Automatic Classification for the Identification of Relationships in a Metadata Repository

Gerd Beuster, Ulrich Furbach,
Margret Groß-Hardt, Bernd Thomas

10/2003

Automatic Classification for the Identification of Relationships in a Metadata Repository

Gerd Beuster   Ulrich Furbach   Margret Gross-Hardt   Bernd Thomas
Universität Koblenz-Landau, Universitätsstr. 1, 56070 Koblenz, Germany
(gb|uli|margret|bthomas)@uni-koblenz.de

February 25, 2003

For a major company a prototype for automatic detection of similar objects in database systems has been developed. This task has been accomplished by transferring the database object classification problem into a text classification problem and applying standard classification algorithms. Although the data provided for the task did not look promising from a technical point of view, the results turned out to be very good.

1 Introduction

Large companies manage huge amounts of data (i.e. data about their customer base, their suppliers, products etc.). Usually, there are many databases and applications that store and provide these data. Since these databases have been developed and managed independently, they are often heterogeneous in logical structure, attribute naming and semantics.

Nowadays, companies face the need for an integrated view on their data. That is, they want to understand the relationships between data in different databases or between applications using the same database. Detecting similar objects and relationships between objects is a crucial integration task in enterprise application integration and business-to-business applications. In order to achieve these goals, heterogeneities within the data stored in different databases have to be recognized or even resolved.

There are various approaches how to deal with heterogeneous data sources. Some are more tightly coupled and define common views on multiple data sources, whereas more loosely coupled approaches maintain the autonomy of different distributes databases [Sheth & J.Larson, 1990; Bouguettaya, Benatallah, & Elmagarmid, 1998].

A recent development is the creation of *meta data repositories* to manage meta data about systems, databases and the data therein [Marco, 2000]. Meta data repositories play the role of information brokers and provide applications and users with the information necessary to determine dependencies between data sources. A meta data repository contains information about objects, tables and relationships in the various databases used in a company. It uses this information to analyze business processes, to provide information about marketing campaigns etc. The amount and accuracy of the meta data is critical for the quality of the meta data repository. Since most databases used in a company are developed independent of each other, information about the same real world entity—e.g customer name and address information—is stored multiple times at different places, under different names and in different formats. In order to make the conglomerate of different database structures manageable and to avoid inconsistencies, these kind of dependencies should be detected and stored in the meta data repository.

The success of a meta data repository is based on the accuracy and completeness of the data. It has to be maintained continuously. This requires a lot of additional work, because meta data usually is maintained manually. When a set of new objects is to be added to the meta data repository, an expert uses her knowledge about the meta data repository and the new objects in order to add it.

In order to reduce the amount of work needed to maintain a meta data repository, the company wizAI did a study on how this process can be automated and developed a prototype. The prototypical system aids the user in adding database objects to the meta data repository. When a new object is added to the repository, the automatic classification system lists potential relationships of the object to other objects. A human is still needed to decide if the suggestions of the system are correct, but a lot less expert knowledge and time is required by the human.

For this task, wizAI adapted a text classification tool developed by University Koblenz-Landau AI Research Group. It turned out that after some transformation of the input data, standard automatic classification methods were able to identify dependencies between database objects with a very high precision.

In the following sections, we give a detailed definition of the task, we show how the problem can be solved by transferring it into a text classification problem, we give some example results, and show the design of the system developed to automate this task.

2 Preliminaries

Next, we give a formal definition of the scenarios and the other concepts we are using throughout this paper. We will use the following terms: *Object* for an entity of a domain, e.g. a system, data model, business object, etc. Let $O_1, \ldots, O_n$ be objects. Each object has a number of characteristics $A_1, \ldots, A_m$ called *attributes*. An object is characterized by its attributes. We write this characterization as $< O_i.A_1, \ldots, O_i.A_m >$

Objects of the same kind are of the same *object type*. The type of an object is an attribute of the object.

(43,Repository Root, Repository Root 4, 1900-01-02, 3000-01-01, PDREP 2001-03-04, BREX_V5, 2001-07-23, PD, PD1)
(1213, Mandant,Mandant, 33, 1900-01-01, 3000-01-01, OAI8PV, 2001-11-22,,)
(45, PPS-Z, PPS-Z, 282, 1900-01-01, 3000-01-01, BREX, 2001-03-07, HKAZJA, 2001-10-31)

Figure 1: Example: Some objects. Each tuple represents one object and table entry. An object is defined by its attributes.

(195, 326, 330, 1900-01-01, 3000-01-01, BREX_V5, 2001-07-03)
(97, 23, 75, 1900-01-01, 3000-01-01, JBI7LRE, 2001-03-13, BREX, 2001-04-17)
(96, 84, 86, 1900-01-01, 3000-01-01, JBI7LR, 2001-03-29, BREX, 2001-04-06)

Figure 2: Example: Relationships between objects: First argument of each tuple is the relationship type, second and third are the two objects.

