Support Vector Machines for Part-of-Speech Tagging

Annika Nietzio
Ruhr-Universität Bochum
Annika.Nietzio@ruhr-uni-bochum.de

Abstract
This paper explores the possible application of Support Vector Machines to part-of-speech tagging. I adopted the approach suggested by Collins and Duffy (2002) who employ another kernel-based method – the voted perceptron algorithm – for a different tagging problem – named entity extraction. The objective of this work is to present an implementation of an SVM-Tagger using the software SVMlight (Joachims, 1998).

Keywords: Machine Learning, Part-of-Speech Tagging, Support Vector Machine, Kernel Function

1 Introduction
Several machine learning techniques have been applied to the task of part-of-speech tagging. There are rule based approaches like the Brill Tagger (Brill, 1995) and statistical taggers using Hidden Markov Models (Manning and Schütze, 1999, chap. 10). These approaches take into account only limited information about the examined objects while kernel-based algorithms allow to represent complex linguistic objects like word-tag-sequences, as arbitrary feature vectors, thus keeping more information about them.

1.1 Support Vector Machines
A Support Vector Machine (SVM) (Burges, 1998) is a learning machine for binary classification. The training instances consist of a vector and a label \((x_i, y_i) \in \mathbb{R}^n \times \{+1, -1\}, i = 1, \ldots, l\). The algorithm tries to find a separating hyperplane \(\langle w, x \rangle + b = 0\) with maximal margin.

The margin is defined as the minimal distance of the hyperplane to any of the training points. The optimal hyperplane can be found by solving the following optimization problem:

\[
\begin{align*}
\text{minimize} & \quad \|w\|^2 \\
\text{subject to} & \quad y_i (\langle w, x_i \rangle + b) \geq 1 \\
& \quad i = 1, \ldots, l
\end{align*}
\]

The Lagrangian formulation of the problem leads to the dual problem. The maximal margin hyperplane can be expressed as a linear combination of the training examples

\[
w = \sum_{i=1}^{l} \alpha_i y_i x_i
\]

The parameters \(\alpha_i\) are called the dual parameters. The training examples \(x_i\) with \(\alpha_i \neq 0\) are the Support Vectors. In this formulation of the optimization problem the training data only occurs in inner products of the feature vectors. This allows the use of kernel functions (cf. section 1.2). Another advantage of the dual formulation is that the resulting problem can be solved by a convex quadratic programming algorithm which always finds the global optimum, i.e. the hyperplane with maximal margin. (Cristianini and Shawe-Taylor, 2000, chapter 5).

1.2 Kernel Functions
In some cases no separating hyperplane can be found in the given vector space. The application of a feature map \(\phi : \mathbb{R}^n \rightarrow \mathbb{R}^N\) projecting the training points into a high dimensional feature space where the data is linearly separable often leads to increased computational complexity referred to as Curse of dimensionality.
SVMs can handle such high dimensional feature spaces because they employ kernel functions. A kernel function provides a way to compute efficiently the inner product of two vectors in the feature space without explicit calculation of the feature mapping.

\[ K(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle \]  

A characterization of kernel functions is given in (Cristianini and Shawe-Taylor, 2000).

**Theorem 1 (Mercer’s Condition)**

Let \( K(x_1, x_2) \) be a symmetric function on a finite input space \( X = \{ x_1, \ldots, x_l \} \). Then \( K \) is a kernel function if and only if the matrix

\[ K = (K(x_i, x_j))_{i,j=1}^l \]

is positive semi-definite (has non-negative eigenvalues).

2 Tagging as Classification Problem

As the previous section shows, the main application of SVMs is classification. This section addresses the task of (re-)formulating the tagging problem as a binary classification problem\(^3\) which can be solved by an SVM.

For tagging a sentence one has to generate several tag sequences and then classify them. Given a training example \( \{ s_i, t_i \} \), where \( t_i \) is the correct tag sequence for \( s_i \), the learner generates a set of candidates for \( s_i \)

\[ C(s_i) = \{ x_{i1}, x_{i2}, \ldots, x_{im} \} \]

The two labels are correct and wrong.\(^4\) Since there are usually more than two candidate tag sequences for one word sequence this leads to problems. This approach doesn’t take into account that for one sentence there is only one correct tag sequence (except for true ambiguities) as the candidates are classified independently.

