A Comparative Study of the Application of Different Learning Techniques to Natural Language Interfaces

Werner Winiwarter and **Yahiko Kambayashi**
Dept. of Information Science
Kyoto University
Sakyo, Kyoto 606-01, Japan
{ww|yahiko}@kuis.kyoto-u.ac.jp

Abstract

In this paper we present first results from a comparative study. Its aim is to test the feasibility of different inductive learning techniques to perform the automatic acquisition of linguistic knowledge within a natural language database interface. In our interface architecture the machine learning module replaces an elaborate semantic analysis component. The learning module learns the correct mapping of a user's input to the corresponding database command based on a collection of past input data. We use an existing interface to a production planning and control system as evaluation and compare the results achieved by different instance-based and model-based learning algorithms.

1 Introduction

One of the main obstacles to the efficient use of natural language interfaces is the often required high amount of manual knowledge engineering (see (Androutsopoulos et al., 1995) for a recent survey). This time-consuming and tedious process is often referred to as "knowledge acquisition bottleneck". It may require extensive efforts by experts highly experienced in linguistics as well as in the domain and the task (Riloff and Lehnert, 1994). Therefore, natural language interfaces represent a domain that is very well suited for the application of machine learning algorithms to automate the acquisition process of linguistic knowledge.

So far, inductive learning has already been applied successfully to a large variety of natural language tasks. This includes basic linguistic problems such as morphological analysis (van den Bosch et al., 1996), parsing (Zelle and Mooney, 1996), word sense disambiguation (Mooney, 1996), and anaphora resolution (Aone and Bennett, 1996). Besides this, there also exists some research on applications, e.g. machine translation (Yamazaki et al., 1996), text categorization (Moulinier and Ganascia, 1996), or information extraction (Soderland et al., 1996).

The learning task in natural language interfaces is to select the correct command class based on semantic features extracted from the user input. Therefore, it can be modeled as classification problem, i.e. the machine learning algorithms construct a theory from the training data that is used for classifying unseen test data (Quinlan, 1996). So far, we consider only supervised learning so that each training case has to be labeled with the correct class.

We apply different existing instance-based and model-based algorithms to this problem and compare the achieved results. In addition, we have also developed several new algorithms, which we present briefly in this paper. We have implemented all algorithms by means of the deductive object-oriented database system *ROCK & ROLL* (Barja et al., 1994).

It solves the problem of updates in deductive databases in that it separates the declarative logic query language *ROLL* from the imperative data manipulation language *ROCK* within the context of a common object-oriented data model. Besides this, ROCK & ROLL makes a clean distinction between *type declarations*, which describe the structural characteristics of a set of instance objects and the methods that can be applied to them, and *class definitions*, which specify the implementation of the methods associated with a type.

The use of the available powerful logic and object-oriented programming language enables an efficient implementation of the different approaches to machine learning. It also gives us a convenient integrated tool that assists in applying the machine learning algorithms to the data collection stored in the same database.

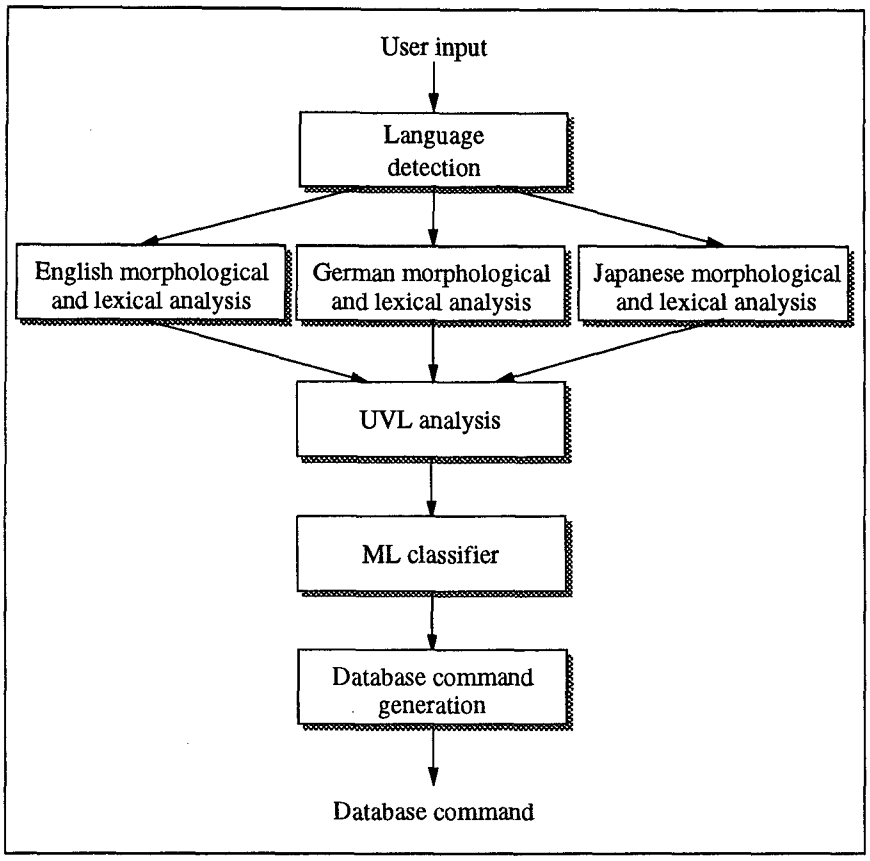
User input
Language detection
English morphological and lexical analysis
German morphological and lexical analysis
Japanese morphological and lexical analysis
UVL analysis
ML classifier
Database command generation
Database command

