Improving Accuracy in Wordclass Tagging through Combination of Machine Learning Systems

Hans van Halteren*
TOSCA / Language & Speech, University of Nijmegen

Jakub Zavrel, Walter Daelemans†
CNTS / Language Technology Group, University of Antwerp

We examine how differences in language models, learned by different data driven systems performing the same NLP task, can be exploited to yield a higher accuracy than the best individual system. We do this by means of experiments involving the task of morpho-syntactic wordclass tagging, on the basis of three different tagged corpora. Four well-known tagger generators (Hidden Markov Model, Memory-Based, Transformation Rules and Maximum Entropy) are trained on the same corpus data. After comparison, their outputs are combined using several voting strategies and second stage classifiers. All combination taggers outperform their best component. The reduction in error rate varies with the material in question, but can be as high as 24.3% with the LOB corpus.

1. Introduction

In all Natural Language Processing (NLP) systems, we find one or more language models that are used to predict, classify and/or interpret language related observations. Because most real-world NLP tasks require something that approaches full language understanding in order to be perfect, but automatic systems only have access to limited (and often superficial) information, as well as limited resources for reasoning with that information, such language models tend to make errors when the system is tested on new material. The engineering task in NLP is to design systems that make the amount of errors as small as possible with as little effort as possible. Common ways to reduce the error rate are devising better representations of the problem, spending more time on encoding language knowledge (in the case of hand-crafted systems) or finding more training data (in the case of data-driven systems). However, given limited resources, these options are not always available.

Rather than devising a new representation for our task, in this paper, we combine different systems employing known representations. The observation that suggests this approach is that systems that are designed differently, either because they use a different formalism or because they contain different knowledge, will typically produce different errors. We hope to make use of this fact and reduce the number of errors with very little additional effort by exploiting the disagreement between different language models. Although the approach is applicable to any type of language model, we focus on the case of statistical disambiguators that are trained on annotated corpora. The examples of the task that are present in the corpus and its annotation are fed into a learning algorithm, which induces a model of the desired input-output mapping in the form of a *classifier*.

* P.O. Box 9103, 6500 HD Nijmegen, The Netherlands, hvh@let.kun.nl
† Universiteitsplein 1, 2610 Wilrijk, Belgium, {zavrel,daelem}@uia.ua.ac.be

Computational Linguistics 27 (2), 199-230, 2001

We use a number of different learning algorithms simultaneously on the same training corpus. Each type of learning method brings its own 'inductive bias' to the task and will produce a classifier with slightly different characteristics, so that different methods will tend to produce different errors.

We investigate two ways of exploiting these differences. First, we make use of the *gang effect*. By simply using more than one classifier, and voting between their outputs, we expect to eliminate the quirks, and hence errors, that are due to the bias of one particular learner. However, there is also a way to make better use of the differences: we can create an *arbiter effect*. We can train a second level classifier to select its output on the basis of the patterns of co-occurrence of the outputs of the various classifiers. In this way, we not only counter the bias of each component, but actually exploit it in the identification of the correct output. This method even admits the possibility of correcting collective errors. The hypothesis is that both types of approaches can yield a more accurate model from the same training data than the most accurate component of the combination, and that given enough training data the arbiter type of method will be able to outperform the gang type.[1]

In the machine learning literature there has been much interest recently in the theoretical aspects of classifier combination, both of the gang effect type and of the arbiter type (see section 2). In general, it has been shown that, when the errors are uncorrelated to a sufficient degree, the resulting combined classifier will often perform better than any of the individual systems. In this paper we wish to take a more empirical approach and examine whether these methods result in substantial accuracy improvements in a situation typical for statistical NLP, *viz.* learning *morpho-syntactic wordclass tagging* (also known as Part-of-Speech or POS tagging) from an annotated corpus of several hundred thousand words.

Morpho-syntactic wordclass tagging entails the classification (*tagging*) of each token of a natural language text in terms of an element of a finite palette (*tagset*) of wordclass descriptors (*tags*). The reasons for this choice of task are several. First of all, tagging is a widely researched and well-understood task (cf. (van Halteren (ed.), 1999)). Second, current performance levels on this task still leave room for improvement: 'state of the art' performance for data driven automatic wordclass taggers on the usual type of material (e.g. tagging English text with single tags from a low detail tagset) is at 96-97% correctly tagged words, but accuracy levels for specific classes of ambiguous words are much lower. Finally, a number of rather different methods that automatically generate a fully functional tagging system from annotated text are available off-the-shelf. First experiments (van Halteren, Zavrel, and Daelemans, 1998; Brill and Wu, 1998) demonstrated the basic validity of the approach for tagging, with the error rate of the best combiner being 19.1% lower than that of the best individual tagger (van Halteren, Zavrel, and Daelemans, 1998). However, these experiments were restricted to a single language, a single tagset and, more importantly, a limited amount of training data for the combiners. This led us to perform further, more extensive, tagging experiments before moving on to other tasks. Since then the method has also been applied to other NLP tasks with good results (see section 6).

In the remaining sections, we first introduce classifier combination on the basis of previous work in the machine learning literature and present the combination methods we use in our experiments (Section 2). Then we explain our experimental setup (Sec-

1 In previous work (van Halteren, Zavrel, and Daelemans, 1998), we were unable to confirm the latter half of the hypothesis unequivocally. As we judged this to be due to insufficient training data for proper training of the second level classifiers, we greatly increase the amount of training data in the present work through the use of cross-validation.

tion 3), also describing the corpora (3.1) and tagger generators (3.2) used in the experiments. In Section 4, we go on to report the overall results of the experiments, starting with a comparison between the component taggers (and hence between the underlying tagger generators) and continuing with a comparison of the combination methods. The results are examined in more detail in Section 5, where we discuss such aspects as accuracy on specific words or tags, the influence of inconsistent training data, training set size, the contribution of individual component taggers and tagset granularity. In Section 6, we discuss the results in the light of related work, after which we conclude with a summary of the most important observations and interesting directions for future research (Section 7).

2. Combination methods

In recent years there has been an explosion of research in Machine Learning on finding ways to improve the accuracy of supervised classifier learning methods. An important finding is that a set of classifiers whose individual decisions are combined in some way (an *ensemble*) can be more accurate than any of its component classifiers if the errors of the individual classifiers are sufficiently uncorrelated (see (Dietterich, 1997; Chan, Stolfo, and Wolpert, 1999) for overviews). There are several ways in which an ensemble can be created, both in the selection of the individual classifiers and in the way they are combined.

One way to create multiple classifiers is to use subsamples of the training examples. In *bagging*, the training set for each individual classifier is created by randomly drawing training examples with replacement from the initial training set (Breiman, 1996a). In *boosting*, the errors made by a classifier learned from a training set are used to construct a new training set in which the misclassified examples get more weight. By sequentially performing this operation, an ensemble is constructed (e.g. ADABOOST, (Freund and Schapire, 1996)). This class of methods is also called *arcing* (for adaptive resampling and combining). In general, boosting obtains better results than bagging, except when the data is noisy (Dietterich, 1997). Another way is to train classifiers on different sources of information about the task by giving them access to different subsets of the available input features (Cherkauer, 1996). Still other ways are to represent the output classes as bitstrings where each bit is predicted by a different component classifier (Error Correcting Output Coding, (Dietterich and Bakiri, 1995)) or to develop learning-method specific methods for ensuring (random) variation in the way the different classifiers of an ensemble are constructed (Dietterich, 1997).

In this paper we take a multi-strategy approach, in which an ensemble is constructed by classifiers resulting from training different learning methods on the same data (see also (Alpaydin, 1998)).

Methods to combine the outputs of component classifiers in an ensemble include simple *voting* where each component classifier gets an equal vote, and *weighted voting*, in which each component classifier's vote is weighted by its accuracy (see e.g. (Golding and Roth, 1999)). More sophisticated weighting methods have been designed as well. E.g. (Ali and Pazzani, 1996) apply the Naive Bayes algorithm to learn weights for classifiers. Voting methods lead to the *gang effect* discussed earlier. The most interesting approach to combination is *stacking* in which a classifier is trained to predict the correct output class when given as input the outputs of the ensemble classifiers, and possibly additional information (Wolpert, 1992; Breiman, 1996b; Ting and Witten, 1997a; Ting and Witten, 1997b). Stacking can lead to an *arbiter effect*. In this paper we compare voting and stacking approaches on the tagging problem.

In the remainder of this section, we describe the combination methods we use in

Let T_i be the component taggers, $S_i(tok)$ the most probable tag for a token tok as suggested by T_i, and let the quality of tagger T_i be measured by

- the precision of T_i for tag tag: $Prec(T_i, tag)$
- the recall of T_i for tag tag: $Rec(T_i, tag)$
- the overall precision of T_i: $Prec(T_i)$

Then the vote $V(tag, tok)$ for tagging token tok with tag tag is given by:

- **Majority:**
$$\sum_i IF\ \ S_i(tok) = tag\ \ THEN\ \ 1\ \ ELSE\ \ 0$$
- **TotPrecision:**
$$\sum_i IF\ \ S_i(tok) = tag\ \ THEN\ \ Prec(T_i)\ \ ELSE\ \ 0$$
- **TagPrecision:**
$$\sum_i IF\ \ S_i(tok) = tag\ \ THEN\ \ Prec(T_i, tag)\ \ ELSE\ \ 0$$
- **Precision-Recall:**
$$\sum_i IF\ \ S_i(tok) = tag\ \ THEN\ \ Prec(T_i, tag)\ \ ELSE\ \ 1 - Rec(T_i, tag)$$

Figure 1
Simple algorithms for voting between component taggers.

our experiments. We start with variations based on weighted voting. Then we go on to several types of stacked classifiers, which model the disagreement situations observed in the training data in more detail. The input to the second stage classifier can be limited to the first level outputs or can contain additional information from the original input pattern. We will consider a number of different second level learners. Apart from using three well-known machine learning methods, Memory-Based Learning, Maximum Entropy and Decision Trees, we also introduce a new method, based on grouped voting.

2.1 Simple voting

The most straightforward method to combine the results of multiple taggers is to do an n-way vote. Each tagger is allowed to vote for the tag of its choice and the tag with the highest number of votes is selected.[2] The question is how large a vote we allow each tagger (cf. Figure 1). The most democratic option is to give each tagger one vote (**Majority**). This does not require any tuning of the voting mechanism on training data.

However, the component taggers can be distinguished by several figures of merit, and it appears more useful to give more weight to taggers which have proved their quality. For this purpose we use *precision* and *recall*, two well-known measures, which can be applied to the evaluation of tagger output as well. For any tag X, *precision* measures which percentage of the tokens tagged X by the tagger are also tagged X in the benchmark. *Recall* measures which percentage of the tokens tagged X in the benchmark are also tagged X by the tagger. When abstracting away from individual tags, precision and recall are equal (at least if the tagger produces exactly one tag per token) and mea-

2 In all our experiments, any ties are resolved by a random selection from among the winning tags.

sure how many tokens are tagged correctly; in this case we also use the more generic term *accuracy*. We will call the voting method where each tagger is weighted by its general quality **TotPrecision**, i.e. each tagger votes its overall precision. To allow for more detailed interactions, each tagger is weighted by the quality in relation to the current situation, i.e. each tagger votes its precision on the tag it suggests (**TagPrecision**). This way, taggers that are accurate for a particular type of ambiguity can act as specialized experts. The information about each tagger's quality is derived from a cross-validation of its results on the combiner training set. The precise setup for deriving the training data is described in more detail below, in section 3.