In the following, we deal with *typed relationships* R over a set of objects $S = \{O_1, \ldots, O_n\}$. R is a subset of $\{(O_i \times O_j \times t) | O_i, O_j \in S, 1 \leq t \leq n\}$. We call the third element of the triple the *type* of the relationship. n is the total number of relationship types. We are only dealing with typed relationships in the rest of this paper.

Examples for objects are given in Figure 1, and examples for relationships between objects are given in Figure 2. It should be noted that the database objects are descriptions of data types, not actual instances of databases entries. In the context of a relational database, the objects are the top rows of each table (which define the table's attributes), not the table's entries in the rest of the rows.

A *meta data repository* is a set of objects, together with a set of typed relationships over these objects: $M = (S, R)$.

3 Problem definition

The classification task was to identify relationships between database objects. We were provided with a list of relationship types, with descriptions of database objects, and with some information about relationships between objects. Figures 1 and 2 give examples for the kind of information provided. To support the user in identify relationships between database objects, two typical usage scenarios were identified.

3.1 Identifying Relationship Types

The first scenario models the situation where the user has identified two objects which might be in one or more relationships to each other. The system should make suggestions about the potential kinds of relationships between the objects. Since the number of relationship types is fairly large (in the example data provided by the client, there were about 100 relationship types),

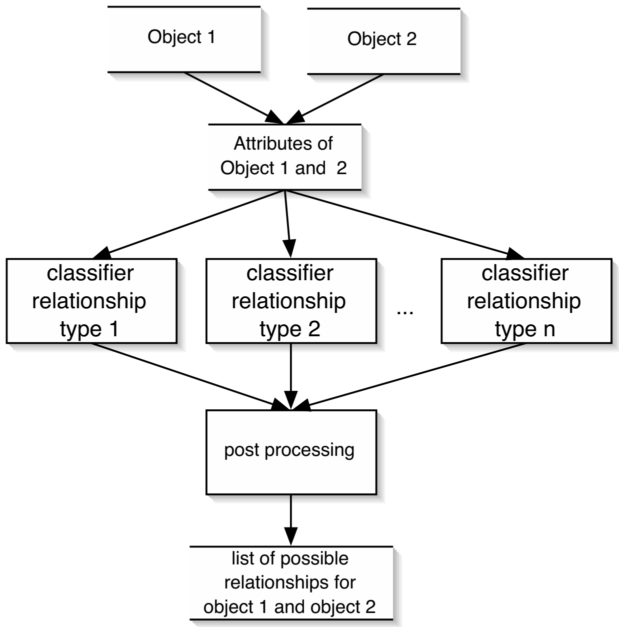
Object 1
Object 2
Attributes of Object 1 and 2
classifier relationship type 1
classifier relationship type 2
...
classifier relationship type n
post processing
list of possible relationships for object 1 and object 2

Figure 3: Scenario 1

an aid of this kind can significantly speed up the process of updating the meta data repository with new data. Scenario 1 can be formally defined as follows:

Definition 1 (Scenario 1) *Given two objects O_1 and O_2, and a meta-data-repository $M = (S, R)$ with $O_2 \in S$, find all potential relationships P that might hold between O_1 and O_2, and present them to the user. The user selects $P' \subseteq P$. An extended repository M' is created by adding these relationships: $M' = (S \cup O_1, R \cup P')$.*

3.2 Identifying Relationship Partner Objects

In scenario 1, the user has to decide if two given objects are in one or more relationships with each other, and the system aids the user in deciding which relationships are applicable. Scenario 2 removes the first requirement. In scenario 2, the user presents a single object to the system, and the system returns a list of other objects from the meta data repository together with the potential relationships between the given objects and the objects found by the system.

Definition 2 (Scenario 2) *Given an object O_1 and a meta-data-repository $M = (S, R)$, find all potential relationships P that might hold between O_1 and some $O_2 \in S$, and present them to the user. The user selects $P' \subseteq P$. An extended repository M' is created by adding these relationships: $M' = (S \cup O_1, R \cup P')$.*