It is more appropriate to model the selection of the correct tag sequence as a ranking problem. The classifier calculates a score for every candidate and selects the one with the highest value. The score for a candidate \( x_{ij} \) is defined as

\[ \text{score}(x_{ij}) = \langle w, f(s_i, x_{ij}) \rangle \]

where \( f \) is a function mapping a word-tag-sequence to a vector in \( \mathbb{R}^d \) (cf section 3.1). The output of the classifier for a sentence \( s \) is

\[ \arg \max_{x \in C(s)} \langle w, f(s, x) \rangle \]

Collins and Duffy (2002) propose a way to transform a ranking problem into a classification problem. Assuming that the first candidate is the correct one \( (x_{ii} = t_i) \), i.e. that it has the highest score, yields \( \text{score}(x_{ii}) > \text{score}(x_{ij}) \). This results in the following conditions for the optimization problem:

\[ \langle w, f(s_i, x_{ii}) \rangle - \langle w, f(s_i, x_{ij}) \rangle > 0 \]

\[ \forall 1 \leq i \leq l, \ 2 \leq j \leq i_m \]

The bilinearity of the inner product gives

\[ \langle w, f(s_i, x_{ii}) \rangle - f(s_i, x_{ij}) \rangle > 0 \]

Thus the positive training examples can be represented as

\[ (f(x_{ii}) - f(x_{ij}), +1) \]

In this approach it is not necessary to produce negative examples like \( (f(x_{ii}) - f(x_{ii}), -1) \) which would contain redundant information. Instead the hypothesis class is restricted to unbiased hyperplanes \( \langle w, x \rangle = 0 \) (i.e. the bias \( b \) is 0), so the margin maximization can be performed with just the positive examples.

3 Implementation

For the experiment I used the software SVM\(^\text{light}\) (Joachims, 1998). The program was written to improve SVM learning with many training examples and support vectors. It has several built-in kernel functions but allows the user to define new ones as well. The provided kernels can only be applied to the elements of an euclidian vector space \( \mathbb{R}^d \).\(^5\) For discrete structures like word-tag-sequences another type of kernel functions

\[ K(x_1, x_2) = (\langle x_1, x_2 \rangle + c)^d \]

\(^3\)Another possibility is to consider PoS-Tagging as a multiclass classification problem. This approach is presented by Nakagawa et al. (2001) who employ the one-versus-rest method. I.e. there is one (binary) classifier \( f_i \) for each category. The tagger selects the tag corresponding to the classifier that yields the highest distance value \( y = \arg \max f_i(x) \).

\(^4\)It is not possible to model directly the process of assigning tags to words with just two classification labels.

\(^5\)E.g. the polynomial kernel \( (c = \text{const.}, d \geq 2) \)

\[ K(x_1, x_2) = (\langle x_1, x_2 \rangle + c)^d \]
called convolution kernels is employed. I implemented such a kernel to be used with SVMlight (cf 3.1).

First I created the training examples for SVMlight from a correctly annotated corpus according to the procedure explained in section 2. This involved generating candidate tag-sequences for a sentence with a simple tagger (cf section 3.2). Then I applied the learner of SVMlight to the training data. This results in a model that correctly classifies the training examples.

The tagger works as follows:

1. Read input sentences.
2. Generate candidates for the sentence.
3. Classify the candidates according to the model. (This is a functionality of SVMlight.)
4. Select the highest scoring candidate as tag sequence for the sentence.

3.1 Calculating the kernel function

The main idea is that a sentence can be described by several attributes or features corresponding to the dimensions of the vector space. The value of an attribute is the frequency of its occurrence in the word-tag-sequence.