Figure 1: System architecture of natural language interface

As comparative evaluation of the implemented algorithms, we applied them to an extensive case study: a natural language interface for a production planning and control system. The system is used in a multilingual environment, which includes the languages English, German, and Japanese. Therefore, an important issue of the evaluation was to check whether the learned knowledge is language-independent, i.e. if it really operates based on semantic deep forms so that it abstracts from linguistic surface phenomena.

The rest of the paper is organized as follows. First, we briefly introduce the learning task before we present the applied machine learning algorithms in more detail. Finally, we explain the set-up of the case study and discuss the achieved results from evaluation.

2 Learning Task

Our interface architecture is displayed in Fig. 1. It represents a multilingual database interface for the languages English, German, and Japanese. First, the *language* of the user input is *detected* and the input is transferred to the corresponding language-specific *morphological and lexical analyzer.*

Morphological and lexical analysis performs the *tokenization* of the input, i.e. the segmentation into individual words or tokens. This task is not always trivial as in the case of Japanese, which uses no spaces for separating words. As next step the input is transformed into a *deep form list (DFL)*, which indicates for each token its surface form, category, and semantic deep form.

For database interfaces, unknown values contained in the input possess particular importance for the meaning of a command. Therefore, we treat those unknown values separately in the *unknown value list (UVL) analyzer.* This module checks the data type of unknown values and looks them up in the database to find out whether they represent identifiers of existing entities. In such a case, the entity type is indicated in the resulting UVL, otherwise we use the data type instead.

DFL and UVL represent the input to the *machine learning (ML) classifier.* It assigns a ranked list of command classes to the input sentence according to the learned classification rules. As last step the classifications are used for *generating* appropri-

Input	New purchase price of St 37 H is 1,7.	St 37 H kostet nun 1.7 Schilling	St37Hの購入代金を 1,7に変えなさい。
DFL	new purchase price of be	cost now schilling	purchase price change
UVL	1 material 1 real	1 material 1 real	1 material 1 real

Figure 2: Example of feature encoding

ate *database commands*.

For the encoding of the training data we only make use of the semantic deep forms contained in the DFL. We use English concepts as deep forms and map them to binary features, i.e. a certain feature equals 1 if the deep form is a member of the DFL, otherwise it equals 0. For the elements of the UVL we apply a more detailed encoding, which maps the number and the type to binary features. Figure 2 shows an example of the features derived from English, German, and Japanese input sentences for the update of the purchase price for a material.

Thus, the learning task replaces an elaborate semantic analysis of the user input. The development of the corresponding underlying rule base might require several man-months. The learning task represents a realistic real-life application, which differs from many other problems studied in machine learning research in that it consists of a large number of features and classes. Furthermore, the command classes are often very similar and even for human experts very difficult to distinguish.