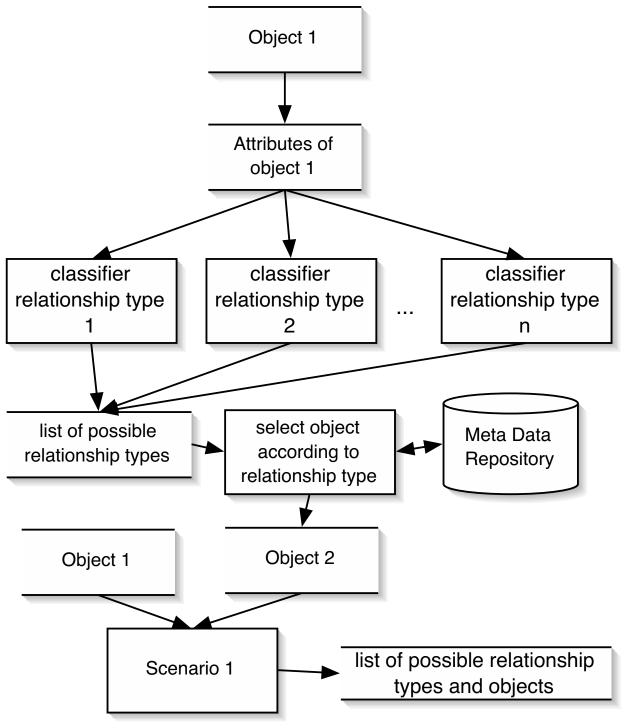
Object 1
Attributes of object 1
classifier relationship type 1
classifier relationship type 2
...
classifier relationship type n
list of possible relationship types
select object according to relationship type
Meta Data Repository
Object 1
Object 2
Scenario 1
list of possible relationship types and objects

Figure 4: Scenario 2

3.3 Problem Discussion

There is an obvious relationship between scenario 2 and scenario 1: In scenario 2, each object from the meta data repository is combined with the new object and run through scenario 1. Thus Scenario 2 can be reduced to scenario 1 by algorithm 1.

Algorithm 1 Reduction of scenario 2 to scenario 1

```
Require: $O_1$ is the new object, and $P = \{\}$
  for all $O_2 \in S$ do
    $P_{new} = P \cup \{r | r \in$ result of running $O_1$ and $O_2$ through scenario 1$\}$
    $P = P_{new}$
  end for
Ensure: $P$ contains all potential new relationships between $O_1$ and another object
```

We assume independence between the relationship types. This means, in order to decide if a given object is in a certain relationship with some other object, we do not take into consideration if the object is in some other relationship with the same or a third object. Therefore, we can treat each relationship type separately.

By treating all relationships separately, the problem of scenario 1 is reduced as follows: Given two objects, decide if they are in the given relationship or not. Thus, the input for the classifier is a tuple of two objects (O_1, O_2), and the output a binary value indicating whether the two objects are in relationship with each other or not.[1] Algorithm 2 shows this classification process.

Algorithm 2 Classification in scenario 1

```
**Require:** A meta data repository $M = (S, R)$; $T$ is the set of relationship types in $M$; an object $O_2 \in S$; a new object $O_1$; $P = \{\}$
  **for all** $t \in T$ **do**
    $M_t = (S, R_t)$, $R_t = \{(O_a, O_b) | (O_a, O_b, t) \in R\}$
    Let $C_t$ be a classifier trained on $M_t$
    **if** $(O_1, O_2)$ potentially in relationship $t$ according to $M_t$ **then**
      $P_{new} = P \cup \{(O_1, O_2, t)\}$
    **else**
      $P_{new} = P$
    **end if**
    $P = P_{new}$
  **end for**
**Ensure:** $P$ contains all potentially new relationships
```

Figure 3 shows this scenario: The objects are combined and transfered into a feature representation. This feature vector is classified by each of the classifiers for the various relationship types. (Note that these classification can be done in parallel for all relationship types.) The classification results of all classifiers are presented to the user.

In scenario two, the user provides an object and is interested in all objects which might be in relation to this object, and the types of these relations.

For scenario 2, algorithm 2 and algorithm 1 could simply be combined. Following this naïve approach is not advisable for two reasons: Since the new object has to be checked against all existing objects for all relationship types, with N objects and R relationship types in the meta data repository, $N \times R$ classification would be necessary for each new object. This does not only increases the complexity of the classification process, we must also assume that there will be quite a number of false results (see Section Results for classification results).

For these reasons, the classification process in scenario 2 (depicted in Figure 4) is split into two parts: In the first step, the object is used as the input data of the classification process, and the result is a (possible empty) set of relationship types. The object might have relations of the given types to other objects. In the second step, the classification system draws each of the objects from the meta-data-repository (in the figure shown as object 2), and feeds the combination of the two

[1] It would also be possible to give a confidence value indicating how likely it is that the two objects are in the given relationship.

objects into all the classifiers for the relationship types from step one (scenario 1). Since the new object has to be checked against all other objects, the number of classifications to compute is proportional to the number of objects already in the meta data repository. So the difference is that the new object is not checked for all relationship types with all other objects, but also for a reduced set of relationship types which look promising.

