Diverse Classifiers for NLP Disambiguation Tasks
Comparison, Optimization, Combination, and Evolution

Jakub Zavrel ♣ Sven Degroeve ♢ Anne Kool ♣
Walter Daelemans ♣
Kristiina Jokinen ♢
♣ CNTS - Language Technology Group, University of Antwerp
{zavrel|kool|daelem}@uia.ua.ac.be
♢ Center for Evolutionary Language Engineering, Ieper, Belgium
{sven.degroeve|kristiina.jokinen}@sail.com

In: TWLT 18. Learning to behave, p. 201-221, 2000

Abstract

In this paper we report preliminary results from an ongoing study that investigates the performance of machine learning classifiers on a diverse set of Natural Language Processing (NLP) tasks. First, we compare a number of popular existing learning methods (Neural networks, Memory-based learning, Rule induction, Decision trees, Maximum Entropy, Winnow Perceptrons, Naive Bayes and Support Vector Machines), and discuss their properties vis à vis typical NLP data sets. Next, we turn to methods to optimize the parameters of single learning methods through cross-validation and evolutionary algorithms. Then we investigate how we can get the best of all single methods through combination of the tested systems in classifier ensembles. Finally we discuss new and more thorough methods of automatically constructing ensembles of classifiers based on the techniques used for parameter optimization.

Keywords: Models and algorithms for computational neural architectures

1 Introduction

In recent years the field of Natural Language Processing (NLP) has been radically transformed by a switch from a deductive methodology (i.e. explaining data from theories or models constructed manually) to an inductive methodology (i.e. deriving models and theories from data) (see e.g. Abney (1996) for a review). An important component of this transformation is the realization that many NLP tasks can be modeled as simple classification tasks or as ensembles of simple classifiers (Daelemans, 1996; Ratnaparkhi, 1997). Thus NLP has been able to capitalize on a large body of research in the field of machine learning and statistical modeling. This, accompanied by the continuing explosion of computer power, storage size, and availability of training corpora, has lead to increasingly accurate language models for a quickly growing number of language modeling tasks. However, which machine learning methods have the best performance on NLP data sets is still an active area of research.

In this paper, we empirically study the performance of a range of supervised learning techniques on a selection of benchmark tasks in NLP. In classification-based, supervised learning, a learning algorithm constructs a classifier for a task by processing a set of examples. Each example associates a feature vector (the problem description) with one of a finite number of classes (the solution). Given a new feature vector, the classifier assigns a class to the problem description it represents by means of extrapolation from a "knowledge structure" extracted from the examples. Different learning algorithms construct different types of knowledge representations: probability distributions, decision trees, rules, exemplars, weight vectors, etc.

Classification-based supervised learning methods seem to be especially well suited for NLP tasks because they fit the main task in all areas of NLP very well: implementing a complex, context-sensitive, mapping between different levels of linguistic representation. Sub-tasks in such

a transformation can be of two types (Daelemans, 1996): segmentation (e.g. decide whether a word or tag is the start or end of an NP), and disambiguation (e.g. decide whether a word is a noun or a verb). We can even carve up complex NLP tasks like syntactic analysis into a number of such classification tasks with input vectors representing a focus item and a dynamically selected surrounding context. Output of one classifier (e.g. a tagger or a chunker) is then used as input by other classifiers (e.g. syntactic relation assignment).

Although the existence of machine learning data repositories (such as UCI (Blake et al., 1998)) has made it easy to compare and benchmark machine learning algorithms, such studies (e.g. Michie et al., 1994) have not usually focused on natural language learning. However, language data sets have characteristics that make them quite different from typical machine learning data sets:

- Size: Millions of example cases, large numbers of features, many of which redundant or irrelevant, and very large numbers of feature values (e.g. all words of a lexicon). This places a high computational burden on many existing learning algorithms.
- Disjunctiveness: Language is characterized by an interplay between rules, (sub)regularities, and exceptions. Even exceptions (that are difficult to distinguish from noise) can be members of small but productive families. Daelemans et al. (1999) have observed that in language datasets low-frequent and exceptional events are important for accurate generalization to unseen events.
- Sparse data: Since language is a system of infinite expression by limited means, the available examples usually cover only a very small portion of the possible space.

With these issues in mind we have conducted a benchmarking study consisting of data sets for several NLP tasks: Grapheme to phoneme conversion, Part of speech tagging, and Word sense disambiguation.

The algorithms which have been evaluated are: Neural networks, Memory-based learning, Rule learning, Decision tree learning, Maximum Entropy learning, Winnow Perceptron, Naive Bayes, and Support Vector Machines. We have run these algorithms on the NLP data sets under identical conditions, and present an overview of the experimental results. These experiments reveal that although some algorithms stick out on average, given a new NLP task, and a particular machine learning algorithm with its default settings, your mileage may vary. It seems worthwhile to look at the application of so called ensemble systems. In ensemble systems, different classifiers are performing the same task, and their differences are leveraged to yield a combined system that has a higher accuracy than the single best component. The reason for this is that, to some degree, the different weaknesses cancel each other out, and the different strengths improve the ensemble system. Thus combination might (always) be a better idea than competition (and selection of the best). As a direct by-product of the system comparison, we already obtain a basic ensemble system, namely one that uses different base learners. The utility of this approach has already been demonstrated to work well for Part-of-speech tagging (Van Halteren et al., 1998; Brill and Wu, 1998), and is a natural fall-out of any system competition (see e.g. Tjong Kim Sang et al., 2000; Kilgarriff and Rosenzweig, 2000). However, the potential of combination is much larger, as there are many ways in which differences between components can be introduced. Preliminary results in building more elaborate ensembles have been obtained and evaluation looks promising.

In the remainder of this paper, we first describe the base machine learning algorithms used in our experiments (Section 2). In Section 3, we then describe the NLP data sets, and next in Section 4.1 the experimental methodology. The algorithms times the data sets define the space of our experiments, the results of which are presented in Section 4. After the benchmark results, we turn to parameter optimization, and present the evolutionary methods we use to optimize large parameter spaces. Then, in Section 6, ensemble methods are introduced, and in Section 7 the design of effective ensembles is rephrased as a large parameter optimization problem. We report the first results with this approach, and finally conclude in Section 8.

2 Base Algorithms

In this section, we give a short description of each of the machine learning methods. These are all supervised classification methods. The basis of this framework is that each algorithm is trained on a set of labeled examples. These examples, which are basically feature-value vectors, are then used to induce decision boundaries in the very high dimensional feature space. As Roth (1998, 2000) shows, under several limiting assumptions, all of the following algorithms can be seen as particular instantiations of a linear classifier in the feature space that consists of all combinations of all features. However, the computational method to arrive at a trained classifier and the representational strategy used by an algorithm can differ greatly. E.g. many learning algorithms are not suited to account for the influence of combinations of features unless this is explicitly represented in their input, some algorithms only use binary features, others multi-valued, some algorithms start from random initialization and others are deterministic, etc. Here we will only give a concise description of each system, in its most common formulation, and describe a few important parameters for each system. Most importantly, we do not manipulate the original feature space to include all feature combinations. This means that Roth's observations about the equivalence of these methods do not hold, and that algorithms which depend on this manipulation of the feature space (e.g. SNoW) are at a somewhat unreasonable disadvantage.

2.1 Memory-Based Learning

Memory-based learning is based on the hypothesis that performance in cognitive tasks is based on reasoning by similarity to *stored representations of earlier experiences*, rather than on the application of rules abstracted from earlier experiences. Historically, memory-based learning algorithms are descendants of the k-nearest neighbor algorithm (Cover and Hart, 1967; Aha et al., 1991).

