Statistical Part-of-Speech Guessing for German: Support Vector Classifiers versus Voting

David Reitter
University of Potsdam
Dept. of Linguistics / Applied Computational Linguistics
P.O. Box 601553 / D-14415 Potsdam / Germany
reitter@ling.uni-potsdam.de

Abstract

In this paper, I present a statistical-based approach to the part-of-speech guessing problem. I see assigning a part-of-speech, such as *Adjective* or *Noun*, as a classification problem. My guessing framework, which relies on automated learning of a language model, is described in detail. The rich feature analysis presented is suitable for linguistic data, such as the ones observed in German. I use a large margin classifier learning algorithm to select relevant features and learn appropriate labelling. The system is evaluated using a German corpus.

1 Introduction

Part-of-speech guessing algorithms are designed to assign one or more possible part-of-speech categories, such as finite-verb, adjective or common noun, to a given word not yet contained in a given lexicon.

Such a task must be accomplished during part-of-speech-tagging, a process that assigns unambiguous part-of-speech tags to the words of a given sentences. Because a word usually has a variety of part-of-speech categories, tagging means much more than merely looking up the words in a lexicon. The actual category depends on the syntactic (and sometimes semantic) context. So, tagging performs morpho-syntactical disambiguation. Tagging algorithms usually access a lexicon to first assign ambiguity classes (several categories) to each word, thus reducing search space. Then, an optimal solution for the assignment is found with the help of a previously learned statistical language model or defined or learned rules.

A good lexicon is vital to the success of the task. Usually, full form lexicons are used; they can be acquired through a corpus that is initially tagged with an existing P.O.S.-tagger and then validated manually. However, the lexicon will never contain all words found in a text. Increasing the lexicon size will not improve the situation much: Following Zipf's law[1] (Brill 1995), increasing the lexicon size will not result in a notable gain in lexicon performance on unrestricted text. There are relatively few high-frequent words and increasingly many words when I look at low-frequent entries in my lexicon. In the Brown corpus, 44 percent of all words occur only once.[2] Thus, part-of-speech-guessing algorithms are needed in order to replace huge parts of the lexicon. These methods will also cover new word-formations which can never be included in a static lexicon.

In the following, I will first look at previous approaches and evaluate them on a theoretical basis with regard to a rich inflectional system such as the one of German. This will lead us to types of features we need to identify when guessing the part-of-speech category of a word. I will then apply the Support Vector Machine technique to this problem in order to implement a supervised learning mechanism. The method is evaluated on a lexicon derived from the German NEGRA corpus.

2 Common Guessing Approaches

All guessing approaches described in the following use a form of rules, which are triggered by word affixes. *Morphological* rules refer to an entry in a lexicon. They account for morphological alternation, such as inflection or derivation. *Non-Morphological* rules do not refer to any lexicon entries. Thus, they can also cope with newly invented words or spelling mistakes.

Eric Brill describes a transformation-based tagger and a transformation-based guessing mechanism. His tagger and guesser are error-driven, i.e. the rules employed in his algorithms revise in multiple iterations the previous assignment of arbitrarily chosen tags. For the guesser, this means that all unknown words may be, e.g., first tagged as 'common noun' (which is the most frequent unknown word). This category is then changed depending on certain preconditions stated in the rules. The rules are learned from a corpus; they are instantiations of the following templates:

- Deleting the prefix/suffix x, $|x| < 4$ results in a known word.
- The first/last n ($1 < n < 4$) characters of the word are x.
- Adding the character string x as a prefix/suffix results in a word ($|x| < 4$).

[1]Zipf's law: When we put our lexicon in a ascending order according to word frequency, the rank of an element divided by its occurrence frequency is constant.

[2]Cited after Daciuk et al. (1998).

Learning is implemented as follows: The initial corpus is annotated using the initial algorithm. Then, the automatically assigned tags are compared to the true ones from the corpus. The learner instantiates the transformation templates to construct transformations that correct the mistakes made by the previous annotator. Then, the transformations are scored; best-scoring ones are chosen and applied. Then, the comparison to the truthful corpus data starts again, until no other successful transformation is found.