Collins and Duffy (2001) propose the following template: A feature is a sequence of part-of-speech tags. The tags can have the corresponding words attached to them. The sentence \( \text{Sie/PPE} \ \text{list/VVF} \ \text{ein/ART} \ \text{Buch/NN} \ \text{.} \) is described by features like

- PPER VVF\_ART
- list/VVF\_ART Buch/NN
- ein/ART Buch/NN \text{.} \$

In the vector representation of a word-tag-sequence each dimension corresponds to such a feature. As the number of features per sentence is exponential in the number of tokens, the calculation of the inner product would take exponential time. To overcome this, a convolution kernel can be used that calculates the similarity (i.e. the number of common subsequences) directly from the word-tag-sequences in polynomial time. Figure 1 shows an adaption of the kernel for paired sequences proposed in (Collins and Duffy, 2002) as it was used in the implementation.

\[
K(s_1, s_2) = \sum_{n_1 \in N_1 \atop n_2 \in N_2} C(n_1, n_2)
\]

/* Recursive calculation of \( C(n_1, n_2) \) */
if (label(n_1) \neq \text{label}(n_2))
then \( C(n_1, n_2) = 0 \)
else if (word(n_1) \neq \text{word}(n_2))
then \( C(n_1, n_2) = 1 + C(\text{next}(n_1), \text{next}(n_2)) \)
else \( C(n_1, n_2) = 2 + 2 \cdot C(\text{next}(n_1), \text{next}(n_2)) \)
fi

Figure 1: Kernel function for two word-tag-sequences \( s_1 \) and \( s_2 \). \( N_i \) is the set of nodes (i.e. word-tag-pairs) in sentence \( s_i \). The functions \( \text{label}(n_i) \) (resp. \( \text{word}(n_i) \)) return the tag (resp. word) at position \( i \), \( \text{next}(n_i) \) moves to the next node.

3.2 Generating the Candidates

A simple way to generate candidates is to build up all the possible tag sequences for a sentence. A lexicon containing all the tags for each entry word can easily be created from the training corpus. Alternatively any other existing lexicon can be used. This approach leads to huge candidate sets. E.g. if a sentence contains three ambiguous words each having two different tags the tagger generates \( 2^3 = 8 \) candidates. This could be avoided by using an initial state tagger with more linguistic knowledge that generates only the most probable candidates.\(^6\)

4 Evaluation and Conclusion

First tests on a small training corpus\(^7\) showed that the presented approach is feasible (cf table 2). As a next step further experiments have to be made to evaluate the performance of the tagger on larger corpora. From the results it is not clear whether more training data leads to higher accuracy. It should be explored how the results compare to the results of other taggers and how the handling of unknown words can be improved.

\(^6\)A further application of the SVM-Tagger could be the use as a postprocessing tool to improve the output of other taggers.

\(^7\)For the evaluation I used the German NEGRA corpus provided by the Saarland University in Saarbrücken. \url{http://www.coli.uni-sb.de/sfb378/negra-corpus}
<table>
<thead>
<tr>
<th>sentences</th>
<th>tokens</th>
<th>hypothesis class</th>
<th>test A</th>
<th>test B</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>4784</td>
<td>unbiased hyperplane</td>
<td>96.1%</td>
<td>-</td>
</tr>
<tr>
<td>1000</td>
<td>9313</td>
<td>unbiased hyperplane</td>
<td>97.9%</td>
<td>93.9%</td>
</tr>
<tr>
<td>2000</td>
<td>18155</td>
<td>unbiased hyperplane</td>
<td>96.7%</td>
<td>91.9%</td>
</tr>
<tr>
<td>500</td>
<td>4784</td>
<td>biased hyperplane</td>
<td>98.4%</td>
<td>92.9%</td>
</tr>
</tbody>
</table>

Table 2: Results for the SVM-Tagger. Test A consists only of words that already occurred in the training corpus, while test B contains unknown words.

Further linguistic questions to be answered are how the initial state tagger and the choice of the lexicon affect the results. It would be very interesting to take a closer look at the support vectors\(^8\) to find out if they have a special structure.

From a computational perspective it should be evaluated how the use of a more complex hypothesis class (biased hyperplanes) affects the performance.

**References**


\(^8\)The support vectors are the most important training examples since they contribute the main information about the position of the hyperplane in the learning process.