During learning, training instances are simply stored in memory. To classify a new instance, the similarity between the new instance X and all examples Y in memory is computed using a *distance metric* $\Delta(X,Y)$, a weighted sum of the distance per feature.

$$\Delta(X,Y) = \sum_{i=1}^{n} w_i \; \delta(x_i, y_i) \tag{1}$$

The test instance is assigned the most frequent category within its k least distant (i.e. similar) neighbors. Depending on the system used, a number of different choices are available for the metric. In our experiments, we have used TiMBL, a system described in detail by Daelemans et al. (2000).

2.1.1 Basic MBL metrics

In TiMBL, we can use either Overlap (Equation 2) or MVDM as the basic metric for patterns with symbolic features. Overlap simply counts the number of mismatching features. The k-nearest neighbor algorithm using Overlap and $k = 1$ is called IB1 (Aha et al., 1991)

$$\delta(x_i, y_i) = \begin{cases} 0 & \text{if } x_i = y_i \\ 1 & \text{if } x_i \neq y_i \end{cases} \tag{2}$$

MVDM (Equation 3; Stanfill and Waltz (1986); Cost and Salzberg (1993)) is a method to determine a graded similarity of the values of a feature by looking at co-occurrence of values with target classes. For two values V_1, V_2 of a feature, we compute the difference of the conditional distribution of the classes C_i for these values.

$$\delta(V_1, V_2) = \sum_{i=1}^{n} |P(C_i|V_1) - P(C_i|V_2)| \tag{3}$$

A further parameter of the metric is the weighting method. TiMBL's default weights are computed using Information Gain (IG) (Quinlan, 1993), which looks at each feature in isolation, and measures how much it reduces, on average, our uncertainty about the class label (Equation 4).

$$w_i = H(C) - \sum_{v \in V_i} P(v) \times H(C|v) \quad (4)$$

Where C is the set of class labels, V_i is the set of values for feature i, and $H(C) = -\sum_{i=1}^{n} P(C_i) \log_2 P(C_i)$ is the entropy of the class labels. The probabilities are estimated from relative frequencies in the training set. Information Gain tends to overestimate the relevance of features with large numbers of values. To normalize for features with different numbers of values, Quinlan (Quinlan, 1993) has introduced a normalized version, called **Gain Ratio**, which is Information Gain divided by the entropy of the feature-values ($-\sum_{v \in V_i} P(v) \log_2 P(v)$).

Unfortunately, as White and Liu (1994) have shown, the Gain Ratio measure still has a bias towards features with more values. TiMBL also supports weights based on a chi-squared statistic, which can be corrected explicitly for the number of degrees of freedom.

So, in sum TiMBL has three tunable parameters, the metric, the number of neighbors (k), and the method to compute weights. Unless explicitly optimizing these settings, we have used TiMBL's defaults: Overlap metric, Gain Ratio weighting, and $k = 1$.

2.1.2 Family-based MBL

FAMBL, or Family-based learning, is an extension to MBL where instances in memory are merged into families. The FAMBL package has many options that allow for many types of abstraction. We will only discuss those options that are used for this research. A detailed description of FAMBL can be found in Van den Bosch (1999).

Instead of just placing every training instance in memory, FAMBL tries to merge instances that have the same annotated class and are close together, given a distance measure (see Section 2.1.1). Families are extracted iteratively by randomly selecting an instance from memory and merging this instance with its neighbors of the same class. Merging data points means replacing mismatching symbolic feature values with a wild-card. The distance between a symbolic feature value and a wild-card is always zero.

Since family extraction is done randomly, FAMBL introduces a probing-phase which defines threshold values for the size of a family and the maximum distances between instances in a family. These threshold values are then used in the actual family extraction phase to limit the size of the extracted families. This is referred to as careful abstraction.

So, the classifier induced by FAMBL is a memory of families. A classifier predicts a classification category for a test instance by searching for the closest family in memory ($k = 1$) and assigning the class of the family found.

2.2 Decision Tree Learning

As a representative of the class of decision tree learners, we used the well-known program C4.5 (Quinlan, 1993), which performs *top-down induction of decision trees*, followed by confidence-based pruning. On the basis of an instance base of examples, C4.5 constructs a decision tree which compresses the classification information in the instance base by exploiting differences in relative importance of different features. Instances are stored in the tree as paths of connected nodes ending in leaves which contain classification information. Nodes are connected via arcs denoting feature values. Feature Information Gain ratio (Equation 4) is used dynamically in C4.5 to determine the order in which features are employed as tests at all levels of the tree (Quinlan, 1993).

C4.5 has three parameters, the *pruning confidence level* (the c parameter), the *minimal number of instances represented at any branch of any feature-value test* (the m parameter), and the choice whether to group feature values or not during tree construction (the sub-setting parameter). The first two parameters directly affect the degree of 'forgetting' of individual instances by C4.5, and in previous work (Daelemans et al., 1999), we have shown that for NLP tasks the best results are obtained at the minimal amount of forgetting. However, in the present experiments we use C4.5's default settings, $c = 25\%$; $m = 2$, and no sub-setting.

2.3 Maximum Entropy Modeling

Maximum Entropy Modeling (ME), tackles the classification task by building a probability model that combines information from all the features, without making any assumptions about the underlying probability distribution.

This type of model represents examples of the task (given by multi-valued features: $F_1 ... F_n$) as sets of binary indicator features ($f_1 ... f_m$), for classification tasks the binary features are typically conjunctions of a particular feature value and a particular category. The model has the form of an exponential model:

$$p_\Lambda(C|F_1 ... F_n) = \frac{1}{Z_\Lambda(F_1 ... F_n)} exp(\sum_i \lambda_i f_i(F_1 ... F_n, C)) \quad (5)$$

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 binary feature i.

Learning is the search for a model (i.e. a vector of weights), within the constraints posed by the observed distribution of the features in the training data, that 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 (Berger et al., 1996; Ratnaparkhi, 1997). 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*.

In our experiments, the training was done using a hundred iterations of the Improved Iterative Scaling algorithm. The implementation which we use is called MACCENT, and is available from `http://www.cs.kuleuven.ac.be/~ldh`.

2.4 Rule Induction

Ripper (RIP) (Cohen, 1995) is a well-known effective rule induction algorithm. During training it grows rules by covering heuristics. The training set is split in two parts. On the basis of one part, rules are induced. When the induced rules classify instances in the second part of the training set below some classification accuracy threshold, they are considered to overfit and are not stored. Rules are induced on a class by class basis, starting with the least frequent class, leaving the most frequent class as the default rule, which, in general, produces small rule sets (i.e. one class is taken as *'positive'* and the remainder of the instances as *'negative*).

2.5 Winnow Perceptrons

The Winnow algorithm (WIN) (Littlestone, 1988), is a multiplicative update algorithm for single layer perceptrons, i.e. very simple linear neural networks. A single perceptron takes as input the set of active features in an example[1], and returns a binary decision as to whether it is a positive or negative example. Let w_i be the weight of the i'th feature. The Winnow algorithm then returns a classification of 1 (positive) iff:

$$\sum_{f \in \mathcal{F}} w_f > \theta,$$

where θ is a threshold parameter. In the experiments reported here, θ was set to 1.

A multi-class classifier is constructed out of as many units as there are classes. Each example is treated as a positive example for the classifier of its class, and as a negative example for all the other classifiers.

Training is done incrementally: an instance is presented to the system, the weights are updated, and the example is then discarded. Weights are only added as needed, initially all connections are empty. The updating of the weights is, as said before, done using the multiplicative Winnow

[1] Active features are a set of indexes of the feature values present in an example.

update rule, updating the weights only when a mistake is made. If the classifier predicts 0 for a positive example (i.e., where 1 is the correct classification), then the weights are promoted:

$$\forall f \in \mathcal{F}, w_f \leftarrow \alpha \cdot w_f,$$

where $\alpha > 1$ is a promotion parameter. If the classifier predicts 1 for a negative example (i.e., where 0 is the correct classification), then the weights are demoted:

$$\forall f \in \mathcal{F}, w_f \leftarrow \beta \cdot w_f,$$

where $0 < \beta < 1$ is a demotion parameter.