We have access to even more information on how well the taggers perform. We not only know whether we should believe what they propose ($precision$) but know as well how often they fail to recognize the correct tag ($1 - recall$). This information can be used by forcing each tagger to add to the vote for tags suggested by the opposition too, by an amount equal to 1 minus the recall on the opposing tag (**Precision-Recall**). As an example, suppose that the MXPOST tagger suggests DT and the HMM tagger TnT suggests CS (two possible tags in the LOB tagset for the token "that"). If MXPOST has a precision on DT of 0.9658 and a recall on CS of 0.8927, and TnT has a precision on CS of 0.9044 and a recall on DT of 0.9767, then DT receives a 0.9658 + 0.0233 = 0.9991 vote and CS a 0.9044 + 0.1073 = 1.0117 vote.

Note that simple voting combiners can never return a tag that was not suggested by a (weighted) majority of the component taggers. As a result, they are restricted to the combination of taggers which all use the same tagset. This is not the case for all the following (arbiter type) combination methods, a fact which we have recently exploited in bootstrapping a wordclass tagger for a new corpus from existing taggers with completely different tagsets (Zavrel and Daelemans, 2000).

2.2 Stacked probabilistic voting

One of the best methods for tagger combination in (van Halteren, Zavrel, and Daelemans, 1998) is the **TagPair** method. It looks at all situations where one tagger suggests tag_1 and the other tag_2 and estimates the probability that in this situation the tag should actually be tag_x. Although it is presented as a variant of voting in that paper, it is in fact also a stacked classifier, because it does not necessarily select one of the tags suggested by the component taggers. Taking the same example as in the voting section above, if tagger MXPOST suggests DT and tagger TnT suggests CS, we find that the probabilities for the appropriate tag are:

CS	subordinating conjunction	0.4623
CS22	second half of a two-token subordinating conjunction, e.g. "so that"	0.0171
DT	determiner	0.4966
QL	quantifier	0.0103
WPR	wh-pronoun	0.0137

When combining the taggers, every tagger pair is taken in turn and allowed to vote (with a weight equal to the probability $P(tag_x|tag_1, tag_2)$ as described above) for each possible tag (cf. Figure 2). If a tag pair tag_1-tag_2 has never been observed in the training data, we fall back on information on the individual taggers, i.e. $P(tag_x|tag_1)$ and $P(tag_x|tag_2)$. Note that with this method (and all the following) a tag suggested by a minority (or even none) of the taggers actually has a chance to win, although in practice the chance to beat a majority is still very slight.

Let T_i be the component taggers and $S_i(tok)$ the most probable tag for a token tok as suggested by T_i. Then the vote $V(tag, tok)$ for tagging token tok with tag tag is given by:

$$V(tag, tok) = \sum_{i,j|i<j} V_{i,j}(tag, tok)$$

where $V_{i,j}(tag, tok)$ is given by

$$IF\ frequency(\ S_i(tok_x) = S_i(tok),\ S_j(tok_x) = S_j(tok)\ ) > 0$$

$$THEN\ V_{i,j}(tag, tok) = P(tag \mid\ S_i(tok_x) = S_i(tok), S_j(tok_x) = S_j(tok)\ )$$

$$ELSE\ V_{i,j}(tag, tok) = \frac{1}{2}P(tag \mid\ S_i(tok_x) = S_i(tok)\ ) + \frac{1}{2}P(tag \mid\ S_j(tok_x) = S_j(tok)\ )$$

Figure 2
The TagPair algorithm for voting between component taggers.

If the case to be classified corresponds to the feature-value pair set

$$F_{case} = \{\{f_1 = v_1\}, ..., \{f_n = v_n\}\}$$

then estimate the probability of each class C_x for F_{case} as a weighted sum over all possible subsets F_{sub} of F_{case}:

$$\hat{P}(C_x) = \sum_{F_{sub} \subset F_{case}} W_{F_{sub}} P(C_x \mid F_{sub})$$

with the weight $W_{F_{sub}}$ for an F_{sub} containing n elements equal to $\frac{n!}{W_{norm}}$, where W_{norm} is a normalizing constant so that $\sum_{C_x} \hat{P}(C_x) = 1$.

Figure 3
The Weighted Probability Distribution Voting (WPDV) classification algorithm, as used in the combination experiments.

Seeing the success of TagPair in the earlier experiments, we decided to try and generalize this stacked probabilistic voting approach to combinations larger than pairs. Among other things, this would let us include word and context features here as well. The method that was eventually developed we have called *weighted probability distribution voting* (henceforth *WPDV*).

A WPDV classification model is not limited to pairs of features (such as the pairs of tagger outputs for TagPair), but can use the probability distributions for all feature combinations observed in the training data (cf. Figure 3). During voting, we do not use a fallback strategy, like TagPair, but use weights to prevent the lower order combinations from excessively influencing the final results when a higher order combination, i.e. more exact information, is present. The original system, as used for this paper, weights a combination of order n with a factor $n!$, a number based on the observation that a combination of order m contains m combinations of order $(m-1)$ which have to be competed with. Its only parameter is a threshold for the number of times a combina-

- Tags suggested by the base taggers, used by all systems:

  $TagTBL = JJ \quad TagMBT = VBN \quad TagMXP = VBD \quad TagHMM = JJ$

- The focus token, used by stacked classifiers at level Tags+Word:

  $Word = restored$

- Full form tags suggested by the base tagger for the previous and next token, used by stacked classifiers at level Tags+Context, except for WPDV:

  $PrevTBL = JJ \quad PrevMBT = NN \quad PrevMXP = NN \quad PrevHMM = JJ$

  $NextTBL = NN \quad NextMBT = NN \quad NextMXP = NN \quad NextHMM = NN$

- Compressed form of the context tags, used by WPDV(Tags+Context), because the system was unable to cope with the large number of features:

  $Prev = JJ + NN + NN + JJ \quad Next = NN + NN + NN + NN$

- Target feature, used by all systems:

  $Tag = VBD$

Figure 4
Features used by the combination systems. Examples are taken from the LOB material.

tion must be observed in the training data in order to be used,[3] which helps prevent a combinatorial explosion when there are too many atomic features.[4]

In contrast to voting, stacking classifiers allows the combination of the outputs of component systems with additional information about the decision's context. We investigated several versions of this approach. In the basic version (Tags), each training case for the second level learner consists of the tags suggested by the component taggers and the correct tag (cf. Figure 4). In the more advanced versions we add information about the word in question (Tags+Word) and the tags suggested by all taggers for the previous and the next position (Tags+Context). These types of extended second level features can be exploited by WPDV, as well as by a wide selection of other machine learning algorithms.

2.3 Memory-Based combination

Our first choice from these other algorithms is a Memory-Based second level learner, implemented in TiMBL (Daelemans et al., 1999), a package developed at Tilburg University and Antwerp University.[5]

Memory-Based learning is a learning method that is based on storing all examples of a task in memory and then classifying new examples by similarity-based reasoning from this memory of examples. Each example is represented by a fixed length vector of feature-values, called a *case*. If the case to be classified has been observed before, i.e. if it is found among the stored cases (in the *case base*), the most frequent corresponding output is used. If the case is not found in the case base, k nearest neighbours are determined with some similarity metric and the output is based on the observed outputs for those neighbours. Both the value of k and the similarity metric used can be selected by parameters of the system. For the Tags version, the similarity metric used is *Overlap* (a count of the number of matching feature-values between a test and a training item) and k is kept

3 In our experiments, this parameter is always set to 5.
4 WPDV has since evolved, using more parameters and more involved weighting schemes, and also been tested on other tasks than tagger combination (van Halteren, 2000a; van Halteren, 2000b).
5 TiMBL is available from `http://ilk.kub.nl/`.

at 1. For the other two versions (Tags+Word and Tags+Context), a value of $k = 3$ is used, and each overlapping feature is weighted by its *Information Gain* (Daelemans, Van den Bosch, and Weijters, 1997). The Information Gain of a feature is defined as the difference between the entropy of the a priori class distribution and the conditional entropy of the classes given the value of the feature.[6]

2.4 Maximum Entropy combination

The second machine learning method, *Maximum Entropy Modeling*, implemented in the MACCENT system (Dehaspe, 1997),[7] does the classification task by selecting the most probable class given a Maximum Entropy Model. This type of model represents examples of the task (Cases) as sets of binary indicator features, for the task at hand conjunctions of a particular tag and a particular set of feature values. The model has the form of an exponential model:

$$p_\Lambda(tag|Case) = \frac{1}{Z_\Lambda(Case)}\, e^{\sum_i \lambda_i f_i(Case,tag)}$$

where i indexes all the binary features, f_i is a binary indicator function for feature i, Z_Λ is a normalizing constant, and λ_i is a weight for feature i. The model is trained by iteratively adding binary features with the largest gain in the probability of the training data, and estimating the weights using a numerical optimization method called *Improved Iterative Scaling*. The model is constrained by the observed distribution of the features in the training data and has the property of having the maximum entropy of all models that fit the constraints, i.e. all distributions that are not directly constrained by the data are left as uniform as possible.[8]

The maximum entropy combiner takes the same information as the memory-based learner as input, but internally translates all multi-valued features to binary indicator functions. The Improved Iterative Scaling algorithm is then applied, with a maximum of one hundred iterations. This algorithm is the same as the one used in the MXPOST tagger described in section 3.2, but without the beam search used in the tagging application.

2.5 Decision Tree combination

The third machine learning method we used is C5.0 (Quinlan, 1993),[9] an example of *top-down induction of decision trees*. A decision tree is constructed by recursively partitioning the training set, selecting, at each step, the feature which most reduces the uncertainty about the class in each partition, and using it as a split. C5.0 uses *Gain Ratio* as an estimate of the utility of splitting on a features. Gain Ratio corresponds to the Information Gain measure of a feature, as described above, except that the measure is normalized for the number of values of the feature, by dividing by the entropy of the feature's values. After the decision tree is constructed, it is pruned to avoid overfitting, using a method which is described in detail in (Quinlan, 1993). A classification for a test case is made by traversing the tree until either a leaf node is found or all further branches do not match the test case, and returning the most frequent class at the last node. The case representation uses exactly the same features as the memory-based learner.

6 This is also sometimes referred to as Mutual Information in the Computational Linguistics literature.
7 MACCENT is available from
`http://www.cs.kuleuven.ac.be/~ ldh`.
8 For a more detailed discussion, see (Berger, Della Pietra, and Della Pietra, 1996; Ratnaparkhi, 1996).
9 C5.0 is commercialy available from `http://www.rulequest.com/`. Its predecessor C4.5 can be downloaded from http://www.cse.unsw.edu.au/~ quinlan/.

3. Experimental setup

In order to test the potential of system combination, we obviously need systems to combine, i.e. a number of different taggers. As we are primarily interested in the combination of classifiers trained on the same data sets, we are in fact looking for data sets (in this case tagged corpora) and systems which can automatically generate a tagger on the basis of those data sets. For the current experiments we have selected three tagged corpora and four tagger generators. Before we go into a detailed description of each of these, we first describe how the ingredients are used in the experiments.

Each corpus is used in the same way to test tagger and combiner performance. First of all, it is split into a 90% training set and a 10% test set. We can evaluate the base taggers by using the whole training set to train the tagger generators on and the test set to test the resulting tagger on. For the combiners a more complex strategy must be followed, since combiner training must be done on material unseen by the base taggers involved. Rather than setting apart a fixed combiner training set,[10] we use a nine-fold training strategy. The 90% training set is split into nine equal parts. Each part is tagged with component taggers which have been trained on the other eight parts. All results are then concatenated for use in combiner training, so that, in contrast to our earlier work, all of the training set is effectively available for the training of the combiner. Finally, the resulting combiners are tested on the test set. Since the test set is identical for all methods, we can compute the statistical significance of the results using McNemar's chi-squared test (Dietterich, 1998).

