

Evaluation of Text and Data Mining Systems for Experience Bases

Michaela Steffan

September 2003

Author: Michaela Steffan
Supervisor: Jörg Rech
Prof. Rombach
Organization: Fraunhofer
Institut
Experimentelles
Software Engineering
Sauerwiesen 6
67661 Kaiserslautern

Contents

1	Introduction	1
2	Text-Mining-Systems	3
2.1	YALE: Yet Another Learning Environment	3
2.1.1	Origin	3
2.1.2	Requirements	3
2.1.3	Function	4
2.1.4	Conclusion	5
2.2	Weka	6
2.2.1	Origin	6
2.2.2	Requirements	6
2.2.3	Function	6
2.2.4	Conclusion	7
2.3	KEA	8
2.3.1	Origin	8
2.3.2	Requirements	8
2.3.3	Function	8
2.3.4	Conclusion	9
3	Experience-Management-System	11
3.1	Origin	11
3.2	Requirements	11
3.3	Function	11
3.3.1	List of Databases	12
3.3.2	Database “Project”	14
3.4	Conclusion	14
3.4.1	Advantages	15
3.4.2	Disadvantages	15
4	Developed System	17
4.1	Yale	17
4.1.1	Input	17

4.1.2	Function	18
4.1.3	Output	19
4.1.4	Results and Interpretation	19
4.2	Weka	22
4.2.1	Weka Clusterer	22
4.2.2	Weka Learner / Classifier	24
4.2.3	Operators with their Output Model	26
4.3	KEA	27
4.3.1	First Experiment	27
4.3.2	Second Experiment	32
5	Summary	35
5.1	Conclusion	35
5.2	Future Work	36
A	XML-Files	39
A.1	Clusterer experiment file	39
A.2	Classifier experiment file	40
A.3	Attribute description file	40
B	Result Writer	43
B.1	Clusterers	43
B.1.1	EM with project data	43
B.1.2	Cobweb with project data	45
	Bibliography	51

Chapter 1

Introduction

The aim of this project thesis is the integration of a text mining system in an experience factory. This system should extract information from given experiences stored in a database. The knowledge contained in this database has to be edited and reused in new processes. It should be revised and provided in a compact manner to benefit from this intellectual capital.

Therefore, it was necessary to gather information about text mining systems and their functionality. The first investigated system is “Yale: Yet another learning environment” from the University of Dortmund. By analyzing this system it became clear that it can not handle continuous text like it is given in many instances of the experience database. This tool is an environment for learning algorithms which normally can only handle data in the form of numbers or single words.

For the fact that Yale is more an environment for external learning tools than a complete data mining system itself, it was necessary to gather information about “Weka” from the University of Waikato. This system provides many learning algorithms and is an important part in the data mining area. To keep track of the task, the investigation was restricted to learning algorithms from the Weka system.

As mentioned already, systems like Yale or Weka are only designed for acting with structured data. So the next idea was to find a tool which is able to handle continuous text. Further research lead to “KEA” from the University of Waikato. This tool can extract key phrases out of continuous text.

So it was possible to extract knowledge from a given experience factory, which consists of a database with the experience the “Fraunhofer Institut Experimentelles Software Engineering” (IESE) has gained. In the following this system for editing and processing the dataset is developed and described. At the end the result and findings are summarized.

The current text mining systems Yale, Weka and KEA are described extensively in chapter 2. The origin, demands and function of each is exposed. An interpretation of the usability, advantages and disadvantages – if they are conspicuous – is given. Chapter 3 is about the experience management system with attention to the use for the text mining systems. A list of the databases with their used or usable columns is given. Some problems

the text mining systems have with the database values are noted as well. The developed system is described in chapter 4 with its special input from the experience factory. One example for an exclusive Yale experiment is given, and two for Yale associated with Weka. Also for KEA two experiments are described which are working with the current experience management system. Chapter 5 lists the result and gives an interpretation of the output created by the different tools. This interpretation can only be superficial. The utilization of the knowledge extracted from the experience factory is part of economy management institutes, for example the marketing branch. A summary and conclusion is posted in chapter 6. It gives an overview of the results with respect to the possibilities of text mining systems. As they cannot handle continuous text but only single words they should be more considered as data mining systems. Calling them text mining systems may create wrong expectations. Some ideas for future work are listed, too, especially for examining the relation between different databases.

Chapter 2

Text-Mining-Systems

2.1 YALE: Yet Another Learning Environment

2.1.1 Origin

Yale is a learning environment for data mining operator chains from the University of Dortmund, Department of Computer Science, Chair of Artificial Intelligence. It is free software and can be downloaded from the world wide web.

The YALE 2.0 Obeta version provides a graphical user interface which makes it more comfortable and is therefore used here. The main program – especially the learning algorithms – are already implemented in the YALE 1.0 version.

For more information see the Yale-Tutorial [2].

Version	YALE 1.0 YALE 2.0 Obeta (GUI version)
URL	http://yale.cs.uni-dortmund.de/
Reference	Paper [1] and Tutorial [2] “YALE: Yet Another Learning Environment”

2.1.2 Requirements

Yale needs at least the Java version 1.4. It runs with the operating system *Linux* as well as with *Windows*.

The input data for Yale must be structured and described very strictly. It has to be ordered in columns separated by special tags like spaces or commas. Each column represents one attribute. These attributes are identified by a name and the type of the data like numeric or nominal. The nominal attributes are single words which must be elements of a predefined domain. Therefore, continuous text can not be given to a learning algorithm without extensive modification.

2.1.3 Function

The handling of the Yale GUI version is self-explaining. If starting with a blank experiment the external operator is given and the user can insert or delete the wanted operators with the menu. The first operator always has to be of the class “Experiment”. It is also possible to change the xml-file itself. This could sometimes be necessary if a parameter is not reachable in the menu because of a bug.

Yale was designed for the preprocessing of data and the executing of several learning steps one after another. These steps are implemented as *Operators* which themselves have an assigned, mandatory input and output. See the following list for the description of some used or useful operators. The experiment program is saved as an xml-file (example: appendices A.1 and A.2).

Operators

- Input/output
 - ExampleSource: Generates the dataset by using an attribute description file in xml (example: appendix A.3). Mandatory for most operators.
 - DatabaseExampleSource: Generates data with SQL queries.
 - ResultWriter: Writes the current result to the named result file (example: appendices B.1.1 and B.1.2).
 - AttributeSetWriter and ExampleSetWriter: Writes the current attributes or data in the named file.
- Machine Learning Algorithms
 - DecisionTreeLearner: Simple decision tree learner delivered by Yale which can handle nominal and numeric attributes. The algorithm of this learner is based on Quinlan’s ID3 decision tree induction algorithm which is very common in the data mining scene and therefore not extended upon here.
 - WekaLearner: see section 2.2.
- WekaClusterer: see section 2.2.
- Validation and performance evaluation: Tests the performance of a given or produced model.
- Feature selection and generation: Generates some new or the best set of features (attributes) for a given dataset. It is also possible to complete automatically empty data with the operator “Missing Value Replenishment”.

The datasets are written in ASCII- respectively text-files with the data represented in columns separated by definable characters. All spaces are identified as separators, comas

respectively semicolons are the default values. It is also possible to define characters which should be ignored.

The value of the columns are the attributes of the data and are described in xml-files (example: appendix A.3). The name of the example source or dataset which is described by the current attribute description file is named in this file as the parameter “default source”. See the following list of attribute value types which are used in the developed system described in chapter 4.

Attribute value types

- Nominal: predefined words, the almost only meaningful possibility for the IESE database entries.
- Integer: the second possibility for IESE database entries. Numeric can be chosen instead.

2.1.4 Conclusion

Advantages

The handling of the GUI version is easy and self explaining. It is not necessary to adapt it for the experience factory as it first was a part of the conceptual formulation.

The realization of several steps one after another is possible. A sequence of operators could process the input data in one pass without latching great datasets.

Disadvantages

There are some bugs, especially in the GUI version, e.g. a wrong parameter name for the weka clusterer, it must be “weka_clusterer_name” instead of “weka_clusterer”.

Often, no expressive error messages are given. Finding out if something is wrong with the input is often only possible by debugging the source code. The error messages are a bit more understandable in the GUI version.

2.2 Weka

2.2.1 Origin

Weka is implemented in Java and delivers many programs for machine learning. It comes from the Department of Computer Science, University of Waikato, New Zealand and is free software that can be downloaded from the world wide web.