In this way, weights on non-active features remain unchanged, and the update time of the algorithm depends on the number of *active* features in the current example, and not on the total number of features in the domain.

The implementation we used is called SNoW (Carlson et al., 1999). We used all its default settings.

2.6 Naive Bayes

Another popular algorithm, also implemented in the SNoW package, is Naive Bayes (NB). Naive Bayes follow the Bayes optimal decision rule, that tells us to assign the class C that maximizes $P(C|F_1...F_n)$. By using Bayes' rule we can rewrite this as:

$$C = argmax_{c_i} \frac{P(F_1...F_n|c_i) \times P(c_i)}{P(F_1...F_n)} \tag{6}$$

The Naive Bayes method then simplifies the problem of estimating $P(F_1...F_n|c_i)$ by making the arguable naive assumption that:

$$P(F_1...F_n|c_i) = \prod_{1<j<n} P(F_j|c_i) \tag{7}$$

Each probability on the right-hand side can now be estimated directly from the training data using a maximum-likelihood estimate.

2.7 Multi-layer Perceptrons

A multi-layer perceptron (denoted below by NN), is a type of neural network that is able to make a nonlinear mapping from input to output, because it develops internal intermediate representations in its so called 'hidden layer'. The classifier induced by training a two-layered feed forward back-propagation neural network is a weighted combination of q hyper-planes.

$$h_j(x) = w_j * x + b \qquad (j = 1...q) \tag{8}$$

where q is a user defined parameter, also known as "the number of hidden nodes", w_j, a weight vector, and b define the hyper-plane and x is an instance (feature vector). For each class C_i, the confidence for a test instance z to belong to C_i is obtained by a weighted combination of the distances of z to each $h_j()$.

$$Conf(C_i) = \sum_{j=1}^{q} (w_i * f(h_j(z))) \tag{9}$$

where w_i is again a weight vector for class C_i and $f()$ is an activation function, used to translate the distance value, allowing the combination of the hyper-planes to separate classes that are not linearly separable. For the activation function, which is a user-defined parameter, we have used a sigmoid function.

Usually, the class with the highest confidence is chosen to be the classifiers' prediction. Training a neural network means optimizing the q hyper-planes such that the amount of errors made by

Equation 9 on the training instances, also known as the empirical risk, is minimal. In the case of back-propagation, the approximation is done by a backward propagation of the error for each training instance t. This means that the distances between t and each of the q hyper-planes is gradually adjusted such that the empirical risk becomes smaller. Usually, a validation set (a subset of the training instances, not used during training) or other techniques such as early stopping (Prechelt, 1998) are used to prevent the network from overfitting on the training set. The experiments were performed using the SNNS package (Zell et al., 1995).

2.8 Support Vector Machines

Support Vector Machines (SVM) are an application of the principle of structural risk minimization, introduced by Vapnik (1982). They can be used to induce classifiers that solve binary classification tasks, i.e. that assign one of two classes to an instance. In the case of an SVM, the induced classifier is represented by one hyper-plane $w * x + b$ that separates the classes in the training set, so that:

1. The largest possible fraction of training instances of the same class are on the same side, i.e. the empirical risk is minimal, and
2. The distance of either class from the hyper-plane, called the margin, is maximal.

The classifier's prediction, 1 or -1, for a test instance z is then defined as

$$sgn(w * x + b) \tag{10}$$

When both constrains (a) and (b) are satisfied, the upper bound on the generalization error (or true risk) of the induced classifier will be minimal and the hyper-plane is optimal. Remember that neural network training is only constrained to (a).

In Burges (1998), it is shown that finding an optimal hyper-plane, or training an SVM, is equal to maximizing:

$$W(a) = \sum a_i - \frac{1}{2} \sum a_i a_j y_i y_j (x_i * x_j) \tag{11}$$
$$\text{(constraint to: } 0 \leq a_i \leq C \text{ and } \sum a_i y_i = 0)$$

where a is a variable vector containing the so called Lagrange multipliers, the y_i are the annotated class-labels for each training instance i and C is a user defined parameter that reduces the effect of outliers and noise. Once a is known, all values $a_i > 0$ are the support vectors. From these support vectors, w and b (Eq 10) can be derived and the optimal hyper-plane is induced.

When the classes cannot be separated by a linear combination of elected training instances, the feature space can be mapped to a different (larger) feature space, using a kernel function $K()$. For many classification tasks, the classes can now be linearly separated with an optimal hyper-plane in this augmented space. Equation 12 becomes:

$$W(a) = \sum a_i - \frac{1}{2} \sum a_i a_j y_i y_j K(x_i, x) \tag{12}$$

or in matrix notation:

$$W(a) = -a^T \ I + \frac{1}{2} a^T Q a \tag{13}$$

with $(Q)_{ij} = y_i y_j K(x_i, x_j)$.

Solving this optimization problem requires matrix Q to be stored in memory. Since the size of Q is quadratic in the number of training instances, this becomes impractical for a large data sets (> 5000). Recent SVM literature proposes some good solutions such as chunking, decomposing and sequential minimal optimization methods (Campbell, 2000). As usual, this implies a tradeoff between time- and space-complexity.

A multi-class classifier is constructed by combining single-class classifiers. Each single-class classifier distinguishes one class from all other classes. The max-operator is used to combine the single-class classifiers, i.e. the class corresponding to the single-classifier with the highest prediction (see Equation 10) is chosen as the multi-class classifiers prediction.

The implementation we have used is called `SVM_light` (Joachims, 1999).

3 Data

In this section we describe the data sets used in our benchmarking experiments (Word Sense Disambiguation, Grapheme to Phoneme conversion, Part-of-speech tagging). The selected tasks reach from low-level phonetic processing, through shallow syntactic processing, to higher level semantic judgments. The choice of these data sets was made to include both small and large values on number of dimensions (size, number of features, number of values, number of categories, regularity). Each of the selected tasks is in itself a challenging problem, but here we do not focus on the solution of these problems, but rather take the selected tasks and data sets as given, and just use them for comparison of algorithms. As said in the introduction, we restrict ourselves to single classification tasks, whereas many interesting NLP tasks would be composed of many such decisions in cascades.

The pre-processing we used for the datasets is mostly common practice and is described in detail in the descriptions of the datasets. To maximize comparability with other published results we tried to keep as closely as possible to publicly available datasets, or datasets extracted from well-known generally available datasets.

3.1 Word Sense Disambiguation

Word sense disambiguation (WSD) is the task to select the appropriate sense for a word from a predefined finite set on the basis of its context. Our dataset is based on the 1998 Senseval competition (Kilgarriff and Rosenzweig, 2000), which compared machine learning methods on a small sample of ambiguous words:

> accident (1279), amaze (327), band (1418), behaviour (1009), bet-n (168), bet-v (102), bitter (193), bother (350), brilliant (481), bury (344), calculate (289), consume (111), derive (294), excess (290), float-a (57), float-n (94), float-v (261), generous (339), giant-a (316), giant-n (389), invade (82), knee (530), modest (415), onion (43), promise-n (622), promise-v (1472), sack-n (125), sack-v (195), sanction (117), scrap-n (81), scrap-v (47), seize (340), shake (1099), shirt (564), slight (427), wooden (378)

The total number of examples for a word is between brackets (all words together form a set of 14648 instances). Training and testing is done for each word separately. The features in these data sets represent the following information: The first nine features represent two words to the left, the word to be disambiguated (focus), and two words to the right, each word is followed by its part of speech (Penn Treebank tagset). After this immediate context come a number of binary features indicating the presence (1) or absence (0) of a number of focus-specific keywords in a wider context around the word of interest. These keyword features are different (also in number) for each word, and they were selected using the default method suggested by Ng and Lee (1996).