As we will see, the increase in combiner training set size, *viz.* 90% of the corpus vs. the fixed 10% tune set in the earlier experiments, indeed results in better performance. On the other hand, the increased amount of data also increases time and space requirements for some systems to such a degree that we had to exclude them from (some parts of) the experiments.

The data in the training set is the only information used in tagger and combiner construction: all components of all taggers and combiners (lexicon, context statistics, etc.) are entirely data driven and no manual adjustments are made. If any tagger or combiner construction method is parametrized, we use default settings where available. If there is no default, we choose intuitively appropriate values without preliminary testing. In these cases, we report such parameter settings in the introduction to the system.

3.1 Data

In the current experiments we make use of three corpora. The first is the LOB corpus (Johansson, 1986), which we used in the earlier experiments as well (van Halteren, Zavrel, and Daelemans, 1998) and which has proved to be a good testing ground. We then switch to Wall Street Journal material (WSJ), tagged with the Penn Treebank II tagset (Marcus, Santorini, and Marcinkiewicz, 1993). Like LOB, it consists of approximately 1M words, but of American English. Furthermore, it is of a different structure (only newspaper text) and tagged with a rather different tagset. The experiments with WSJ will also let us compare our results with those reported by (Brill and Wu, 1998), which show a much less pronounced accuracy increase than ours with LOB. The final corpus is the slightly smaller (750Kw) Eindhoven corpus (Uit den Boogaart, 1975) tagged with the Wotan tagset (Berghmans, 1994). This will let us examine the tagging of

10 Cf. the "Tune" set in (van Halteren, Zavrel, and Daelemans, 1998). This consisted of 114K tokens, but, because of a 92.5% agreement over all four taggers, it yielded less than 9K tokens of useful training material to resolve disagreements. This was suspected to be the main reason for the relative lack of performance by the more sophisticated combiners.

a different language than English, *viz.* Dutch. Furthermore, the Wotan tagset is a very detailed one, so that the error rate of the individual taggers tends to be higher. Moreover, we can more easily use projections of the tagset and thus study the effects of levels of granularity.

LOB. The first data set we use for our experiments consists of the tagged Lancaster-Oslo/Bergen corpus (LOB; (Johansson, 1986)). The corpus comprises about one million words of British English text, divided over 500 samples of 2000 words from 15 text types.

The tagging of the LOB corpus, which was manually checked and corrected, is generally accepted to be quite accurate. Here we use a slight adaptation of the tagset. The changes are mainly cosmetic, e.g. non-alphabetic characters such as "$" in tag names have been replaced. However, there has also been some retokenization: genitive markers have been split off and the negative marker "n't" has been reattached.

An example sentence tagged with the resulting tagset is:

The	ATI	singular or plural article
Lord	NPT	singular titular noun
Major	NPT	singular titular noun
extended	VBD	past tense of verb
an	AT	singular article
invitation	NN	singular common noun
to	IN	preposition
all	ABN	pre-quantifier
the	ATI	singular or plural article
parliamentary	JJ	adjective
candidates	NNS	plural common noun
.	SPER	period

The tagset consists of 170 different tags (including ditto tags[11]), and has an average ambiguity of 2.82 tags per wordform over the corpus. An impression of the difficulty of the tagging task can be gained from the two baseline measurements in Table 2 (in 4.1 below), representing a completely random choice from the potential tags for each token (Random) and selection of the lexically most likely tag (LexProb).[12]

The training/test separation of the corpus is done at utterance boundaries (each 1st-9th utterance is training and each 10th is test) and leads to a 1046K token training set and a 115K token test set. Around 2.14% of the test set are tokens unseen in the training set[13] and a further 0.37% are known tokens but with unseen tags.

WSJ. The second data set consists of 1M words of Wall Street Journal material. It differs from LOB in that it is American English and, more importantly, in that is completely made up of newspaper text. The material is tagged with the Penn Treebank tagset (Marcus, Santorini, and Marcinkiewicz, 1993), which is much smaller than the LOB one. It consists of only 48 tags.[14] There is no attempt to annotate compound words, so there are

11 Ditto tags are used for the components of multi-token units, e.g. if "as well as" is taken to be a coordinating conjunction, it is tagged "as_CC-1 well_CC-2 as_CC-3", using three related but different ditto tags.

12 These numbers are calculated on the basis of a lexicon derived from the whole corpus. An actual tagger will have to deal with unknown words in the test set, which will tend to increase the ambiguity and decrease Random and LexProb. Note that all actual taggers and combiners in this paper do have to cope with unknown words as their lexicons are based purely on their training sets.

13 Because of the way in which the tagger generators treat their input, we do count tokens as different even though they are the same underlying token, but differ in capitalization of one or more characters.

14 In the material we have available, quotes are represented slightly differently, so that there are only 45 different tags. In addition, the corpus contains a limited number of instances of 38 'indeterminate' tags, e.g. JJ|VBD indicates a choice between adjective and past participle which cannot be decided or about which the annotator was unsure.

no ditto tags.

An example sentence is:

By	IN	preposition/subordinating conjunction
10	CD	cardinal number
a.m.	RB	adverb
Tokyo	NNP	singular proper noun
time	NN	singular common noun
,	,	comma
the	DT	determiner
index	NN	singular common noun
was	VBD	past tense verb
up	RB	adverb
435.11	CD	cardinal number
points	NNS	plural common noun
,	,	comma
to	TO	"to"
34903.80	CD	cardinal number
as	IN	preposition/subordinating conjunction
investors	NNS	plural common noun
hailed	VBD	past tense verb
New	NNP	singular proper noun
York	NNP	singular proper noun
's	POS	possessive ending
overnight	JJ	adjective
rally	NN	singular common noun
.	.	sentence-final punctuation

Mostly because of the less detailed tagset, the average ambiguity of the tags is lower than LOB's, at 2.34 tags per token in the corpus. This means that the tagging task should be an easier one than that for LOB. This is supported by the values for Random and Lex-Prob in Table 2. On the other hand, the less detailed tagset also means that the taggers have less detailed information to base their decisions on. Another factor which is of influence on the quality of automatic tagging is the consistency of the tagging over the corpus. The WSJ material has not been checked as extensively as the LOB corpus and is expected to have a much lower consistency level (see section 5.3 below for a closer examination).

The training/test separation of the corpus is again done at utterance boundaries and leads to a 1160K token training set and a 129K token test set. Around 1.86% of the test set are unseen tokens and a further 0.44% are known tokens with previously unseen tags.

Eindhoven. The final two data sets are both based on the Eindhoven corpus (Uit den Boogaart, 1975). This is slightly smaller than LOB and WSJ. The written part, which we use in our experiments, consists of about 750K words, in samples ranging from 53 to 451 words. In variety, it lies between LOB and WSJ, containing 150Kw each of samples from Dutch newspapers (subcorpus CDB), weeklies (OBL), magazines (GBL), popular scientific writings (PWE) and novels (RNO).

The tagging of the corpus, as used here, was created in 1994 as part of a Master's thesis project (Berghmans, 1994). It employs the Wotan tagset for Dutch, newly designed during the project. It is based on the classification used in the most popular descriptive grammar of Dutch, the *Algemene Nederlandse Spraakkunst* (ANS; (Geerts et al., 1984)). The actual distinctions encoded in the tagset were selected on the basis of their importance to the potential users, as estimated from a number of in-depth interviews with interested parties in the Netherlands. The Wotan tagset is not only very large (233 base tags, leading to 341 tags when counting each ditto tag separately), but furthermore contains distinctions which are very difficult for automatic taggers, such as verb transitivity, syntactic use of adjectives and the recognition of multi-token units. It has an average ambiguity of 7.46 tags per token in the corpus. For our experiments, we also designed a simplification of the tagset, dubbed WotanLite, which no longer contains the most problematic distinctions. WotanLite has 129 tags (with a complement of ditto tags leading to a total of 173) and an average ambiguity of 3.46 tags per token.

An example of Wotan tagging is (only underlined parts remain in WotanLite):

Mr. (*Master*, title)[15]	N(eigen,ev,neut):1/2	first part of singular neutral case proper noun
Rijpstra	N(eigen,ev,neut):2/2	second part of singular neutral case proper noun
heeft (*has*)	V(hulp,ott,3,ev)	3rd person singular present tense auxiliary verb
de (*the*)	Art(bep,zijd-of-mv,neut)	neutral case non-neuter or plural definite article
Commissarispost (*post of Commissioner*)	N(soort,ev,neut)	singular neutral case common noun
in (*in*)	Prep(voor)	adposition used as preposition
Friesland	N(eigen,ev,neut)	singular neutral case proper noun
geambieerd (*aspired to*)	V(trans,verl-dw,onverv)	base form of past participle of transitive verb
en (*and*)	Conj(neven)	coordinating conjunction
hij (*he*)	Pron(per,3,ev,nom)	3rd person singular nominative personal pronoun
moet (*should*)	V(hulp,ott,3,ev)	3rd person singular present tense auxiliary verb
dus (*therefore*)	Adv(gew,aanw)	demonstrative non-pronominal adverb
alle (*all*)	Pron(onbep,neut,attr)	attributively used neutral case indefinite pronoun
kans (*opportunity*)	N(soort,ev,neut)	singular neutral case common noun
hebben (*have*)	V(trans,inf)	infinitive of transitive verb
er (*there*)	Adv(pron,er)	pronominal adverb "er"
het (*the*)	Art(bep,onzijd,neut)	neutral case neuter definite article
beste (*best*)	Adj(zelfst,overtr,verv-neut)	nominally used inflected superlative form of adjective
van (*of*)	Adv(deel-adv)	particle adverb
te (*to*)	Prep(voor-inf)	infinitival "te"
maken (*make*)	V(trans,inf)	infinitive of transitive verb
.	Punc(punt)	period

The annotation of the corpus was realized by a semi-automatic upgrade of the tagging inherited from an earlier project. The resulting consistency has never been exhaustively measured for either the Wotan or the original tagging.

The training/test separation of the corpus is done at sample boundaries (each 1st-9th sample is training and each 10th is test). This is a much stricter separation than applied for LOB and WSJ, as for those two corpora our test utterances are related to the training ones by being in the same samples. Partly as a result of this, but also very much because of word compounding in Dutch, we see a much higher percentage of new tokens, viz. 6.24% tokens unseen in the training set. A further 1.45% known tokens have new tags for Wotan, and 0.45% for WotanLite. The training set consists of 640K tokens and the test set of 72K tokens.

15 I.e. *Master Rijpstra has aspired to the post of Commissioner in Friesland and he should therefore be given every opportunity to make the most of it.*

Table 1
The features available to the four taggers in our study. Except for MXPOST, all systems use different models (and hence features) for known (k) and unknown (u) words. However, Brill's Transformation Based Learning system (TBL) applies its two models in sequence when faced with unknown words, thus giving the unknown word tagger access to the features used by the known word model as well. The first five columns in the table show features of the focus word: capitalization (C), hyphen (H) or digit (D) present, and number of suffix (S) or prefix (P) letters of the word. Brill's TBL system (for unknown words) also takes into account whether the addition or deletion of a suffix results in a known lexicon entry (indicated by an L). The next three columns represent access to the actual word (W) and any range of words to the left (W_{left}) and/or right (W_{right}). The last three columns show access to tag information for the word itself (T) and any range of words left (T_{left}) and/or right (T_{right}). Note that the expressive power of a method is not purely determined by the features it has access to, but also by its algorithm, and what combinations of the available features this allows it to consider.