4 Transforming the problem into a text classification problem

The main idea of our approach is to assume that database objects are similar to text objects, as the example objects in Figure 1 show. Since the field of automatic text classification is well researched, we decided to adapt a system for automatic text classification for the database classification problem. wizAI uses MIC, a text classification system developed at University Koblenz AI Research Group, which is especially suited for the treatment of structured texts [Beuster, 2001].

Automatic classification methods take vector representations of the objects to be classified as inputs. In Section Preliminaries, an object was defined as a vector with its attributes as the vector elements. It would be possible to use this vector directly as the input data for the classification algorithms, but this representation is not well suited for our classification task. Instead, we used a well known method from text classification: The descriptions of database objects are treated as texts. The first object from Figure 1 is represented as:

> "43 Repository Root Repository Root 4 1900-01-02 3000-01-01 PDREP 2001-03-04 BREX_V5 2001-07-23 PD PD1"

The second object is represented as:

> "1213 Mandant Mandant 33 1900-01-01 3000-01-01 OAI8PV 2001-11-22"

For the classification tasks in scenario one and the second step in scenario two, the input data for the classifiers are combinations of two objects. In these cases, the textual representations of the two objects are concatenated. So the input "text" for the combination of the first two objects is:

> "43 Repository Root Repository Root 4 1900-01-02 3000-01-01 PDREP 2001-03-04 BREX_V5 2001-07-23 PD PD1 1213 Mandant Mandant 33 1900-01-01 3000-01-01 OAI8PV 2001-11-22"

In general, the textual representation of an object is created by concatenating the attribute strings of the object where each attribute string is separated by a whitespace-character.

In order to classify a text, it has to be transformed into a vector representation. We used the relative frequency of words in a text as the input feature vector. For each word of the vocabulary of the training data set, the relative frequency of the word in the document is calculated. This is the number of appearances of the word in the text, divided by the total number of words in the text. This is a common method for text representation, described e.g. in [Mitchell, 1997, page 183]. Since we need fixed vector sizes for some of the classifiers, only the n most informative words for the classification task are used in the vector representation of the text.

5 Classifiers

For text classification, two functions are needed: The first function transfers a text document into an input feature representation. This has been described in the previous section. The second function calculates a classification from this feature representation. Within the University Koblenz-Landau AI Research Group, MIC, a text classification system has been implemented [Beuster, 2001]. MIC supports a variety of classification methods. For this project, we have used three widely used standard classification methods: Naive Bayes Classifier, Decision Tree and Neural Network.

Naive Bayes Classification is the most simple form of a bayesian classifier. Bayesian classifiers use the probabilities of the input vector elements for classification. The assumption of the *naive bayes classifier* is that the input vector elements are independent of each other. Naive bayes classifier are one of the oldest text classification methods.[Maron, 1961] Because of their simplicity, they became kind of a reference method for estimating the difficulty of a classification problem.

Decision trees use a tree structure in which each node is labeled with a decision, and each link with a potential answer. Leaf nodes are labeled with categories. Starting from the top most node, a decision tree is descended until a leaf node is reached. The label of the leaf node represents the result of the classification. For decision tree construction, we used the standard ID3-algorithm[Quinlan, 1979] which always splits on the most informative attribute.

One reason why we used a decision tree classifier is that in difference to the other classifiers presented in this paper, the classification process of a decision tree is easy to comprehend for a human. This allows us an analysis of the characteristics used by the classifier to come up with a classification.

Artificial neural networks mimic the behavior of natural neural networks as found in humans and animals by using large number of interconnected simple computational units which can be trained. Neural networks have been proven as a robust methods for all kinds of classification problems. There is a large number of different neural network architectures. For our classification task, we used standard fully connected feed-forward neural network with one hidden layer. Each input node is associated with an input vector element, and each output node is associated with a category. Standard backpropagation is used for training [Rumelhart, Hinton, & Williams, 1986].

6 Input data selection

Two data sets were provided for classification. The first data set contained 3579 database objects, 79 different relationship types, and 18605 relationships between database object. The second, considerably smaller data set, contained 748 database objects, 92 different relationship types, and 1080 relationships between objects. Since we used separate classifiers for each of the relationship types , we had on average 235 instances of a relationships between two objects for the first data set, and 12 instances of relationships between two objects for the second data set.