An example instance for the word "accident" is:

> after, IN, an, DT, accident, NN, at, IN, the, DT, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 532675

The target category (here 532675) is a six-digit code that corresponds to a sense-entry in the HECTOR dictionary[2].

In sum the WSD data has few training examples, many features, some of which have large numbers of values, and others are just binary. The number of categories is relatively small, but

[2] See http://www.itri.brighton.ac.uk/events/senseval/

differs considerably from word to word, as does the ratio between regularity and exceptional cases. For some words, the task is very difficult, because of the interaction between features, and the lack of sorely needed common-sense knowledge.

3.2 Grapheme-Phoneme With Stress

In this data set, the mapping to be learned is from a letter in context to a phonetic representation with stress markers. It will further be referred to as GS. This dataset is based on the CELEX dictionary Baayen et al. (1993) for English. For every word in that dictionary, the letter to be transcribed (the focus), and a context window of three letters to the left and to the right are given as features.

An example of the word "above" converted to windowed training instances is:

```
_,_,_,a,b,o,v,0@.
_,_,a,b,o,v,e,1b.
_,a,b,o,v,e,_,0V.
a,b,o,v,e,_,_,0v.
b,o,v,e,_,_,_,0-.
```

The first character of the target category represents whether the syllable starting with that letter receives stress in the pronunciation (1) or not (0). The second letter is the phoneme corresponding to the 4th feature (the focus letter).

The dataset consists of 77565 words divided into a training set GS-DATA (69808) and a test set GS-TEST (7757). The total number of instances in training and test set is respectively 608228 and 67517. For some experiments we have considered the DATA and TEST parts as separate tasks (large and small version). The number of features is modest compared to the WSD task. Each feature has the same amount of values (number of symbols in the alphabet). In previous research (Van den Bosch, 1997), it has been shown that this task is one where exceptions, and sub-regularities play a large role. Also, obviously, the interaction between the features (and in particular the focus and variable sized portions of the context) is crucial.

3.3 Part-Of-Speech Tagging

Part-of-speech (POS) tagging is the task of assigning the single most appropriate morpho-syntactic category to a word on the basis of its context. If the word has been observed in the training data, we have lexical information available (possible categories, also called "ambiguity classes"); if the word has not been seen before, we must guess on the basis of form and context features. Hence there are two versions of the data, one involved with predicting the POS for known words, and one for unknown words[3].

Our dataset is based on the TOSCA tagged LOB corpus Johansson (1986) of English[4]. The features represent information about the word to be tagged (focus) and its context, and are similar but slightly different for the two sets.

3.3.1 Known Words

The known words set (1045541 cases, henceforth: POS-KNOWN, was made from every 1st through 9th (90the following ten features:

```
W : The focus word itself
d : the POS tag of the word at position n-2
d : the POS tag of the word at position n-1
f : the ambiguity class of the focus word (position n)
```

[3] Note that this is a task that *resembles* POS tagging, but is not actually comparable to the tagging of unseen text. Here each word is processed in isolation, assuming a correctly disambiguated left context. Also the unknown words are not really unknown, they are just infrequent.

[4] Kindly provided to us by Hans van Halteren of the TOSCA Research Group at the University of Nijmegen.

```
a : the ambiguity class of the word at position n+1
a : the ambiguity class of the word at position n+2
s : the 3rd last letter of the focus word
s : the 2nd last letter of the focus word
s : the last letter of the focus word
h : does the word contain a hyphen?
c : does the word start with a capital letter?
```

3.3.2 Unknown Words

The unknown words set (65275 cases, henceforth: POS-UNKNOWN) was also made from every 1st through 9th (90corpus. However, as the distribution of unknown words closely resembles that of the low-frequent words, the only words that are included in this set are words that occurred 5 or less times in the whole dataset. The features for this set are the same as for the known words, except that the focus word itself and its ambiguity class are omitted.

The POS-KNOWN set is very large, whereas the POS-UNKNOWN set is of intermediate size. The number of features is intermediate, and some features (e.g. the focus word) have very large numbers of values, whereas others (e.g. hyphen) are only binary. The number of categories is 201 for KNOWN, and for 118 for UNKNOWN. The KNOWN words data is quite regular (i.e. the most frequent category of a word, regardless of context, already scores more than 90% correct; most capitalized unknown words are proper nouns, etc.), but there seems to be a large number of infrequent exceptions.

4 Comparison Of Algorithms

4.1 Methodology

In this benchmark study, we are especially interested in differences between algorithms with regard to their generalization ability. How well does a particular algorithm process new data, when trained on a particular training set. We estimate this by the *accuracy* (or its inverse, error), operationalized as the percentage of previously unseen data items classified correctly by the algorithm.

The underlying assumption in most work on machine learning is that new data is drawn according to the same distribution as the training data. We operationalize this by randomizing the data set, and then splitting into train and test partitions. To ensure statistical reliability, we do a *10-fold cross-validation* (10CV) (Weiss and Kulikowski, 1991) (divide the data set into ten partitions after randomization, use each partition in turn as test set and the other nine as training set, compute evaluation criteria by averaging over the results for the ten test sets).

For those algorithms that parameterize their settings, we used the default settings in most experiments, except there where an optimal set of parameters settings was explicitly selected. This was done by cross-validating different parameter settings (within reasonable bounds) on the training set. This means that part of the training set was used as a validation test set, and the parameter settings providing the best result on this validation test set were used for the real test set.

In the case of 10CV experiments, the train/test split were identical for all algorithms, allowing direct comparison. For some experiments 10CV was computationally not feasible. In such cases only a single train/test run was done on the first partition of the 10CV.

4.2 Experimental Results

As of October 2000, a large part of the experimental matrix has already been completed. Only for the small task WSD, we have succeeded so far in getting results for all described algorithms (TiMBL, FAMBL, C4.5, RIP, ME, WIN, NB, NN, and SVM). The results for all 36 words of the WSD task are given in Table 6. We see that the difficulty of the task varies considerably across words, and that different algorithms give the best performance for different words. Nonetheless, some clear tendencies can be observed. The bottom of the table summarizes these, by giving the

	Accuracy			
Algorithm	POS-KNOWN	POS-UNKNOWN	GS-DATA	GS-TEST
TiMBL	97.5	82.8	**92.8**	**81.9**
FAMBL	–	80.5	91.3	–
C4.5	–	79.2	–	80.3
ME	**98.1**	**83.7**	79.3	76.7
RIP	96.4	80.1	76.2	73.7
WIN	97.4	74.4	64.5	62.1
NB	96.6	79.2	70.1	68.4

Table 1: Generalization accuracies (10CV) for the POS and GS tasks.

number of words for which an algorithm is the winner, and the average rank of the algorithm. From the average rank we can obtain an overall order between the algorithms in terms of their consistent performance (from best to worst):

$$\text{SVM} > \text{TiMBL} > \text{ME} > \text{NB} > \text{FAMBL} > \text{RIP} > \text{NN} > \text{WIN} > \text{C4.5}.$$

From the number of first places it is very clear that SVM performs very well on this data set, and the other algorithms remain far behind. This clearly shows that SVM's can both maintain a rich representation of the decision boundaries, while at the same avoiding overfitting on the small data sets provided for each word. Several other studies (Mooney, 1996; Escudero et al., 2000) have shown similar results, in particular the good performance of NB on WSD is a recurring reason for wonder, given the simple model this algorithm makes.