System	Features										
	C	D	N	S	P	W	W_{left}	W_{right}	T	T_{left}	T_{right}
TBL (k)						x	1–2	1–2	x	1–3	1–3
TBL (u)	x	x	x	4,L	4,L		1–2	1–2		1–3	1–3
MBT (k)						x			x	1–2	1–2
MBT (u)	x	x	x	3						1	1
MXP (all)	x	x	x	4	4	x	1–2	1–2		1–2	
TNT (k)	x					x			x	1–2	
TNT (u)	x			10						1–2	

3.2 Tagger generators

The second ingredient for our experiments is a set of four tagger generator systems, selected on the basis of variety[16] and availability. Each of the systems represents a popular type of learning method, each uses slightly different features of the text (see Table 1), and each has a completely different representation for its language model. All publicly available systems are used with the default settings that are suggested in their documentation.

Error-driven Transformation Based Learning is a learning method that finds a set of rules that transforms the corpus from a baseline annotation so as to minimize the number of errors (we will refer to the system with "**TBL**" below). A tagger generator using this learning method is described in (Brill, 1992; Brill, 1994). The implementation that we use is Eric Brill's publicly available set of C programs and Perl scripts.[17]

When training, this system starts with a baseline corpus annotation A_0. In A_0, each known word is tagged with its most likely tag in the training set, and each unknown word is tagged as a noun (or proper noun if capitalized). The system then searches through a space of transformation rules (defined by rule-templates) in order to reduce the discrepancy between its current annotation and the provided correct one. There are separate templates for known words (mainly based on local word and tag context), and for unknown words (based on suffix, prefix, and other lexical information). The exact features used by this tagger are shown in Table 1. The learner for the unknown words is trained and applied first. Based on its output, the rules for context disambiguation are learned. In each learning step, all instantiations of the rule-templates that are present in the corpus are generated and receive a score. The rule that corrects the highest num-

16 The systems have to differ as much as possible in their learning strategies and biases, as otherwise there will be insufficient differences of opinion for the combiners to make use of. This was shown clearly in early experiments in 1992, where only n-gram taggers were used, and which produced only a very limited accuracy improvement (cf. (van Halteren, 1996)).
17 Brill's system can be downloaded from
`ftp://ftp.cs.jhu.edu/pub/brill/Programs/RULE_BASED_TAGGER_V.1.14.tar.Z`

ber of errors at step n is selected and applied to the corpus to yield an annotation A_n, which is then used as the basis for step $n+1$. The process stops when no rule reaches a score above a predefined threshold. In our experiments this has usually yielded several hundreds of rules. Of the four systems, TBL has access to the most features: contextual information (the words and tags in a window spanning three positions before and after the focus word) as well as lexical information (the existence of words formed by suffix/prefix addition/deletion). However, the conjunctions of these features are not all available in order to keep the search space manageable. Even with this restriction, the search is computationally very costly. The most important rule-templates are of the form

`if` $context = x$ `change` tag_i `to` tag_j

where $context$ is some condition on the tags of the neighbouring words. Hence learning speed is roughly cubic in the tagset size.[18]

When tagging, the system again starts with a baseline annotation for the new text, and then applies all rules that were derived during training, in the sequence in which they were derived. This means that application of the rules is fully deterministic. Corpus statistics have been at the basis of selecting the rule sequence, but the resulting tagger does not explicitly use a probabilistic model.

Memory-Based Learning is another learning method that does not explicitly manipulate probabilities, but, rather than extracting a concise set of rules, it is based on storing all examples of a task in memory in an efficient way (cf. Section 2.3). New examples are then classified by similarity-based reasoning from this memory of examples. A tagger using this learning method, **MBT**, was proposed by (Daelemans et al., 1996).[19]

During the training phase, the training corpus is transformed into two case bases, one which is to be used for known words and one for unknown words. The cases are stored in an IGTree (a heuristically indexed version of a case memory (Daelemans, Van den Bosch, and Weijters, 1997)), and during tagging new cases are classified by matching cases with those in memory going from the most important feature to the least important. The order of feature relevance is determined by Information Gain.

For known words, the system used here has access to information about the focus word and its potential tags, the disambiguated tags in the two preceding positions and the undisambiguated tags in the two following positions. For unknown words, only one preceding and following position, three suffix letters and information about capitalization and presence of a hyphen or a digit are used as features. The case base for unknown words is constructed from only those words in the training set which occur five times or less.

Maximum Entropy Modeling (cf. Section 2.4) can also be used for tagging: a maximum entropy tagger, called MXPOST, was developed by (Ratnaparkhi, 1996) (we will refer to this tagger as "**MXP**" below).[20] This system uses a number of word and context features rather similar to system MBT, and trains a Maximum Entropy model using the Improved Iterative Scaling algorithm for one hundred iterations. The final model has a weighting parameter for each feature value that is relevant to the estimation of the probability $P(tag|features)$, and combines the evidence from diverse features in an explicit probability model. In contrast to the other taggers, both known and unknown words are processed by the same model. A striking difference is also that this tagger does not

18 Because of the computational complexity, we have had to exclude the system from the experiments with the very large Wotan tagset.
19 An online version of the tagger is available at `http://ilk.kub.nl/`.
20 Ratnaparkhi's Java implementation of this system is freely available for non-commercial research purposes at `ftp://ftp.cis.upenn.edu/pub/adwait/jmx/`.

have a separate storage mechanism for lexical information about the focus word (i.e. the possible tags). The word is merely another feature in the probability model. As a result, no generalizations over groups of words with the same set of potential tags are possible. In the tagging phase a beam search is used to find the highest probability tag sequence for the whole sentence.

Hidden Markov Models view the tagging task as finding the maximum probability sequence of states in a stochastic finite state machine. The transitions between states emit the words of a sentence with a probability $P(w|S_t)$, the states S_t themselves model tags or sequences of tags. The transitions are controlled by Markovian state transition probabilities $P(S_{t_i}|S_{t_{i-1}})$. Because a sentence could have been generated by a number of different state sequences, the states are considered to be "Hidden". Although methods for unsupervised training of HMM's do exist, training is usually done in a supervised way by estimation of the above probabilities from relative frequencies in the training data. The HMM approach to tagging is by far the most studied and applied (Church, 1988; DeRose, 1988; Charniak, 1993).

In (van Halteren, Zavrel, and Daelemans, 1998) we used a straightforward implementation of HMM's, which turned out to have the worst accuracy of the four competing methods. In the present work, we have replaced this by the TnT system (we will refer to this tagger as "**HMM**" below).[21] TnT is a trigram tagger (Brants, 2000), which means that it considers the previous two tags as features for deciding on the current tag. Moreover, it considers the capitalization of the previous word as well in its state representation. The lexical probabilities depend on the identity of the current word for known words and on a suffix tree smoothed with Successive Abstraction (Samuelsson, 1996) for guessing the tags of unknown words. As we will see below, it shows a surprisingly higher accuracy than our previous HMM implementation. When we compare it with the other taggers used in this paper, we see that a trigram HMM tagger uses a very limited set of features (Table 1). On the other hand, it is able to access some information about the rest of the sentence indirectly, through its use of the Viterbi algorithm.

4. Overall results

The first set of results from our experiments is the measurement of overall accuracy for the base taggers. In addition, we can observe the agreement between the systems, from which we can estimate how much gain we can possibly expect from combination. The application of the various combination systems, finally, shows us how much of the projected gain is actually realized.

4.1 Base tagger quality

An additional benefit of training four popular tagging systems under controlled conditions on several corpora is an experimental comparison of their accuracy. Table 2 lists the accuracies as measured on the test set.[22] We see that TBL achieves the lowest accuracy on all data sets. MBT is always better than TBL, but is outperformed by both MXP and HMM. On two data sets (LOB and Wotan) the Hidden Markov Model system (TnT) is better than the Maximum Entropy one (MXPOST). On the other two (WSJ and Wotan-Lite) MXPOST is the better system. In all cases, except the difference between MXP and HMM on LOB, the differences are statistically significant ($p < 0.05$, McNemar's chi-

21 The TnT system can be obtained from its author through
`http://www.coli.uni-sb.de/˜ thorsten/tnt/`

22 In this and several following tables, the best performance is indicated with bold type.

Table 2
Baseline and individual tagger test set accuracy for each of our four datasets. The bottom four rows shows the accuracies of the four tagging systems on the various data sets. In addition, we list two baselines, viz. the selection of a completely random tag from among the potential tags for the token (Random) and the selection of the lexically most likely tag (LexProb).

	LOB	WSJ	Wotan	WotanLite
Baseline				
Random	61.46	63.91	42.99	54.36
LexProb	93.22	94.57	89.48	93.40
Single tagger				
TBL	96.37	96.28	–[23]	94.63
MBT	97.06	96.41	89.78	94.92
MXP	97.52	**96.88**	91.72	**95.56**
HMM	**97.55**	96.63	**92.06**	95.26

Table 3
Pairwise agreement between the base taggers. For each base tagger pair and data set, we list the percentage of tokens in the test set on which the two taggers select the same tag.

Dataset	Tagger pair					
	MXP HMM	MXP MBT	MXP TBL	HMM MBT	HMM TBL	MBT TBL
LOB	97.56	96.70	96.27	97.27	96.96	96.78
WSJ	97.41	96.85	96.90	97.18	97.39	97.21
Wotan	93.02	90.81	–	92.06	–	–
WotanLite	95.74	95.12	95.00	95.48	95.36	95.52

squared test).

We can also see from these results that WSJ, although it is about the same size as LOB, and has a smaller tagset, has a higher difficulty level than LOB. We suspect that an important reason for this is the inconcistency in the WSJ annotation (cf. (Ratnaparkhi, 1996)). We examine this effect in more detail below. The Eindhoven corpus, both with Wotan and WotanLite tagset is yet more difficult, but here the difficulty lies mainly in the complexity of the tagset and the large percentage of unknown words in the test sets. We see that the reduction in the complexity of the tagset from Wotan to WotanLite leads to an enormous improvement in accuracy. This granularity effect is also examined in more detail below.

4.2 Base tagger agreement

On the basis of the output of the single taggers we can also examine the feasibility of combination, as combination is dependent on different systems producing different errors. As expected, a large part of the errors are indeed uncorrelated: the agreement between the systems (Table 3) is at about the same level as their agreement with the benchmark tagging. A more detailed view of inter-tagger agreement is shown in Table 4, which lists the (groups of) patterns of (dis)agreement for the four data sets.

It is interesting to see that although the general accuracy for WSJ is lower than for LOB, the inter-tagger agreement for WSJ is on average higher. It would seem that the less consistent tagging for WSJ makes it easier for all systems to fall into the same traps. This becomes even clearer if we examine the patterns of agreement and see, e.g., that the number of tokens where all taggers agree on a wrong tag is practically doubled.

23 The training of TBL on the large Wotan tagset was aborted after several weeks of training failed to produce any useful results.

Table 4
The presence of various tagger (dis)agreeement patterns for the four data sets. In addition to the percentage of the test sets for which the pattern is observed (%), we list the cumulative percentage (%Cum).