Since there were on average 235 relationship instances (data set one) resp. 12 relationship instances (data set two) for each relationship type, we had $\frac{235}{3579} \approx 6.5\%$ resp. $\frac{12}{748} \approx 1.6\%$ positive examples in the data sets. This got even worse for scenario 1, where two database objects are given and the classification task is to tell if these two objects are in the given relationship. In this scenario the input data is the combination of the two objects. Since each object can be combined with every other object, for n objects we had $\frac{n \cdot (n-1)}{2}$ combined objects. For the first data set, these are 6402831 combined objects. Thereby, the number of positive examples was reduced drastically. We had only $\frac{235}{6402831} \approx 0.004\%$ positive examples. For the second scenario, we had $\frac{748 \cdot (748-1)}{2} = 279378$ combined objects. This results in a ratio of 0.004% positive examples.

The drastic disproportion between positive and negative examples did not allow to use automatic classification algorithms successfully on these data sets. We used two methods to alleviate the disproportions in the data sets:

Only objects of the same type: One attribute of the objects is their *type*. A constraint of the meta data repository is that only objects of the same type can be in a given relation. There are 67 object types in the first data set, and 74 object types in the second data set.

Restricted number of negative examples: Since the disproportion between positive and negative examples is still too large, we took a drastic approach: The number of negative examples used for the training of classifiers were reduced to fixed numbers. In the following section, we present results for runs where the number of negative examples was limited to 500, 2000, and 5000.

We always used two data sets, one for training and one for testing. The positive examples were distributed equally among these two data sets. After this, negative examples were added until the desired size of the data sets was reached. Negative examples were selected as shown in Table 1.

Size restrictions for negative example sets have been implemented on three levels: *S1* max. 500 negative examples, *S2* max. 2000 negative examples and *S3* max. 5000 negative examples.

7 Results

For each classification algorithm, scenario, type of object selection, and example size, the quality of the classifier was calculated. Since we assume independence between relationship types,

scenario 1	
N3	Random object pairs which are not in the set of positive examples. This is the set of all object pairs, minus the positive examples.
N2	Object pairs who are in a relation, but not in the relation the classifier is trained on. *N2* is a subset of *N1*. Only those object pairs are used as negative examples, which share some other relationship than the relationship to be learned.
N1	Only object pairs of the same object types as the objects in the positive examples *N1* is a subset of *N2*. It contains only those elements of *N2* whose elements (object-pairs) are of the same type as the object-pairs in the relationship we want to learn.
scenario 2	
N3	Random objects which are not in the set of positive examples.
N2	Objects who are in relation, but not in the relation the classifier is trained on.
N1	Objects of the same type as the objects in the positive examples.
$N1 \subseteq N2 \subseteq N3$	

Table 1: Negative example test classes

these calculations were done independently for each relationship type. The data set was split randomly into two halves of roughly the same size. One half of the data set was used to train the classifier, and the other half was used to evaluate the quality of the learned classifier. Results were accumulated over all relationship types. Tables 3 to 8 show the classification results. The meaning of the table column headings are described in Table 2. *Recall*, the percentage of objects classified, and *precision*, the percentage of objects classified correctly, are the common figures to give the quality of a classifier. In these tables, the recall-rate is not given explicitly, because it is always 100%, i.e. all objects from the test data set are classified.

Some test constellations were not applicable, because for some class of examples the data set

test	Test scenario as previously described.
e	Total number of elements (objects or object-pairs, depending on the test scenario) in the testing data set.
p%	Percentage of positive examples in the testing set.
c	Number of examples classified correctly.
cp%	Percentage of positive examples classified correctly.
prec	Percentage of examples classified correctly.
fp	Number of negative examples which have been classified wrongly as positive.
fn	Number of positive examples which have been classified wrongly as negative.

Table 2: result table legend

test	e	p %	c	cp %	prec	fp	fn
S1,N2	8997	6.19	8974	97.49	99.74	9	14
S2,N2	18287	3	18013	72.68	98.5	124	150
S3,N2	18090	2.9	17814	79.81	98.47	170	106

Table 3: Decision Tree results scenario 1

test	e	p %	c	cp %	prec	fp	fn
S1,N1	2193	30.1	2127	95.45	96.99	36	30
S1,N2	9024	7.73	8873	81.81	98.33	24	127
S2,N1	2254	29.77	2165	93.89	96.05	48	41
S2,N2	10974	6.54	10787	76.74	98.3	20	167
S3,N1	2218	29.08	2153	96.43	97.07	42	23
S3,N2	11026	6.09	10874	80.06	98.62	18	134

Table 4: Decision Tree results scenario 2

is empty. In some cases for any relationship type, there are no objects O_1 and O_2 such that O_1 and O_2 are of the same object type as the objects in the set of positive examples, and in some relationship to each other, but not in the relationship that should be learned (e.g. test scenario 1 and setting [S1,N1]).

8 Conclusion

Existing methods of text classification proved to be surprisingly robust for application to a very different task. The data used for classification was not well suited for three reasons: It was not actual text, but descriptions of data base objects. These descriptions were very short. There was a blatant disproportion between positive and negative examples.

