In Hans van Halteren (Ed.) Syntactic Wordclass Tagging.
Kluwer Academic Publishers, 285-304, 1999.

17 MACHINE LEARNING APPROACHES

Walter Daelemans

17.1 INTRODUCTION

The usefulness and feasibility of *automatically training* a syntactic wordclass tagger instead of hand-crafting it motivated a large body of work on statistical and rule-learning approaches to the problem. Syntactic wordclass taggers trained on corpora are claimed to be equally accurate as, and more robust and more portable than, hand-crafted systems[1]. Moreover, development time is considerably faster. Recently, inductive machine learning approaches such as connectionist learning algorithms, decision tree induction and case-based learning have also been applied to the syntactic wordclass disambiguation problem. In some cases these approaches have interesting properties not present in existing statistical and rule-based approaches.

A succesful inductive machine learning algorithm works by extracting generalizations from a set of examples of a desired input-output mapping. The relations between input and output, implicit in these examples, are discovered by the algorithm and are used to predict the correct output when presented with a new, previously unseen, input

[1] Although we will report published accuracy figures for the taggers discussed, we should be cautious about them. Evaluation of the different methods has not been achieved the same way in each case.

Table 17.1 Tagging as a mapping from sentences to tag strings.

Input	Output
John will join the board	np md vb dt nn

Table 17.2 Tagging as a mapping from focus words with context to tags.

Input					Output
Left context		Focus	Right context		
=	=	John	will	join	np
=	John	will	join	the	md
John	will	join	the	board	vb
will	join	the	board	=	dt
join	the	board	=	=	nn

pattern. In other words, the algorithm *classifies* a new input pattern as belonging to a particular output category.

Many problems in Natural Language Processing (NLP), especially disambiguation problems, can be formulated as classification tasks (Magerman 1994; Daelemans 1995; Cardie 1996). Tagging, e.g., can be seen as a mapping from sentences to strings of tags. In syntactic wordclass tagging, abbreviated tagging from here, a sentence should be mapped into a string of morphosyntactic tags (table 17.1).

By approximating this mapping with a function from a focus word and its context to the disambiguated tag belonging to the focus word in that context (table 17.2), the mapping becomes a *classification* task amenable to symbolic and connectionist Machine Learning (ML) approaches. Context size can vary from one word at each side of the focus word (comparable to trigram models in statistics) to the complete sentence. Some machine learning methods dynamically determine the context size needed to disambiguate a particular focus word. The information provided to the learning algorithm can consist also of 'morphological' features (suffixes, presence or absence of a hyphen, a capital or a digit), syntactic information (features representing the syntactic context in which a word has to be tagged), or any other available linguistic information. In general, there will be as many examples for the learning algorithm as there are words in the training corpus.

In this chapter, we will provide an overview of basic concepts in inductive learning methods and discuss recent research on the application of these methods to tagging.

We will discuss case-based classifiers, decision tree induction methods and neural networks.

Another promising approach is Inductive Logic Programming (ILP; e.g. Muggleton and De Raedt 1994), in which background knowledge and positive and negative examples are used to induce a logic program compatible with the background knowledge and all of the positive examples, but none of the negative examples. The induced logic programs are more expressive than the propositional language of feature vectors, which makes the approach potentially interesting to induce recursive tagging rules. See Cussens (1997) for an early example of this approach.

17.2 INDUCTIVE LEARNING FROM EXAMPLES

17.2.1 Concepts

Machine Learning (ML) is a sub-discipline of *Artificial Intelligence* (AI) which studies algorithms that can learn either from experience or by reorganizing the knowledge they already have. See Langley (1996) and Carbonell (ed.) (1990) for introductory material, Weiss and Kulikowski (1991) for methodological issues and Natarajan (1991) for a formal-theoretical approach.

Conceptually, a learning system consists of a *performance component* which achieves a specific task (given an input, it produces an output) and a *learning component* which modifies the performance component on the basis of its experience in such a way that performance of the system in doing the same or similar tasks improves (figure 17.1). Experience is interpreted rather narrowly here and is represented as a set of examples used to train the system. Examples usually take the form of pairs of inputs with their associated desired output. In tagging, the input is (a description of) a focus word and its context and the desired output is the disambiguated tag to be assigned to the focus word.

To achieve its task, the performance component uses an internal representation. The task of the learning component may therefore be construed as a search in the space of possible representations for a representation that is optimal for performing the mapping. In this chapter, we will consider decision trees, case bases and sets of connection weights as types of languages/formalisms for internal representations for trainable taggers. In most cases, finding *the* optimal representation given a set of examples and a representation language is computationally intractable. Some form of heuristic search is therefore used by all learning systems.

In Machine Learning, the concept of *bias* refers to domain-dependent constraints on the search process: knowledge about the task may be used to make the search simpler. There may also be bias in the way the experience presented to the learning component (the training examples) is preprocessed. The addition of linguistic bias to a learning system is the obvious way to let learning taggers profit from linguistic knowledge about the task.