For the remainder of the tasks not all algorithms could be applied. The data sets were either too big to fit in memory with a particular implementation, or the algorithm did not scale up well and took too long to terminate[5]. This is disappointing, as one of the victims of this was SVM which produced excellent results for WSD. More sophisticated task decomposition strategies, such as e.g. pairwise coupling (Moreira and Mayoraz, 1998), should however improve this situation in future research.

The results of the experiments for POS and GS are shown in Table 1. The systems tested here are: TiMBL, FAMBL (POS-KNOWN not done), C4.5 (POS-KNOWN terminated), ME, RIP, WIN, NB. For the POS-KNOWN task we get the order (from best to worst):

$$\text{ME} > \text{TiMBL} > \text{WIN} > \text{NB} > \text{RIP}.$$

The systems which allow a better modeling of inter feature dependencies seem to be superior to the systems which consider the features in isolation (NB) or produces small rule sets (RIP). On the POS-UNKNOWN task we get the following order, from which a similar conclusion can be drawn, although the lower echelons are slightly different:

$$\text{ME} > \text{TiMBL} > \text{FAMBL} > \text{RIP} > \text{NB}||\text{C4.5} > \text{WIN}.$$

The resulting ordering for the large version of the GS task is:

$$\text{TiMBL} > \text{FAMBL} > \text{ME} > \text{RIP} > \text{NB} > \text{WIN}.$$

and for the small version (GS-TEST):

$$\text{TiMBL} > \text{C4.5} > \text{ME} > \text{RIP} > \text{NB} > \text{WIN}.$$

Here, again, the algorithms that can model complex feature interactions win over those that cannot, and moreover TiMBL is at an advantage as this tasks is well-known for being ridden with exceptions, and semi-regularities. This is also strikingly demonstrated by the fact that when going from 10% (GS-TEST) to 90% (GS-DATA) of the data, the other algorithms improve only slightly ($< 2.6\%$), whereas TiMBL gains an extra 10.9%.

[5] we have run the experiments on dual Pentium III machines, 512 MB, Redhat Linux. If an algorithm did not produce results within a week it was terminated.

5 Parameter Optimization

So far we have only used default parameter settings for each algorithm. This results in a certain ordering of the algorithms on each task. To make this result worthwile, it should help us in picking an appropriate learning algorithm for a new task. However, we see that the ordering depends on the task at hand. In this section we will first look into additional improvements of the single algorithms, and after that (in Section 6) we will look at system combination as a method to always do as well or better than the best single algorithm. It turns out that the methods to tune and improve a single algorithm, can be reused to get good combinations, i.e. tuned ensembles.

There are at least two ways in which the results for each of the algorithms could be improved. First, we could fine-tune the parameter settings for each system for each task. And second, we could try to adapt the problem representation to make a task better fit the bias of a particular algorithm. This will be left for future research. In this section we consider a case-study with TiMBL as a part of a limited excursion into the first territory (parameter tuning). A full study of all parameter tunings of all algorithms is beyond the scope of this paper. The main point we want to make here, is that a) parameter tuning can make a huge difference to the outcome of any benchmarking study, and b) exhaustive parameter tuning is often impossible.

In memory-based learning, is we want to tune the number of k nearest neighbors, the metric, and the weighting scheme. Given a few settings per parameter, it is not unfeasible to exhaustively (EX) explore this parameter space on a validation set. However, it can also be beneficial to select a subset of features. Moreover, parameter optimization and feature selection or weighting are likely to interact. This situation, typical for an algorithm with a medium to large number of parameters calls for non-exhaustive optimization capable of efficiently avoiding local minima. Therefore, evolutionary algorithms algorithms promise to be of use.

In the experiments, we linked TiMBL to PGAPACK[6]. During the feature subset selection experiments the string is composed of binary values, indicating presence or absence of a feature. During the simultaneous optimization experiments, the first gene in the string encodes the values for k (only odd values are used, to avoid ties), the second gene indicates which weight settings are used and the remaining genes are reserved for the features. In these experiments we look at feature selection as an optimization process, where each feature has three possible values: a feature can either be present, it can be absent or its MVDM can be calculated. Each feature-gene can take on any of these three values and subset selection is then optimization of these values for the specific features. The fitness of the strings is determined by running the memory-based learner with each string on a validation set, and returning the resulting accuracy as a fitness value for that string. Hence, selection with the GA is an instance of a *wrapper* approach as opposed to a *filter* approach such as information gain (John et al., 1994).

For comparison with evolutionary feature selection, we include two popular classical wrapper methods: backward elimination (henceforth BA) and forward selection (henceforth FO).

In Table 2 we show the results of our experiments on POS-KNOWN, POS-UNKNOWN, and GS-TEST. We can see that *a)* exhaustive search for optimal parameter settings improves the classification accuracy and that *b)* selection of a subset of features leads to similar or better results with a reduction in the number of features used. For *c)* simultaneous parameter optimization and feature selection, show improvement for the POS-KNOWN task (significant; McNemar's chi-square; $p<0.001$) , and the GS-TEST task (not significant; $p=0.318$). The exhaustive search for optimal parameters is better than the simultaneously optimized case for POS-UNKNOWN unknown (but not significantly; $p=0.684$). For a more detailed discussion of these results, see Kool et al. (2000).

Although the improvements are by no means dramatic, they do already have consequences for the ranking of algorithms in our benchmark. Compare the best results on POS (resp. 98.1% for KNOWN and 83.7% for UNKNOWN) which were obtained by ME, with the best results obtained here with a parameter optimized version of TiMBL (resp. 98.4% and 85.4%). These are indeed much larger differences than e.g. between TiMBL and ME with their defaults on. A less

[6] A software environment for evolutionary computation developed by D. Levine, Argonne National Laboratory, available from ftp://ftp.mcs.anl.gov/pub/pgapack/

Task	Results		
POS-UNKNOWN		Default Parameters	Optimized Parameters
	All Features	DE 82.6	EX 85.4
	Optimized Features	GA 84.4	GA 84.9
		BA 84.4	BA 85.2
		FO 84.5	FO 85.0
POS-KNOWN		Default Parameters	Optimized Parameters
	All Features	DE 97.5	EX 98.3
	Optimized Features	GA 98.3	GA 98.2
		BA 98.3	BA 98.4
		FO 98.3	FO 98.4
GS-TEST		Default Parameters	Optimized Parameters
	All Features	DE 81.6	EX 81.7
	Optimized Features	GA 81.6	GA 82.0
		BA 81.6	BA 81.5
		FO 81.6	FO 81.6

Table 2: Feature and parameter optimization results.

pronounced, but still interesting, difference is found for GS, where TiMBL is able to improve its result from 81.6% to 82.0%.

So, the optimization of small numbers of parameters is always to be recommended, and can be done by an exhaustive search on the validation set. Simultaneous application of feature selection and parameter optimization has shown some performance gains, but further work on better search algorithms is needed to realize the full potential of the approach. The applicability of this approach goes well beyond TiMBL. Other machine learning algorithms are confronted with similar feature weighting, feature selection, and parameter optimization problems, and these results are likely to be relevant for these other algorithms as well. For example, an small optimization run of SVM parameters on the WSD task (C and the dimension of the kernel function) resulted in an average improvement of 3.7 percentage points per word over the already very good results in Table 6.

6 System Combination

As argued throughout this paper, disambiguation tasks in NLP can be characterized as complex mappings from large amounts of features to large amounts of categories. From the benchmarks we can see that learning these tasks from corpora tends to push existing learning algorithms to their limits. A possible solution for this problem is to modularize a task as a series of more simple problems. However, for most tasks a good decomposition is difficult to design.