Brill's guessing algorithm accounts for morphological alternation such as the -ly ending in English adverbials. It is trained on a corpus and learns suffixes (word endings) that are related to an already known word or simply have a certain ending.

Andrei Mikheev extends this technique with a mutative segment in the rules (Mikheev 1997). Translated to one of Brill's conditions, it means assigning a given tag, if

- Deleting the prefix/suffix x and adding the character string y as prefix/suffix results in a known word.

While Brill's guesser needs to find a lexicon entry with the isolated stem (which is always possible in English), Mikheev's mutative segment is able to exchange segments of an input word before looking it up in the lexicon.

Mikheev also describes a supervised learning algorithm. He acquires rules similar to the ones Brill suggested. The guessing algorithm is non-statistical. Mikheev applies ending-guessing rules, which determine a word's category by identifying its suffix. Besides, Mikheev applies *morpholocial* rules, which refer to a lexicalized known part of the word. For example

```
AˆP: [ |- un ?(VBD VBN) -> (JJ)]
```

states that if an unknown word carries the prefix 'un', and segmenting this prefix from it results in a lexicalized word of the category *VBD* or *VBN*, the unknown word is an adjective. These morphological rules are reported to increase the guessing accuracy by about two percent.

Other approaches exist, such as Dermatas & Kokkinakis (1995). This probabilistic solution is based solely on so-called hapax words – those words which occur only once in the corpus. Mikheev, in contrast, learns only from non-hapax words. Schmid (1994) presents a statistical-based suffix learner. Daciuk et al. (1998) is concerned with a representation of lexical material (in particular: affix lists), that is both space- and time-efficient. He presents an algorithm to create a minimal automaton from the lexicon. Daciuk uses this representation for guessing in a memory-based approach. Speaking on the level of morphology and ignoring the pruning process, Daciuk tries to identify the longest affix that can be segmented away from the word and that had been found previously in the training corpus. He returns all the tags of the training words with the given suffix or any longer one.

Rule-based approaches provide good results for English. With huge training corpora, Mikheev reports 95.2 percent recall, 85.2 percent precision in each assignment of a tag.

Brill's and Mikheev's guessers return a set of tags for a given word. No probability measure is given for the tags; the algorithms aim at reaching high recall (all acceptable tags are included) and reasonably good precision (the tags proposed are truly the ones of the word).

Modern statistical tagging algorithms such as the one in Brants (2000), however, let probabilities on lexical level contribute to their final probability estimate. Consequently, they will theoretically perform better with a known probability measure for each tag that is assigned to a word by the lexicon unit.

Moreover, there are several issues connected with the morphological structure of German, which suggest a modification of the approaches depicted before.

3 Particularities of Guessing for German

Several linguistic phenomena in German suggest a modification of the guessing strategies described above. German morphological alternation comprises deviation, composition as well as morpho-syntactical inflection, such as case or number marking.

- Inflection via suffixes is not monotonous, meaning, it will not only add a marker to the lexical item, but do so just after removing some inflectional suffix. In other words: the pure stem form does not occur in a corpus, or, only occurs with low frequency. Example:

  (1) *Ich schwimme, du schwimmst, er schwimmt.*
  I swim, you swim, he swims.

  (2) *?schwimm*
  swim-

  (2) is marked, because it will only occur as imperative or part of a compound, which is, for many verbs, very rare in a corpus.

  Brill's rules cannot represent these changes. Mikheev's rules account for them with the specification of a *mutative segment* in the rules.