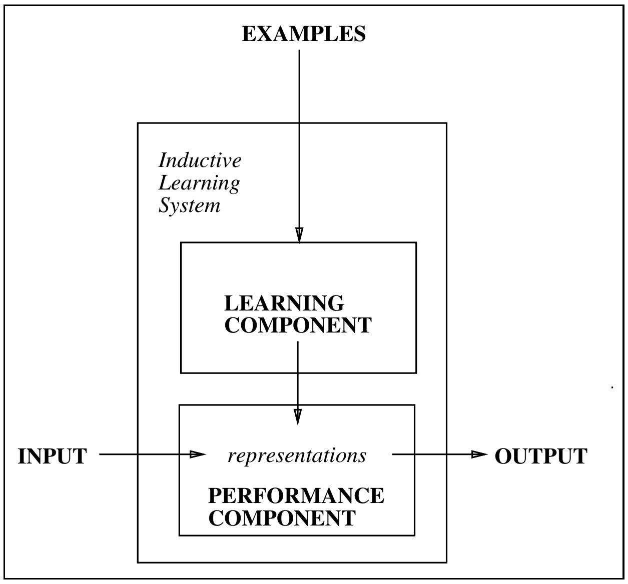

Figure 17.1 General architecture of an inductive learning system.

17.2.2 Classification of learning methods

Given this very general model of inductive learning, a number of dimensions can be distinguished that should be considered in comparing and experimenting with these techniques.

- Amount of *Supervision*. In supervised learning, experience takes the form of *examples* of inputs and the corresponding desired output for each of these inputs. The examples are presented to the system in a training phase. In *unsupervised learning*, examples are presented without information about the desired output. It is up to the system to find similarities in the examples in such a way that they can be exploited by the performance component to solve the task. In *reinforcement learning*, no examples are given; only an indication of the correctness of the output the performance component produces given an input (*feedback*). Most of the research described in this chapter concerns supervised approaches (learning from examples).

- *Input Representation*. Representations used in the ML literature include vectors of bits, vectors of feature-value pairs (numeric or nominal values; compare 'flat' feature structures in linguistics) and complex recursive representations such as

Table 17.3 Comparison of inductive learning methods discussed in this chapter.

	Decision Tree	Case-based	Neural Network
Supervised	YES	YES	YES
Input Rep.	feature value vector	feature value vector	bit string
Output Rep.	symbolic category	symbolic category	bit string
Internal Rep.	tree	cases	connection weights
Incremental	NO/yes[2]	YES	NO
Noise tolerant	YES	YES	YES

semantic nets (compare recursive feature structures in linguistics). For tagging problems, vectors of bits and of nominal feature-value pairs have been proposed.

- *Output Representation*. Output can be a binary category (yes/no) decision, a symbolic category (a finite, discrete set of labels), a continuous category (a real number) or a vector of any of these. In machine learning for tagging, vectors of binary categories and symbolic categories are used (see also Chapter 4).

- *Internal Representation*. The representation used by the performance component, and optimized by the learning component, can be numeric (e.g. connection weights with neural networks) or symbolic (semantic nets, rules, decision trees, examples, . . .).

- *Incremental Learning*. A learning system can be incremental. In that case, relevant information in additional examples can be integrated by the learning component into the performance component without re-learning everything from scratch. In non-incremental or *batch learning* systems (such as neural networks), this is not possible. In batch learning, the complete set of examples has to be inspected (sometimes several times) before learning is completed and addition of new examples makes complete relearning necessary.

- *Noise Tolerance*. Different algorithms can be more or less sensitive to noise in the input (wrongly coded examples, missing values, or even ambiguous examples, i.e. examples which have been assigned contradictory outputs in the training set). Algorithms dealing with linguistic data should be noise-resistant.

Table 17.3 gives a characterization of the different inductive learning algorithms discussed in this chapter along these dimensions.

[2] In most versions, the decision tree learning algorithm is batch learning, but incremental versions have been developed (e.g. Utgoff 1989).

17.2.3 Performance evaluations

The success of a learning component in improving performance can be evaluated using a number of different quantitative and qualitative measures:

- *Generalization accuracy*. Performance accuracy of the system on previously unseen inputs (i.e. inputs it was not trained on). This aspect of learning is of course crucial: it gives an indication of the quality of the *inductive leap* made by the algorithm on the basis of the examples. A good generalization accuracy indicates that the learning system has not *overfit* its training examples, as would happen by generalizing on the basis of errors or exceptions present in them. To get a good estimate of the real generalization accuracy, *cross-validation* can be used, e.g. in 10-fold cross-validation an algorithm is tested on ten different partitions (90% training material, 10% testing material) of the full data set available. Each data item occurs once in one of the test sets. The average generalization accuracy on the ten test sets is then a good statistical estimate of the real accuracy (see also Chapter 6).
- *Space and time complexity*. The amount of storage and processing involved in learning (training the system) and performance (producing output given the input).
- *Explanatory Quality*. Usefulness of the representations found by the learning system as an explanation of the way the task is achieved.

17.2.4 Overview of methods

To sum up this introductory section, we will give an intuitive description of how each of the studied algorithms works, using tagging as an example application. We discuss the algorithms in an order of increasing abstraction of the internal representation used by the performance component and created by the learning component. We start from storage and *table-lookup* of the 'raw' examples as a non-learning baseline.