	LOB		WSJ		Wotan		WotanLite	
Pattern	%	%Cum	%	%Cum	%	%Cum	%	%Cum
All taggers agree and are correct.	93.93	93.93	93.80	93.80	85.68	85.68	90.50	90.50
A majority is correct.	3.30	97.23	2.64	96.44	6.54	92.22	4.73	95.23
Correct tag is present but is tied.	1.08	98.31	1.07	97.51	0.82	93.04	1.59	96.82
A minority is correct.	0.91	99.22	1.12	98.63	2.62	95.66	1.42	98.24
The taggers vary, but are all wrong.	0.21	99.43	0.26	98.89	1.53	97.19	0.46	98.70
All taggers agree but are wrong.	0.57	100.00	1.11	100.00	2.81	100.00	1.30	100.00

Table 5
Projected accuracies for increasingly successful levels of combination achievement. For each level we list the accuracy (%) and the percentage of errors made by the best individual tagger that can be corrected by combination (Δ_{Err}).

	LOB		WSJ		Wotan		WotanLite	
Pattern	%	Δ_{Err}	%	Δ_{Err}	%	Δ_{Err}	%	Δ_{Err}
Best single tagger	HMM 97.55	–	MXP 96.88	–	HMM 92.06	–	MXP 95.56	–
Ties randomly correct	97.77	9.0	96.97	2.8	92.49	5.5	96.01	10.1
All ties correct	98.31	31.3	97.50	19.9	93.04	12.4	96.82	28.3
Minority vs 2 taggers correct	98.48	48.5	97.67	25.4	95.66	45.3	97.09	34.3
Minority vs 3 taggers correct	99.22	68.4	98.63	56.0	–	–	98.24	60.3

The agreement pattern distribution enables us to determine levels of combination quality. Table 5 lists both the accuracies of several ideal combiners (%) and the error reduction in relation to the best base tagger for the data set in question (Δ_{Err}).[24] E.g. on LOB, “All ties correct” produces 1941 errors (corresponding to an accuracy of 98.31%), which is 31.3% less than HMM’s 2824 errors. A minimal level of combination achievement is that a majority or better will lead to the correct tag and that ties are handled appropriately about 50% of the time for the (2-2) pattern and 25% for the (1-1-1-1) pattern (or 33.3% for the (1-1-1) pattern for Wotan). In more optimistic scenarios, a combiner is able to select the correct tag in all tied cases, or even in cases where a two- or three-tagger majority must be overcome. Although the possibility of overcoming a majority is present with the arbiter type combiners, the situation is rather improbable. As a result,

24 We express the error reduction in the form of a percentage, i.e. a relative measure, instead of by an absolute value, because we feel this is the more informative of the two. After all, there is a vast difference between an accuracy improvement of 0.5% from 50% to 50.5% (a Δ_{Err} of 1%) and one of 0.5% from 99% to 99.5% (a Δ_{Err} of 50%).

Table 6
Accuracies of the combination systems on all four corpora. For each system we list its accuracy (%) and the percentage of errors made by the best individual tagger that is corrected by the combination system (Δ_{Err}).

	LOB		WSJ		Wotan		WotanLite	
	%	Δ_{Err}	%	Δ_{Err}	%	Δ_{Err}	%	Δ_{Err}
Best single tagger	HMM		MXP		HMM		MXP	
	97.55	–	96.88	–	92.06	–	95.56	–
Voting								
Majority	97.76	9.0	96.98	3.1	92.51	5.7	96.01	10.1
TotPrecision	97.95	16.2	97.07	6.1	92.58	6.5	96.14	12.9
TagPrecision	97.82	11.2	96.99	3.4	92.51	5.7	95.98	9.5
Precision-Recall	97.94	16.1	97.05	5.6	92.50	5.6	96.22	14.8
TagPair	97.98	17.8	97.11	7.2	92.72	8.4	96.28	16.2
Stacked Classifiers								
WPDV(tags)	98.06	20.8	97.15	8.7	92.86	10.1	96.33	17.2
WPDV(tags+word)	98.07	21.4	97.17	9.3	92.85	10.0	96.34	17.5
WPDV(tags+context)	**98.14**	**24.3**	**97.23**	**11.3**	93.03	12.2	**96.42**	**19.3**
MBL(tags)	98.05	20.5	97.14	8.5	92.72	8.4	96.30	16.7
MBL(tags+word)	98.02	19.2	97.12	7.6	92.45	5.0	96.30	16.6
MBL(tags+context)	98.10	22.6	97.11	7.2	92.75	8.7	96.31	16.8
DecTrees(tags)	98.01	18.9	97.14	8.3	92.63	7.2	96.31	16.8
DecTrees(tags+word)	–[26]	–	–	–	–	–	–	–
DecTrees(tags+context)	98.03	19.7	97.12	7.7	–	–	96.26	15.7
Maccent(tags)	98.03	19.6	97.10	7.1	92.76	8.9	96.29	16.4
Maccent(tags+word)	98.02	19.3	97.09	6.6	92.63	7.2	96.27	16.0
Maccent(tags+context)	98.12	23.5	97.10	7.0	**93.25**	**15.0**	96.37	18.2

we ought to be more than satisfied if any combiners approach the level corresponding to the projected combiner which resolves all ties correctly.[25]

4.3 Results of combination

In Table 6 the results of our experiments with the various combination methods are shown. Again we list both the accuracies of the combiners (%) and the error reduction in relation to the best base tagger (Δ_{Err}). E.g. on LOB, "TagPair" produces 2321 errors (corresponding to an accuracy of 97.98%), which is 17.8% less than H's 2824 errors.

Although the combiners generally fall short of the "All ties correct" level (cf. Table 5), even the most trivial voting system (Majority), significantly outperforms the best individual tagger on all data sets. Within the simple voting systems, it appears that use of more detailed voting weights does not necessarily lead to better results. TagPrecision is clearly inferior to TotPrecision. On closer examination, this could have been expected. Looking at the actual tag precision values (cf. Table 9 below), we see that the precision is generally more dependent on the tag than on the tagger, so that TagPrecision always tends to select the easier tag. In other words, it uses less specific rather than more specific information. Precision-Recall is meant to correct this behaviour by the involvement of recall values. As intended, Precision-Recall generally has a higher accuracy than TagPrecision, but does not always improve on TotPrecision.

25 The bottom rows might be viewed in the light of potential future extremely intelligent combination systems. For the moment, however, it is better to view them as containing recall values for n-best versions of the combination taggers, e.g. an n-best combination tagger for LOB which simply provides all tags suggested by its four components will have a recall score of 99.22%.

26 C5.0 was not able to cope with the large amount of data involved in all Tags+Word experiments and the Tags+Context experiment with Wotan.

Our previously unconfirmed hypothesis, that arbiter type combiners would be able to outperform the gang type ones, is now confirmed. With the exception of several of the Tags+Word versions and the Tags+Context version for WSJ, the more sophisticated modelling systems have a significantly better accuracy than the simple voting systems on all four data sets. TagPair, being somewhere between simple voting and stacking, also falls in the middle where accuracy is concerned. In general, it can at most be said to stay close to the real stacking systems, except for the cleanest data set, LOB, where it is clearly being outperformed. This is a fundamental change from our earlier experiments, where TagPair was significantly better than MBL and Decision Trees. Our explanation at the time, that the stacked systems suffered from a lack of training data, appears to be correct. A closer investigation below shows at which amount of training data the cross-over point in quality occurs (for LOB).

Another unresolved issue from the earlier experiments is the effect of making word or context information available to the stacked classifiers. With LOB and a single 114K tune set (van Halteren, Zavrel, and Daelemans, 1998), both MBL and Decision Trees degraded significantly when adding context and MBL when adding the word.[27] With the increased amount of training material, addition of the context generally leads to better results. For MBL, there is a degradation only for the WSJ data, and of a much less pronounced nature. With the other data sets there is an improvement, significantly so for LOB. For Decision Trees, there is also a limited degradation for WSJ and WotanLite, and a slight improvement for LOB. The other two systems appear to be able to use the context more effectively. WPDV shows a relatively constant significant improvement over all data sets. Maccent shows more variation, with a comparable improvement on LOB and WotanLite, a very slight degradation on WSJ, and a spectacular improvement on Wotan, where it even yields an accuracy higher than the “All ties correct” level.[28] Addition of the word is still generally counterproductive. Only WPDV sometimes manages to translate the extra information into an accuracy improvement, and even then a very small one. It would seem that vastly larger amounts of training data are necessary if the word information is to become useful.

5. Combination in detail

The observations about the overall accuracies, although the most important, are not the only interesting ones. We can also examine the results of the experiments above in more detail, evaluating the results of combination for specific words and tags, and trying to discover why such disappointing results are found for WSJ. Furthermore, we can run additional experiments, to determine the effects of the size of the training set, the number of base tagger components involved and the granularity of the tagset.

5.1 Specific words

The overall accuracy of the various tagging systems gives a good impression of relative performance, but it is also useful to have a more detailed look at the tagging results. Most importantly for this paper, the details give a better feel for the differences between the base taggers and for how well a combiner can exploit these differences. More generally, users of taggers or tagged corpora are rarely interested in the whole corpus. They rather focus on specific words or wordclasses, for which the accuracy of tagging may

27 Just as in the current experiments, the Decision Tree system could not cope with the amount of data when the word was added.

28 We have no clear explanation for this exceptional behaviour, but conjecture that Maccent is able to make optimal use of the tagging differences caused by the high error rate of all four taggers.

Table 7
Error rates for the most confusing words. For each word, we list the total number of instances in the test set (n), the number of tags associated with the word (tags), and then, for each base tagger and WPDV(tags+context), the rank in the error list (rank), the absolute number of errors (err) and the percentage of instances that is mistagged (%).

		MXP			HMM			MBT			TBL			WPDV(t+c)		
word	n/tags	rank	err	%	rank	err	%	rank	err	%	rank	err	%	rank	err	%
as	719/17	1	102	14.19	1	130	18.08	3	120	16.69	1	167	23.23	1	**82**	**11.40**
that	1108/6	2	98	8.84	2	105	9.48	1	130	11.73	2	134	12.09	2	**80**	**7.22**
to	2645/9	3	81	2.76	3	59	2.23	2	122	4.61	3	131	4.27	3	**40**	**1.51**
more	224/4	4	52	23.21	4	42	18.75	4	46	20.54	5	53	23.76	5	**30**	**13.39**
so	247/10	6	32	12.96	6	40	16.19	6	40	16.19	4	63	25.51	4	**31**	**12.55**
in	2102/14	11	**22**	**1.05**	7	35	1.67	5	43	2.46	6	48	2.28	6	25	1.19
about	177/3	5	37	20.90	5	41	23.16	7	30	16.95	17	23	12.99	7	**22**	**12.43**
much	117/2	7	30	25.64	10	27	23.08	8	27	23.08	9	35	29.91	9	**20**	**17.09**
her	373/3	18	13	3.49	21	10	2.68	17	18	4.83	7	39	10.46	25	**7**	**1.88**

differ greatly from the overall accuracy.

We start our detailed examination with the words that are most often mistagged. We use the LOB corpus for this evaluation, as it is the cleanest data set and hence the best example. For each base tagger, and for WPDV(tags+context), we list the top seven mistagged words, in terms of absolute numbers of errors, in Table 7. Although the base taggers have been shown (in 4.2) to produce different errors, we see that they do tend to make errors on the same words, as the five top-sevens together contain only nine words.

A high number of errors for a word is due to a combination of tagging difficulty and frequency. Examples of primarily difficult words are "much" and "more". Even though they have relatively low frequencies, they have high ranks on the error lists. These words can be recognized by their high error percentage scores. Examples of merely frequent words are "to" and "in". The error percentages show that these two words are actually tagged surprisingly well, as "to" is usually quoted as a tough case and for "in" the taggers have to choose between 14 possible tags. The first place on the list is taken by "as" with both a high frequency and a high difficulty level (it is also the most ambiguous word with 17 possible tags in LOB).