- Inflection in German (and other languages) is in another way not monotonous. Stem-Alternation occurs on a regular basis in so-called strong verbs, such as in

  (3) *ich lade, du lädst, er lädt [sie ein]*
  I invite, you invite, he invites [her]

Are these alternations used for productive categories as well? Nouns are certainly more productive than verbs. So, for a guessing algorithm, it would be interesting whether vowel-shift occurs also for nouns. Look at the following data:

(4) *das Stadtjugendhaus,*
the municipal youth house,
die Stadtjugendhäuser
the municipal youth houses

Plural formation makes heavy use of regular vowel shift.
Mikheev's morpholocial rules cannot deal with this property; Daciuk's automaton ignores the stem of a word in most cases.[3]

- Compounds are a very productive morphological word-class in German, i.e. Germans love to create new words. In contrast to English, composition is not clearly marked (as, e.g., with a dash):

  (5) *der Mitsubishifahrer*
  the Mitsubishi driver

- As in many languages spoken in western cultures, new lexical material (many nouns, some adjectives) is integrated, mainly from the English language.

  (6) *Pullover mit* ***Zip***
  pullover with zip
  zip pullover.

  (7) *Du musst den Rechner neu* ***booten****.*
  You need the computer anew boot.
  You need to reboot the computer.

  Obviously, none of the common guessing methods is able to cope with words from varying languages. Usually, as in (7), verbs are integrated into the inflectional system of the host language. Then, ending-guessing rules will be able to deal with foreign words. (6) can be analyzed, because it is capitalized.

4 A Statistical Approach to Guessing

The statistical algorithm I will present in the following involves a language model that can encode several relevant features.

To combine these features, I will discuss the use of a set of Support Vector Machines for classification and compare my results to the ones with a much simpler voting system.

My language model is acquired by supervised learning. The algorithm returns, in contrast to the approaches shown before, a vector containing a probability for the assignment of each tag.

[3]This challenges Daciuk's data of German ("eingeladet", Daciuk et al. (1998), pp. 24).

4.1 Features in categorization

Statistical categorization, such as part-of-speech guessing, involves finding a class assumption about a class of given data C_w. This estimation is found by observing the occurrence of one or more features in the data (f_1 ... f_n); usually, binary features with the values 0,1 are used. I will indicate the positive occurrence of a feature f as event (or *feature predicate*) $F_f \rightarrow f(w) = 1$.

A straightforward, probabilistic approach would be to find the conditional (maximum-likelihood) probability $\hat{P}(C_w|F_1 \cap F_2 \cap ...F_n)$ for all classes and choose the most probable one(s). Note that we can use use Bayes' formula to turn around the order of dependence between events:

$$\hat{P}(C|F) = \frac{\hat{P}(F|C)\hat{P}(C)}{\hat{P}(F)}$$

$\hat{P}(C)$ and $\hat{P}(F \cap C)$ can be easily estimated in a corpus, and $\hat{P}(F|C) = \frac{\hat{P}(F \cap C)}{\hat{P}(C)}$. We still need to observe F. But this is hardly possible - we cannot observe the conjunction of F_1, F_2, ... F_n in the training corpus with significant frequency. An approach adopted by many researchers is to approximate it through the chain formula

$$P(\bigcap_j F_j|C) \approx \prod_j \hat{P}(F_j|C)$$

$\hat{P}(F_j)$ is observed in the training corpus. However, this approach assumes that all the features F_i combined to B using the chain formula are independent of each other. This assumption does not hold for the kind of linguistic data we are working with in part-of-speech guessing.

So, the crucial question is how to combine the predictions of different, yet interdependent features. I will compare two learning and guessing algorithms: A voting mechanism and a set of (more complicated) large margin classifiers, called *support vector machine* (SVM). The features employed in both strategies are similar. In the following, feature types and the language model store are defined formally. Then, the models employed in learning are described.

4.1.1 Identification of Feature Types

The set of possible features (*feature space*) needs to be determined before learning. I will first informally describe the types of information to be considered during the learning process.

The following parameters are considered to have significant predictive capabilities. We will call them *feature types* as opposed to *features*.[4]

In the following, I shortly examine possible feature types that are interesting for German.