For more information see the “Data Mining” book [3].

Version WEKA 3.2.3

URL <http://www.cs.waikato.ac.nz/ml/weka/>

Reference Book “Data Mining” [3]

2.2.2 Requirements

The Weka package is delivered with the Yale GUI version, so it is not necessary to install Weka itself.

The Weka algorithms are only used in connection with the Yale environment. Yale automatically transforms its input dataset to the form Weka needs. The input of the Weka algorithms has to be in the ARFF-format which is explained in detail in the book “Data Mining” [3]. Note that the attribute description file of Yale needs a label value if you want to use a Weka algorithm for processing the data. Omitting the label causes an error message.

2.2.3 Function

The first experiment uses the Weka clusterers Cobweb and EM. Clusterers are algorithms which try to cluster or classify the input dataset. They assign a class to every instance of the dataset with different methods. For the exact calculations the data mining book [3] can be consulted.

Clusterers

- Cobweb: An instance based clusterer which sorts every dataset respective instance into a tree (example: appendix B.1.2). This algorithm takes every single instance and tries to sort it into the current tree. If the actual instance fits into a leaf node it will count for the class this leaf node represents. Otherwise a suitable branch is split and it becomes a new leaf node. The merging of already existing branches is also possible.
- EM: A probability based clusterer (example: appendix B.1.1) which name is deduced from Expectation and Maximization. The expectancy value for each dataset inherent

to a class is calculated. In parallel the algorithm tries to maximize the expected probability of a dataset of belonging to a class respective cluster by rearranging the classes.

Weka provides a great number of learning algorithms, not all of them can be examined within the scope of this project thesis. Only the Learner “J48” and “J48.Part” with their results are described here. Learning algorithms are called “Classifiers” in Weka. They normally produce a model with respect to a given dataset. This model can also be used for examining a new dataset.

Classifiers / Learners

- J48: A decision tree C4.5 based classifier which delivers a tree. It uses the ID3 algorithm like the Yale internal decision tree learner.

Example: section 4.2 weka.classifiers.j48.J48.

- J48 part: A decision tree C4.5 based classifier which delivers classification rules. This rules are derived from the decision tree.

Example: section 4.2 weka.classifiers.j48.PART.

2.2.4 Conclusion

Advantages

A great number of learning algorithms are implemented. The Weka package seems to be leading in the data mining scene.

Most of the Weka learning algorithms are able to deal with nominal attributes. This distinguishes once more the Weka package from other data mining systems.

A GUI version is available, too, and guarantees an user-friendly handling.

Disadvantages

They are not discussed in this topic because Weka is not the main part of it.

2.3 KEA

2.3.1 Origin

KEA is also a product of the Department of Computer Science, University of Waikato, New Zealand. It is implemented by Frank Eibe and is free software. The source can be downloaded from the world wide web.

The program has a tool for extracting key words from a given text namely the *extractor*. Therefore, it uses a model which could be created respectively trained by KEA with the tool *model builder*.

For more information, especially usable parameters, read the “README”-file which is part of the KEA 2.0 distribution.

Version KEA 2.0

URL <http://www.nzdl.org/Kea/>

Reference Paper “KEA: Practical Automatic Keyphrase Extraction” [4]

2.3.2 Requirements

KEA is a set of Java classes and so it runs on the Java Virtual Machine. No demands for the needed Java version is given in the installing instruction. Java 1.4 as it is needed for Yale is sufficient.

The input data for the *extractor* and also the *model builder* has to be in text files with the ending *.txt* and all in the same directory. The text should be as clean as possible i.e. without code tags.

The language of the input should be English. There exists a file “Stopword.java” with words that will never be key phrases (e.g. “yes” and “no”) and are therefore never chosen. The stop words are at the moment only in English.

2.3.3 Function

KEA chooses meta data or key phrases directly from a given text. Firstly it cleans the input by replacing punctuation marks, brackets and numbers with boundaries, removes apostrophes, splits hyphenated words and deletes non-token characters.

Secondly it tries to find a set of candidates for phrases. They have to be of a certain maximum length (variable, usually three words), be no proper names and not begin or end with a *stop word*.

Finally the stem of the candidates are searched with the help from *Lovins stemmer*. This is an external, common and proved program for finding a words stem. This is necessary for not getting several key phrases from the same term.

At least two features or attributes are calculated for each candidate:

- First occurrence of the candidate and
- (TFxIDF), an indicator for the document frequency of the phrase in relation to its rarity in general use.

For the exact calculation of this features see the KEA paper [4].

The result of this calculation is that the earlier a candidate occurs and the more common it is, the less is the probability to be a key phrase. With this data, KEA offers you a requested number of keywords, usually five for each document.

For getting an optimal result, two steps are necessary:

1. Training:

Building a key phrase extraction model by training the system with your own texts and their key phrases. These *training documents* are ASCII text documents and must have the ending “.txt”. Their key phrase documents are also text documents with the same name but with the ending “.key” and every key phrase in a separate line. All these documents have to be in an extra directory. Now build the model with:

```
java KEAModelBuilder -l <training_directory_name> -m <model_name>
```

2. Extraction:

Use this model for extracting key phrases from more documents. If you have similar documents from whom you want to extract key phrases, you can use your model. All these *test documents* have to be also in an extra directory. Now extract key phrases with:

```
java KEAKeyphraseExtractor -l <test_directory_name> -m <model_name>
```

2.3.4 Conclusion

Advantages

KEA is one of the few tools which is able to work with continuous text.

It can be used for classifying text, e.g. papers. These text classifiers perhaps can be used as input for data mining systems.

The key phrase extracting delivers words which characterize the input text by marking it out from other text in the same area.

Disadvantages

There is not enough continuous text material in the experience factory for using KEA in an optimal way. Often the database entries are names of related enterprises or persons which should not be a key phrase for the text.

The calculation of the probability to be a key phrase is not optimal for this topic. KEA uses phrases which are not common in the same area but tries to find the discriminating elements for each document. This is a disadvantage for the used experience management system.

Chapter 3

Experience-Management-System

3.1 Origin

The basis of the experience factory are the databases from the “Fraunhofer Institut Experimentelles Software Engineering (IESE), Kaiserslautern”. In these databases all projects of the IESE are stored with different information entries. Problems that occurred are stored and related to the projects also. The state of the databases is from march 23rd 2003.

3.2 Requirements

The databases were given as pdf files. I converted them to text with the Unix command “pdftotext” and extracted the needed data with shell scripts. Because I did not have the permission to work with the original IESE data I can give no warranty for the results and their interpretation.

Only numeric fields or such with nominal text can be used by the current text mining systems Yale and Weka. The columns which consists of continuous text are worked on with KEA.

Often it was also necessary to edit the data because of empty values. The empty nominal values are filled with the class “unknown”, the empty integer values are filled with “0”.

3.3 Function

The intention of the structure and connections between these databases are not described here. As the databases are given in this form and with these entries it could only be theoretical.

The main part is the database “Project” with a list of projects the Fraunhofer IESE worked on. These projects are referenced in other databases, for example in the database “Problem”.

3.3.1 List of Databases

In the following all of the given databases are listed with only their suitable and not empty respectively not too sparse occupied columns.

The nominal and numeric columns are partially used by Yale and Weka. At the end of each database section the number of usable attributes and the number of entries is given.

The continuous text columns are used by KEA.

Guideline

- Nominal
 - case_mode: entries are confirmed and unconfirmed.
 - category: cat/Best practice experience, cat/Customer experience, etc are redundant to category.
- Numeric
 - projects: connected to database “Project”.
 - iq_processes
 - prog_languages
- Continuous text
 - description

in all: 5 attributes / 94 entries

Improvement-suggestion

- Nominal
 - case_mode (entry is confirmed or empty)
 - status (entry is new or empty)
- Continuous text
 - description

in all: 2 attributes / 19 entries

Observation

- Nominal
 - category: cat/Best practice experience, cat/Customer experience, etc are redundant to category.