3 Learning Algorithms

Instance-based Learning

Instance-based approaches represent the learned knowledge simply as collection of training cases or *instances*. For that purpose they use the same language as for the description of the training data (Quinlan, 1993a). A new case is then classified by finding the instance with the highest similarity and using its class as prediction. Therefore, instance-based algorithms are characterized by a very low training effort. On the other hand, this leads to a high storage requirement because the algorithm has to keep all training cases in memory. Besides this, one has to compare new cases with all existing instances, which results in a high computation cost for classification.

Different instance-based algorithms vary in how they assess the similarity (or distance) between two instances. Two very commonly used methods are *IB1* (Aha et al., 1991) and *IB1-IG* (Daelemans and van den Bosch, 1992). Whereas IB1 applies the simple approach of treating all features as equally important, IB1-IG uses the information gain (Quinlan, 1986) of the features as weighting function.

We have developed an algorithm called *BIN-CAT* for binary features with class-dependent weighting and asymmetric treatment of the feature values. The similarity between a new case X and a training case Y is calculated according to the following formula:

$$\begin{aligned} \mathrm{SIM}_{X,Y} &= \sum_{i=1}^{n} p\left(D_i, C_Y\right) \cdot w_i \cdot \sigma\left(x_i, y_i\right) - \\ &\quad \sum_{i=1}^{n} p\left(D_i, C_Y\right) \cdot w_i \cdot \delta_Y\left(x_i, y_i\right) - \\ &\quad \sum_{i=1}^{n} \left[1 - p\left(D_i, C_Y\right)\right] \cdot w_i \cdot \delta_X\left(x_i, y_i\right) \quad . \end{aligned} \tag{1}$$

In this formula, n indicates the number of features, D_i the number of instances that have value 1 for feature i, and C_Y the class of the training case Y. The term $p(D_i, C_Y)$ then denotes the proportion of instances in D_i that belong to class C_Y. $\sigma(x_i, y_i)$, $\delta_Y(x_i, y_i)$, and $\delta_X(x_i, y_i)$ are determined as follows:

$$\begin{aligned} \sigma\left(x_i, y_i\right) &= \begin{cases} 1 & \text{if } x_i = 1 \wedge y_i = 1 \\ 0 & \text{otherwise} \end{cases} \\ \delta_Y\left(x_i, y_i\right) &= \begin{cases} 1 & \text{if } x_i = 0 \wedge y_i = 1 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

$$\delta_X(x_i, y_i) = \begin{cases} 1 & \text{if } x_i = 1 \wedge y_i = 0 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

so that the second sum in (1) is rated higher for a larger number of occurrences of the ith feature for class C_Y whereas the third sum is rated lower. This means that if the training case Y contains a certain feature and the new case X does not, then we rate this difference the stronger the more often the feature occurs for class C_Y. On the other hand, for features appearing in the new case X but not in Y, the opposite is true.

Finally, w_i represents the weight of feature i. It is calculated by making use of the following formula:

$$w_i = \frac{1}{c} \cdot \sum_{j=1}^{c} 1 - 4 \cdot p(D_i, j) \cdot [1 - p(D_i, j)] \ . \quad (3)$$

The term under the summation symbol represents the selectivity of feature i for class j. It equals 1 if either all or none of the cases have value 1 for this feature. In other words, all instances for class j then either possess or do not possess this feature, which makes it a very discriminative characteristic. The other extreme is that $p(D_i, j)$ equals 50 %. In that case, this feature allows for no prediction of the class and the term under the summation symbol becomes 0.

We have implemented all above-mentioned algorithms for binary features in ROCK & ROLL in that we store the instances as objects and assign to them the features as ordered lists sorted by the feature numbers. The calculation of the similarity between two cases is then realized as method invocation on the feature list. For example, Fig. 3 shows the ROCK method to compute the distance between two feature lists according to IB1.

Besides pure instance-based learning we have also developed an algorithm *BIN-PRO*, which creates a *prototype* for each class. Those prototypes are then used for the comparison with new cases. This has the big advantage that one does not have to store all the training instances and that the number of required comparisons for classification is reduced to the number of existing classes. As similarity function between a new case X and a certain class C we use the following formula:

$$\begin{aligned} \mathrm{SIM}_{X,C} = & \sum_{f \in X} |D_C| \cdot p(D_f, C) \cdot w_f - \\ & \sum_{f \notin X} p(D_f, C) \cdot w_f \ . \end{aligned} \quad (4)$$

In this formula, we give more emphasis to features f that are present in X in that we multiply them by $|D_C|$, the number of instances for class C. However, the second sum takes also important features for class C into account that are missing in the new case X. As weighting function w_f we use again (3). The implementation in ROCK & ROLL is performed by creating an object for each prototype and by invoking the associated method for computing the similarity to a new test case.