The two techniques Decision Tree and Naive Bayes showed overall high values for precision and very good values for the recognition of positive examples (cp%). It should be noted that a good precision value alone does not prioritize one learning technique over the other. In the scenarios we given in this paper, potential relationships are presented to a human user who decides whether the system's suggestions is correct (it is an relationship indeed) or not (there is no relationship). In these scenarios, false negative (existing relationships that are not detected) are a lot worse than false positives (non-existing relationships that are detected wrongly). False negatives literally get lost: They are not presented to the user, and therefore can not be added manually. On the other hand, false positives are presented to the user, so she can remove them.

Based on these considerations, we can conclude that the optimization by restricting the negative example sets according to N1 yields the overall best results. The small decrease in precision when using N1 and N2 are not significant, and acceptable because of the improved recognition of positive examples.

test	e	p %	c	cp %	prec	fp	fn
S1,N2	8983	6.13	8734	95.64	97.23	225	24
S2,N2	18346	2.94	17779	96.11	96.91	546	21
S3,N3	18249	3.06	17610	94.44	95.5	608	31

Table 5: Naive Bayes results scenario 1

test	e	p %	c	cp %	prec	fp	fn
S1,N1	2213	30.37	2135	94.94	96.48	44	34
S1,N2	9011	7.11	8825	92.04	97.94	135	51
S2,N1	2191	28.75	2117	96.51	96.62	52	22
S2,N2	10901	5.68	10709	91.92	98.24	142	50
S3,N1	2213	29.46	2142	96.17	96.79	46	25
S3,N2	11027	6.01	10831	91.4	98.22	139	57

Table 6: Naive Bayes results scenario 2

All of the three learning techniques yield good results for scenario 2 and negative example class N1 (precision: 96.37%–97.07%; cp%: 96.17%–96.43%). For scenario 1, the neural network classifier gives slightly worse *correct positive* values (cp%: 61.44%) than Naive Bayes and Decision Tree. For all three techniques, the precision values ranges from 96% to $98,47\%$.

Using the classification system presented drastically reduced the necessity for human intervention. Still, the system is not completely autonomous. Although the error rate is fairly low, it is still advisable to let a human review the suggestions of the system. Beside reducing the amount of human intervention, the human supervisor also needs less expert knowledge than a human classifying database objects unaided. When using a decision tree algorithm for classification, the researcher additionally can get insights into the structure of the data, and may develop meta-knowledge about what kinds of objects are in relationships.

So far, we investigated the problem with three standard machine learning techniques. We plan to use other classification algorithms like SVMs, which perform excellent in a number of fields. The approach presented in this paper can be improved further by changing the representation of the objects. So far, we treat them as plain text, ignoring all structural information. We expect improved behavior from better representations which preserve the structural information of the objects.

test	e	p %	c	cp %	prec	fp	fn
S1,N2	9005	5.95	8968	94.78	99.59	9	28
S2,N2	18283	3.08	17961	58.97	98.24	91	231
S3,N2	18251	3.11	17924	61.44	98.21	108	219

Table 7: Neural Network results scenario 1

test	e	p %	c	cp %	prec	fp	fn
S1,N1	2201	29.62	2118	95.71	96.23	55	28
S1,N2	9016	7.55	8878	85.32	98.47	38	100
S2,N1	2271	29.94	2193	96.18	96.57	52	26
S2,N2	11086	5.65	10938	81.79	98.66	34	114
S3,N1	2232	29.08	2151	96.3	96.37	57	24
S3,N2	10976	65.01	10827	81.67	98.64	28	121

Table 8: Neural Network results scenario 2

Acknowledgments

G. Beuster has been partially supported by a DAAD postgraduate grant in the framework of the common special academia program III of the federal states and the federal government of Germany.

References

[Beuster, 2001] Beuster, G. 2001. MIC — A System for Classification of Structured and Unstructured Texts. Master's thesis, University Koblenz. http://www/~gb/papers/thesis_mic/mic.pdf.

[Bouguettaya, Benatallah, & Elmagarmid, 1998] Bouguettaya, A.; Benatallah, B.; and Elmagarmid, A. K. 1998. *Interconnecting Heterogeneous Information Systems*. Kluwer Academic Publishers.

[Marco, 2000] Marco, D. 2000. *Building and Managing the Meta Data Repository: A Full Lifecycle Guide*. John Wiley & Sons.

[Maron, 1961] Maron, M. 1961. Automatic indexing: An experimental inquiry. *Journal of the ACM (JACM)* 8:404–417.

[Mitchell, 1997] Mitchell, T. M. 1997. *Machine Learning*. McGraw-Hill International Editions.