[4]The feature type 'suffix' subsumes instances like -en, -st, -chen, which we call *features* following machine learning terminology.

- Capitalization is the strongest indicator for a noun in German.
- Occurrence of Umlaut ("shifted") vowels - ä - ü - ö

  Shifting vowels is common in German flexicon, such as in

  (8) das H**au**s/Noun(Singular)
  die H**äu**ser/Noun(Plural)

  Umlauts seen independently of other features, however, did not show a significant distribution in my German lexicon for the three main ambiguity classes Verb, Noun and Adjective.
- Occurrence of suffixes and prefixes

  Needless to say, most inflection is done in suffixes. A few prefixes are good indicators as well.
- Occurence of markers

  I looked for the inflection markers 'ge', 'zu', 'ver' and the dash '-'. They all can occur in the middle of a German form:

  (9) ein**ge**laden

  (10) heraus**ge**sucht

  (11) an**zu**nehmen

  Significant statistical effects of these markers only occur, if occurrence at the beginning and end of a word are excluded, and if occurrence is counted in relation to word length, because every given string will occur more frequently in longer words.

  I found statistical evidence for a good recall of the markers 'ge', '-' (χ^2 estimate shows significant marker-tag correlation). If taken as single clue, either precision is rather low (as in 'ge'), or they only give us information we already have other indicators for ('-' for nouns, which can be recognized by word capitalization). If employed in a statistical model that can deal with interdependent features, markers do improve overall performance. The performance with a small subset of the training/test-corpus and a support vector model went up from .69-.56 (precision/recall) to .76-.68 with marker feature.

  A prominent German example of this are infinite verbs with 'zu', which can be recognized much better with the 'zu' marker.

4.1.2 Filtering features

The frequency of a word (if available in the training data) may serve as a filter. Filtering out frequent items has proven to increase model performance. I explain this phenomenon by a tendency of languages to lexicalize forms with high frequency; thus these forms are not as regular and won't generalize well. From the NEGRA corpus, only samples with a maximal frequency of 2 are learned.

Affix feature selection. The most interesting types of features are affixes. In the first step of learning, I acquire sets of prefixes and suffixes.

It depends on the model whether affixes need to be identified and selected. If a model is able to learn feature interdependencies, it can be used to identify affixes on its own. The support vector model in my approach will only get single letters at a known position as binary features, both counted from the beginning and counted from the end of a word. Both learning algorithms (Joachims 1998, Collobert & Bengio 2001) will iteratively reduce the feature space dimensionality. This can be shown in practice, too.

The voting approach profits from a frequency-based selection of affixes. All forms of the learning corpus are traversed and checked for any suffix up to a given length k_s, and any prefix up to a given length k_p. Affixes are only considered if the segmented rest of the word (the presumed stem) is longer than a parameter k_w, which is usually set to 3. In my tests, k_s has shown to perform best at around 7, k_p at 4-7. Of course, this is a space & time vs. performance trade-off.

Affixes are filtered based on their observed frequency. The higher the number of affix-feature correlations observed, the better this affix should generalize.[5] Affix-Class combinations with less occurrences than a relatively low threshold (my value: 3) are filtered out. Higher values decrease recall.

4.1.3 Calculating feature weights

For each affix and each category, we are interested in the conditional feature probability $\hat{P}(C|F_{<a,t>})$. This probability is gained with a maximum likelihood estimation from the training corpus. This will serve as feature weight in the voting method.

The SVM training algorithm receives a binary value for each known feature.

4.2 Category selection

It seemed quite helpful to combine some classes into one, if we know a priori that features are not distributed over these classes in any way that would serve a good prediction. For example, this seems to be the case for different types of adjectives in German: Attributive, predicative and adverbial adjectives have empty markers in German (or go unmarked, depending on your favorite linguistic theory.) Other tags were considered to be of closed-classes (such as modal verbs or prepositions) and thus covered entirely by a lexicon with no need for guessing. In case they are needed, they may be expanded after the guessing process.