An alternative, and fully automated approach towards modularization is offered by recent work in Machine Learning. Starting from the observation that different learning systems make different errors when trained to perform the same task, and among all the system's outputs the right output is more likely to be present somewhere than in any single system, so called "combination methods" attempt to train an ensemble of diverse classifiers and combine these to yield a composite classifier with higher accuracy. There are four dimensions on which diversification can be attempted (Dietterich, 2000), and we can in fact consider these as possible paths towards modularization:

1. Data modularization. E.g. in AdaBoost (Freund and Schapire, 1997), each consecutive component system receives a training set in which the items classified wrong by the previous components are given a higher weight.

2. Target category modularization. Error Correcting Output Codes (Dietterich and Bakiri, 1991) train an ensemble in which each component learns one binary split between categories. A similar approach is followed by Pairwise Coupled Classifiers (Moreira and Mayoraz, 1998).

3. Feature modularization: E.g in Bay (1998), a performance gain is obtained by combining several nearest neighbor classifiers, each trained with a random subset of the available features.

4. Bias modularization: Different learning algorithms can be used as components (e.g. Van Halteren et al., 1998), or the same algorithm with different parameter settings.

Interestingly, each dimension of variation can results in accuracy gains, even though the only criterion used to make ensemble members is to ensure that they have some diversity (see e.g. evidence in Dietterich, 2000). Oftentimes, it is also stated that the components must be "accurate enough". As we will see, however, this criterion depends on the combination method used.

6.1 Combination Methods

Once we have trained a set of diverse components, there is a number of ways to combine their outputs. The most straightforward way to do this is **voting**. Voting can be very simple, i.e. each component cast a single vote for its own output, or more sophisticated, by casting weighted votes, and perhaps even countervotes. Although it is certainly by far the most popular combination method, certain properties of voting make it a bad choice for constructing ensembles. First, voting can only result in an ensemble output that is present between the component outputs[7]. Second, following from this, for voting to work, all the components should use the same class labels. This can be a problem when we want to integrate diverse sources of knowledge in the ensemble. Third, and not least, bad components will drag the whole ensemble down.

A much more powerful and effective way to do combination, called **stacking** was proposed by Wolpert (1992). Stacking involves two levels of learning. On top of the components, we place a second level, or meta learner, which is trained to map the vector of component outputs to the correct ensemble output. This gives as much greater freedom: we can use a completely different code at the intermediate level than at the output level, we can use components with diverse codes, and we can even use very misguided components, as long as their outputs are is some way systematic. In fact we can even emulate voting, because (weighted) voting is a special case of stacking, where each output class has one codebook vector [8] The downside of this freedom, is that the second level must be trained, and for training we need enough data. If we train the second level using training data that was also used to train the components, these will be too correct to reliably estimate error patterns from, and the second level will fail to learn any error-correction. Hence we must use a separate tuning set, or produce a cross-validated output of the components on the training set (which we do in the experiments below). Another point that complicates stacking somewhat is that the choice of the second level learner is as much an open issue as the choice of components. (In our experiments we have found that unweighted TiMBL-IB1 works well, as does TiMBL-MVDM with $k = 9$).

Stacking is a very powerful framework, because, given the freedom of different intermediate representations, we can also use diverse recodings of the original features as 'ensemble components'. A special case of this is what we will call **arbiter** learning, where in a stacked ensemble, the meta-learner is also given all the original input features. This allows the meta-learner to error-correct the patterns produced by the component outputs based on their place in the input space. Another way to see this, is that the components are producing compressed representations of their inputs. But when their compression rate is too high, because of a coarse-grained class scheme, the second level looses to much information about the context of the decision. The arbiter method allows us to partially remedy this.

6.2 Basic Ensembles

In this section we report experiments with two of the easiest dimensions of variation. The first (bias) naturally falls out of the benchmark experiments. Since we have done a 10CV of each algorithm on the same data, we can easily construct an ensemble of these.

[7] Unless a special voting code book is used, as in ECOC's.

[8] This insight is due to Dietterich (2000).

	POS-UNKNOWN	POS-KNOWN	GS-DATA	GS-TEST
Single components				
TiMBL	82.6	97.5	92.8	81.6
ME	83.2	98.1	79.2	77.0
RIP	80.1	96.4	–	–
WIN	74.1	97.5	63.7	62.5
Ensembles				
majority	84.7	98.3	83.2	78.8
stacked (IB1)	85.1	98.4	92.6	81.5
arbiter (IB1)	86.4	98.4	93.4	83.4
arbiter (MVDM-k9)	86.4	98.6	93.1	83.6
oracle	93.3	99.4	95.9	90.5

Table 3: System combination ensemble results for POS and GS tasks. These experiments have been performed on one train/test partition only.

Table 3 shows the results for the POS and GS tasks of combining diferent base learners (systemcombi). For POS we used TiMBL, ME, RIP and WIN as the components, for GS, the outputs of RIP were not available, so we only used the three remaining components[9]. These ensembles were only tested on one partition of the 10CV split, so we provide the corresponding single system results as a comparison. We see that for POS majority voting already shows a performance increase. For GS this is not the case–we see a big drop for voting here, because the first and second runner up are much worse than TiMBL. As we can see in the table, this is not a problem for stacking. For POS stacking produces a considerable accuracy increase, and for GS it at least approximates the best single component. The arbiter method produces an improvement even larger for POS (up from 83.2% for ME to 86.4% for UNKNOWN and up from 98.1% to 98.6% for KNOWN) and considerable for GS as well (from 92.8% for a single TiMBL to 93.4% on GS-DATA, and from 81.6% to 83.6% for GS-TEST). The bottom row of the table shows the accuracy of an oracle system, which would always suggest the correct category if one of the components would propose it. As we can see, there is still room for improvement[10]. In future work we should also look at the issue how the number of components influences the performance of the ensemble.

A second ensemble that is relatively easy to construct for our data sets, is an ensemble based on feature variation. Each component uses a default TiMBL learner, on the basis of a different feature subset (featurecombi). We constructed nine components for each ensemble by hand, trying to ensure variation in the set. This is similar to the experiments of Bay (1998), who uses random feature subsets for each component. The various feature subsets, and the results, are shown in Table 4. As we can see, most of the components are much simpler than the full feature representation, and hence do not perform very well by themselves. Neither does the majority voting ensemble perform well. As an illustration of the power of stacking, however, this experiment is sufficient. Both stacked and arbiter version produce better results than the systemcombi ensemble for POS. For GS, the stacked version is better than systemcombi, but, disappointingly, the arbiter version is not. Finally, if we put the components of systemcombi and featurecombi together, the results further improve upon both simple stacked ensembles for POS.

7 Evolution Of Modular Ensemble Systems

In Section 6 four dimensions of variation have been identified that can be used to cause variation among classifiers, and hence possibly improve the performance of an ensemble. Two of these dimensions (bias and feature set) were shown to be effective on two of our data sets. Ensemble methods are a very active area of research and these and other variations are shown to work well time after time (see Dietterich (2000) and the references therein). What is striking however, is that so far:

[9] These are again the systems with default settings.

[10] Note however, that in theory the oracle is only an upper bound for voting, not for stacking.

	POS-UNKNOWN		GS-TEST	
Single components				
	ddaasssch		pppfsss	
	111111111	82.6	1111111	81.6
	110000000	44.8	0001000	47.0
	001100000	41.6	0011000	61.1
	111100000	53.4	0001100	63.0
	000011100	62.9	1110000	28.6
	000000011	38.7	0000111	32.1
	000011111	72.4	0011100	76.3
	110011100	72.7	0111110	81.7
	001111100	68.5	0010100	35.0
Ensembles				
majority		81.6		80.6
oracle		95.5		95.0
stacked (IB1)		84.8		82.5
stacked (MVDM-k9)		85.3		82.6
arbiter (MVDM-k9)		86.9		82.7
stacked+systems (MVDM-k9)		86.6		82.5

Table 4: Feature combination ensemble results for POS-UNKNOWN and GS-TEST. These experiments have been performed on one train/test partition only. The **stacked+systems** entry (bottom row) includes all systems from Table 3 and all feature subsets components.