[Quinlan, 1979] Quinlan, J. 1979. *Expert systems in the Micro-Electronic Age*. Edinburgh: Edinburgh University Press. chapter Discovering rules by induction from a large collection of examples, 168–201.

[Rumelhart, Hinton, & Williams, 1986] Rumelhart, D. D.; Hinton, G. E.; and Williams, R. J. 1986. Learning representations by back-propagating errors. *Nature* 533–536.

[Sheth & J.Larson, 1990] Sheth, A., and J.Larson. 1990. Federated database systems for managing distributed, heterogeneous, and autonomous databases.

Available Research Reports (since 1998):

2003

10/2003 *Gerd Beuster, Ulrich Furbach, Margret Groß-Hardt, Bernd Thomas.* Automatic Classification for the Identification of Relationships in a Metadata Repository.

9/2003 *Nicholas Kushmerick, Bernd Thomas.* Adaptive information extraction: Core technologies for information agents.

8/2003 *Bernd Thomas.* Bottom-Up Learning of Logic Programs for Information Extraction from Hypertext Documents.

7/2003 *Ulrich Furbach.* AI - A Multiple Book Review.

6/2003 *Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt.* Living Books.

5/2003 *Oliver Obst.* Using Model-Based Diagnosis to Build Hypotheses about Spatial Environments.

4/2003 *Daniel Lohmann, Jürgen Ebert.* A Generalization of the Hyperspace Approach Using Meta-Models.

3/2003 *Marco Kögler, Oliver Obst.* Simulation League: The Next Generation.

2/2003 *Peter Baumgartner, Margret Groß-Hardt, Alex Sinner.* Living Book – Deduction, Slicing and Interaction.

1/2003 *Peter Baumgartner, Cesare Tinelli.* The Model Evolution Calculus.

2002

12/2002 *Kurt Lautenbach.* Logical Reasoning and Petri Nets.

11/2002 *Margret Groß-Hardt.* Processing of Concept Based Queries for XML Data.

10/2002 *Hanno Binder, Jérôme Diebold, Tobias Feldmann, Andreas Kern, David Polock, Dennis Reif, Stephan Schmidt, Frank Schmitt, Dieter Zöbel.* Fahrassistenzsystem zur Unterstützung beim Rückwärtsfahren mit einachsigen Gespannen.

9/2002 *Jürgen Ebert, Bernt Kullbach, Franz Lehner.* 4. Workshop Software Reengineering (Bad Honnef, 29./30. April 2002).

8/2002 *Richard C. Holt, Andreas Winter, Jingwei Wu.* Towards a Common Query Language for Reverse Engineering.

7/2002 *Jürgen Ebert, Bernt Kullbach, Volker Riediger, Andreas Winter.* GUPRO – Generic Understanding of Programs, An Overview.

6/2002 *Margret Groß-Hardt.* Concept based querying of semistructured data.

5/2002 *Anna Simon, Marianne Valerius.* User Requirements – Lessons Learned from a Computer Science Course.

4/2002 *Frieder Stolzenburg, Oliver Obst, Jan Murray.* Qualitative Velocity and Ball Interception.

3/2002 *Peter Baumgartner.* A First-Order Logic Davis-Putnam-Logemann-Loveland Procedure.

2/2002 *Peter Baumgartner, Ulrich Furbach.* Automated Deduction Techniques for the Management of Personalized Documents.

1/2002 *Jürgen Ebert, Bernt Kullbach, Franz Lehner.* 3. Workshop Software Reengineering (Bad Honnef, 10./11. Mai 2001).

2001

13/2001 *Annette Pook.* Schlussbericht “FUN - Funkunterrichtsnetzwerk”.

12/2001 *Toshiaki Arai, Frieder Stolzenburg.* Multiagent Systems Specification by UML Statecharts Aiming at Intelligent Manufacturing.

11/2001 *Kurt Lautenbach.* Reproducibility of the Empty Marking.

10/2001 *Jan Murray.* Specifying Agents with UML in Robotic Soccer.

9/2001 *Andreas Winter.* Exchanging Graphs with GXL.

8/2001 *Marianne Valerius, Anna Simon.* Slicing Book Technology — eine neue Technik für eine neue Lehre?.

7/2001 *Bernt Kullbach, Volker Riediger.* Folding: An Approach to Enable Program Understanding of Preprocessed Languages.

6/2001 *Frieder Stolzenburg.* From the Specification of Multiagent Systems by Statecharts to their Formal Analysis by Model Checking.

5/2001 *Oliver Obst.* Specifying Rational Agents with Statecharts and Utility Functions.