4.3 Learning

I understand learning as the process of finding parameters (i.e. the language model M) such that the return value of some imaginary error function $e(G_M, R)$ is minimal. This function compares the performance of two

[5] An even better method would be a filter using likelihood ratios.

functions: G_M, my guessing function using the model M, and R, the function delivering the 'good' standard. W be the set of words, C be the set of categories:

$$G : W \rightarrow \{\vec{v}|\vec{v} \text{ is a vector of length } |C|\}$$
$$R : W \rightarrow \{c^*|c \in C\}$$

The error function depends on the specific algorithm used to realize learning.

I adopt a supervised learning algorithm: it expects an annotated corpus, i.e. a lexicon with a set of part-of-speech tags assigned to each entry. The learning algorithm creates a data structure (*Language Model*) to be used in guessing.

Learning is a two-step process: First, a set of possible features is identified. In particular, affixes are identified. Then, each sample from the training corpus is analyzed in terms of values of present features. Out of a relatively high number of binary features (in P.O.S. guessing, every affix is seen as feature), only a few are present, thus get the value 1. Written as (sparse) vector, each sample describes a point in a k-dimensional space, where k is the number of possible features. used as input to the learning algorithm.

In the following, we describe the use of Support Vector Machines as models for training in comparison to a simpler voting mechanism.

4.3.1 Voting

Observed features were first selected using a frequency threshold. The same features as shown in 4.4 were used, however, they were scalar instead of binary, and the feature vectors were of length $|C|$. Feature weights in the feature vector for a given sample and a certain class contained $P(C|F_n)$. Each feature observed in a word may be seen as its own classifier with a confidence measure for each class. A combination algorithm lets these classifiers vote: This combination algorithm tries to estimate $P(C|F_1 \cap F_2 \cap ...F_n)$:

$$P(C|\bigcap_j F_j) \approx P'_C = \frac{1}{\sum_j \lambda_j} \sum_j \lambda_j \hat{P}(C|F_j)$$

The λ weights are set manually. For the resulting P'_C, we still need a decision function to get a binary result: Does w belong to C? The decision predicate applies iff $P' > \theta_C$. θ_C as separation constant (or: threshold) is estimated automatically: The average value of P'_C is calculated among the positive and the negative examples. θ_C is the mean of those two average values.

This method can be reinterpreted in terms of large-margin classification. Note that the λ weights and the θ_C threshold form a linear decision function: they construct a hyperplane in $|C|$-dimensional space. Feature weights, estimated (not trained) with a maximum-likelihood calculation, scale single features in space. Type-based weights (manually set, not trained) scale them again. The averaging function shown before transforms the multi-dimensional feature vectors into a scalar, where the θ_C (i.e. b) separates the data. θ_C is automatically chosen to achieve optimal (large-margin) separation between the two thought 'centers' of the positive and negative examples.

However, in comparison to training algorithms for the SVMs to be described next, the selection of features (i.e. support vectors) is not as sophisticated because it is not iterative. Kernels to make a non-linear transform[6] are also not available in the voting mechanism. The λ values need to set manually in the voting mechanism, which is another disadvantage.

4.3.2 Support Vector Machines (SVM)

The main idea of SVMs for binary classification problems is the following: we aim at constructing a hyperplane to separate the two classes in feature space so that the distance between the hyperplane and the nearest point(s) is maximal. This results in the following optimization problem:

Find $w \in R^d, b \in R,$ and $\zeta_i, i = 1, 2, ..., n,$ to minimize $\frac{(\sum_i \zeta_i)^q}{n} + \lambda||w||^2$

under the constraints

$$wx_i + b \geq 1 - \zeta_i, \text{for } y_i = +1,$$
$$wx_i + b \leq -1 + \zeta_i, \text{for } y_i = -1,$$
$$\zeta_i \geq 0, i = 1, ..., l.$$

This is a quadratic optimization problem which can be solved. Note that ζ_i allows for a 'soft' margin, which is necessary in the non-separable case.