- *Table Look-Up*. Store all examples (patterns of target words with their context and their corresponding disambiguated tag) in a table. When a new input pattern is given to the performance system, look it up in the table and retrieve the output of the stored example. In this approach the system does not actually learn anything and it fails miserably whenever an input pattern is not present in the table (there is no generalization).
- *Case-Based Learning*. Store all examples in a table. When a new input pattern is given to the performance system, look up the most *similar examples* (in terms of number of feature values common to the stored pattern and the new pattern, for example) and extrapolate from the tags assigned to these nearest matches

to the new case. Various statistical and information-theoretic techniques can be used to design the *similarity metric*. The similarity metric is also a place where linguistic bias can be introduced in the learning algorithm.

- *Rule and Decision Tree Induction*. Use similarities and differences between examples to construct a decision tree or a rule set (these two are largely equivalent and can be translated to each other) and use this constructed representation to assign a tag to a new input pattern. Forget the individual examples.
- *Connectionism, Neural Networks*. Use the examples to train a network. In back-propagation learning, this training is done by repeatedly iterating over all examples, comparing for each example the output predicted by the network (random at first) to the desired output and changing connection weights between network nodes in such a way that performance increases. Keep the connection weight matrix and forget the examples.

The place of stochastic (statistical) approaches deserves some discussion at this point. In this popular approach to tagging, statistical models (e.g. about the N-grams occurring in a language) are computed on the examples (the corpus) and these are used to extrapolate to the most probable analysis of new input. In terms of abstraction versus data-orientation, stochastic, neural network and rule induction approaches are *greedy learning* techniques. These techniques abstract knowledge from the examples as soon as they are presented. Case-Based Learning is a *lazy learning* technique: generalization only occurs when a new pattern is offered to the performance component and abstraction is therefore implicit in the way the contents of the case base and the similarity metric interact.

One succesful statistical approach, recently applied to the tagging problem, is Ratnaparkhi's use of Maximum Entropy Models (1996). In this classification-based approach, diverse sources of contextual information (comparable to those used by the machine learning approaches discussed below) are expressed as binary features, and are combined in a statistical model that makes no further distributional assumptions on the training data by maximizing the entropy of the distribution subject to the constraints of the training data. The model parameters for the distribution are estimated using an iterative procedure called generalized iterative scaling.

In the remainder of this chapter, we will discuss each of the learning methods and their application to tagging in turn. We conclude with a general discussion and evaluation of the methods described.

17.3 CASE-BASED LEARNING

The case-based learning paradigm is founded on the hypothesis that performance in cognitive tasks (in our case language processing) is based on reasoning on the basis of analogy of new situations to *stored representations of earlier experiences* rather than on the application of *mental rules* abstracted from representations of earlier experiences as in rule induction and rule-based processing.

The concept has appeared in several AI disciplines (from computer vision to robotics) several times, using apart from case-based also labels such as similarity-based, example-based, exemplar-based, analogical, nearest-neighbour, instance-based and memory-based (Stanfill and Waltz 1986; Kolodner 1993; Aha *et al.* 1991; Salzberg 1990). These different names are conveniently captured under the term *lazy learning* (Aha 1997).

17.3.1 Algorithm

Examples are represented as a vector of feature values with an associated category label. Features define a pattern space. During training, a set of examples (the training set) is presented in an incremental fashion to the learning algorithm and added to memory. During processing, a vector of feature values (a previously unseen test pattern) is presented to the system. Its distance to all examples in memory is computed using a *similarity metric* and the category of the most similar instance(s) is used as a basis to predict the category for the test pattern.

In this type of lazy learning, performance crucially depends on the similarity metric used. The most straightforward metric for a problem like tagging with nominal (non-numeric) feature values would be an *overlap metric*: similarity is defined as the number of feature values that are equal in two patterns being compared. In such a distance metric, all features describing an example are interpreted as being equally important in solving the classification problem, but this is not necessarily the case: the category of the word immediately before a word to be tagged is obviously more important than the category of the word three positions earlier in the sentence. We call this problem the *feature relevance problem*. Various feature weighting and selection methods have been proposed to differentiate between the features on the basis of their relevance for solving the task (see Wettscherek *et al.* (1996) for an overview).

Another addition to the basic algorithm that has proved relevant for many natural language processing tasks is a value difference metric (Stanfill and Waltz 1986; Cost and Salzberg 1993). Such a metric assigns different distances to pairs of values for the same feature. In tagging, e.g., such a metric would assign a smaller distance between NP and NN than between NN and VBG. These biases can of course also be added by hand to the learner (e.g. by a domain expert). Several other improvements and modifications to the basic case-based learning scheme have been proposed and should be investigated for linguistic problems. Two promising further extensions are weighting the examples in memory and minimizing storage by keeping only a selection of examples. In example weighting, examples are differentiated according to their quality as predictors for the category of new input patterns. This quality can be based on their typicality or on their actual performance as predictors on a held-out test set. In example selection, memory is pruned by deleting those examples which are bad predictors or which are redundant.

