the word in question: if a tag from the vector does not occur in the set of tags associated with the word, the tag is removed from the vector.
   - On the other hand, if the word does not exist in the dictionary, the model for unknown words is employed for prediction. Additional filtering is not possible in this case.

3 Implementation

The tagger consists of two command-line applications, specifically SvmPosTaggerTrain and SvmPosTaggerTag, built on top of a stack of software libraries providing the required functionality. This “hierarchy” of software components is shown in Figure 4. In the following subsections we briefly discuss each of these components.

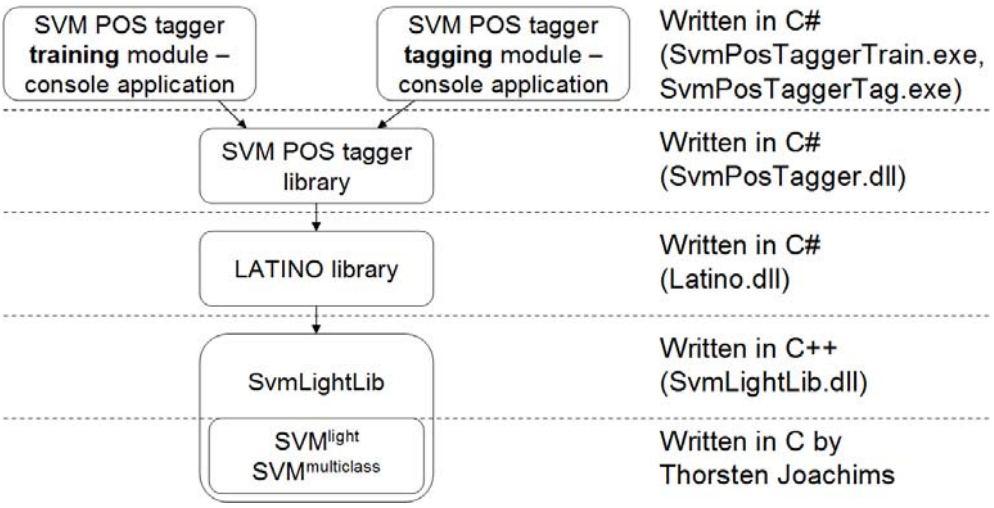

Figure 4: Software components that underlie the tagger and their hierarchical relationship.

3.1 SvmLightLib

SvmLightLib is a software library based on Thorsten Joachims' SVM^{light} [3] and SVM^{multiclass} [2]. The original source code, a set of command-line utilities, was modified to provide its (high-level) functionality through a DLL interface. In short, SvmLightLib has the following properties:

- It is written in C++.
- It is easy to use, much like the Thorsten's command-line utilities.
- It supports working with several SVM models in the same process space.
- It is thread-safe. The original code was modified so that all global variables were removed. It is thus possible to train several models in parallel (e.g. in different threads).
- It supports saving models in a binary file format (smaller in size, faster to load).
- It is meant to be used from managed environments (e.g. C#).

3.2 LATINO Software Library

LATINO (Link Analysis and Text Mining Toolbox) is a software library for machine learning, text mining, data visualization, and network analysis tasks. It is implemented in C# as a class library (i.e. managed DLL) and apart from its native modules also exposes functionalities of several C/C++ software libraries through a set of C# interfaces. Currently (October 2008), only the part of LATINO needed by the tagger (i.e. several basic data structures and a part of the machine learning module) is publicly available. The library requires .NET Framework 2.0.

3.3 SVM POS Tagger Library

This library implements the high-level functionality required by the tagger. Specifically, it implements the dictionary, corpus, and model data structures. On top of this library, a fully functional SVM-based POS tagger can be implemented with only a few lines of code as demonstrated with the implemented console applications for tagging. The library requires .NET Framework 2.0.

3.4 Console Applications

Two console applications are implemented on top of the stack: the application for training POS models (SvmPosTaggerTrain.exe) and the application for tagging (SvmPosTaggerTag.exe). The two applications are used as explained in Section 5. Note that the entire stack of libraries discussed in Section 3 (i.e. SvmLightLib.dll, Latino.dll, and SvmPosTagger.dll) is required to run these applications. In addition, .NET Framework 2.0 is required.

4 Licensing Information

This section presents license agreements corresponding to the software components discussed in Section 3.

4.1 SvmLightLib

The license is as follows:

```
“
 SvmLightLib.dll consists of three tightly integrated parts:
* SVM^light, by Thorsten Joachims
* SVM^multiclass (a derivative of SVM^struct), by Thorsten Joachims
* The DLL wrapper, by Miha Grcar

Authors:
* Thorsten Joachims
             thorsten@joachims.org
             Cornell University
             Department of Computer Science
             4153 Upson Hall
             Ithaca, NY 14853
             USA
* Miha Grcar
             Department of Knowledge Technologies
             Jozef Stefan Institute
             Jamova cesta 39
             1000 Ljubljana
             Slovenia

LICENSING TERMS

This library is granted free of charge for non-commercial research and education purposes.
However you must obtain a license from the author or SVM^light and SVM^struct (Thorsten
Joachims) to use it for commercial purposes.

Scientific results produced using the software provided shall acknowledge the use of
SVM^light and SVM^struct. Please cite as

* I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, Large Margin Methods for
Structured and Interdependent Output Variables, Journal of Machine Learning Research
(JMLR), 6(Sep):1453-1484, 2005.
http://jmlr.csail.mit.edu/papers/volume6/tsochantaridis05a/tsochantaridis05a.pdf
* T. Joachims, Making Large-scale SVM Learning Practical. Advances in Kernel Methods -
Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999.
http://www-ai.cs.uni-dortmund.de/DOKUMENTE/joachims_99a.pdf

Moreover shall the author of SVM^light and SVM^struct (Thorsten Joachims) be informed
about the publication.

The software and derivatives of the software must not be distributed without prior permis-
sion of the author of SVM^light and SVM^struct (Thorsten Joachims).

By using SvmLightLib you agree to the licensing terms.

NO WARRANTY

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO
THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPY-
RIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY
KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
```

```
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY
AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU AS-
SUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT
HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM, BE LIABLE TO
YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARIS-
ING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A
FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER
PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
```