- Numeric
 - validity: entry is 1 or empty.
 - projects: connected to database “Project”.
 - iq_processes
- Continuous text
 - description

in all: 4 attributes / 112 entries

Problem

- Nominal
 - category: cat/Best practice experience, cat/Customer experience, etc are redundant to category.
 - priority: entries are new and in progress.
- Numeric
 - projects: connected to database “Project”.
 - iq_processes
 - affected_roles
- Continuous text
 - description

in all: 5 attributes / 153 entries

Project

- Nominal
 - funding
 - project_type: type/Transfer, type/Training, etc are redundant to project_type.
- Numeric
 - duration
 - project_team_size: mostly not occupied.
- Continuous text
 - project_name

- objective
- goals
- contract_reasons and acquisition_reasons.
- comment

in all: 4 attributes / 331 entries

Relation-Case-Project

no suitable columns

3.3.2 Database “Project”

To keep track of the topic, only the database “Project” is examined in the scope of this project thesis. Therefore a more detailed description of its usable data is given. Subsequent to this paragraph, the used columns with their entries are listed.

The ones which are describable through a nominal attribute are itemized with their possible entries as they appear in the database. The empty values are filled with the dummy value “unknown” to avoid problems with data mining algorithms which cannot deal with empty columns. The value “unknown” is not present in the original data.

The second kind of attributes are numeric ones. Here, the empty values are filled with zero, because of the same reason mentioned above.

Columns

- funding
values: Industrial, IESE, root, Public, EU, FhG, State, German and unknown.
- project_type
values: Study, Seminar, Workshop, Training, Education, RD, Transfer, Infrastructure, false, Consulting and unknown.
- duration
values: all integers possible.
- team_size
values: all integers possible.

3.4 Conclusion

The advantages and disadvantages of the given experience factory are only seen in the context of data mining systems. A valuation about other tasks like the usability as an experience management system is not part of the topic here.

3.4.1 Advantages

All numeric fields can be used in the data mining systems Yale and Weka. The columns with limited entries e.g. funding in “Project” can be used as nominal data.

The columns with continuous text can be handled with KEA.

3.4.2 Disadvantages

Most of the database columns can not be used or are occupied too sparsely. There are too many empty fields. No information can be retrieved while it is not clear if the values are empty or just not filled in. The data mining systems Yale and Weka are not able to handle empty columns. A dummy value has to be filled in and may falsify the data.

The database has a big problem of inconsistent entries. Both English and German are used. The information of this should be separated and regarded isolated from each other as the data mining systems are not able to differentiate between both languages. This would be a unnecessarily complex process of data management.

Redundant information appears often, e.g. in Project the columns “type/Transfer = true” and “project-type = Transfer” express the same. An increase of knowledge is not given with these columns, so the memory cell for them is wasted.

The text mining systems are not able to decide about the usability and the content of the data. Their only use is to structure and edit it for the human. All interpretations must be left up to the user.

Chapter 4

Developed System

In this chapter some developed experiments for the different tools are described in detail with their input, function and output.

A possible interpretation of the different results is provided. This can only be seen as a basis for discussions. It should give an impression of what could be possible with the examined tools.

4.1 Yale

The input and function for the Yale data mining tool and Yale in connection with Weka is described in detail in this section. There is no difference in both, as all experiments run under the Yale environment. The result for the Weka learners and classifiers are interpreted in chapter 4.2.

4.1.1 Input

The origin of the data are the IESE databases as described in chapter 3. The original data is edited by several steps as mentioned. Only the database “Project” is examined in detail.

The usable columns are “funding”, “project_type”, “duration” and “team_size”. For the empty nominal value of “funding” and “project_type” the dummy value “unknown” is used. If possible, empty values of “project_type” are reconstructed by the redundant columns “type/Transfer”, etc. The empty numeric data “duration” and “team_size” are filled with zero.

The needed input for Yale alone and Yale in combination with Weka:

1. The Yale program file which describes the experiment as a xml file (example: appendices A.1 and A.2).
2. The attribute description file that is suitable to the dataset. See appendix A.3 for an example (here: dataset of the database “Project”). This is implicitly defined in the program file.

3. An implicit input too, is the dataset as an ASCII text file with the different data in columns separated by spaces. This file is named in the attribute description file.

Important: Some operators can not handle empty values, it is necessary to use a dummy value like “unknown” instead.

Program file	classifier.xml
Attribute description file	project_attrib3.xml implicitly in the program file
Data file	project3.dat implicitly in the attribute description file

4.1.2 Function

The outer operator of a Yale program always has to be of the class “Experiment”. All other operators are chained between the beginning and the end tag of this first one. The experiment operator is already included when a new experiment is started.

In XML:

```
<operator name='Global' class='Experiment'>  
... other operators / operator chain  
</operator>
```

To run the Yale data mining program it is necessary to define the input data. This is done with an operator from the class *ExampleSource*. The operator has the parameter “attributes” with the name of the attribute description file as value.

In XML:

```
<operator name='Input' class='ExampleSource'>  
  <parameter key='attributes'  
    value='<name of the attribute description file>.xml'>  
</operator>
```

It is also possible to write some results in different files. The first operator *ResultWriter* writes the current learning result in a predefined result file every time such an operator occurs in the operator chain (example for a result file: appendices B.1.1 and B.1.2).

The next necessary operator is a learning algorithm. In this section one learner is described in detail:

Decision Tree Learner

The operator *DecisionTreeLearner* is a Yale internal learner based on the ID3 Learner but being able to handle nominal and numeric attributes. This learner delivers a model in the shape of a picture (see figure 4.2 and figure 4.3 for examples). The program is shown as an operator chain (see figure 4.1) or as a xml code (see appendix A.2).

4.1.3 Output

The calculated output is a model for the input data based on the used learning operator. This model is automatically shown after the process has stopped.

It could be helpful for the user's understanding to have this model as an ASCII file for processing. This is possible by inserting an operator *ResultWriter* at the end of the program. Then the calculated model is written down and saved in a text file. The name of this file has to be defined in the outer operator *Experiment* as the result file value.

The model is also written down in a model file which could be used in another Yale program as an input model. The name of this file must also be defined in the outer operator *Experiment*. As being used as an internal Yale file this model is not usable for other processes like examining it through a human. If you want to have the model in an understandable form you have to use the operator *ResultWriter* at the end of the operator chain.

The input and the output of all operators are listed in the Yale Tutorial [2]. The interested reader can get more information from there.

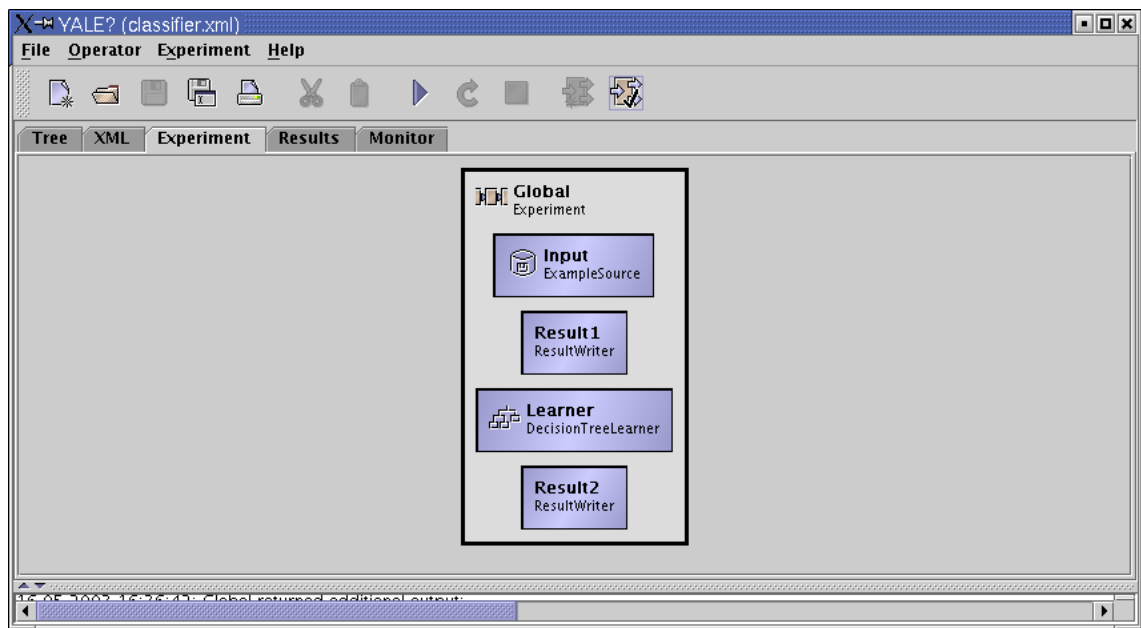


FIGURE 4.1: Yale machine learning algorithm.