17.3.2 Case-based tagging

KENMORE (Cardie 1994, 1996) is presented as a general framework for knowledge acquisition for NLP using different symbolic machine learning techniques. As an instance of this general methodology, a case-based learning approach is suggested for both morphosyntactic and semantic tagging. The architecture presupposes a corpus, a sentence analyser and a learning algorithm. During knowledge acquisition (training) for a specific disambiguation task (e.g. tagging), a case is created for each instance of the problem in the corpus. Each case is an example of the input-output mapping to be learned; the input part is a context describing the ambiguity and the output part is the solution to the particular ambiguity. The examples may be produced from an annotated version of the corpus or through human interaction. During application, the case-base is used to predict the solution to a new instance of the ambiguity given the input (the context) without intervention.

In a tagging experiment based on 2056 cases from the Tipster JV corpus, a fairly complex case representation based on output from the CIRCUS conceptual sentence analyser is used. Figure 17.2[3] shows the case representation containing context features of the focus word "parts", for which the word sense and the part of speech has to be decided.

Local context features describe the syntactic and semantic information about a five-word window centered on the word to be tagged (the words themselves, their part of speech and their word sense). Global context features provide information about the major constituents parsed already (word sense of the subject and type, concept and semantic feature associated with the last parsed constituent). The class to be predicted is the part of speech and the semantic sense of the middle word. All words (except 129 function words) are initially assumed unknown.

As a solution to the feature relevance problem, Cardie (1993) applies a decision tree learning algorithm (see below) to the dataset and uses only those features which have been selected by the decision tree induction algorithm as being relevant during similarity comparison. This turns out to be an effective way to discard irrelevant features.

MBT (Memory-Based Tagging; Daelemans 1995; Daelemans *et al.* 1996) is a case-based approach in which the feasibility of the approach on a larger scale is investigated, with simpler case representations and with a more elegant solution to the feature rele-

[3] Reprinted from Cardie (1996) with permission.

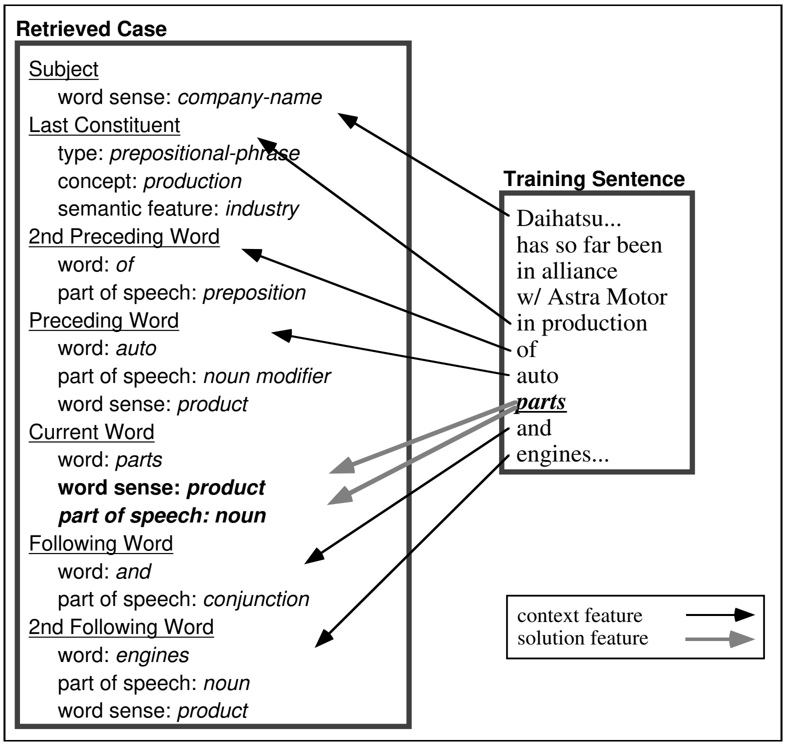

Figure 17.2 Case representation in KENMORE.

vance problem. Experiments were performed on the 3 million word tagged Wall Street Journal corpus[4].

In order to adopt the memory-based approach to the problem of tagging, the following procedure is used:

Lexicon Construction. A lexicon is extracted from the training corpus by computing for each word the number of times it occurs with each category. A new, possibly *ambiguous* tag (e.g. N-V for words which can be both a noun and a verb) is assigned to each word based on this lexical definition.

[4] ACL Data Collection Initiative CD-ROM 1, September 1991; the tagset is the Penn Treebank tagset, consisting of 40 tags. See Appendix 17.6 for a full list.

Case-Base Construction. Two case bases are constructed, one for known words and one for unknown words. The former contains as information the possibly ambiguous category of the word to be tagged (the focus word) and of one context word to the right and the disambiguated category of two words to the left. The latter contains the same context information, but instead of the ambiguous category of the focus word (which is unknown), the first letter and the last three letters of the focus word are added as features. These features provide information about the 'morphology' of the word. The unknown-words case base is constructed on the basis of open class words only.