Table 7 shows yet again that there are clear differences between the base taggers, providing the opportunity for effective combination. For all but one word, "in", the combiner manages to improve on the best tagger for that specific word. If we compare to the overall best tagger, HMM, the improvements are sometimes spectacular. This is of course especially the case where HMM has particular difficulties with a word, e.g. "about" with a 46.3% reduction in error rate, but in other cases as well, e.g. "to" with a 32.2% reduction, which is still well above the overall error rate reduction of 24.3%.

5.2 Specific tags

We can also abstract away from the words and simply look at common wordclass confusions, e.g. a token which should be tagged VBD (past tense verb) is actually tagged VBN (past participle verb). Table 8 shows the tag confusions which are present in the top seven confusion list of at least one of the systems (again the four base taggers and WPDV(tags+context) used on LOB). The number on the right in each system column is the number of times the error was made and the number on the left is the position

Table 8
Confusion rates for the most often confused tag pairs. For each pair (tagger, correct), we first take the two possible confusion directions separately and list the corresponding error list ranks (rank) and absolute number of errors (err) for the four base taggers and for WPDV(tags+context). Then we list the same information for the pair as a whole, i.e. for the two directions together.

		MXP		HMM		MBT		TBL		WPDV(t+c)	
tagger	correct	rank	err	rank	err	rank	err	rank	err	rank	err
VBN	VBD	6	**92**	1	154	1	205	1	236	3	102
VBD	VBN	3	118	3	117	3	152	3	149	4	**100**
	pair		210		271		357		385		**202**
JJ	NN	2	132	2	150	2	168	2	205	2	**109**
NN	JJ	1	153	6	**75**	4	148	4	148	1	110
	pair		285		225		316		353		**219**
IN	CS	4	105	4	93	5	122	8	97	5	**79**
CS	IN	10	55	7	70	10	64	6	122	8	**48**
	pair		160		163		186		219		**127**
NN	VB	5	98	5	78	6	116	5	132	6	**59**
VB	NN	25	**28**	14	45	12	60	7	100	15	35
	pair		126		123		176		232		**94**
IN	RP	7	59	10	61	7	99	12	83	7	**50**
RP	IN	24	**30**	18	38	27	34	21	42	18	**30**
	pair		89		99		133		125		**80**

Table 9
Precision and recall for tags involved in most often confused tag pairs. For each tag, we list the percentage of tokens in the test set that are tagged with that tag (%test), followed by the precision (Prec) and recall (Rec) values for each of the systems.

		MXP	HMM	MBT	TBL	WPDV(t+c)
tag	%test	Prec/Rec	Prec/Rec	Prec/Rec	Prec/Rec	Prec/Rec
CS	1.48	92.69/90.69	90.14/91.10	89.46/89.05	84.85/91.51	**93.11/93.38**
IN	10.57	97.58/98.95	97.83/98.59	97.14/98.17	97.33/97.62	**98.37/99.03**
JJ	5.58	94.52/94.55	94.07/95.61	92.79/94.38	90.66/94.06	**95.64/96.00**
NN	13.11	96.68/97.85	**97.91**/97.24	96.59/97.22	96.00/96.31	97.66/**98.25**
RP	0.79	95.74/91.82	94.78/92.27	95.26/88.84	93.05/90.28	**95.95/94.14**
VB	2.77	98.04/95.55	97.95/95.99	96.79/94.55	95.09/93.36	**98.13/97.06**
VBD	2.17	94.20/**95.22**	94.23/93.06	92.48/90.29	91.74/87.40	**95.26**/95.14
VBN	2.30	94.07/93.29	90.93/93.37	89.59/90.54	87.09/90.99	**94.25/94.50**

in the confusion list. The rows marked with tag values show the individual errors.[29] In addition, the 'pair' rows show the combined value of the two inverse errors preceding it.[30]

As with the word errors above, we see substantial differences between the base taggers. Unlike the situation with words, there are now a number of cases where base taggers perform better than the combiner. Partly, this is because the base tagger is outvoted to such a degree that its quality cannot be maintained, e.g. NN→JJ. Furthermore, it is probably unfair to look at only one half of a pair. Any attempt to decrease the number of errors of type X→Y will tend to increase the number of errors of type Y→X. The balance between the two is best shown in the 'pair' rows, and here the combiner is again performing excellently, in all cases improving on the best base tagger for the pair.

For an additional point of view, we show the precision and recall values of the

29 The tags are: CS=subordinating conjunction, IN=preposition, JJ=adjective, NN=singular common noun, RP=adverbial particle, VB=base form of verb, VBD=past tense of verb, VBN=past participle.
30 RP→IN is not actually in any top seven, but has been added to complete the last pair of inverse errors.

Table 10
A comparison of benchmark consistency on a small sample of WSJ and LOB. We list the reasons for differences between WPDV(tags+context) output and the benchmark tagging, both in terms of absolute numbers and percentages of the whole test set.

	WSJ		LOB	
	tokens	%	tokens	%
Tagger wrong, Benchmark right	250	1.97	200	1.75
Benchmark wrong, Tagger right	90	0.71	11	0.10
Both wrong	7	0.06	1	0.01
Benchmark left ambiguous, Tagger right	2	0.02	-	-

systems on the same tags in Table 9, as well as the percentage of the test set which should be tagged with each specific tag. The differences between the taggers are again present, and in all but two cases the combiner produces the best score for both precision and recall. Furthermore, as precision and recall form yet another balanced pair, i.e. as improvements in recall tend to decrease precision and vice versa, the remaining two cases (NN and VBD), can be considered to be handled quite adequately as well.

5.3 Effects of inconsistency

Seeing the rather bad overall performance of the combiners on WSJ, we feel the need to identify a property of the WSJ material that can explain this relative lack of success. A prime candidate for this property is the allegedly very low degree of consistency of the WSJ material. We can investigate the effects of the low consistency by way of comparison with the LOB data set, which is known to be very consistent.

We have taken one tenth of the test sets of both WSJ and LOB and manually examined each token where the WPDV(tags+context) tagging differs from the benchmark tagging. The first indication that consistency is a major factor in performance is found in the basic correctness information, given in Table 10. For WSJ, there is a much larger percentage where the difference in tagging is due to an erroneous tag in the benchmark. This does not mean, however, that the tagger should be given a higher accuracy score, as it may well be that the part of the benchmark where tagger and benchmark do agree contains a similar percentage of benchmark errors. It does imply, though, that the WSJ tagging contains many more errors than the LOB tagging, which is likely to be detrimental to the derivation of automatic taggers.

The cases where the tagger is found to be wrong provide interesting information as well. Our examination shows that 109 of the 250 erroneous tags occur in situations which are handled rather inconsistently in the corpus.

In some of these situations we only have to look at the word itself. The most numerous type of problematic word (21 errors) is the proper noun ending in 's'. It appears to be unclear whether such a word should be tagged NNP or NNPS. When taking the words leading to errors in our 1% test set and examining them in the training data, we see a near-even split for practically every word. The most frequent ones are "Securities" (146 NNP vs 160 NNPS) and "Airlines" (72 NNP vs 83 NNPS). There are only two very unbalanced cases, viz. "Times" (78 NNP vs 6 NNPS) and "Savings" (76 NNP vs 21 NNPS). A similar situation occurs, although less frequently, for common nouns, e.g. "headquarters" gets 67 NN and 21 NNS tags.

In other cases, difficult words are handled inconsistently in specific contexts. Examples here are "about" in cases such as "'about 20" (405 IN vs 385 RB) or "about \$20" (243 IN vs 227 RB), "ago" in cases such as "years ago" (152 IN vs 410 RB) and "more" in "more than" (558 JJR vs 197 RBR).

Finally, there are more general wordclass confusions, such as adjective/particle or

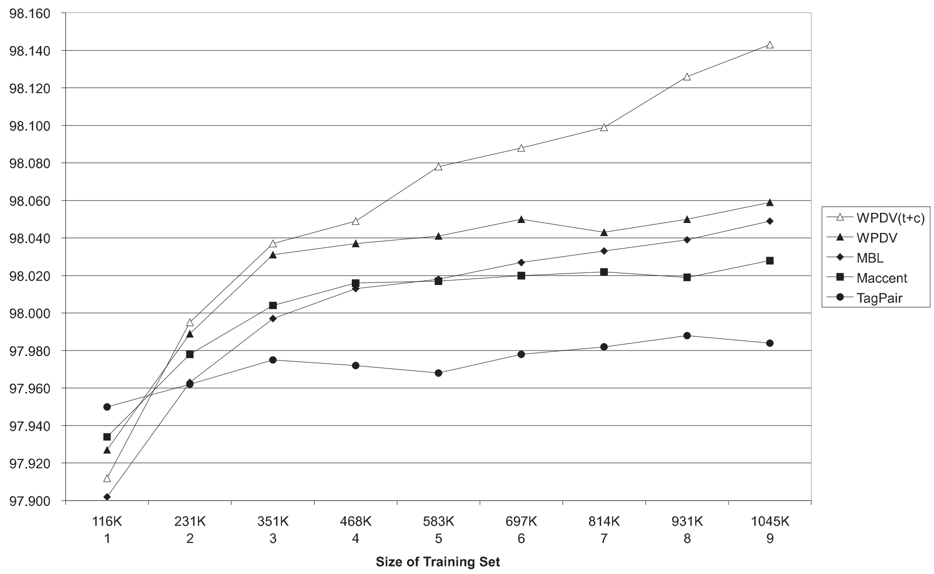

Figure 5
The accuracy of combiner methods on LOB as a function of the number of tokens of training material.

noun/adjective in noun-premodifying positions. Here it is much harder to provide numerical examples, as the problematic situation must first be recognized. We therefore limit ourselves to a few sample phrases. The first is "stock-index" which leads to several errors in combinations like "stock-index futures" or "stock-index arbitrage". In the training set, "stock-index" in premodifying position is tagged JJ 64 times and NN 69 times. The second phrase "chief executive officer" has three words so that we have four choices of tagging: JJ-JJ-NN is chosen 90 times, JJ-NN-NN 63 times, NN-JJ-NN 33 times and NN-NN-NN 30 times.

Admittedly, all of these are problematic cases and many other cases are handled quite consistently. However, the inconsistently handled cases do account for 44% of the errors found for our best tagging system. Under the circumstances, we feel quite justified in assuming that inconsistency is the main cause of the low accuracy scores.[31]

5.4 Size of the training set

The most important result which has undergone a change between (van Halteren, Zavrel, and Daelemans, 1998) and our current experiments is the relative accuracy of TagPair and stacked systems such as MBL. Where TagPair used to be significantly better than MBL, the roles are now well reversed. It appears that our hypothesis at the time, that the stacked systems were plagued by a lack of training data, is correct, since they can now hold their own. In order to see at which point TagPair is overtaken, we have

31 Another property that might contribute to the relatively low scores for the WSJ material is the use of a very small tagset. This makes annotation easier for human annotators, but it provides much less information to the automatic taggers and combiners. It may well be that the remaining information is insufficient for the systems to discover useful disambiguation patterns in. Although we cannot measure this effect for WSJ, because of the many differences with the LOB data set, we feel that it has much less influence than the inconsistency of the WSJ material.

Table 11
WPDV(tags+context) accuracy measurements for various component tagger combinations. For each combination, we list the tagging accuracy (Test), the error reduction expressed as a percentage of the error count for the best component base tagger ($\Delta_{Err}(best)$) and any subsequent error reductions when adding further components (Gain).