- Few attempts have been made to optimize the divergence in an ensemble directly in order to get better performance[11]. In most research on combined methods, an ensemble is constructed by making a diverse set in some ad-hoc fashion–as we have done–, and then combining these components (usually by using voting).
- Moreover, the four dimensions of variation have mainly been studied in isolation. This in spite of the fact that a simultaneous variation in bias *and* output coding *and* data set *and* feature set might have much more far-reaching effects.

In this section we sketch the outline of a method that can exploit all of the above mentioned dimensions of modularization, while at the same time explicitly optimizing the composition of the ensemble using Genetic Algorithms. It is a derivative of the method used for feature and parameter optimization in Section 5. We simply encode the whole ensemble as a large vector of parameters.

A first population of ensembles is generated with random settings for each component, all components are trained on the same task, a 'stacked' second level algorithm learns to combine the outputs of the ensemble, and the combiner's test score is used as a fitness value. Subsequent generations are formed by cross-over and mutation from the fittest ensembles. Using the score of the whole ensemble as a fitness measure ensures the selection of good 'team-players' as constituents, even though we have no good understanding how quality (accuracy) and specialization (de-correlation of errors) contribute to the global solution. This should typically lead to automatic modularization of the task. As shown in Table 5, in our first experiments, this method has a higher accuracy than both the best individual system, and the 'hand-designed' systemcombi and featurecombi systems for the GS task. For POS results are better than systemcombi but the same (stacked version) or slightly worse (arbiter version) than featurecombi.

However, these experiments are only starting to scratch the surface of what is possible with these methods. We hope that additional performance gains can be realized when more dimensions of variation are included in the optimized ensembles. But, it remains to be seen whether GA's are an appropriate optimization method for this application.

[11] We are only aware of work in this direction in the Neural Networks field (e.g. Moriarty and Miikkulainen, 1997; Yao, 1999), but have not yet encountered such work in symbolic Machine Learning.

	POS-UNKNOWN		GS-TEST	
Single components				
	`ddaasssch`		`pppfsss`	
	011011011	71.7	0110000	26.0
	110100110	65.5	1001110	70.0
	001011111	76.8	1101000	57.3
	111111001	70.8	0010111	44.9
	010111111	80.4	0110010	33.0
	010111100	72.4	0001111	70.2
	100111111	74.7	0111111	81.5
	011000010	52.3	1111100	77.9
	001100010	41.7	1000001	18.8
Ensembles				
handmade-featurecombi stacked		85.3		82.6
handmade-featurecombi arbiter		86.9		82.7
GA-stacked		85.3		83.3
GA-arbiter		86.5		84.4
GA-**feat+param**-stacked		85.8		83.4
GA-**feat+param**-arbiter		86.4		84.0

Table 5: GA optimized feature modular ensembles for POS-UNKNOWN and GS-TEST. These experiments have been performed on one train/test partition only. Each component uses a default TiMBL learner, on the basis of a different feature subset. For the **feat+param** entries (bottom two rows) the GA optimized both the feature subset as well as the TiMBL parameters. The stacked and arbiter systems all use TiMBL with MVDM-k9 as the second level learner.

8 Conclusions And Future Work

- In line with previously obtained results (Daelemans et al., 1999) TiMBL performs well across our benchmark of NLP tasks. However, on particular tasks, there are strong competitors: On WSD, Support Vector Machines show an outstanding performance. This algorithm, however, still has trouble scaling up to large NLP tasks. On POS, Maximum Entropy modeling is slightly better than TiMBL, but this is mitigated after tuning TiMBL's parameters.

- We have shown how important feature and parameter tuning is and how it can be done using GA's or more traditional search methods.

- We have provided arguments and empirical evidence that stacking is superior to voting in ensemble systems. In particular, using stacking, it seems a good idea (if there is enough training data) to always use ensembles rather than the best single algorithm.

- Using stacked and arbiter combination we have been able to build effective ensembles (beating every single component system) from natural collections of components, such as divergent feature sets, and different learners participating in a benchmark. In future work, the remaining two dimensions of variation (i.e. data set modularization, and output coding modularization will be investigated as well.

- We have proposed a new framework for constructing optimized modular ensemble classifiers using GA's. The first evaluations are very promising. In future work, we will continue in this direction. In particular, we plan to investigate the development of intermediate representations that work well for stacking.

References

Abney, S. (1996). Statistical methods and linguistics. In Klavans, J. L. and Resnik, P., editors, *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*, pages 1–26. MIT Press, Cambridge, MA.

Aha, D. W., Kibler, D., and Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, 6:37–66.

Baayen, R. H., Piepenbrock, R., and van Rijn, H. (1993). *The CELEX lexical data base on CD-ROM*. Linguistic Data Consortium, Philadelphia, PA.

Bay, S. (1998). Combining nearest neighbor classifiers through multiple feature subsets. In *Proc. 17th Intl. Conf. on Machine Learning*, pages 37–45.

Berger, A., Della Pietra, S., and Della Pietra, V. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1), March 1996.

Blake, C., Keogh, E., and Merz, C. (1998). UCI repository of machine learning databases.

Brill, E. and Wu, J. (1998). Classifier combination for improved lexical disambiguation. In *Proceedings of the Seventeenth International Conference on Computational Linguistics (COLING-ACL'98), Montreal, Canada*, pages 191–195.

Burges, C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):955–974.

Campbell, C. (2000). Algorithmic approaches to training support vector machines: A survey. In *Proceedings of ESANN2000*, pages 27–36. D-Facto Publications, Belgium.

Carlson, A., Cumby, C., Rosen, J., and Roth, D. (1999). SNoW User's Guide. Technical Report UIUC-DCS-R-99-210, University of Illinois at Urbana-Champaign.

Cohen, W. (1995). Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, Lake Tahoe, California.

Cost, S. and Salzberg, S. (1993). A weighted nearest neighbour algorithm for learning with symbolic features. *Machine Learning*, 10:57–78.

Cover, T. M. and Hart, P. E. (1967). Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 13:21–27.

Daelemans, W. (1996). Experience-driven language acquisition and processing. In Van der Avoird, M. and Corsius, C., editors, *Proceedings of the CLS Opening Academic Year 1996-1997*, pages 83–95. CLS, Tilburg.

Daelemans, W., Van den Bosch, A., and Zavrel, J. (1999). Forgetting exceptions is harmful in language learning. *Machine Learning, Special issue on Natural Language Learning*, 34:11–41.

Daelemans, W., Zavrel, J., van der Sloot, K., and van den Bosch, A. (2000). TiMBL: Tilburg memory based learner, version 3.0, reference manual, technical report ILK-0001. Technical report, ILK, Tilburg University.

Dietterich, T. and Bakiri, G. (1991). Error-correcting output codes: A general method for improving multiclass inductive learning programs. In *Proc. of the 9th National Conference on Artificial Intelligence (AAAI-91)*, pages 572–577.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In Kittler, J. and Roli, F., editors, *Proc. of the First International Workshop on Multiple Classifier Systems (MCS 2000)*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer Verlag, Berlin.

Escudero, G., Márquez, L., and Rigau, G. (2000). A comparison between supervised learning algorithms for word sense disambiguation. In *Proc. of CoNLL-2000*. ACL.

Freund, Y. and Schapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Joachims, T. (1999). Making large-scale SVM learning practical. In Schölkopf, B., Burges, C., and Smola, A., editors, *Advances in Kernel Methods - Support Vector Learning*. MIT-Press.