Tables 17.4 and 17.5 list samples of the known-words and unknown-words case bases for part of the first sentence of the corpus. In the first table, we use the following abbreviations for the known-words case base: f for focus word (the word to be disambiguated, represented by its ambiguous category), d for disambiguated word (a previously contextually disambiguated word to the left of the focus word), a for ambiguous word, a still to be disambiguated word to the right of the focus word, represented by its ambiguous category, and t for the target, disambiguated, category of the focus word. In the unknown-words case base, we find features for left (d) and right (a) context, and instead of the lexical representation of the focus word, we have features representing prefix letters (p) and suffix letters (s) of the focus word.

Tagging. In tagging, new input text is transformed into case representations for case-based reasoning on the basis of either the known- or unknown-words case base.
In MBT, the feature relevance problem is solved by weighting each feature with the average amount of case base information entropy reduction it can provide (i.e. its Information Gain, IG; see Daelemans *et al.* (1996) for more information). The weights for the different features are also listed in tables 17.4 and 17.5. They express the relative relevance of the features and are used as a weight during similarity computation. Advantages of this approach (compared to KENMORE's) are that feature relevance is not interpreted as a yes/no property but as a gradual one and that it does not presuppose using two inductive classification learning algorithms (decision tree induction and case-based learning), one of which is only used for feature selection.

To solve the computational complexity problem inherent in matching all feature values of a new case to the corresponding values of all stored cases, MBT uses IGTREE, a memory- and processing-time-saving heuristic implementation of memory-based reasoning. This formalism (fully described in Daelemans *et al.* 1996, 1997) compresses a memory base into a tree using an information-theoretic heuristic, reducing storage and retrieval complexity considerably without an adverse effect on generalization accuracy. An additional advantage is that this tree structure allows dynamic selection of context width (see also 17.4 below on decision trees).

Table 17.4 Case representation and information gain pattern for known words.

Word	Case Representation				
	d	d	f	a	t
IG	.06	.22	.82	.23	
Pierre	=	=	np	np	np
Vinken	=	np	np	,	np
,	np	np	,	cd	,
61	np	,	cd	nns	cd
years	,	cd	nns	jj-np	nns
old	cd	nns	jj-np	,	jj

Table 17.5 Case representation and information gain pattern for unknown words.

Word	Case Representation						
	d	p	s	s	s	a	t
IG	.21	.21	.15	.20	.32	.14	
Pierre	=	P	r	r	e	np	np
Vinken	np	V	k	e	n	,	np
61	,	6	=	6	1	nns	cd
years	cd	y	a	r	s	jj-np	nns
old	nns	o	o	l	d	,	jj

17.3.3 Evaluation

In the experiment with KENMORE, accuracy of tagging turned out to be 95% overall and 91% on contents words only. In MBT on the Wall Street Journal corpus, overall generalization accuracy was 96.4% (96.7% on known words, 90.6% on unknown words). The level of accuracy attained by probabilistic taggers seems to be well in reach of case-based taggers. Already at small data set sizes, performance is relatively high. We obtained similar scores when adapting the tagger architecture to Dutch by training it on a Dutch tagged corpus.

An important advantage of the case-based approach is the flexibility of case representations: there are several types of information which can be stored in the memory base, ranging from the words themselves to intricate lexical representations. Combined with feature-weighting approaches, this flexibility offers a new approach to information source integration (data fusion) in tagging. The unknown-words case base in MBT, e.g., integrates context information and 'morphological' information (suffix letters) in a smooth way. It would be impossible at present, for reasons of sparseness of data and computational complexity, to estimate probabilities for such intricate contexts in a stochastic approach.

Additional advantages include incremental learning (new cases can be added incrementally to the case bases without need for relearning[5]), explanation capabilities (the best memory matches serve as explanations for the tagging behaviour of the system) and, at least in MBT, fast learning and tagging (more than 1000 words per second).

17.4 DECISION TREE INDUCTION

The *decision tree learning* paradigm is based on the assumption that similarities between examples can be used to automatically extract decision trees and categories with explanatory and generalization power. In other words, the extracted structure can be used to solve new instances of a problem and to explain why a performance system behaves the way it does. In this paradigm, learning is *greedy* and abstraction occurs at learning time. There are systematic ways in which decision trees can be transformed into rule sets (the two representations are equivalent).

Decision tree induction is a well-developed field within AI. See, e.g., Quinlan (1993) for a synthesis of major research findings. More ancient statistical pattern recognition work (such as Hunt *et al.* 1966; Breiman *et al.* 1984) also still makes for useful reading.

17.4.1 Algorithm

A decision tree is a data structure in which nodes represent tests, and arcs between nodes represent possible answers to tests. Leaf nodes represent answers to problems. A problem is solved by following a path from the root node through the decision tree until a leaf node is reached. The path taken depends on the answers that a particular problem provides to the tests at the nodes. Decision tree *learning* works by repeatedly dividing the set of examples into subsets according to whether the examples in a particular subset have a feature-value pair in common, until the subsets are homogeneous, i.e. all examples in the subset have the same category. The algorithm achieves this according to the simplified recursive scheme in Figure 17.3.
To classify new input patterns with a decision tree, start at the root node of the tree and find the value in the input pattern for the corresponding feature. Take the branch corresponding to that value and perform this process recursively until a leaf node is reached. The category corresponding to this leaf node is the output.

[5]Computation of feature weights is not incremental, it presupposes access to a complete batch of training examples, but usually, the weight values become stable after only a few hundred training examples.