The non-linear support vector machine maps the non-linear, higher dimensional input data into a linear feature space and applies the linear SVM there. The mapping function is called *kernel*.

Binary classification schemes can be extended to solve multi-class problems with a one-against-all classification, which train a separate model (i.e. hyper-planes) for each class with the binary classes $f(x) = c; f(x) \neq c$.

[6]Besides observing the features in the words, which can be seen as transformation.

	French	NEGRA
prefixes	15	10
suffixes	20	20
prefixes-morph	15	10
suffixes-morph	15	10
suffixes-morph-mutative	15	10
markers	0	0
capitalization	15	30

Table 1: Manually chosen lambda weights for voting as employed in the two evaluation corpora.

During solving, all classifiers are run and may vote for the final solution. However, since part-of-speech guessing is a multi-variate problem, the classifiers won't vote. Instead, they will simply decide about whether their respective class is additionally assigned to the solution.

Training algorithms such as SVMLight (Joachims 1998) and SVMTorch (Collobert & Bengio 2001) allow for iterative (optimal) selection of support vectors from the training data. Training time depends on the number of samples supplied, not on the number of features given, which makes these algorithms suitable for rich-feature problems. SMVTorch gives a training time complexity of about $O(N^2)$.

4.4 Solving

Support Vector Models are dichotomizers. Since we need to assign each model classes out of more than two classes, a One-Against-All strategy is applied. This means training k binary classifiers if k classes are present. A one-against-one method would work as well, probably even more accurate, but it poses a time problem – we would need $\frac{k(k-1)}{2}$ classifiers.[7]

If no class is chosen by the set of binary classifiers, the best-scoring class is accepted, since we know that each input must be assigned at least one class. If more than a parametrized number of classes are chosen, the worst scoring ones are thrown out.

Balancing the corpus. Usually, the bi-classification learning algorithm expects positive and negative examples to be equally important. Considering that we are using a one-against-all algorithm to map a polychotomizer into a set of dichotomizers (one for each class) and that each sample belongs to one, two or (seldomly) three classes, it is quite clear that each dichotomizer will be presented with a high number of negative examples (out of the class) and a small number of positive ones (in class).

Each resulting classifier separates the two groups optimally. This means that the resulting separation classifies the least possible number of examples. This results in an optimal accuracy count. (Accuracy is a measure of counting the correct classifications in relation to the number of all classifications.) Recall, measuring performance on positive examples, however, will be quite low, because the model focusses on negative examples, since there were more of them in the training corpus.

The straightforward way of giving positive examples more weight would be to present each positive example n times so that the number positive examples equals the number of negative ones in the training corpus. This is not practical, because the training data-sets get too big.

SVMLight (Joachims 1998) offers separate c cost factors for misclassified positive and negative examples during SVM training phase. However, SVMLight performed slightly worse than SVMTorch, even with differing positive / negative cost factors.[8] A remaining option is provided by Y. Lin (2002). I have not implemented it, yet it seems to be promising.

The Radial Basis Function (RBF) kernel proved to be optimal among the kernel functions provided by SVMLight and SVMTorch.

Feature types. I identified the following feature types to be observed in a word. A feature is identified as ordered list; usually $< a, t >$ where t is a feature type parameter (prefix, suffix, marker, capitalization, length) and a is some value compatible with the type. For each type, I will now define a function $F_{feature}(w)$:

- Prefixes and suffixes. A word begins/ends with a known affix α. While I had to always select the longest known affix of a word in order to get the most accurate feature in previous investigations, the training algorithm will take further care of an individual selection of the best features.
- Morphological pre/suffixes. $F_{<}a, marker >$ is $true \iff$ removing an affix α results in a known word of category c.
- Morphological pre/suffixes as mutative segments. $F_{<}a, marker >$ is $true \iff$ substituting a known affix α with an affix β results in a known word of category c.
- Markers. $F_{<}a, marker >$ is $true \iff$ the word w contains a, and a's position in w is within the limits m_l and $|w| - m_r$.
- Capitalized. $F_{<}a, cap >$ is $true \iff$ the word begins with a capitalized letter.
- Length of the word. $F_{<}a, length > equals$ the number of letters of w.
- Single letters. $F_{<}a, p, singlePrefix >$ is $true \iff$ the p-th position of the word (counted from the beginning) contains letter a. $F_{<}a, p, singleSuffix >$ is $true \iff$ the p-th position of the word (counted from the end of the word) contains letter a.

[7]Raaijmakers (2001) proposes to disseminate classes into several binary decisions, usually taken over by classifiers trained with different feature-sets. Investigating this was beyond our resources for this work.

[8]Modifications to the SVMTorch algorithm with regard to a sample-dependent 'c' were not successful either (the model did not converge). Multiple insertions of positive examples (while negative examples were presented only once) during training resulted in a loss of both precision and accuracy.

Tag	Baseline	Voting	SVM
Noun	.42-1.0-.42	.98-.99-.98	1.0-1.0-1.0
Adj	0-0-.56	.88-.98-.93	.93-.94 -.94
Verb-fin	0-0-.89	.54-.41-.89	.70-.83-.94
Verb-inf-1	0-0-.94	.78-.39-.95	.64-.59-.95
Verb-inf-2	0-0-.99	.42-.30-.99	1.0-.51-.99
Verb-part.	0-0-.96	.75-.58-.97	.75-.72-.98
Total	.42-.40-.80	.89-.88-1.00	.91-.93-.97

Table 2: Guessing performance on German NEGRA corpus (35 000 tokens). Values denote precision, recall and accuracy. Baseline is given with regard to the assignment of the most frequent class, which is a Noun.

Tag	Baseline	Voting	SVM
Noun	.62-1.0-.62	.87-.90-.89	.85-.90-.84
Adj	0-0-.83	.70-.67-.90	.65-.62-.88
Verb	0-0-.78	.85-.79-.92	.78-.81-.91
Verb-ind	0-0-.99	.97-.77-.99	.90-.77-.99
Verb-subj	0-0-.97	.97-.84-.99	.97-.71-.99
Verb-imp	0-0-.99	.93-.80-.99	.90-.77-.99
Verb-part	0-0-.97	1.00-.73-.99	.71-.21-.99
Total	.62-.58-.88	.85-.84-.95	.81-.83-.94

Table 3: Performance on French corpus. Verb-participle has only 23 examples of 4371 in the test corpus, Verb-imperative only 36. (22500 Tokens)

4.5 Evaluation

The SVM approach and the voting method were evaluated against each other with a lexicon with 30.900 entries extracted from NEGRA, a manually annotated German newspaper corpus (Skut et al. 1997) and a French corpus with 20.000 entries.

As tagset, the *Stuttgart-Tübingen-Tagset* was used in NEGRA, described in Schiller et al. (1995). As non-functional categories from this tagset I separated: Noun (*NE*/*NN*), Adjective (*ADJ*), Finite Verb (*VVFIN*), Imperative (*VVIMP*), Infinite Verb (*VVINF*), Infinite Verb (‘zu’ form) (*VVIZU*), Past Participle (*VVPP*).

Common Noun (*NN*) and Proper Noun (*NE*) from the STTS tagset were unified (N). So were different forms of adjectives as described below.

I gained the language model from 85 percent of the lexicon and evaluated with 15 percent.

The French corpus run shows that SVMs and their training algorithm seem to be successful in learning feature interdependencies, but won’t improve performance with very sparse vectors: In the French corpus, mostly suffixes seem to be of importance.

Compared to other work, the performance of SVMs on German P.O.S. guessing can compete with published results for different languages. Daciuk et al. (1998) states up to 91.64 percent precision, 93.92 percent recall on a French corpus. Daciuk’s reimplementation of Mikheev (1997) (a method that resembles my approach) showed 88.27 percent precision, 92.47 percent recall on the same corpus.