4.1.4 Results and Interpretation

The result is a decision tree for the input data in a graphical representation (see figure 4.2 and figure 4.3).

The decision tree learner tries to classify the data. A tree is built with an attribute as root and the others ordered behind in branches. The calculation is based on the ID3

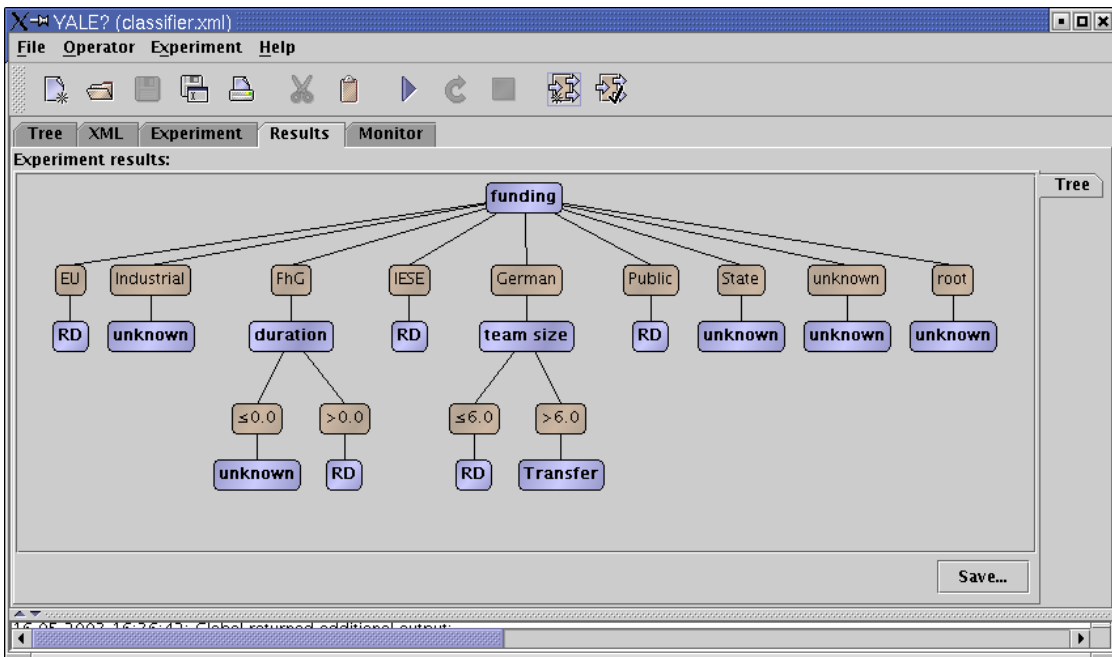


FIGURE 4.2: Yale decision tree with project data and label “project_type”.

learning algorithm. The leaf nodes can be seen as a class inherent to the input data and therefore the label attribute is chosen.

This decision tree could be used for classifying new data, e.g. if the funding value of the new dataset is “German” and the team size is bigger than “6” the probability is high that the value of project_type is “Transfer”.

The relations between the different columns of the database “Project” can be examined with the help of such a tree. In the second step the column “funding” is chosen as label and a new tree is calculated (see figure 4.3). This allows a cross-validation and deeper insights in the data structure.

With this different approaches the user may be able to recognize patterns in the data structure or to re-classify the data for the optimization of data retrieval.

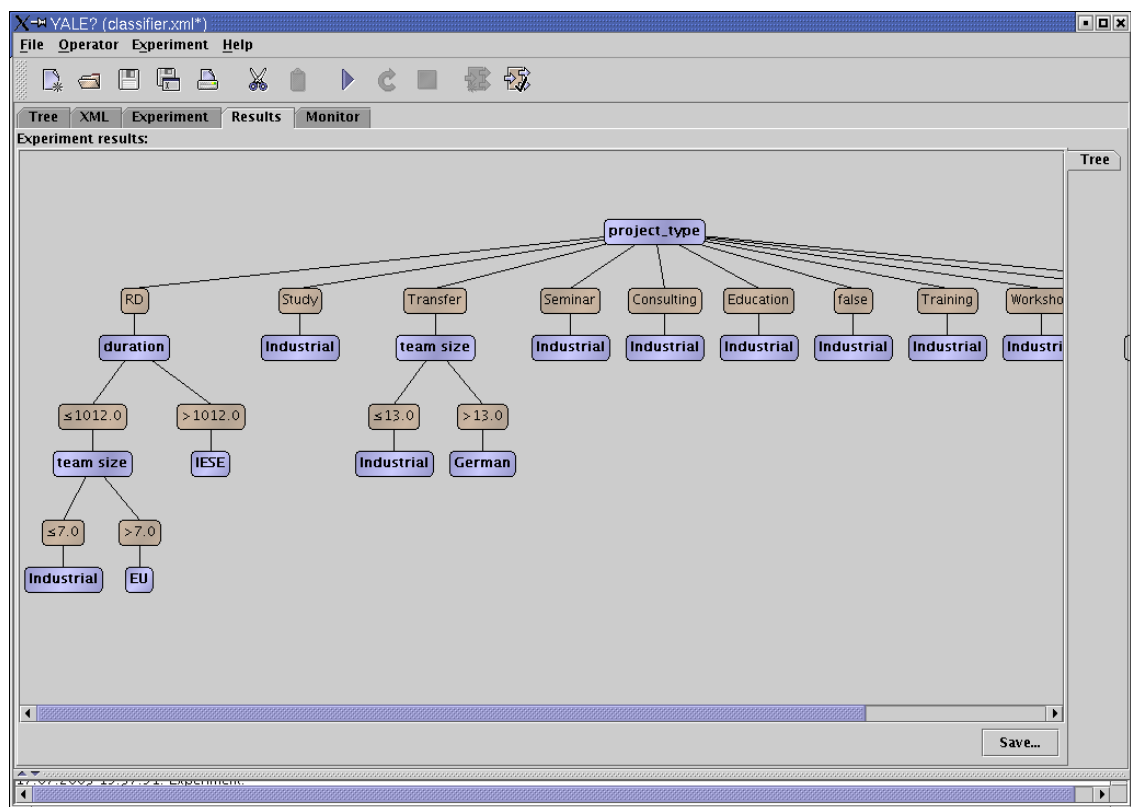


FIGURE 4.3: Yale decision tree with project data and label “funding”.

4.2 Weka

The input, function and output is described in a short form. A more detailed description is given in section 4.1.

Important: Defining a label for using a Weka operator is mandatory.

4.2.1 Weka Clusterer

Input

Program file	clusterer.xml
Attribute description file	project_attrib3.xml implicitly in the program file
Data file	project3.dat implicitly in the attribute description file

Function

The operator *WekaClusterer* is an external cluster algorithm from the Weka package. A more detailed explanation of how the clusterers are working is given in section 2.2.

The program is also shown as an operator chain with one important difference: Instead of the operator *DecisionTreeLearner* as before the operator *WekaClusterer* is called (see figure 4.4 and appendix A.1 for the XML code).

It is also possible to write the current dataset with the operator *ExampleSetWriter* or the current attribute set with the operator *AttributeSetWriter* in a predefined file as shown in figure 4.4. These operators are not necessary for the developed system but mentioned here for future work.

Output

This learner tries to cluster the current data. The input data is clustered respectively classified. A new column is added to every data row with the number of the class this row belongs to.

Results and Interpretation

The clusterer algorithms can be used if nothing is known about the structure and the context of the input data. It clusters the data in sets respectively classes of mated items which gives a first overview of the coherence.