Given a set of examples T

If T contains one or more cases all belonging to the same class C_j,
then the decision tree for T is a leaf with category C_j.

If T contains different classes then

- Choose a feature, and partition T into subsets that have the same value for the feature chosen. The decision tree consists of a node containing the feature name and a branch for each value leading to a subset.
- Apply the procedure recursively to subsets created this way.

Figure 17.3 Recursive scheme for constructing decision trees.

Again, we are confronted with a *feature relevance problem* in this approach. In order to obtain a concise tree with good generalization performance (i.e. a tree reflecting the structure of the domain), we have to select at each recursion of the above algorithm a test which is optimal in achieving this goal). The algorithm is non-backtracking and considering all trees consistent with the data is an NP-complete problem, so a reliable heuristic feature selection criterion is essential. Information-theoretic or statistical techniques maximizing homogeneity of subsets by selecting a particular feature are usually applied to this end. Several variants and extensions have been developed to the basic algorithm for pruning (making the tree more compact by cutting off subtrees on the basis of a statistical criterion), grouping similar values of a feature into classes, making tree building incremental, etc.

17.4.2 Decision tree tagging

Work on parsing (including tagging) of text with decision trees was pioneered at IBM (Black *et al.* 1992; Magerman 1994, 1995). SPATTER (Magerman 1995) starts from the premise that a parse tree can be viewed as the result of a series of classification problems (tagging, choosing between constituents, labelling constituents, etc.). The most probable sequence of decisions for a sentence, given a training corpus, is its most probable analysis. In the statistical decision tree technology used (based on Breiman *et al.* 1984), decision trees are constructed for each sub-problem in the parsing task (tagging is one of them). In such a decision tree, leaf nodes contain distributions over categories instead of a single category. E.g., in tagging, the feature associated with the root node of the decision tree might be the word to be tagged. In case its value is “the”, the category “article” can be returned with certainty. In case its value is “house”, a test at the next level of the tree corresponds to the feature “tag of the previous word”. In case its value is “article”, the probability distribution returned by the decision tree would be “noun (.8); verb (.2)” (Magerman 1995). In practice, SPATTER uses binary trees, however. Searching for the most probable tag series for a sentence is done by

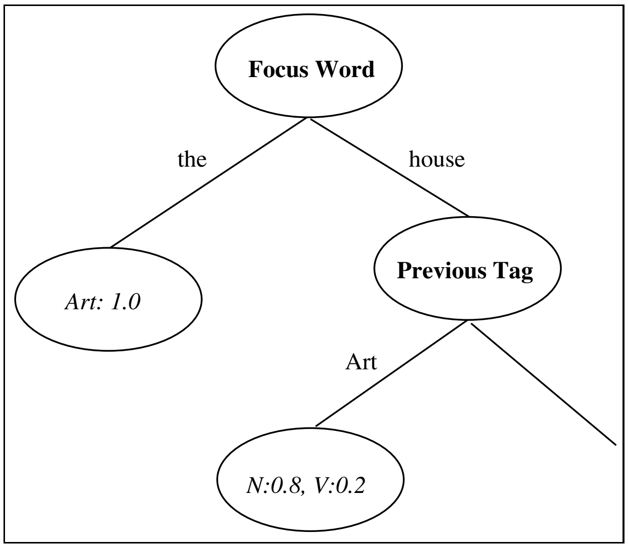

Figure 17.4 A statistical decision tree for tagging.

means of stack decoder search with a breadth-first algorithm and probabilistic pruning. Figure 17.4 shows such a tree, based on Magerman (1995).

Schmid (1994b) describes TREETAGGER, a tagger which takes basically the same approach as SPATTER. Transition probabilities between tags in a tag sequence are estimated using a decision tree induced from a set of N-grams occurring in the Penn Treebank corpus. The features are the tags of the words preceding the word to be tagged. He experimented with one, two and three such context features. The category to be predicted is the tag of the focus word. Using an information-theoretic heuristic feature selection method, the tree is built using the recursive algorithm discussed earlier. As in SPATTER, all tests have binary results. Instead of having a subtree for each possible tag in a context position, corresponding to questions like "what is the tag of the word before the target?", tests are instead individual feature-value combinations with a binary branching, corresponding to questions like "is the tag of the word before the target equal to ADJ?". This results in deeper trees. Again as in SPATTER, at the leaf nodes a probability distribution over the categories is given for those patterns for which the sequence of tests leading to this leaf node are true. The information-theoretic feature selection method ensures that the test chosen maximizes the distinctiveness of the probability distribution of the subtrees. For pruning the resulting decision tree, an information-theoretic heuristic is used as well. Finally, the Viterbi algorithm (Viterbi 1967; see also Chapter 16) is used to find the best tag sequence for a sentence, given the probability distributions obtained by decision tree lookup. The approach is combined with a lexicon system containing a priori tag probabilities for each word and a probabilistic suffix analyser.