Comb	Test	$\Delta_{Err}(best)$	Gain +TBL	Gain +MBT	Gain +MXP	Gain +HMM
TBL	96.37	–	–	29.1	**40.2**	38.9
MBT	97.06	–	12.5	–	**28.4**	26.0
MBT+TBL	97.43	12.5 (MBT)	–	–	**20.6**	17.2
MXP	97.52	–	12.3	15.0	–	**16.2**
HMM	97.55	–	9.5	11.3	**15.3**	–
HMM+TBL	97.78	9.5 (HMM)	–	4.0	**11.8**	–
HMM+MBT	97.82	11.3 (HMM)	2.0	–	**13.7**	–
MXP+TBL	97.83	12.3 (MXP)	–	6.0	–	**9.9**
HMM+MBT+TBL	97.87	13.1 (HMM)	–	–	**12.9**	–
MXP+MBT	97.89	15.0 (MXP)	3.0	–	–	**10.8**
MXP+HMM	97.92	15.3 (HMM)	5.7	**9.6**	–	–
MXP+MBT+TBL	97.96	17.6 (MXP)	–	–	–	**9.1**
MXP+HMM+TBL	98.04	20.1 (HMM)	–	**5.2**	–	–
MXP+HMM+MBT	98.12	23.4 (HMM)	**1.1**	–	–	–
MXP+HMM+MBT+TBL	98.14	24.3 (HMM)	–	–	–	–

trained several systems on increasing amounts of training data from LOB.[32] Each increment is one of the 10% training corpus parts described above. The results are shown in Figure 5.

TagPair is only best when a single part is used (as in the earlier experiments). After that it is overtaken and quickly left behind, as it is increasingly unable to use the additional training data to its advantage.

The three systems using only base tagger outputs have comparable accuracy growth curves, although the initial growth is much higher for WPDV. The curves for WPDV and Maccent appear to be leveling out towards the right end of the graph. For MBL, this is much less clear. However, it would seem that the accuracy level at 1Mw is a good approximation of the eventual ceiling.

The advantage of the use of context information becomes clear at 500Kw. Here the tags-only systems start to level out, but WPDV(tags+context) keeps showing a constant growth. Even at 1Mw, there is no indication that the accuracy is approaching a ceiling. The model seems to be getting increasingly accurate in correcting very specific contexts of mistagging.

5.5 Interaction of Components

Another way in which the amount of input data can be varied is by taking subsets of the set of component taggers. The relation between the accuracy of combinations for LOB (using WPDV(tags+context)) and that of the individual taggers is shown in Table 11. The first three columns show the combination, the accuracy and the improvement in relation to the best component. The other four columns show the further improvement gained when adding yet another component.

The most important observation is that every combination outperforms the combination of any strict subset of its components. The difference is always significant, except in the cases MXP+HMM+MBT+TBL vs MXP+HMM+MBT and HMM+MBT+TBL vs

32 Only combination uses a variable number of parts. The base taggers are always trained on the full 90%.

HMM+MBT.

We can also recognize the quality of the best component as a major factor in the quality of the combination results. HMM and MXP always add more gain than MBT, which always adds more gain than TBL. Another major factor is the difference in language model. MXP, although having a lower accuracy by itself than HMM, yet leads to better combination results, again witnessed by the "Gain" columns. In some cases, MXP is even able to outperform pairs of components in combination: both MXP+MBT and MXP+HMM are better than HMM+MBT+TBL.

5.6 Effects of granularity

The final influence on combination which we measure is that of the granularity of the tagset, which can be examined with the highly structured Wotan tagset. Part of the examination has already taken place above, as we have added the WotanLite tagset, a less granular projection of Wotan. As we have seen, the WotanLite taggers undeniably have a much higher accuracy than the Wotan ones. However, this is hardly surprising, as they have a much easier task to perform. In order to make a fair comparison, we now measure them at their performance of the same task, viz. the prediction of WotanLite tags. We do this by projecting the output of the Wotan taggers (i.e. the base taggers, WPDV(tags) and WPDV(tags+context)) to WotanLite tags. Additionally, we measure all taggers at the main wordclass level, i.e. after the removal of all attributes and ditto tag markers.

All results are listed in Table 12. The three major vertical blocks each represent a level at which the correctness of the final output is measured. Within the lower two blocks, the three lines represent the type of tags used by the base taggers. The lines for Wotan and WotanLite represent the actual taggers, as described above. The line for BestLite does not represent a real tagger, but rather a virtual tagger which corresponds to the best tagger from among Wotan (with its output projected to WotanLite format) and WotanLite. This choice for the best granularity is taken once for each system as a whole, not per individual token. This leads to BestLite being always equal to WotanLite for TBL and MBT, and to projected Wotan for MXP and HMM.

The three major horizontal blocks represent combination strategies, viz. no combination, combination using only the tags and combination using tags and direct context. The two combination blocks are divided into three columns, representing the tag level at which combination is performed, e.g. for the Lite column the output of the base taggers is projected to WotanLite tags which are then used as input for the combiner.

We hypothesized beforehand that, in general, the more information a system can use, the better its results are. Unfortunately, even for the base taggers, reality is not that simple. For both MXP and HMM, the Wotan tagger indeed yields a better WotanLite tagging than the WotanLite tagger itself, thus supporting the hypothesis. On the other hand, the results for MBT do not confirm this, as here the WotanLite tagger is more accurate. However, we have already seen that MBT has severe problems in dealing with the complex Wotan data. Furthermore, the lowered accuracy of the MBL combiners when provided with words (cf. 4.3) also indicate that memory-based learning sometimes has problems in coping with a surplus of information. This means that we have to adjust our hypothesis: more information is better, but only up to the point where the wealth of information overwhelms the machine learning system. Where this point is found obviously differs per system.

For the combiners, the situation is rather inconclusive. In some cases, especially for WPDV(tags), combining at a higher granularity (i.e. using more information) produces better results. In others, combining at a lower granularity works better. In all cases, the

Table 12
Accuracy for base taggers and different levels combiners, as measured at various levels of granularity. The rows are divided into blocks, each listing accuracies for a different comparison granularity. Within a block, the individual rows list which base taggers are used as ingredients in the combination. The columns contain, from left to right, the accuracies for the base taggers, the combination accuracies when using only tags (WPDV(tags)) at three different levels of combination granularity (Full, Lite and Main) and the combination accuracies when adding context (WPDV(tags+context)), at the same three levels of combination granularity.

	base taggers				WPDV(tags)			WPDV(tags+context)		
	TBL	MBT	MXP	HMM	Full	Lite	Main	Full	Lite	Main
Measured as Wotan tags										
Wotan	–	89.78	91.72	92.06	92.83	–	–	93.03	–	–
Measured as WotanLite tags										
Wotan	–	94.56	95.71	95.98	96.50	96.49	–	96.53	96.54	–
WotanLite	94.63	94.92	95.56	95.26	–	96.32	–	–	96.42	–
BestLite	94.63	94.92	95.71	95.98	–	96.58	–	–	96.64	–
Measured as main wordclass tags										
Wotan	–	96.55	97.23	97.54	97.88	97.87	97.85	97.88	97.89	97.91
WotanLite	96.37	96.76	97.12	96.96	–	97.69	97.71	–	97.76	97.77
BestLite	96.37	96.76	97.23	97.54	–	97.91	97.90	–	97.94	97.93

difference in scores between the columns is extremely small and hardly supports any conclusions either way. What is obviously much more important for the combiners is the quality of the information they can work with. Here, higher granularity on the part of the ingredients is preferable, as combiners based on Wotan taggers perform better than those based on WotanLite taggers,[33] and ingredient performance seems to be even more useful, as BestLite yields yet better results in all cases.

6. Related Research

Combination of ensembles of classifiers, although well-established in the Machine Learning literature, has only recently been applied as a method for increasing accuracy in Natural Language Processing tasks. There has of course always been a lot of research on the combination of different methods (e.g. knowledge-based and statistical) in hybrid systems, or on the combination of different information sources. Some of that work even explicitly uses voting and could therefore also be counted as an ensemble approach, e.g. (Rigau, Atserias, and Agirre, 1997) combine different heuristics for word sense disambiguation by voting and (Agirre et al., 1998) do the same for spelling correction evaluation heuristics. The difference between single classifiers learning to combine information sources, i.e. their input features (see (Roth, 1998) for a general framework), and the combination of ensembles of classifiers trained on subsets of those features is not always very clear anyway.

For part of speech tagging, significant accuracy increase by combining the output of different taggers was first demonstrated in (van Halteren, Zavrel, and Daelemans, 1998) and (Brill and Wu, 1998). In both approaches, different tagger generators were applied to the same training data and their predictions combined using different combination methods, including stacking. Yet, the latter paper reported much lower accuracy im-

33 However, this comparison is not perfect, as the combination of Wotan tags does not include TBL. On the one hand, this means the combination has less information to go on and we should hence be even more impressed with the better performance. On the other hand, TBL is the lowest scoring base tagger, so maybe the better performance is due to not having to cope with a flawed ingredient.

Table 13
A comparison of our results for WSJ with those by Brill et al. (1998).

	Brill and Wu		Our experiments
training/test split	80/20	=	90/10
Unigram	93.26	LexProb	94.57
Trigram	96.36	TnT	96.63
–		MBT	96.41
Transformation	96.61	Transformation	96.28
Maximum Entropy	96.83	Maximum Entropy	96.88
Transf. based comb.	97.16	WPDV(tags+context)	97.23
Error rate reduction	10.4%	Error rate reduction	11.3%

provement figures. As we now apply the methods of (van Halteren, Zavrel, and Daelemans, 1998) to WSJ as well, it is easier to make a comparison. An exact comparison is still impossible, as we have not used the exact same data preparation and taggers, but we can put roughly corresponding figures side by side (Table 13). As for base taggers, the first two differences are easily explained: Unigram has to deal with unknown words, while LexProb does not, and TnT is a more advanced trigram system. The slight difference for MaxEnt might be explained by the difference in training/test split. What is more puzzling is the substantial difference for the transformation based tagger. Possible explanations are that Brill and Wu used a much better parametrisation of this system or that they used a different version of the WSJ material. Be that as it may, the final results are comparable and it is clear that the lower numbers in relation to LOB are caused by the choice of test material (WSJ) rather than by the methods used.

In (Tufiş, D., 1999), a single tagger-generator is trained on different corpora representing different language registers. For the combination, a method called *credibility profiles* worked best. In such a profile, for each component tagger, information is kept about its overall accuracy, its accuracy for each tag, etc. In another recent study, (Màrquez et al., 1999) investigate several types of ensemble construction in a decision tree learning framework for tagging specific classes of ambiguous words (as opposed to tagging all words). The construction of ensembles was based on bagging, selection of different subsets of features (e.g. context and lexical features) in decision tree construction, and selection of different splitting criteria in decision tree construction. In all experiments simple voting was used to combine component tagger decisions. All combination approaches resulted in a better accuracy (an error reduction between 8% and 12% on average compared to the basic decision tree trained on the same data). But as these error reductions refer to only part of the tagging task (18 ambiguity classes), they are hard to compare with our own results.

In (Abney, Schapire, and Singer, 1999), ADABOOST variants are used for tagging WSJ material. Component classifiers here are based on different information sources (subsets of features), e.g. capitalization of current word, and the triple "string, capitalization, and tag" of the word to the left of the current word are the basis for the training of some of their component classifiers. Resulting accuracy is comparable to, but not better than, that of the maximum entropy tagger. Their approach is also demonstrated for prepositional phrase attachment, again with results comparable to but not better than state-of-the-art single classifier systems. High accuracy on the same task is claimed by (Alegre, Sopena, and Lloberas, 1999) for combining ensembles of neural networks. ADABOOST has also been applied to text filtering (Schapire, Singer, and Singhal, 1998) and text categorization (Schapire and Singer, 1998).