1. Cobweb:

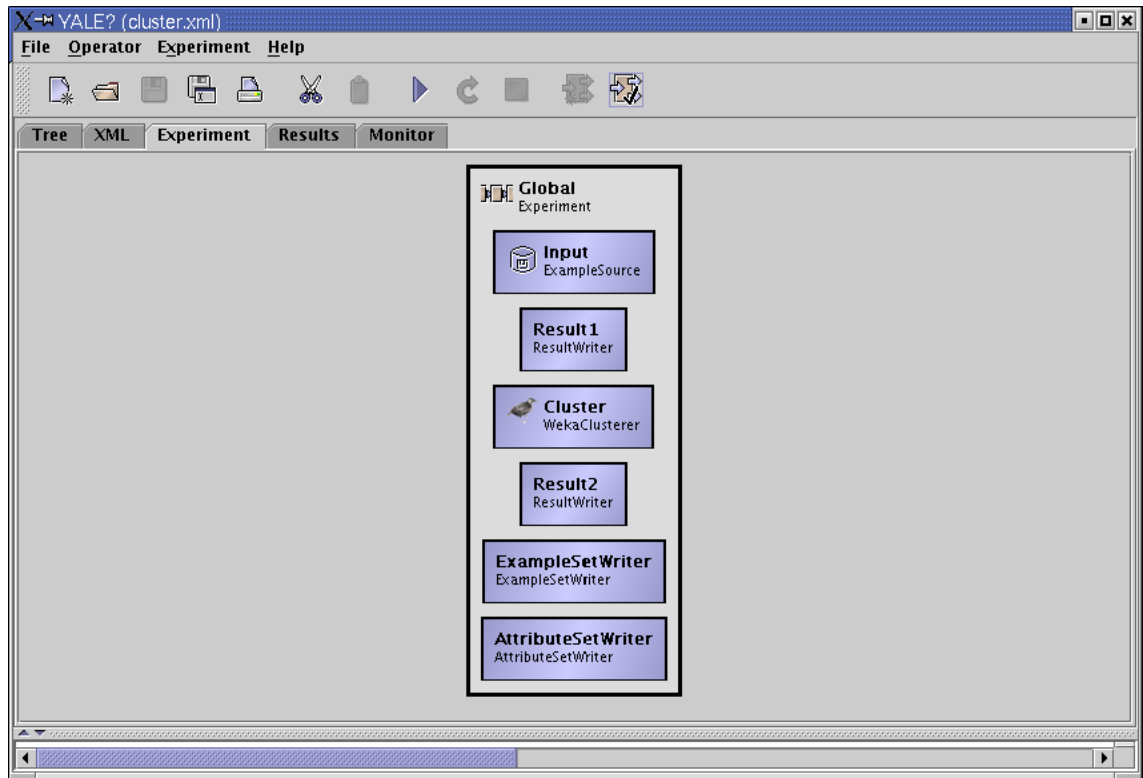


FIGURE 4.4: Weka clusterer algorithm in Yale.

This clusterer tries to sort every new instance in the leafs of a calculated tree. Every leaf node of this tree is a class. See section 2.2 for a more detailed description. The input data of the database “Project” is divided into 89 leafs respectively classes. These are too many for a clear analysis and therefore this algorithm is not useful for the current data set. Some auxiliary results are given in the output of the result writer (see appendix B.1.2).

- *Numberofmerges* = 49: How many branches are merged during the calculation of the tree.
- *Numberofsplits* = 42: How many branches are split.
- *Numberofclusters* = 89: The final number of calculated classes.

2. EM:

The number of clusters the input data is divided into is four. This is a more useful subdivision than the first one. Some auxiliary results are also given in the result writer (see appendix B.1.1). For the nominal attributes a discrete estimation and for the numeric ones a normal distribution is calculated for every class. Some attribute properties of the different clusters are tried to extract. The number of items of each cluster is listed.

- Cluster 0:
team_size > 0, *duration* > 0, counts = 16
- Cluster 1:
miscellaneous, frequently *project_type* = *RD*, counts = 56
- Cluster 2:
miscellaneous, frequently *funding* = *Industrial*, counts = 248
- Cluster 3:
team_size = 0, *duration* > 0, counts = 11

4.2.2 Weka Learner / Classifier

Input

Program file	classifier.xml
Attribute description file	project_attrib3.xml implicitly in the program file
Data file	project3.dat implicitly in the attribute description file

Function

The operator *WekaLearner* is also external from the Weka package. It could be used instead of the operator *DecisionTreeLearner* from section 4.1 and delivers a similar operator chain picture as figure 4.1 and is therefore not shown here. The input data is classified by associating every row of the input data to a class.

Output

A model for the classification of the input data. The model could be in an ASCII tree representation (see the result of the `weka.classifiers.j48.J48`) or a ruled-based representation (see the result of the `weka.classifiers.j48.PART`).

Results and Interpretation

1. `weka.classifiers.j48.J48`

The output of this learning algorithm is a decision tree like the one delivered by the Yale internal decision tree learner. The basis for its calculation is different which could be seen in the differences of the two result trees. The integer values of the attribute “*team_size*” is divided in more intervals and the attribute “*duration*” is not used.

The model representation is in ASCII form and not a graphical tree as in section 4.1:

```
J48 pruned tree
-----
funding = EU: RD (13.0/6.0)
funding = Industrial
|   team size <= 0: unknown (226.0/171.0)
|   team size > 0
|   |   team size <= 5
|   |   |   team size <= 3: RD (9.0/4.0)
|   |   |   team size > 3: Study (2.0/1.0)
|   |   team size > 5: Transfer (4.0/1.0)
funding = FhG: RD (5.0/3.0)
funding = IESE: RD (6.0/2.0)
funding = German: RD (27.0/10.0)
funding = Public: RD (9.0/3.0)
funding = State: unknown (18.0/11.0)
funding = unknown: unknown (11.0)
funding = root: unknown (1.0)
Number of Leaves   :    12
Size of the tree   :    16
```

The root attribute is the column “funding” but the tree has more splits especially in the column “team_size” as the decision tree in section 4.1. The leaf nodes are again the label attribute “project_type”. The label attribute represents the class the dataset belongs to and has to be of the type nominal. Its entries are always chosen as leaf nodes.

For every possible value of the root attribute “funding” a branch is made. The first integer in the brackets after every leaf node gives the number of datasets which have this funding value. The second integer counts the failed classifications, e.g. there are 13 data rows with the funding “EU” but only 7 have indeed the project_type “RD”. The remaining 6 belong to miscellaneous classes and are miss-classified.

2. weka.classifiers.j48.PART

This learning algorithm is also a classifying one. With the difference that now the output model consists of rules and is not a tree. New input data can be labeled or classified by following this rules.

The rule-based representation of the model:

PART decision list

```

-----
funding = German: RD (27.0/10.0)
funding = State: unknown (18.0/11.0)
funding = EU: RD (13.0/6.0)
funding = unknown: unknown (11.0)
funding = Industrial AND
  team size > 0 AND
  team size <= 5 AND
  team size <= 3: RD (9.0/4.0)
funding = Public: RD (9.0/3.0)
funding = IESE: RD (6.0/2.0)
duration > 67 AND
  duration <= 282: Study (3.0)
team size > 2 AND
  team size <= 7: RD (4.0/2.0)
: unknown (231.0/173.0)
Number of Rules :      10

```

This model is not limited to tree representation, not every rule has to start from a root attribute. Additional rules are introduced, e.g. if the duration is between 67 and 282 the value of `project_type` is always “Study” without classification errors.

4.2.3 Operators with their Output Model

1. Decision Tree Learner: picture of a decision tree (example: figure 4.2).
2. Weka Clusterer: a model for clustering a similar dataset.
3. Weka Learner / Classifier: depends on the Weka learner, possibilities:
 - Tree in text form (example: section 4.2 `weka.classifiers.j48.J48`)
 - Rules (example: section 4.2 `weka.classifiers.j48.PART`)

The operator *Weka Clusterer* calculates a second output. This leads to an error message by executing the clusterer in the GUI version which could be ignored. The clusterer result is written in the result file (see appendices B.1.1 and B.1.2 for an example). The clustered dataset is written in the example file and can be stored by using an *ExampleSetWriter*.

4.3 KEA

Some universal and important conditions for KEA are mentioned at first:

1. The input text must be in ASCII. For optimal solution it should be as clean as possible (without code, tags, etc) in files with the ending “.txt”
2. The key phrases for a text must be in a file with the same name, but with the ending “.key”. Every key phrase has to be in a separate line.
3. For extracting key phrases a model is needed which has been built with the KEA model builder.
4. The input and output files or directories are defined with parameters in the program call.

For KEA two experiments are developed. The first experiment is composed of three parts. The second experiment has only two parts.

4.3.1 First Experiment

Part I

The first part deals with the continuous text from the description column of the databases “Guideline”, “Improvement-suggestion”, “Observation” and “Problem”.

- Input:

The descriptions are stored for each database in an extra file:

guideline.description.txt, improv.description.txt, observ.description.txt and problem.description.txt.

These four files are together in the directory “IESE_train”. This working directory is defined with the parameter “-l”.

As the key phrases should be extracted, no key phrase files have been applied before running the experiment.