17.4.3 Evaluation

Decision tree models are equivalent in expressive power to interpolated N-gram models (Magerman 1995), but whereas in N-gram models the number of parameters to be estimated grows exponentially with N, in decision-tree learning, the size of the model depends on the number of training examples and remains constant with the number of decisions taken into account. Also, the decision tree approach automatically selects relevant context size: uninformative context positions are not used in the tree and because of its computational properties (constant with wider context) larger contexts (corresponding to 4 or 5-grams) can initially be considered. That way, decision tree approaches are potentially more sensitive to context and therefore better equiped to solve long-distance dependencies. Finally, Schmid also reports robustness relative to training set size: TREETAGGER 'degrades gracefully' with smaller training set sizes.

As far as performance is concerned, decision tree methods seem to be comparable to stochastic appoaches: Schmid reports 96.4% generalization on the Penn Treebank using 4-grams (0.3% better than a similar probabilistic trigram tagger, which, however, uses a different lexicon system). Magerman reports 96.5% for sentences up to 40 words in length in the Wall Street Journal corpus. Schmid also reports fast tagging speed performance (10,000 words per second).

In the examples discussed here, decision tree technology does not deliver a solution to the complete tagging problem. The tag probabilities returned by the tree are used by a search mechanism (stack decoder or Viterbi) to find the best series of tags. Also, both versions of the decision tree approach are not completely non-parametric; SPATTER requires smoothing of the decision trees and TREETAGGER requires a pruning threshold. In principle, however, it would be possible to produce a complete tagger on the basis of a learned statistical decision tree. Recently, this approach has indeed been explored (Màrquez & Rodríguez, 1998).

17.5 NEURAL NETWORK METHODS

Multilayer Perceptrons (Rumelhart *et al.* 1986) are the most popular neural network architecture. As an inductive learning technique, supervised neural network learning is a greedy learning approach. During learning, the set of examples is repeatedly inspected to find an optimal set of connection weights between layers of simple units. The training material is thus abstracted into a set of numeric weights which is then used to predict the output of new input patterns. There is an immense literature on neural network algorithms (see Aleksander and Morton (1990), Bishop (1995) and Fausett (1994) for recent introductions). There is also a considerable body of research

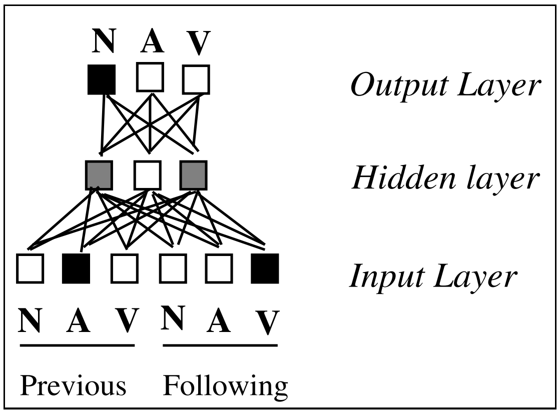

Figure 17.5 A simple neural network architecture for tagging. Activation of units is shown by colouring. In a situation where the previous word is an adjective and the following word a verb, this hypothetical 3-layer network would predict the current word to be a noun.

on applying neural network technology to language processing problems (Reilly and Sharkey (eds.) 1992; Sharkey 1992).

17.5.1 Algorithm

A multilayer perceptron consist of an input and an output layer of simple processing units and one or more intermediate, 'hidden' layers (figure 17.5). The input layer is used to code the input part of an example, the output layer to encode the output part. We will assume a single hidden layer here. All adjacent layers are fully interconnected, i.e. each unit in each layer connected to each unit in the next layer. Each unit has an activation and a threshold. Each connection between two units has a weight. Activation can be expressed as 1 or 0, or as a real number; thresholds and connection weight are usually expressed as real numbers. Activation flows from the input layer to the output layer via the hidden layer.

Two simple rules govern the training and use of a multilayer perceptron. The *activation rule* is a local rule which is used by each unit to compute its activation. Consider for instance the activation flow from input layer to hidden layer. The input activation of a particular unit of the hidden layer (a_j), is equal to $\sum_i a_i \times w_{i,j}$, where the a_i are the activations of the units in the input layer connected to that unit at the hidden layer and the $w_{i,j}$ are the weights of the connections from those input layer units to that hidden layer unit. The threshold value determines whether, given the input activation to a unit, that unit will become active (if the input activation exceeds the threshold the unit becomes active, otherwise it stays 'off'). By making use of such a threshold value, the activation of a unit is a *non-linear* function of activation values of the units in the previous layer connected to it.

The *learning rule* (in this case back-propagation learning) incrementally adapts the connection weights until an acceptable performance is reached by the system (this involves repeatedly cycling through all training examples). This adaptation process works by changing the connection weights depending on the quality of the output of the network (i.e. the activation pattern at the output layer). The output of the network is compared to the desired output. Those connections that contributed to the wrong activation of an output unit are weakened, those that were responsible for not activating an output unit that should have been active are strengthened. These error corrections are also 'back-propagated' to the connections between input and hidden layer.