Model-based Learning

In contrast to instance-based learning, model-based approaches represent the learned knowledge in a theory language that is richer than the language used for the description of the training data (Quinlan, 1986). Such learning methods construct explicit generalizations of training cases resulting in a large reduction of the size of the stored knowledge base and the cost of testing new test cases.

In our research we consider the subtypes of decision trees and rule-based learning as well as hybrid approaches between them. The main difference between the various methods for constructing *decision trees* is the selection of the feature for splitting a node. The following two main categories are distinguished:

- *static splitting*: selects the best feature for splitting always on the basis of the complete collection of instances,
- *dynamic splitting*: re-evaluates the best feature for splitting for each node based on the current local set of instances.

Static splitting requires less computational effort because it performs the feature ranking only once for the construction process. However, it entails overhead to keep track of already used features and to eliminate features that provide no proper splitting of the set of instances. Besides that, dynamic splitting methods produce much more compact trees with fewer nodes, leaves, and levels. This results in a sharp reduction of the storage requirement as well as the number of comparisons during classification.

We have implemented decision trees for static (*BS-tree*) and dynamic splitting (*BD-tree*) by using the weighting function (3) as ranking scheme for the splitting criterion. In addition, we have also implemented the *IGTree* algorithm (Daelemans et al., 1997), which uses the information gain as static splitting criterion, and *C4.5* (Quinlan, 1993b), which applies the information gain to dynamic splitting. The decision trees are implemented in

```
type E.featurelist:                                type declaration for feature lists
    public [feature];                              list of features
    ROCK:                                          ROCK methods
        distance(x: featurelist): int;             method for calculation of distance to
end-type                                           feature list of new instance X
class E.featurelist                                persistent class definition
    public:                                        visibility
    distance(x: featurelist): int                  method for distance calculation
    begin
        var ix: int;                               index for instance X
        var iy: int;                               index for instance Y
        var fx: int;                               feature for instance X
        var fy: int;                               feature for instance Y
        var dist: int;                             computed distance
        ix := 1;                                   initialization of index ix
        iy := 1;                                   initialization of index iy
        while (ix <= upper@x) do                   while index ix ≤ that of last feature for X do
        begin
            fx := get_fnr@(f[ix]);                 get feature number for instance X at index ix
            fy := get_fnr@(get_member_at(iy)@self); get feature number for instance Y at index iy
            if (fx = fy) then                      if same feature, then
            begin
                ix := ix + 1;                      increment indices
                iy := iy + 1;
            end
            else                                   else
            begin
                dist := dist + 1;                  increment distance
                if (fx < fy) then                  if feature number for X smaller than
                    ix := ix + 1;                  that for Y, then increment index ix
                else                               else
                    iy := iy + 1;                  increment index iy
            end
            if (ix > upper@x) then                 if index ix > that of last feature for X, then
                while (iy <= upper@self) do        while index iy ≤ that of last feature for Y do
                begin
                    iy := iy + 1;                  increment index iy
                    dist := dist + 1;              increment distance
                end
            if (iy > upper@self) then              if index iy > that of last feature for Y, then
                while (ix <= upper@x) do           while index ix ≤ that of last feature for X do
                begin
                    ix := ix + 1;                  increment index ix
                    dist := dist + 1;              increment distance
                end
        end
        dist                                       return distance
    end
end-class
```

Figure 3: ROCK & ROLL code segment for IB1 distance calculation

ROCK & ROLL by creating an object for each node and by linking the nodes according to the tree structure. The classification of a new case is then simply performed as top-down traversal of the tree starting from the root. Besides this exact search we have also implemented an approximate search method, which allows one incorrect edge along the traversal to find a larger number of similar cases.

Rule-based learning represents a second large category of model-based techniques. It aims at deriving a set of rules from the instances of the training set. A *rule* is here defined as a conjunction of *literals*, which, if satisfied, assigns a class to a new case. For the case of binary features, the literals correspond to *feature tests* with positive or negative *sign*. This means that they check whether a new case possesses a certain feature (for positive tests) or not (for negative tests).