The model “CSTR_abstracts_model” delivered by KEA is used for the first extracting process. The model is defined with the parameter “-m”.

- Function:

Key phrases from the experience factory text are extracted with the delivered KEA model. These key phrases are stored in the same directory in four different files with the same name as the text files but with the ending “.key”: guideline.description.key, improv.description.key, observ.description.key and problem.description.key.

```
java -l IESE_train -m CSTR_abstracts_model
```

- Output:

Files with five key phrases for each of the four input files stored in the directory “IESE_train”.

Part II

With these text files and their key phrase files an IESE model is built.

- Input:

The directory with the text files and the key phrase files is the input for the KEA model builder. This working directory is defined with the parameter “-l”.

- Function:

A special IESE model with the name “IESE_model” is created. Its name is defined with the parameter “-m”.

```
java KEAModelBuilder -l IESE_train -m IESE_model
```

- Output:

An IESE special model is built.

Part III

After that the experiment deals with the continuous text from the columns Comment, Goals, Name, Objective and Reasons from the database Project.

- Input:

The text of the different columns are stored in extra files:

project.comment.txt, project.goals.txt, project.name.txt, project.objective.txt and project.reasons.txt.

These five files are placed in the directory “IESE_test”. This working directory is defined with the parameter “-l”.

As the key phrases should be extracted, no key phrase files have been applied before running the experiment.

The model “IESE_model” from part II of this experiment is used for the extracting process. The new created model with the parameter “-m’

- Function:

Key phrases from the experience factory text are extracted with the new created IESE model. These key phrases are stored in the same directory in five different files with the same name as the text files but with the ending “.key”:

project.comment.key, project.goals.key, project.name.key, project.objective.key and project.reasons.key.

```
java KEAKeyphraseExtractor -l IESE_test -m IESE_model
```

- Output:

Files with five key phrases for each of the five input files.

Results and Interpretation

Part I: The result is represented in the table 4.1 with the extracted key phrases from the four databases. The second column contains the key phrases as they are stored in the key phrase files i.e. every key phrase in a separate line.

TABLE 4.1: Result First Experiment Part I

database	key phrase
Guideline	project customer plan Check partners
Improvement	IESE Institute courses management consulting
Observation	customer project wanted expected scientific
Problem	project customer person effort planned

The relation and content of the experience factory is not determined within the scope of this project thesis. Therefore a interpretation of the extracted key phrases for the databases “Guideline” and “Improvement-suggestion” is not given here, as it is not clear what information they manage.

The key phrases for the databases “Observation” and “Problem” seem to be significant and are therefore interpreted superficially by myself. The topic that is “wanted” and “expected” should be observed. Problems occur mostly with “planned” “projects” and “customers”.

Part II: For this part the output is a model and can only be evaluated implicitly by using it for key phrase extracting.

Part III: The extracted key phrases of the columns from the database “Project” are presented with the same design as above in the table 4.2.

The problem of the language inconsistency is visible here, the words “bei” and “eines” should be part of a german stopword class and therefore never chosen as a key phrase. KEA gives the possibility to upgrade the stopword file. The words could be integrated in the source code of the java program and recompiled.

Nevertheless some significant key phrases are found and interpreted superficially by myself:

The goals aspired to the “Fraunhofer IESE” projects are often to “keep strategic alliance” with the industry, gain “experience” especially in software “development”.

The extracted key phrases for the column “Reason” leads in the same direction. Having a “foot” in the “door” of a company, practice the “continuation” or “get” a “contact”.

TABLE 4.2: Result First Experiment Part III

column name	key phrase
Comment	manager Project manager Actual effort Partners' ist
Goals	Fraunhofer IESE Keep strategic alliance alliance with Telekom experience development
Project_name	bei eines Software Network Software-Technologie-Transfer
Objective	validation cost compare Investigation improvement
Reasons	door foot Continuation Contact Get

4.3.2 Second Experiment

Part I

- Input:

This experiment only deals with continuous text of the database “Project”. The single projects should be examined. At first twenty projects are chosen randomly. The complete text of these projects are stored in twenty files in the directory “Projekt_train”.

Origin of the texts for the Projekt_train files:

```
# random numbers - Case-Id - page numbers:
# 8 - 2009 - 19/20           # 28 - 2029 - 65/67
# 37 - 2039 - 86/88         # 55 - 2057 - 124/125
# 59 - 2061 - 132/133      # 73 - 2075 - 160/161
# 113 - 2448 - 252/253     # 126 - 2461 - 278/279
# 131 - 2466 - 288/289    # 154 - 2492 - 334/335
# 156 - 2494 - 338/339    # 161 - 2499 - 348/349
# 171 - 2509 - 368/369    # 214 - 2608 - 455/456
# 228 - 2640 - 485/486    # 268 - 2830 - 566/567
# 273 - 2835 - 576/577    # 278 - 2840 - 586/587
# 300 - 2874 - 630/631    # 305 - 2879 - 641/642
```

The key phrases for each of these project files are extracted by myself and stored in a file with the same name as the project file but with the ending “.key”. All the project files and their key phrase files are placed in the directory “Project_train”.

As a new model should be built, no input model file is chosen.

- Function:

A new model for the database Project should be built with the KEA model builder. The name of this new model is “Project_model”. Building a model with this text and their key phrases:

```
java KEAModelBuilder -l Projekt_train -m Project_model
```

- Output:

Project_model: A new created model suitable to the project database.

Part II

- Input:

The created new model “Project_model” from part I of this experiment should be tested. So the whole text of the first 101 projects is stored in a text file for each project in the directory “Projekt_test”.

As the key phrases should be extracted, no key phrase files have been applied before running the experiment.

The input model for the key phrase extractor is the new created model “Projekt_model”.

- Function:

The usefulness of the new model from part I should be tested. This model is used for extracting key phrases from the first 101 projects of the database “Project”.

The parameter “-n” is used for setting the number of extracted key phrases to four instead of the default value of five. The parameter “-a” is used for delivering the used word stem and the probability for each key phrase to be one. This should give an insight of the key phrase extracting process.

```
java KEAKeyphraseExtractor -l Projekt_test -m Project_model -n 4 -a
```

- Output:

Four key phrases for the first 101 projects with stemmed phrases and score.

Results and Interpretation

Three examples for the delivered key phrases with word stem and score are presented in the table 4.3. The first column of each table contains the actual key phrase. The second column is filled with the word stem of this key phrase as found by Lovin stemmers. And the last one contains the probability for this word being a key phrase.

In detail, for the second key phrase of the first example: “project-members Oliver Flege” is the extracted key phrase, the word stem of it is “project-member oliver fleg” and it has the probability of 0.86 percent to be a key phrase.

It is obvious that a key phrase can consist of several words. This can be changed with setting the parameter “-x”. The default value is three.

Another result of this experiment is that the probability of the chosen words being a key phrase is very small, often smaller than 1 percent. This leads to the conclusion that the calculation KEA uses for key phrase extraction do not fit to the given experience factory. As explained in section 2.3 KEA chooses phrases which are not common in the environment of the text, i.e. the directory all the related texts are stored. On the other hand the KEA key phrases characterize a single text and show the difference to the others. It could be helpful for finding the discriminating attributes.

TABLE 4.3: Results Second Experiment

key phrase	word stem	probability
Project-no. 10:		
STT	st	0.8603
project-members Oliver Flege	project-member oliver fleg	0.0086
maxess systemhaus GmbH	mac systemhaus gmbh	0.0086
sponsor Markant	spon mark	0.0086
Project-no. 46:		
Dietmar Pfahl	dietmar pfahl	0.0086
project-manager Dietmar Pfahl	project-manager dietmar pfahl	0.0086
case-name Management von	ca-nam manag von	0.0086
Projekten und Prozessen	projekt und proz	0.0086
Project-no. 101:		
Schlumberger Retail Petroleum	schlumberger retail petroleum	0.0086
CEFRIEL	cefriel	0.0086
Jürgen Wüst	jürg wüst	0.0086
Establishment of Measurement	establishm of measur	0.0086

Chapter 5

Summary

5.1 Conclusion

In the context of this project thesis different text mining systems have been examined and described. Their usefulness for the experience management system of the Fraunhofer IESE have been discussed in detail above. The findings once again in a nutshell:

- The tools Yale and Weka are data mining systems. Their input data has to be structured in columns. The data values can be numbers or nominal values (predefined single words). No empty fields are allowed.
- The GUI version of Yale is easy to handle, even though there are some bugs and inexpressive error messages. Large datasets can be processed by a sequence of operators without latching.
- Weka provides a great number of algorithms for data handling which could be used in the Yale environment. Many of these algorithms can deal with nominal attributes. In the case of using Weka within Yale a label attribute has to be defined for the input data.
- The result of the decision tree learning algorithm in combination with the given data can be used for classifying new data. A detailed explanation is given in section 4.1 paragraph “Results and Interpretation”.
- KEA extracts key phrases from a continuous text in English which is in ASCII format. These words are the discriminating elements for the text files. The key phrases may be used for classifying data and as input for data mining systems.
- The entries of the experience factory are incomplete and inconsistent. Often, fields are empty and two different languages are used. Redundant information is stored.
- Only the few given numeric or nominal fields can be used for the data mining systems. Columns with continuous text are too small for extracting expressive key phrases with KEA.