4/2001 *Torsten Gipp, Jürgen Ebert.* Conceptual Modelling and Web Site Generation using Graph Technology.

3/2001 *Carlos I. Chesñevar, Jürgen Dix, Frieder Stolzenburg, Guillermo R. Simari.* Relating Defeasible and Normal Logic Programming through Transformation Properties.

2/2001 *Carola Lange, Harry M. Sneed, Andreas Winter.* Applying GUPRO to GEOS – A Case Study.

1/2001 *Pascal von Hutten, Stephan Philippi.* Modelling a concurrent ray-tracing algorithm using object-oriented Petri-Nets.

2000

8/2000 *Jürgen Ebert, Bernt Kullbach, Franz Lehner (Hrsg.).* 2. Workshop Software Reengineering (Bad Honnef, 11./12. Mai 2000).

7/2000 *Stephan Philippi.* AWPN 2000 - 7. Workshop Algorithmen und Werkzeuge für Petrinetze, Koblenz, 02.-03. Oktober 2000 .

6/2000 *Jan Murray, Oliver Obst, Frieder Stolzenburg.* Towards a Logical Approach for Soccer Agents Engineering.

5/2000 *Peter Baumgartner, Hantao Zhang (Eds.).* FTP 2000 – Third International Workshop on First-Order Theorem Proving, St Andrews, Scotland, July 2000.

4/2000 *Frieder Stolzenburg, Alejandro J. García, Carlos I. Chesñevar, Guillermo R. Simari.* Introducing Generalized Specificity in Logic Programming.

3/2000 *Ingar Uhe, Manfred Rosendahl.* Specification of Symbols and Implementation of Their Constraints in JKogge.

2/2000 *Peter Baumgartner, Fabio Massacci.* The Taming of the (X)OR.

1/2000 *Richard C. Holt, Andreas Winter, Andy Schürr.* GXL: Towards a Standard Exchange Format.

1999

10/99 *Jürgen Ebert, Luuk Groenewegen, Roger Süttenbach.* A Formalization of SOCCA.

9/99 *Hassan Diab, Ulrich Furbach, Hassan Tabbara.* On the Use of Fuzzy Techniques in Cache Memory Managament.

8/99 *Jens Woch, Friedbert Widmann.* Implementation of a Schema-TAG-Parser.

7/99 *Jürgen Ebert, and Bernt Kullbach, Franz Lehner (Hrsg.).* Workshop Software-Reengineering (Bad Honnef, 27./28. Mai 1999).

6/99 *Peter Baumgartner, Michael Kühn.* Abductive Coreference by Model Construction.

5/99 *Jürgen Ebert, Bernt Kullbach, Andreas Winter.* GraX – An Interchange Format for Reengineering Tools.

4/99 *Frieder Stolzenburg, Oliver Obst, Jan Murray, Björn Bremer.* Spatial Agents Implemented in a Logical Expressible Language.

3/99 *Kurt Lautenbach, Carlo Simon.* Erweiterte Zeitstempelnetze zur Modellierung hybrider Systeme.

2/99 *Frieder Stolzenburg.* Loop-Detection in Hyper-Tableaux by Powerful Model Generation.

1/99 *Peter Baumgartner, J.D. Horton, Bruce Spencer.* Merge Path Improvements for Minimal Model Hyper Tableaux.

1998

24/98 *Jürgen Ebert, Roger Süttenbach, Ingar Uhe.* Meta-CASE Worldwide.

23/98 *Peter Baumgartner, Norbert Eisinger, Ulrich Furbach.* A Confluent Connection Calculus.

22/98 *Bernt Kullbach, Andreas Winter.* Querying as an Enabling Technology in Software Reengineering.

21/98 *Jürgen Dix, V.S. Subrahmanian, George Pick.* Meta-Agent Programs.

20/98 *Jürgen Dix, Ulrich Furbach, Ilkka Niemelä .* Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations.

19/98 *Jürgen Dix, Steffen Hölldobler.* Inference Mechanisms in Knowledge-Based Systems: Theory and Applications (Proceedings of WS at KI ’98).

18/98 *Jose Arrazola, Jürgen Dix, Mauricio Osorio, Claudia Zepeda.* Well-behaved semantics for Logic Programming.

17/98 *Stefan Brass, Jürgen Dix, Teodor C. Przymusinski.* Super Logic Programs.

16/98 *Jürgen Dix.* The Logic Programming Paradigm.

15/98 *Stefan Brass, Jürgen Dix, Burkhard Freitag, Ulrich Zukowski.* Transformation-Based Bottom-Up Computation of the Well-Founded Model.

14/98 *Manfred Kamp.* GReQL – Eine Anfragesprache für das GUPRO–Repository – Sprachbeschreibung (Version 1.2).