The methods for deriving the rules originate from the field of *inductive logic programming* (Muggleton, 1992). One of the most prominent algorithms for rule-based learning is *FOIL* (Quinlan and Cameron-Jones, 1995), which learns for each class a set of rules by applying a separate-and-conquer strategy. The algorithm takes the instances of a certain class as *target relation*. It iteratively learns a rule and removes those instances from the target relation that are covered by the rule. This is repeated until no in-

```
type E.literal:                         type declaration for literals
    properties:                         attributes
        public:                         visibility
            litf: feature,              feature
            sign: bool;                 sign of feature test
    ROLL:                               ROLL methods
        differ(featurelist);            method for performing feature test
end-type
class E.literal                         persistent class definition
    public:                             visibility
    differ(featurelist)                 method for performing feature test
    begin                               returns true if test is not satisfied, otherwise false
        differ(Flist) :-                test for positive sign
            S == get_sign@self,         get sign of feature test
            S == true,                  test if sign is positive
            F == get_litf@self,         get feature
            ~is_in(F)@Flist;            test if feature is not member of feature list
        differ(Flist) :-                test for negative sign
            S == get_sign@self,         get sign of feature test
            S == false,                 test if sign is negative
            F == get_litf@self,         get feature
            is_in(F)@Flist;             test if feature is member of feature list
    end
end-class
type E.rule:                            type declaration for rule
    properties:                         attributes
        public:                         visibility
            rulenr: int,                rule number
            ruleclass: int;             class of rule
    public [literal];                   list of literals
    ROLL:                               ROLL methods
        differ(featurelist);            method for performing test of rule
end-type
class E.rule                            persistent class definition
    public:                             visibility
    differ(featurelist)                 method for performing test of rule
    begin                               returns true if test is not satisfied, otherwise false
        differ(Flist) :-
            is_in(L)@self,              get individual literals
            differ(Flist)@L;            invoke method for all literals
    end
end-class
```

Figure 4: ROCK & ROLL code segment for test of rules

stances are left in the target relation. A rule is grown by repeated specialization, adding literals until the rule does not cover any instances of other classes. In other words, the algorithm tries to find rules that possess some *positive bindings*, i.e. instances that belong to the target relation, but no *negative bindings* for instances of other classes. Therefore, the reason for adding a literal is to increase the relative proportion of positive bindings.

As weighting function for selecting the next literal, FOIL uses the information gain. We have implemented FOIL, and besides this, we also use the algorithm *BIN-rules* with the following weighting function:

$$W_{f,s,C} = b_f^+ \cdot (b^- - b_f^-) \cdot w_{f,s,C} \ . \qquad (5)$$

In this formula, s indicates the sign of the feature test. The number of positive (negative) bindings after adding the literal for the test of feature f is written as b_f^+ (b_f^-). Finally, b^- indicates the number of negative bindings before adding the literal so that $b^- - b_f^-$ calculates the reduction of negative bindings achieved by adding the literal. The weights $w_{f,s,C}$ are calculated as class-dependent weights for class C by making use of the feature weights w_f from (3):

$$w_{f,s,C} = \begin{cases} w_f \cdot p(D_f, C) & \text{if } s \text{ positive} \\ w_f \cdot [1 - p(D_f, C)] & \text{otherwise} \ . \end{cases} \qquad (6)$$

We have implemented the test of rules as deductive ROLL method as shown in Fig. 4. The invocation of the method is a query with the parameter `fl` for the feature list of the new case. The test returns

false for those rules that are satisfied by the new case. The result of the query can then be assigned to the set of satisfied rules **rs** by using the command: `rs := [{R}|~differ(!fl)@R];`. As in the case of decision trees, we have developed an approximate test, which tolerates one divergent literal.

As last group of model-based algorithms we look at *hybrid approaches* between decision trees and rule-based learning. There exist two ways in principle to combine the advantages of the two paradigms. The first one is to extract rules from a decision tree whereas the second one follows the opposite direction by constructing a decision tree from a rule base.

As example of the first type of approach we have implemented *C4.5-RULES* (Quinlan, 1993b), which extracts rules from the decision tree built by C4.5. Rules are computed as paths along the traversal from the root to all leaves. In a second run, rules are pruned by removing redundant literals and rules.