To increase the utility of the data mining systems Yale and Weka a new experience management system based on the already stored information should be developed. This proposed project could be adapted to the requirements of machine learning algorithms. The main task is to find attributes which describe the experience in a way computers can handle. The classification of the already available data can be helpful for this purpose.

In the next section some points are listed which should be tried and tested. Especially the use of SQL queries is a promising function as the IESE experience management system consists of databases.

5.2 Future Work

Some tasks which could not be examined in detail within this project thesis are summarized and presented.

- In this topic, only the database “Project” is examined. The first interesting function of the Yale data mining system is to generate a data set with SQL queries. Therewith it would be possible to combine different databases with their foreign keys. This could be used for examining the connection between problems and projects, e.g. what properties have the projects which caused the problems.
- As mentioned in the previous section the urgent task is the development of an experience factory which is not only able to store experiences but also to retrieve it. Therefore the objective of the experience factory has to be defined and attributes which express it must be selected. This is an extensive project where different IESE members with their knowledge and requirements should take part of.
- The next tool that should be tested is the Weka GUI version. The advantages and disadvantages of Weka should be detected and compared with Yale.
- In the scope of that other clustering and learning features from the Weka program package could be tried. Some learning algorithms which are able to handle the current input data without modification are for example “Decision stump”, “Decision table”, “IBk”, “IB1” and “Neural Networks”.
- An interesting assignment for text mining is the examination of Support Vector Machines (SVM). These are typically used for pattern recognition in numerical data. To prepare them for finding patterns in continuous texts is a promising field of research.
- And finally a necessary but not pretentious task is to write a Java class for German stop words.

Appendix A

XML-Files

A.1 Clusterer experiment file

```
<operator name="Global" class="Experiment">
  <parameter key="logfile"
    value="/home/Michaela/Yale/daten/new_yale/cluster.log"/>
  <parameter key="temp_dir"
    value="/home/Michaela/Yale/daten/new_yale/tmp"/>
  <parameter key="resultfile"
    value="/home/Michaela/Yale/daten/new_yale/cluster.res"/>
  <operator name="Input" class="ExampleSource">
    <parameter key="attributes"
      value="/home/Michaela/Yale/daten/new_yale/project_attrib3.xml"/>
  </operator>
  <operator name="Result1" class="ResultWriter">
  </operator>
  <operator name="Cluster" class="WekaClusterer">
    <list key="weka_parameters">
    </list>
    <parameter key="weka_clusterer"
      value="weka.clusterers.Cobweb"/>
    <parameter key="cluster_file"
      value="/home/Michaela/Yale/daten/new_yale/cluster_file.txt"/>
    <parameter key="weka_clusterer_name"
      value="weka.clusterers.Cobweb"/>
  </operator>
  <operator name="Result2" class="ResultWriter">
  </operator>
  <operator name="ExampleSetWriter" class="ExampleSetWriter">
    <parameter key="example_set_file"
```

```
    value="/home/Michaela/Yale/daten/new_yale/my_example.txt"/>
  <parameter key="attribute_description_file"
    value="/home/Michaela/Yale/daten/new_yale/my_attribute.txt"/>
</operator>
<operator name="AttributeSetWriter" class="AttributeSetWriter">
  <parameter key="attribute_set_file"
    value="/home/Michaela/Yale/daten/new_yale/my_attribute2.txt"/>
</operator>
</operator>
```

A.2 Classifier experiment file

```
<operator name="Global" class="Experiment">
  <parameter key="logfile"
    value="/home/Michaela/Yale/daten/new_yale/classifier.log"/>
  <parameter key="temp_dir"
    value="/home/Michaela/Yale/daten/new_yale/tmp"/>
  <parameter key="resultfile"
    value="/home/Michaela/Yale/daten/new_yale/project_res.txt"/>
  <operator name="Input" class="ExampleSource">
    <parameter key="attributes"
      value="/home/Michaela/Yale/daten/new_yale/project_attr3.xml"/>
  </operator>
  <operator name="Result1" class="ResultWriter">
  </operator>
  <operator name="Learner" class="DecisionTreeLearner">
    <parameter key="model_file"
      value="/home/Michaela/Yale/daten/new_yale/classifier.mod"/>
  </operator>
  <operator name="Result2" class="ResultWriter">
  </operator>
</operator>
```

A.3 Attribute description file

```
<attributeset default_source=
  "/home/Michaela/Yale/daten/new_yale/project3.dat">
  <attribute
    name      ="funding"
    sourcecol ="1"
    valuetype ="nominal"
```

```
    blocktype = "single_value"
    classes   = "Industrial IESE root unknown Public EU FhG State German"
  />
  <label
    name      = "project_type"
    sourcecol = "2"
    valuetype = "nominal"
    blocktype = "single_value"
    classes   = "Study Seminar Workshop Training unknown Education
                RD Transfer Infrastructure false Consulting"
  />
  <attribute
    name      = "duration"
    sourcecol = "3"
    valuetype = "integer"
    blocktype = "single_value"
  />
  <attribute
    name      = "team size"
    sourcecol = "4"
    valuetype = "integer"
    blocktype = "single_value"
  />
</attributeset>
```


Appendix B

Result Writer

B.1 Clusterers

B.1.1 EM with project data

```
19.05.2003 11:23:42 Results of ResultWriter 'Result1':
19.05.2003 11:23:42 ExampleSet: 331 examples; no partition;
  attributes = {
    #0: funding:=funding[]: nominal/single_value/no block
      [EU,Industrial,FhG,IESE,German,Public,State,unknown,root]
      <on>,
    #2: duration:=duration[]: integer/single_value/no block<on>,
    #3: team size:=team size[]: integer/single_value/no block<on>
  }; 3/3 used
  label = #1: project_type:=project_type[]:
    19.05.nominal/single_value/no block
    [RD,Study,Transfer,Seminar,Consulting,Education,false,Training,
    Workshop,Infrastructure,unknown]
19.05.2003 11:23:45 Results of ResultWriter 'Result2':
19.05.2003 11:23:45
EM
==
Number of clusters selected by cross validation: 4
Cluster: 0 Prior probability: 0.693
Attribute: funding
Discrete Estimator. Counts =  2.17 202.37 3.52 1.82 3.35
                             2.05 8.47 11.57 1.83
(Total = 237.15)
Attribute: duration
Normal Distribution. Mean = 0 StdDev = 0
```

```

Attribute: team size
Normal Distribution. Mean = 0 StdDev = 0
Attribute: project_type
Discrete Estimator. Counts = 15 18.89 51.9 10.61 38.39
                             3.59 4.68 3.65 18.19 1 73.25
(Total = 239.15)
Cluster: 1 Prior probability: 0.2254
Attribute: funding
Discrete Estimator. Counts = 9.83 23.63 2.48 3.18 23.65
                             7.95 11.53 1.43 1.17
(Total = 84.85)
Attribute: duration
Normal Distribution. Mean = 0 StdDev = 0
Attribute: team size
Normal Distribution. Mean = 0 StdDev = 0
Attribute: project_type
Discrete Estimator. Counts = 49 6.11 9.1 1.39 1.61
                             3.41 1.32 3.35 1.81 1 8.75
(Total = 86.85)
Cluster: 2 Prior probability: 0.0332
Attribute: funding
Discrete Estimator. Counts = 3 9 1 1 2 1 1 1 1 (Total = 20)
Attribute: duration
Normal Distribution. Mean = 0 StdDev = 0
Attribute: team size
Normal Distribution. Mean = 6 StdDev = 4.156
Attribute: project_type
Discrete Estimator. Counts = 6 3 4 1 1 2 1 1 1 1 1 (Total = 22)
Cluster: 3 Prior probability: 0.0483
Attribute: funding
Discrete Estimator. Counts = 2 10 2 4 2 2 1 1 1 (Total = 25)
Attribute: duration
Normal Distribution. Mean = 466.625 StdDev = 405.9096
Attribute: team size
Normal Distribution. Mean = 5.625 StdDev = 5.8189
Attribute: project_type
Discrete Estimator. Counts = 8 5 4 1 2 1 1 1 1 2 1 (Total = 27)
19.05.2003 11:23:45 ExampleSet: 331 examples; no partition;
  attributes = {
    #0: funding:=funding[: nominal/single_value/no block