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

John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 121–129, San Mateo, CA. Morgan Kaufmann.

Kilgarriff, A. and Rosenzweig, J. (2000). Framework and results for english senseval. *Computers and the Humanities, special issue on Senseval*, 34(1–2).

Kool, A., Zavrel, J., and Daelemans, W. (2000). Simultaneous feature selection and parameter optimization for memory-based natural language processing. In *submitted*.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear threshold

algorithm. *Machine Learning*, 2:285–318.

Michie, D., Spiegelhalter, D. J., and Taylor, C. C. (1994). *Machine learning, neural and statistical classification*. Ellis Horwood, New York.

Mooney, R. J. (1996). Comparative experiments on disambiguating word senses: An illustration of the role of bias in machine learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 82–91.

Moreira, M. and Mayoraz, E. (1998). Improved pairwise coupling classification with correcting classifiers. In *Proceedings of the 10th European Conference on Machine Learning*, pages 160–171.

Moriarty, D. E. and Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5:373–399.

Ng, H. T. and Lee, H. B. (1996). Integrating multiple knowledge sources to disambiguate word sense: An exemplar-based approach. In *Proc. of 34th meeting of the Assiociation for Computational Linguistics*.

Prechelt, L. (1998). Early stopping – but when? In *Neural Networks: Tricks of the trade*, Lecture Notes in Computer Science 1524, Springer Verlag, Heidelberg, pages 55–69.

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

Ratnaparkhi, A. (1997). A linear observed time statistical parser based on maximum entropy models. Technical Report cmp-lg/9706014, Computation and Language, http://xxx.lanl.gov/list/cmp-lg/.

Roth, D. (1998). Learning to resolve natural language ambiguities: A unified approach. In *Proc. of AAAI*.

Roth, D. (2000). Learning in natural language: Theory and algorithmic approaches. In *Proc. of CoNLL'00*. ACL.

Stanfill, C. and Waltz, D. (1986). Toward memory-based reasoning. *Communications of the* ACM, 29(12):1213–1228.

Tjong Kim Sang, E., Daelemans, W., Déjean, H., Koeling, R., Krumolowski, Y., Punyakanok, V., and Roth, D. (2000). Applying system combination to base noun phrase identification. In *Proceedings of COLING 2000*, Saarbrücken, Germany.

Van den Bosch, A. (1997). *Learning to pronounce written words: A study in inductive language learning*. PhD thesis, Universiteit Maastricht.

Van den Bosch, A. (1999). Instance-family abstraction in memory-based learning. In *Proc. of the 16th International Conference on Machine Learning (ICML'99), Bled, Slovenia*, pages 39–48.

Van Halteren, H., Zavrel, J., and Daelemans, W. (1998). Improving data driven wordclass tagging by system combination. In *Proceedings of the Seventeenth International Conference on Computational Linguistics (COLING-ACL'98), Montreal, Canada*, pages 491–497.

Vapnik, V. (1982). *Estimation of Dependencies Based on Empirical Data*. Nauca, Moskow, 1979, Enlish translation: Springer Verlag, New York.

Weiss, S. and Kulikowski, C. (1991). *Computer systems that learn*. San Mateo, CA: Morgan Kaufmann.

White, A. and Liu, W. (1994). Bias in information-based measures in decision tree induction. *Machine Learning*, 15(3):321–329.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.

Zell, A., Mamier, G., Vogt, M., et al. (1995). *SNNS, Stuttgart Neural Network Simulator, User Manual*. University of Stuttgart, version 4.1 edition. Technical report 6/95.

Word	Accuracy								
	TiMBL	FAMBL	C4.5	ME	RIP	WIN	NB	NN	SVM
accident	87.1	85.2	69.0	87.8	88.8	88.3	–	87.7	**90.8**
amaze	**99.7**	**99.7**	99.1	93.1	98.8	99.1	97.2	99.4	99.1
band	87.0	86.7	81.7	82.8	85.8	81.4	85.3	83.5	**89.9**
behaviour	95.5	96.3	95.4	96.1	**96.7**	95.3	93.3	95.7	94.5
bet-n	**76.9**	66.3	70.0	71.3	62.5	70.0	70.0	59.4	75.0
bet-v	79.0	78.0	69.0	84.0	77.0	72.0	71.0	84.0	**86.0**
bitter	56.3	55.8	45.8	57.9	52.1	37.4	59.5	50.5	**67.4**
bother	83.4	77.4	72.3	77.7	77.1	77.4	79.1	77.4	**84.3**
brilliant	52.1	51.7	44.8	55.4	52.9	17.3	55.6	47.3	**60.4**
bury	48.2	50.9	34.1	50.9	48.2	37.9	50.3	40.3	**51.8**
calculate	80.0	74.6	75.4	76.8	**81.8**	74.3	81.4	80.0	79.3
consume	62.7	65.5	51.8	67.3	70.9	63.6	64.6	62.7	**71.8**
derive	65.5	59.0	55.5	**70.3**	65.2	65.9	61.0	63.8	67.2
excess	83.8	84.8	83.5	85.5	82.4	**85.9**	84.5	81.4	85.5
float-a	60.0	52.0	74.0	62.0	66.0	70.0	**80.0**	54.0	72.0
float-n	66.7	64.4	47.8	60.0	50.0	46.7	66.7	54.4	**72.2**
float-v	50.0	47.7	36.2	51.2	41.2	44.6	50.0	37.3	**52.7**
generous	43.6	43.0	37.6	51.5	40.6	44.5	**52.1**	40.3	50.9
giant-a	90.3	**92.6**	**92.6**	**92.6**	92.3	**92.6**	90.0	91.9	91.3
giant-n	80.0	77.4	75.8	77.9	80.8	76.3	78.7	76.3	**82.9**
invade	52.5	50.0	37.5	60.0	48.8	41.3	58.8	48.8	**62.5**
knee	**78.3**	71.1	68.5	74.9	71.7	69.2	71.9	65.8	77.5
modest	58.3	59.8	55.6	**65.6**	64.4	38.5	61.2	57.6	63.4
onion	95.0	**97.5**	80.0	80.0	80.0	82.5	82.5	92.5	95.0
promise-n	67.3	67.4	61.3	**73.6**	72.3	67.4	69.2	72.6	73.2
promise-v	89.2	87.3	85.9	87.6	87.3	85.5	88.6	89.4	**92.4**
sack-n	**83.3**	72.5	65.8	71.7	75.8	58.3	77.5	73.3	**83.3**
sack-v	**99.5**	98.4	96.3	96.3	96.3	96.8	97.4	96.3	97.9
sanction	80.0	72.3	71.8	74.6	76.4	61.8	80.9	72.7	**82.7**
scrap-n	82.5	77.5	62.5	77.5	68.8	77.5	78.8	47.5	**87.5**
scrap-v	87.5	92.5	92.5	85.0	**95.0**	**95.0**	92.5	85.0	87.5
seize	60.0	61.5	59.7	62.9	55.0	55.3	58.5	58.5	**63.2**
shake	69.6	68.7	62.6	69.1	67.3	62.2	68.2	67.4	**76.2**
shirt	86.4	85.7	87.2	80.2	87.9	84.6	80.2	83.9	**89.5**
slight	91.9	91.2	90.2	89.5	**92.2**	90.2	90.0	87.8	90.9
wooden	97.6	97.3	97.6	95.4	**97.8**	95.7	97.0	93.5	97.0
best one	5	3	1	4	5	3	2	0	19
avg. rank	3.9	4.8	7.1	4.3	5.0	6.6	4.6	6.4	2.3

Table 6: Generalization accuracies (10CV) for the WSD task. The bottom two rows summarize the table by listing how many times an algorithm was the best one, resp. what its average ranking per word is.