Regarding the second type of approach, we start from the rule base produced by BIN-rules and use it for building an *SE-tree* (Rymon, 1993). SE-trees are a generalization of decision trees in that they allow not only one but several feature tests at one node. Therefore, a much flatter and more compact tree structure is achieved. For the construction of the tree we sort the feature tests of the rules first. Starting from a root node, we then construct paths according to the literals of the individual rules. For this process we make use of existing paths as far as possible before creating new branches.

4 Evaluation

As case study for investigating the feasibility of the implemented machine learning algorithms, we use a multilingual natural language interface to a *production planning and control system (PPC)*. The PPC performs the mean-term scheduling of products and resources involved in the manufacturing processes, i.e. material, machines, and labor. The resulting master production schedule forms the basis of the coordination of related business services such as engineering, manufacturing, and finance. The modeled enterprise makes precision tools by using job order production and serial manufacture as basic strategies. The efficient realization of the high demands of the application exceeds the power of relational database technology. Therefore, it represents an excellent choice for deriving full advantage of the extended functionality of deductive object-oriented database systems. Furthermore, the sophisticated functionality justifies the effective use of a natural language interface.

During previous research (Winiwarter, 1994) we developed a German natural language interface based on 1000 input sentences that had been collected from users by means of questionnaires. The input sentences were then mapped to 100 command classes (10 for each class). The mapping was performed by elaborate semantic analysis; for the development of the underlying rule base we spent several man-months.

Therefore, we were eager to see if we could replace this extensive effort by a machine learning component that learns the same linguistic knowledge automatically. For this purpose we divided the 1000 sentences into 900 training cases and 100 test cases. In addition, we collected 100 Japanese and 100 English test sentences to check whether the learned knowledge really operates at a semantic level independent from language-specific phenomena.

As result of the encoding of the training set (see Sect. 2), we obtained the large number of 316 features, 289 for the DFL and 27 for the UVL. For the evaluation of the different machine learning algorithms we used as performance measures the *success rate*, i.e. the proportion of correctly classified test cases, and the *top-3 rate*. The latter indicates the proportion of cases where the correct classification is among the first three predicted classes. For the case of model-based approaches we had to produce additional candidates for classes. This was achieved by applying approximate methods that allow one incorrect edge along the traversal of decision trees or one divergent literal for the test of rules (see Sect. 3).

Our first experiment was the comparison of the four instance-based algorithms IB1, IB1-IG, BIN-CAT, and BIN-PRO. As can be seen from the results in Table 1, BIN-CAT clearly outperforms IB1 and IB1-IG. Concerning the method BIN-PRO, which uses prototypes of classes, we achieved results at the same quality level as for BIN-CAT. This is remarkable if one considers the much more condensed representation of the learned knowledge.

The comparison between the results for the individual languages shows that there is no advantage for the German test sentences. On the contrary, the test results for German are inferior to that for English or Japanese. This may be partly due to a greater deviation of the German expressions and phrases used in the test set from the ones used in the training set. Besides this, the restriction of extracted features during encoding the test set to those learned from the training set certainly performs an important filtering function. It removes language-specific syntactic particles that do not contribute to the meaning of the input. This is especially true for the case of Japanese sentences, which possess a

	GERMAN		ENGLISH		JAPANESE	
	Success rate	Top-3 rate	Success rate	Top-3 rate	Success rate	Top-3 rate
IB1	82 %	94 %	98 %	99 %	94 %	98 %
IB1-IG	84 %	98 %	97 %	100 %	90 %	99 %
BIN-CAT	94 %	100 %	99 %	100 %	99 %	100 %
BIN-PRO	95 %	100 %	97 %	100 %	97 %	100 %

Table 1: Test results for instance-based learning

	GERMAN		ENGLISH		JAPANESE	
	Success rate	Top-3 rate	Success rate	Top-3 rate	Success rate	Top-3 rate
IGTree	80 %	94 %	92 %	100 %	86 %	97 %
BS-tree	86 %	97 %	95 %	100 %	90 %	96 %
C4.5	94 %	100 %	94 %	100 %	89 %	100 %
BD-tree	93 %	99 %	94 %	99 %	91 %	99 %
SE-tree	94 %	97 %	96 %	97 %	91 %	95 %

Table 2: Test results for decision trees

completely different syntactic structure in comparison with English or German including many particles with no equivalent words in the other two languages.