```

```

    [EU,Industrial,FhG,IESE,German,Public,State,unknown,root]
    <on>,
    #2: duration:=duration[]: integer/single_value/no block<on>,
    #3: team size:=team size[]: integer/single_value/no block<on>
}; 3/3 used
label = #1: project_type:=project_type[]:
    nominal/single_value/no block
[RD,Study,Transfer,Seminar,Consulting,Education,false,Training,
Workshop,Infrastructure,unknown]

```

B.1.2 Cobweb with project data

19.05.2003 11:17:26 Results of ResultWriter 'Result1':

19.05.2003 11:17:26 ExampleSet: 331 examples; no partition;

```

attributes = {
    #0: funding:=funding[]: nominal/single_value/no block
        [EU,Industrial,FhG,IESE,German,Public,State,unknown,root]
        <on>,
    #2: duration:=duration[]: integer/single_value/no block<on>,
    #3: team size:=team size[]: integer/single_value/no block<on>
}; 3/3 used
label = #1: project_type:=project_type[]:
    19.05.nominal/single_value/no block
[RD,Study,Transfer,Seminar,Consulting,Education,false,Training,
Workshop,Infrastructure,unknown]

```

19.05.2003 11:17:28 Results of ResultWriter 'Result2':

19.05.2003 11:17:28 Number of merges: 49

Number of splits: 42

Number of clusters: 89

```

node 0 [331]
|  node 1 [247]
|  |  node 2 [80]
|  |  |  node 3 [25]
|  |  |  |  leaf 4 [11]
|  |  |  node 3 [25]
|  |  |  |  node 5 [7]
|  |  |  |  |  leaf 6 [2]
|  |  |  |  node 5 [7]
|  |  |  |  |  leaf 7 [2]
|  |  |  |  node 5 [7]
|  |  |  |  |  leaf 8 [2]

```

```
| | | | node 5 [7]
| | | | | leaf 9 [1]
| | | node 3 [25]
| | | | leaf 10 [7]
| | node 2 [80]
| | | leaf 11 [55]
| node 1 [247]
| | node 12 [106]
| | | node 13 [25]
| | | | leaf 14 [20]
| | | node 13 [25]
| | | | node 15 [5]
| | | | | leaf 16 [3]
| | | | node 15 [5]
| | | | | leaf 17 [2]
| | node 12 [106]
| | | node 18 [38]
| | | | leaf 19 [37]
| | | node 18 [38]
| | | | leaf 20 [1]
| | node 12 [106]
| | | node 21 [43]
| | | | node 22 [18]
| | | | | leaf 23 [17]
| | | | node 22 [18]
| | | | | leaf 24 [1]
| | | node 21 [43]
| | | | node 25 [15]
| | | | | leaf 26 [4]
| | | | node 25 [15]
| | | | | node 27 [5]
| | | | | | leaf 28 [1]
| | | | | node 27 [5]
| | | | | | leaf 29 [3]
| | | | | node 27 [5]
| | | | | | leaf 30 [1]
| | | | node 25 [15]
| | | | | node 31 [6]
| | | | | | node 32 [2]
| | | | | | | leaf 33 [1]
```

RESULT WRITER

```
| | | | | | node 32 [2]
| | | | | | | leaf 34 [1]
| | | | | | node 31 [6]
| | | | | | | node 35 [4]
| | | | | | | | leaf 36 [3]
| | | | | | | | node 35 [4]
| | | | | | | | leaf 37 [1]
| | | | node 21 [43]
| | | | | leaf 38 [10]
| | node 1 [247]
| | | node 39 [61]
| | | | leaf 40 [51]
| | | | node 39 [61]
| | | | | node 41 [10]
| | | | | | leaf 42 [6]
| | | | | | node 41 [10]
| | | | | | | node 43 [4]
| | | | | | | | node 44 [2]
| | | | | | | | | leaf 45 [1]
| | | | | | | | | node 44 [2]
| | | | | | | | | leaf 46 [1]
| | | | | | | | | node 43 [4]
| | | | | | | | | leaf 47 [1]
| | | | | | | | | node 43 [4]
| | | | | | | | | leaf 48 [1]
node 0 [331]
| | | node 49 [84]
| | | | node 50 [65]
| | | | | node 51 [29]
| | | | | | leaf 52 [28]
| | | | | | node 51 [29]
| | | | | | | leaf 53 [1]
| | | | | | | node 50 [65]
| | | | | | | | node 54 [20]
| | | | | | | | | leaf 55 [6]
| | | | | | | | | node 54 [20]
| | | | | | | | | | node 56 [9]
| | | | | | | | | | | leaf 57 [2]
| | | | | | | | | | | node 56 [9]
| | | | | | | | | | | leaf 58 [6]
```

```
| | | | node 56 [9]
| | | | | leaf 59 [1]
| | | node 54 [20]
| | | | leaf 60 [5]
| | node 50 [65]
| | | leaf 61 [16]
| node 49 [84]
| | node 62 [19]
| | | node 63 [8]
| | | | node 64 [3]
| | | | | leaf 65 [1]
| | | | node 64 [3]
| | | | | leaf 66 [1]
| | | | node 64 [3]
| | | | | leaf 67 [1]
| | | node 63 [8]
| | | | node 68 [4]
| | | | | node 69 [2]
| | | | | | leaf 70 [1]
| | | | | node 69 [2]
| | | | | | leaf 71 [1]
| | | | node 68 [4]
| | | | | leaf 72 [2]
| | | node 63 [8]
| | | | leaf 73 [1]
| | node 62 [19]
| | | node 74 [7]
| | | | leaf 75 [1]
| | | node 74 [7]
| | | | leaf 76 [1]
| | | node 74 [7]
| | | | node 77 [2]
| | | | | leaf 78 [1]
| | | | node 77 [2]
| | | | | leaf 79 [1]
| | | node 74 [7]
| | | | node 80 [2]
| | | | | leaf 81 [1]
| | | | node 80 [2]
| | | | | leaf 82 [1]
```

```
| | | node 74 [7]
| | | | leaf 83 [1]
| | node 62 [19]
| | | node 84 [4]
| | | | leaf 85 [1]
| | | node 84 [4]
| | | | leaf 86 [1]
| | | node 84 [4]
| | | | leaf 87 [1]
| | | node 84 [4]
| | | | leaf 88 [1]
```

```
19.05.2003 11:17:28 ExampleSet: 331 examples; no partition;
  attributes = {
    #0: funding:=funding[]: nominal/single_value/no block
      [EU,Industrial,FhG,IESE,German,Public,State,unknown,root]
      <on>,
    #2: duration:=duration[]: integer/single_value/no block<on>,
    #3: team size:=team size[]: integer/single_value/no block<on>
  }; 3/3 used
label = #1: project_type:=project_type[]:
  nominal/single_value/no block
[RD,Study,Transfer,Seminar,Consulting,Education,false,Training,
Workshop,Infrastructure,unknown]
```


Bibliography

- [1] Oliver Ritthoff, Ralf Klinkenberg, Simon Fischer, Ingo Mierswa, and Sven Felske. YALE: Yet Another Learning Environment.
- [2] Simon Fischer, Ralf Klinkenberg, Ingo Mierswa, and Oliver Ritthoff. YALE: Yet Another Learning Environment - Tutorial. 2002.
- [3] Ian H. Witten and Eibe Frank. *Data Mining: Praktische Werkzeuge und Techniken für das maschinelle Lernen*. Carl Hanser Verlag, München, 2001.
- [4] Ian H. Witten, Gordon W. Paynter, Eibe Frank, Carl Gutwin, and Craig G. Nevill-Manning. KEA: Practical automatic keyphrase extraction. In *ACM DL*, pages 254–255, 1999.