4.2 LATINO, SVM POS Tagger Library, and the Console Applications

5 Usage Instructions

5.1 Installing the Tagger

To install the tagger, follow this procedure:

1. Unpack SvmPosTaggerOct08.zip into a folder, say C:\SvmPosTagger.
2. Download .NET Framework 2.0 (if you don't have it already installed on your machine) from the following URL:
   - http://www.microsoft.com/downloads/details.aspx?familyid=0856eacb-4362-4b0d-8edd-aab15c5e04f5&displaylang=en
3. Install .NET Framework: execute dotnetfx.exe and follow the instructions provided by the setup process.
4. You are now ready to run the tagger from C:\SvmPosTagger\Release. Note: if you wish to run the tagger from elsewhere, you need to put folder C:\SvmPosTagger\Release into the PATH environment variable.

5.2 Running the Tagger

5.2.1 Training

If you execute SvmPosTaggerTrain.exe without specifying any arguments, you will be provided with the following instructions:

```
*** SVM POS Tagger 1.0 - Training Module ***

Usage:
SvmPosTaggerTrain [<options>] <corpus_file_name> <model_file_name>

<options>:          See below.
<corpus_file_name>: Tagged corpus for training (input).
<model_file_name>:  Trained model (output).

Options:
-v             Verbose.
               (default: not set; quiet mode)
-c:<float>0>   SVM parameter; tradeoff between error and margin.
               (default: 5000)
-eps:<float>0> SVM parameter; termination criterion.
               (default: 0.1)
-mff:<int>0>   Minimum feature frequency.
               (default: 2)
```

For your convenience, two sets of training and test corpora are available in the Release folder. We suggest you

use the "orwell_1" corpus to get familiar with the tagger. To build a POS model, execute the following command:

```
C:\SvmPosTagger\Release> SvmPosTaggerTrain.exe -v orwell_1_train.txt orwell_1_model.bin
```

After a while (you will be able to follow the training progress), three new files will appear in C:\SvmPosTagger\Release. These are the models built by the training module. One of the files bears the name specified above (orwell_1_model.bin); the other two have more cryptic names. All three files are required to invoke tagging (see Section 5.2.2). Training on other datasets may result in even more files, depending on how many disjunctive tag classes are discovered in the training corpus (see Section 2.1.2). Note: the files with cryptic names are not overwritten if you re-run SvmPosTaggerTrain with the same output file name setting. Instead, new files with unique names are created. You therefore need to delete obsolete files manually.

Parameters –c and –eps allow the user to set a suitable balance between training time and accuracy. Decreasing –c while increasing –eps will reduce the training time but also the accuracy. For example, if –c is reduced from 5000 to 500 and –eps is increased from 0.1 to 1.0, the training time will decrease for 65%[3] and the accuracy will fall from 98.23% / 87.11% to 97.63% / 75.33%[4]. To better understand the meaning of these two parameters, please consult the SVM^{multiclass} Web page[5].

Increasing –mff will reduce the number of features taken into account and at some point result in smaller models at the expense of lower accuracy. For example, if –mff is increased from 2 to 50, the size of the model will decrease from 222 MB to 104 MB and its accuracy will fall from 98.23% / 87.11% to 98.08% / 83.33%.

5.2.2 Tagging

Tagging can be employed yet after the models were built. If you execute SvmPosTaggerTag.exe without specifying any arguments, you will be provided with the following instructions:

```
*** SVM POS Tagger 1.0 - Tagging Module ***

Usage:
SvmPosTaggerTag [<options>] <corpus_file_name> <model_file_name>
  <tagged_corpus_file_name>

<options>:                See below.
<corpus_file_name>:       Corpus for tagging (input).
<model_file_name>:        Trained model (input).
<tagged_corpus_file_name>: Tagged corpus (output).

Options:
-v Verbose.
   (default: not set; quiet mode)
```

In order to tag the "orwell_1" test corpus by employing the model trained in the previous section, execute the following command:

```
C:\SvmPosTagger\Release> SvmPosTaggerTag.exe -v orwell_1_test.txt orwell_1_model.bin
  orwell_1_tagged.txt
```

If the test corpus is tagged (orwell_1_test is), the tagger will report the accuracy on known words, accuracy on unknown words, and overall accuracy. The predicted tags will be written into the specified output file (orwell_1_tagged.txt in the above example).

---

[3] From 632 to 224 seconds on Intel Core Duo at 2.4 GHz, 2 GB RAM, Windows Vista.

[4] Accuracy on known words / Accuracy on unknown words

[5] http://svmlight.joachims.org/svm_multiclass.html

References

[1] Giménez, J. and Márquez, L. (2004): SVMTool: A General POS Tagger Generator based on Support Vector Machines. In Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04). Lisbon, Portugal.

[2] Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005): Large Margin Methods for Structured and Interdependent Output Variables. In Journal of Machine Learning Research (JMLR), 6 : 1453–1484.

[3] Joachims, T. (1999): Making Large-scale SVM Learning Practical. In Advances in Kernel Methods – Support Vector Learning, Schölkopf, B., Burges, C., Smola, A. (eds.), MIT Press.