The second part of the evaluation was the comparison of the four algorithms for building decision trees: IGTree, BS-tree, C4.5, and BD-tree. Besides this, we also included the SE-tree constructed by a hybrid approach (see Sect. 3). The test results in Table 2 indicate that the trees with dynamic splitting are superior to those with static splitting and that C4.5, BD-tree, and SE-tree produce results of similar quality. Table 3 compares the number of nodes, leaves, and levels for the individual trees. The two trees with dynamic splitting are much more compact than those with static splitting, with C4.5 clearly outperforming BD-tree. Finally, the hybrid SE-tree is much flatter than C4.5 but possesses a larger number of nodes and leaves.

	Nodes	Leaves	Levels
IGTree	865	433	33
BS-tree	719	360	86
C4.5	339	170	26
BD-tree	451	226	52
SE-tree	559	209	8

Table 3: Characteristics for decision trees

As last part of our comparative study we tested the rule-based techniques FOIL, BIN-rules, and the hybrid approach C4.5-RULES. As Table 4 shows, FOIL produces the most compact representation of learned knowledge, followed by C4.5-RULES and BIN-rules. However, according to Table 5 both BIN-rules and C4.5-RULES outperform FOIL with almost identical results.

	Rules	Literals	Max. length
FOIL	215	534	5
BIN-rules	209	726	7
C4.5-RULES	167	677	24

Table 4: Characteristics for rule-based learning

An advantage of rule-based learning in comparison with other methods is that the learned knowledge can be easily presented to the user in a clear and understandable form. The derived rules allow a transparent knowledge representation that one can use for explaining decisions of the system to the user. Figure 5 gives some examples of rule sets learned by BIN-rules for several command classes.

If we take a final look at Table 1, Table 2, and Table 5, we can see that independent from the applied machine learning paradigm the achieved results reached satisfactory quality for all three groups. By considering the three best representatives BIN-CAT, C4.5, and BIN-rules, we obtain an average success rate for all three languages of 94.3 % and a top-3 rate of 98.8 %. This result is surprisingly high if one considers the complexity of the task at hand. Unfortunately, we had no possibility of a direct comparison with the results of the hand-engineered interface because the previous interface had been developed only for German based on the complete collection of 1000 sentences by using a different software. In any case, we could show that machine learning represents a sound alternative to manual knowledge acquisition for the application in natural language interfaces.

	GERMAN		ENGLISH		JAPANESE	
	Success rate	Top-3 rate	Success rate	Top-3 rate	Success rate	Top-3 rate
FOIL	85%	97%	92%	97%	88%	96%
BIN-rules	94%	97%	95%	97%	91%	95%
C4.5-RULES	94%	98%	94%	96%	91%	96%

Table 5: Test results for rule-based learning

Class description	**Rule set**
update of purchase price for material	*1 material* **AND** *1 real*
liquidation of stock for product	*1 product* **AND** liquidate *1 product* **AND** stock
update of salary for operator	*1 operator* **AND** *1 real* *1 operator* **AND** salary *1 operator* **AND** earn
list of product orders grouped by status	status **AND** product order
query of master data for operator	*1 operator* **AND** about *1 operator* **AND** data **AND NOT** stoppage

Figure 5: Examples of learned rules

5 Conclusion

In this paper we have presented first results from a comparative study of applying different inductive learning techniques to natural language interfaces. We have implemented a representative selection of instance-based and model-based algorithms by making use of deductive object-oriented database functionality. The extensive case study for an interface to a production planning and control system shows the feasibility of the approach in that linguistic knowledge is learned the acquisition of which normally takes a large effort of human experts.

Future work will concentrate on the important point of increasing the reliability of test results in that we apply cross-validation trials and statistical tests for the significance of performance differences between two algorithms. Furthermore, we also want to generate learning functions that plot success rates as function of the size of the training collection. Besides this, we plan to test our learning algorithms on standard benchmark machine learning datasets and other typical natural language learning datasets.