The algorithms presented perform worse on the same French corpus. This may be due to the fact that I did only little parameter-tuning for the corpora. Also, my (French) corpus might have been smaller than the one used in Daciuk’s experiments. Most importantly, SVMs cannot take advantage of their feature-integration capabilities.

The results show that guessing difficulty differs greatly among languages. My initial understanding that German is much harder in P.O.S. analysis is confirmed.

Training times for the model based on SVMLight was 1m37s, for SVMTorch it was 0m50s.[9]

Contribution of single features. My experiments with disabling some of the feature types show that, for SVMs, the single letter feature could basically do a huge part of the job. The learning algorithm recognizes combined features for itself. Prefixes and suffixes do not contribute significantly to the performance of the guessers. Morphological features, however, do add valuable information.

4.6 Implementation

The training and evaluation system is implemented in C++. Machine learning libraries may be interfaced with a hierarchy of interface classes derived from a common wrapper class so that different learners may be evaluated against each other. Currently, SVMTorch, SVMLight and a simple voting mechanism are included. [10]

Feature types, threshold constants and, of course, the models and their parameters may be chosen in the source code.

5 Conclusion

I described the use of advanced statistical models in part-of-speech guessing. They improve guessing performance for German significantly. SVMs showed the properties described in the literature. The major drawback of SVMs is a bad time complexity in training. But besides the ability to handle highly-dimensional features spaces, their performance did not decrease significantly when adding non-relevant features. This way, they can successfully integrate features that are only relevant to certain classes. Thus, support vector models make an excellent classifier in every natural language classification task, where diverse features need to be integrated.

Acknowledgements

I am grateful to Lothar Lemnitzer, Dale Gerdemann, Jakub Zavrel and Caroline Sporleder for their helpful comments regarding earlier versions of this paper.

[9]On an Athlon 1.4GHz, 2GB RAM machine running Linux & GCC.

[10]Neither SVMLight nor SVMTorch come with a well-documented API that would allow integration of the source code.

References

Brants, T. (2000), ‘Tnt – a statistical part-of-speech tagger’.

Brill, E. (1993), A Corpus-Based Approach to Language Learning, PhD thesis, Philadelphia, PA.

Brill, E. (1995), ‘Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging’, *Computational Linguistics* **21**(4), 543–565.

Collobert, R. & Bengio, S. (2001), ‘SVMTorch: Support vector machines for large-scale regression problems’, *Journal of Machine Learning Research* **1**, 143–160.

Daciuk, J., Watson, B. W. & Watson, R. E. (1998), Incremental construction of minimal acyclic finite state automata and transducers, *in* L. Karttunen, ed., ‘FSMNLP’98: International Workshop on Finite State Methods in Natural Language Processing’, Association for Computational Linguistics, Somerset, New Jersey, pp. 48–55.

Dermatas, E. & Kokkinakis, G. K. (1995), ‘Automatic stochastic tagging of natural language texts’, *Computational Linguistics* **21**(2), 137–163.

Joachims, T. (1998), ‘Making large-scale support vector machine learning practical’.

Mikheev, A. (1997), ‘Automatic rule induction for unknown-word guessing’, *Computational Linguistics* **23**(3), 405–423.

Raaijmakers, S. (2001), Large margin plug-in classification of natural language.

Schiller, A., Teufel, S. & Thielen, C. (1995), ‘Guidelines fur das tagging deutscher textkorpora mit stts’.

Schmid, H. (1994), ‘Part-of-speech tagging with neural networks’.

Skut, W., Krenn, B., Brants, T. & Uszkoreit, H. (1997), ‘An annotation scheme for free word order languages’.

Y. Lin, Y. Lee, G. W. (2002), ‘Support vector machines for classification in nonstandard situations’, *Machine Learning* **46**, 191–202.