In (Chen, Bangalore, and Vijay-Shanker, 1999), classifier combination is used to overcome the sparse data problem when using more contextual information in *supertag-*

ging, an approach in which parsing is reduced to tagging with a complex tagset (consisting of partial parse trees associated with lexical items). When using pairwise voting on models trained using different contextual information, an error reduction of 5% is achieved over the best component model. Parsing is also the task to which (Henderson and Brill, 1999) apply combination methods with reductions of up to 30% precision error and 6% recall error compared to the best previously published results of single statistical parsers.

This recent research shows that the combination approach is potentially useful for many NLP tasks apart from tagging.

7. Conclusion

Our experiments have shown that, at least for the wordclass tagging task, combination of several different systems enables us to raise the performance ceiling which can be observed when using data driven systems. For all tested data sets, combination provides a significant improvement over the accuracy of the best component tagger. The amount of improvement varies from 11.3% error reduction for WSJ to 24.3% for LOB. The data set that is used appears to be the primary factor in the variation, especially the data set's consistency.

As for the type of combiner, all stacked systems using only the set of proposed tags as features reach about the same performance. They are clearly better than simple voting systems, at least as long as there is sufficient training data. In absence of enough data, one has to fall back to less sophisticated combination strategies. Addition of word information does not lead to improved accuracy, at least with the current training set size. However, it might still be possible to get a positive effect by restricting the word information to the most frequent and ambiguous words only. Addition of context information does lead to improvements for most systems. WPDV and Maccent make the best use of the extra information, with WPDV having an edge for less consistent data (WSJ) and Maccent for high error rate material (Wotan).

Although the results reported in this paper are very positive, many directions of research remain to be explored in this area. In particular, we have high expectations of the following two directions. First, there is reason to believe that better results can be obtained by using the probability distributions generated by the component systems, rather than just their best guesses (see e.g. (Ting and Witten, 1997a)). Second, in the present paper we have used disagreement between a fixed set of component classifiers. However, there exist a number of dimensions of disagreement (inductive bias, feature set, data partitions, and target category encoding) that might fruitfully be searched to yield large ensembles of modular components that are evolved to cooperate for optimal accuracy.

Another open question is whether and, if so, when combination is a worthwile technique in actual NLP applications. After all, the natural language text at hand has to be processed by each of the base systems, and then by the combiner. Now none of these is especially bothersome at runtime (most of the computational difficulties being experienced during training), but when combining N systems, the time needed to process the text can be expected to be at least a factor $N + 1$ more than when using a single system. Whether this is worth the improvement that is achieved, which is as yet expressed in percents rather than in factors, will depend very much on the amount of text that has to be processed and the use that is made of the results. There are a few clear-cut cases, such as a corpus annotation project where the CPU-time for tagging is negligible in relation to the time needed for manual correction afterwards (i.e. do use combination), or information retrieval on very large text collections where the accuracy improvement does not have enough impact to justify the enormous amount of extra CPU-time (i.e. do not use combination). However, most of the time the choice between combining or not combining will have to be based on evidence from carefully designed pilot experiments, for which this paper can only hope to provide suggestions and encouragement.

8. Acknowledgements

The authors would like to thank the creators of the tagger generators and classification systems used here for making their systems available, and Thorsten Brants, Guy De Pauw, Erik Tjong Kim Sang, Inge de Mönnink, the other members of the CNTS, ILK and TOSCA research groups, and the anonymous reviewers for comments and discussion.

This research was done while the second and third author were at Tilburg University. Their research was done in the context of the "Induction of Linguistic Knowledge" (ILK) research programme, supported partially by the Netherlands Organization for Scientific Research (NWO).

References

Abney, S., R.E. Schapire, and Y. Singer. 1999. Boosting Applied to Tagging and PP Attachment. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 38–45.

Agirre, E., K. Gojenola, K. Sarasola, and A. Voutilainen. 1998. Towards a Single Proposal in Spelling Correction. In *COLING-ACL '98* , pages 22–28.

Alegre, M., J. Sopena, and A. Lloberas. 1999. PP-Attachment: A Committee Machine Approach. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 231–238.

Ali, K.M. and M.J. Pazzani. 1996. Error Reduction through Learning Multiple Descriptions. *Machine Learning*, 24(3):173–202.

Alpaydin, E. 1998. Techniques for Combining Multiple Learners. In E. Alpaydin, editor, *Proceedings of Engineering of Intelligent Systems*, pages 6–12.

Berger, A., S. Della Pietra, and V. Della Pietra. 1996. Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 22(1).

Berghmans, J. 1994. Wotan, een automatische grammatikale tagger voor het Nederlands. Master's thesis, Dept. of Language and Speech, University of Nijmegen.

Brants, T. 2000. TnT – a statistical part-of-speech tagger. In *Proceedings of the 6th Applied NLP Conference, ANLP-2000, April 29 – May 3, 2000, Seattle, WA.*

Breiman, L. 1996a. Bagging Predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. 1996b. Stacked Regressions. *Machine Learning*, 24(3):49–64.

Brill, E. 1992. A simple rule-based part-of-speech tagger. In *Proceedings of the Third ACL Conference on Applied NLP*, pages 152–155, Trento, Italy.

Brill, E. 1994. Some advances in transformation-based part-of-speech tagging. In *Proceedings AAAI'94.*

Brill, E. and Jun Wu. 1998. Classifier combination for improved lexical disambiguation. In *COLING-ACL'98*, pages 191–195, Montreal, Canada, August 10-14.

Chan, P.K., S.J. Stolfo, and D. Wolpert. 1999. Guest Editors' Introduction. Special Issue on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms. *Machine Learning*, 36(1–2):5–7.

Charniak, E. 1993. *Statistical Language Learning.* Cambridge, MA: The MIT Press.

Chen, J., S. Bangalore, and K. Vijay-Shanker. 1999. New Models for Improving Supertag Disambiguation. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 188–195.

Cherkauer, K.J. 1996. Human Expert–level Performance on a Scientific Image Analysis Task by a System Using Combined Artificial Neural Networks. In P. Chan, editor, *Working Notes of the AAAI Workshop on Integrating Multiple Learned Models*, pages 15–21.

Church, K. W. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proc. of Second Applied NLP (ACL).*

Daelemans, W., A. Van den Bosch, and A. Weijters. 1997. IGTree: using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review*, 11:407–423.

Daelemans, W., J. Zavrel, P. Berck, and S. Gillis. 1996. MBT: A memory-based part of speech tagger generator. In E. Ejerhed and I. Dagan, editors, *Proc. of Fourth Workshop on Very Large Corpora*, pages 14–27. ACL SIGDAT.

Daelemans, W., J. Zavrel, K. Van der Sloot, and A. Van den Bosch. 1999. TiMBL: Tilburg Memory Based Learner, version 2.0, reference manual. Technical Report ILK-9901, ILK, Tilburg University.

Dehaspe, L. 1997. Maximum entropy modeling with clausal constraints. In *Inductive Logic Programming: Proceedings of the 7th International Workshop (ILP-97), Lecture Notes in Artificial Intelligence, 1297*, pages 109–124. Springer Verlag.

DeRose, S. 1988. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14:31–39.

Dietterich, T. G. 1997. Machine Learning Research: Four Current Directions. *AI Magazine*, 18(4):97–136.

Dietterich, T. G. and G. Bakiri. 1995. Solving Multiclass Learning Problems via Error–Correcting Output Codes. *Journal of Artificial Intelligence Research*, 2:263–286.

Dietterich, T.G. 1998. Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Computation*, 10(7):1895–1924.

Freund, Y. and R. E. Schapire. 1996. Experiments with a New Boosting Algorithm. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning, ICML'96*, pages 148–156, San Francisco, CA. Morgan Kaufmann.

Geerts, G., W. Haeseryn, J. de Rooij, and M. van der Toorn. 1984. *Algemene Nederlandse Spraakkunst*. Wolters-Noordhoff, Groningen and Wolters, Leuven.

Golding, A.R. and D. Roth. 1999. A Winnow-Based Approach to Context-Sensitive Spelling Correction. *Machine Learning*, 34(1–3):107–130.

van Halteren, H. 1996. Comparison of tagging strategies, a prelude to democratic tagging. In S. Hockey and N. Ide, editors, *Research in Humanities Computing 4. Selected papers from the ALLC/ACH Conference, Christ Church, Oxford, April 1992*, pages 207–215, Oxford, England. Clarendon Press.

van Halteren, H. 2000a. A default first order weight determination procedure for WPDV models. In *Proceedings of CoNLL-2000*.

van Halteren, H. 2000b. Chunking with WPDV models. In *Proceedings of CoNLL-2000*.

van Halteren, H., J. Zavrel, and W. Daelemans. 1998. Improving data-driven wordclass tagging by system combination. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Montréal, Quebec, Canada*, pages 491–497, Montreal, Canada, August 10-14.

van Halteren (ed.), H. 1999. *Syntactic Wordclass Tagging*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Henderson, J. and E. Brill. 1999. Exploiting Diversity in Natural Language Processing: Combining Parsers. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 187–194.

Johansson, S. 1986. *The tagged LOB Corpus: User's Manual*. Bergen, Norway: Norwegian Computing Centre for the Humanities.

Marcus, M., B. Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Màrquez, L., H. Rodríguez, J. Carmona, and J. Montolio. 1999. Improving POS Tagging Using Machine-Learning Techniques. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 53–62.

Quinlan, J.R. 1993. C4.5*: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Ratnaparkhi, A. 1996. A maximum entropy part-of-speech tagger. In *Proc. of the Conference on Empirical Methods in Natural Language Processing, May 17-18, 1996, University of Pennsylvania*.

Rigau, G., J. Atserias, and E. Agirre. 1997. Combining Unsupervised Lexical Knowledge Methods for Word Sense Disambiguation. In *Proceedings of ACL/EACL '97*, pages 48–55.

Roth, D. 1998. Learning to Resolve Natural Language Ambiguities: A Unified Approach. In *Proceedings of AAAI '98*, pages 806–813.

Samuelsson, Christer. 1996. Handling sparse data by successive abstraction. In *Proceedings of the 16th International Conference on Computational Linguistics, COLING-96, Copenhagen, Denmark*.

Schapire, R. E. and Y. Singer. 1998. BoosTexter: A system for multiclass multi-label text categorization. Technical report, To appear in Machiene Learning, AT&T Labs.

Schapire, R. E., Y. Singer, and A. Singhal. 1998. Boosting and Rocchio applied to text filtering. In *Proceedings of the 21st Annual International Conference on Research and Development in Information Retrieval, SIGIR '98*.

Ting, K.M. and I.H. Witten. 1997a. Stacked generalization: when does it work? In *International Joint Conference on Artificial Intelligence, Japan*, pages 866–871.

Ting, K.M. and I.H. Witten. 1997b. Stacking bagged and dagged models. In *International Conference on Machine Learning, Tennessee*, pages 367–375.

Tufiş, D. 1999. Tiered tagging and Combined Language Models Classifiers. In *Proceedings Workshop on Text, Speech, and Dialogue*.

Uit den Boogaart, P.C. 1975. *Woordfrequenties in geschreven en gesproken Nederlands*. Oosthoek, Scheltema & Holkema, Utrecht.

Wolpert, D. H. 1992. Stacked Generalization. *Neural Networks, Pergamon Press*, 5:241–259.

Zavrel, J. and W. Daelemans. 2000. Bootstrapping a tagged corpus through combination of existing heterogeneous taggers. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC 2000)*, pages 17–20.