Finally, we intend to extend the implemented algorithms to include also unsupervised methods as well as connectionist and evolutionary techniques. In addition, we will implement incremental learning techniques, which continue the learning process during the test phase, and adaptive boosting methods, which apply several classifiers instead of just one. We believe that our study is a first promising step towards the challenging task of carrying out comparative evaluations of the performance of different machine learning algorithms for specific linguistic problems.

References

David W. Aha, Dennis Kibler, and Marc Albert. 1991. Instance-based learning algorithms. *Machine Learning*, 7:37–66.

Ioannis Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databases — an introduction. *Journal of Natural Language Engineering*, 1(1):29–81.

Chinatsu Aone and Scott W. Bennett. 1996. Applying machine learning to anaphora resolution. In S. Wermter, E. Riloff, and G. Scheler, editors, *Connectionist, Statistical, and Symbolic*

Approaches to Learning for Natural Language Processing, pages 302–314. Springer-Verlag, Berlin, Germany.

Maria L. Barja, Norman W. Paton, Alvaro A.A. Fernandes, M. Howard Williams, and Andrew Dinn. 1994. An effective deductive object-oriented database through language integration. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 463–474, Athens, Greece. Morgan Kaufmann, San Mateo, California.

Walter Daelemans and Antal van den Bosch. 1992. Generalisation performance of backpropagation learning on a syllabification task. In M. Drossaers and A. Nijholt, editors, *TWLT3: Connectionism and Natural Language Processing*, pages 27–37. Twente University Press, Enschede, Netherlands.

Walter Daelemans, Antal van den Bosch, and Ton Weijters. 1997. IGTree: Using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review*. To appear.

Raymond J. Mooney. 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*, pages 82–91, Philadelphia, Pennsylvania, May.

Isabelle Moulinier and Jean-Gabriel Ganascia. 1996. Applying an existing machine learning algorithm to text categorization. In S. Wermter, E. Riloff, and G. Scheler, editors, *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, pages 343–354. Springer-Verlag, Berlin, Germany.

Stephen Muggleton, editor. 1992. *Inductive Logic Programming*. Academic Press, London, England.

J. Ross Quinlan. 1986. Induction of decision trees. *Machine Learning*, 1:81–206.

J. Ross Quinlan. 1993a. Combining instance-based and model-based learning. In *Proceedings of the 10th International Conference on Machine Learning*, pages 236–243, Amherst, Massachusetts. Morgan Kaufmann, San Mateo, California.

J. Ross Quinlan. 1993b. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California.

J. Ross Quinlan and R. Michael Cameron-Jones. 1995. Induction of logic programs: FOIL and related systems. *New Generation Computing*, 13:287–312.

J. Ross Quinlan. 1996. Learning first-order definitions of functions. *Journal of Artificial Intelligence Research*, 5:139–161.

Ellen Riloff and Wendy Lehnert. 1994. Information extraction as a basis for high-precision text classification. *ACM Transactions on Information Systems*, 12(3):296–333.

Ron Rymon. 1993. An SE-tree-based characterization of the induction problem. In *Proceedings of the 10th International Conference on Machine Learning*, pages 268–275, Amherst, Massachusetts. Morgan Kaufmann, San Mateo, California.

Stephen Soderland, David Fisher, Jonathan Aseltine, and Wendy Lehnert. 1996. Issues in inductive learning of domain-specific text extraction rules. In S. Wermter, E. Riloff, and G. Scheler, editors, *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, pages 290–301. Springer-Verlag, Berlin, Germany.

Antal van den Bosch, Walter Daelemans, and Ton Weijters. 1996. Morphological analysis as classification: An inductive-learning approach. In *Proceedings of the Second International Conference on New Methods in Language Processing*, Ankara, Turkey, September.

Werner Winiwarter. 1994. *The Integrated Deductive Approach to Natural Language Interfaces*. PhD thesis, University of Vienna, Austria.

Takefumi Yamazaki, Michael J. Pazzani, and Christopher Merz. 1996. Acquiring and updating hierarchical knowledge for machine translation based on a clustering technique. In S. Wermter, E. Riloff, and G. Scheler, editors, *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, pages 329–342. Springer-Verlag, Berlin, Germany.

John M. Zelle and Raymond J. Mooney. 1996. Comparative results on using inductive logic programming for corpus-based parser construction. In S. Wermter, E. Riloff, and G. Scheler, editors, *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, pages 355–369. Springer-Verlag, Berlin, Germany.