17.5.2 Neural network tagging

NETGRAM (Nakamura *et al.* 1980) is a multilayer neural network for word category prediction in the context of a speech recognition system. The input consists of the category of two (or more) preceding words, the output is the category of the current word. Words are encoded the following way. Each word is represented by a number of units equal to the number of categories. The unit associated with the category of the word is made active, the other units are inactive. Given two words preceding context and 89 categories, the network has an input layer of 178 units and an output layer of 89 units. Back-propagation is used to train the network. When an input is presented to the network, the two input units corresponding to the categories of the two preceding words are made active and the output unit with the highest activation is taken as the category of the current word.

NET-TAGGER (Schmid 1994a) also used back-propagation learning, but in this case the problem handled is disambiguation (tagging) rather than tag prediction. In the input layer, information about the word to be tagged and one or more preceding and following words is encoded. Again, for each tag and each word pattern position, a unit is created. E.g., supposing 40 tags, 3 words left context, a focus word and 2 words right context, an input layer of 240 (40×6) units is needed. For the left context words, the previous output of the network (the activation levels of the output layer units) is used as input. For the focus word and the right context, the lexical probabilities of the words are used. Adding a hidden layer to a two-layer network did not improve performance. The output layer has one unit for each possible wordclass. The output unit with the highest activation, given an input pattern, is interpreted as the tag of the focus word. In order to attach lexical probabilities to words, a lexicon system based on Cutting *et al.* (1992) is used, combined with an unknown-word guesser using information about suffix letters.

17.5.3 Evaluation

Accuracy of NETGRAM and NET-TAGGER is comparable to stochastic trigram methods, the main advantage being that unseen patterns are interpolated effectively, without requiring expensive special interpolation methods as in stochastic approaches. Connectionist approaches also require the computation of fewer parameters (weights) than statistical models (N-gram probabilities), which becomes especially useful when considering a wider context than trigrams. The number of weight-parameters grows quadratic with the number of input nodes. As in the learning methods discussed earlier, it is possible to output more than one possible tag for a word, e.g. a list of tags ordered according to their likelihood.

17.6 DISCUSSION

With the availability of only a relatively small body of empirical data and theoretical analysis on the applicability of inductive machine learning techniques to tagging, it is too early for strong conclusions. On the empirical side, there is a hard-felt need for a methodologically sound, reliable, empirical comparison of statistical and machine learning approaches to automatic tagging. On the theoretical side, there is a need for more insight into the differences and similarities in how *generalization* is achieved in this area by different statistical and machine learning techniques. In the absence of this knowledge, our discussion will necessarily turn out to be preliminary and superficial. Our discussion will take the form of a number of theses.

- *Learning is preferable to programming.* Compared to hand-crafted rule-based (or constraint-based approaches), an inductive learning approach, such as in the methods discussed here and in stochastic approaches, provides a solution to the knowledge-acquisition and reusability bottlenecks and to robustness and coverage problems. As an example, a fast and accurate tagger for Dutch was learned with the MBT tagger-generator in half a day, with a minimum of linguistic engineering. On the other hand, it should be noted that the accuracy which can be obtained using hand-crafted constraint-based methods (cf. Chapter 14) still seems to be out of reach for automatic learning approaches.

- *Machine Learning is a different type of statistics.* Decision tree induction, case-based learning and neural networks are statistical methods, but they use a different kind of statistics than the well-known maximum-likelihood and Markov model methods, e.g. in case-based learning, no assumptions are made about the distribution of the data whereas most statistical techniques presuppose normal distributions. Different statistical methods have different properties which make them more or less suited for a particular type of application. If only for that reason, the applicability of all types of statistics to the tagging problem should be studied thoroughly. Already from the preliminary empirical data, important advantages of these methods compared to current statistical methods suggest themselves:

1. They require less training data.
2. They require less parameters to be computed and can therefore take into account more context.
3. They provide elegant and computationally attractive solutions to the smoothing problem and to the integration of different information sources.
4. Training is often much faster.

- *Abstraction can be harmful.* In many linguistic tasks, we have found (see Daelemans, Van den Bosch and Zavrel 1999) that an approach keeping complete memory of all training data provides better performance than techniques that abstract from low-frequency and exceptional events, such as rule(learning)-based systems. Neural networks and stochastic approaches are similar to rule- and decision-tree-induction methods in that they abstract from their experience (to a matrix of connection weights in neural networks, to a set of probabilities in stochastic approaches and to a set of rules in rule-induction approaches) and forget about the original data on which these abstractions were based. The effect that full memory of all examples yields better generalization is probably related to the fact that natural language processing tasks such as morphosyntactic disambiguation can be characterised by the interaction of regularities, sub-regularities and pockets of exceptions. Abstracting away from these exceptions causes a performance degradation because new similar exceptions are overgeneralized: being there is better than being probable.

Compared to the well-developed theoretical and empirical foundations of statistical approaches to tagging, the machine learning approach to this problem has only just started. In all methods described, there is still a lot of room for improvement, especially in three areas: exploring variations or extensions of the basic algorithms, adding linguistic bias to the learning algorithms and combining them with other approaches in hybrid architectures. A fourth area where machine learning methods may provide increased tagging accuracy is in the development of machine learning algorithms that take as input the outputs of different taggers (trained, statistical or even hand-crafted) and learn when to trust which tagger. Initial work on this approach is described by van Halteren, Zavrel and Daelemans (1998) and by Brill and Wu (1998).