Comparison of Adaptive Classification Methods

Gabor Kecskemeti G-5S8
Supervisor: **Dr. Laszlo Kovacs**
Department of Information Technology

April 29, 2004

Contents

Chapter 1

Abstract

The objectives of this paper are the next:

1. Provide an introduction to the document classification and clustering tasks and techniques and systematise them.
2. Choose two classification methods which can create usable results for humans, and analyse their effectiveness.
3. Analyse the ability of these methods to learn about other documents without repeating the whole learning process.
4. Implement a test system to compare the base and the adaptive methods

Chapter 2

Background Research

2.1 Introduction

The large amount of data which has to be processed by the project implies data mining techniques, thus to understand what the program could do the reader has to be aware to the concepts of these methods.

2.1.1 The Concept Of Data Mining

[2] If we want to understand the concept of the Data Mining we have to know the background of the causative theories.

Nowadays we have very much information in local databases and on the Internet. The aggregation of this information usually made by a relational database system, which has it's roots in the early 1980s. This large amount of data can not reach it's original aim to get knowledge from it, because the relational database systems are query oriented and for the best performance the data have to stored in normalised form. Normalisation made to decrease the redundancy and simplify updating of data. But it make the needed data fragmented, and if we want to collect the results of a complex query then we have to walk through unnecessary relations to connect the requested elements. The normalisation causes slowly reveal-able relations. Furthermore the person who does not know the actual database management system's specialities the query writing could be difficult. The relations contains a lot of data which seem to be unimportant for the query but without these data we can not get to our needs.

Accordingly a simply usable data management layer becomes necessary. This layer is intended to use anyone who want to gather information from the database, so it should be easily understandable and fast. Because the collected data, which is stored in the database system, are for decision support

usually. These facts lead to On-Line Analytical Processing (OLAP) technique. Nowadays it is the other big branch of the database systems, which usually cooperates with the legacy transaction based systems. These systems usually based on data cube model which is specially designed to store large amount of data which can be accessed in every context. The OLAP technique developed for the users. But if we want to know how it is working we need to know the concept of the data warehouses, which is usually in the background. But if we need to interact with an OLAP system as an information engineer we need to know a lot more about this hidden thing. The data warehouses contains data from heterogeneous sources which are filtered and checked to prove the best composition of the information gathered from there. Because the main aim is to be fast in execution of complex queries, we make data available in every context thus these techniques uses the storage capacities wastefully.

Information lie not only in certain data values, but in the meaning of these series of values. And at last we have arrived: the data mining is a method set which is able to find background information or coherency of a large data set. To retrieve the information from the system we should use artificial intelligence methods. Nowadays the data mining integrates artificial intelligence to achieve its aims, because it contains learning and knowledge representation methods which was developed in the last 40 years.

2.1.2 The Process Of Data Mining

This process has several stages which are strongly connected with the operations of OLAP systems. These are data loading, filtering and standardisation procedures which was developed for each newly deployed data warehouse or OLAP system. And these systems are the lower layers of a data mine, so the performance of the data mining applications depends on the preparation of the sources of these applications. The main property of an adequate data warehouse is to be sparing and extensible any time.

Unlike the users of the legacy database systems, the data mining applications usually have to analyse the whole database for a simple query, so the data warehouse behind the application should be the smallest which could satisfy the possible requests. For this reason we can achieve high efficiency with the data set's reduction which can be made by a deviance analyser or a similarity searcher. These methods are basically for throwing out the elements which can not add more information for the system or which make the system unstable. The other way to get the results faster is to calculate the aggregated values in every possible context, so when a new query is running it can use these previously calculated values. But a data cube could have a lot of viewing point which contains hardly more information than others, so we can leave these

aggregated values to be calculated when they are needed. Then the knowledge extraction could be easier and faster.

The main phase of data mining begins after the preparation of data, and this phase usually based on statistical methods which could be hid behind some artificial intelligence technique. These methods have several aims which are listed here [4]:

Cluster analysis is to find any natural group structure of a currently undifferentiated data set. Usually this uses a distance function to determine the borders of the newly defined groups. There we could use unsupervised learning methods such as: self organising neural networks (SOM), FP growth trees.

Classification: this technique's goal is to categorise elements. This categorisation is based on a sample set, which was filled with special manually categorised elements. During the preparation methods supervised learning could be used, such as back propagation and Hopefield neural networks, inductive learning methods (like decision trees, case based reasoning), etc.

Prediction: this technique's aim is to get future trends and data with using the data distribution in time, so the data cube have to be accessed through the dimension of time.

Deviation analysis: The purpose of deviation analysis is to find extremities in the data set which could be used for special dissection.

Exploration of associative rules: this technique's goal is to find causality between specific data elements. With this method we can identify events, attributes or entities which implicate others in the same transaction or time period.

So at the end of the process we get highly filtered concentrate of information which can reveal unknown facts about the data set.

2.2 Machine Learning

[8] Machine learning algorithms used to gather *concept descriptions* from prepared *examples* with *background knowledge* which usually a man made simplification of the learning algorithm about the current problem. These algorithms can be grouped by three criteria defined by Michie in 1988 on "3rd European working session on Learning":

1. *Weak criterion*: The system uses sample data to generate an updated basis for improved performance of subsequent data.

2. *Strong criterion*: weak criterion is satisfied. Moreover, the system can communicate its internal updates in explicit symbolic form.

3. *Ultra-strong criterion*: strong criteria is satisfied. Moreover, the system can communicate its internal updates in an operationally effective symbolic form.

Every learning system satisfies the weak criterion. Only the algorithms which are able to give symbolic form of the learned concepts can satisfy the strong one. If the symbolic description of the concepts easily usable by people then the system which produced this satisfies the ultra-strong criteria. It means that if it found a description then we do not need to use the system to analyse the subsequent data because a person can do on an effective way.

2.2.1 Knowledge Representation

The computer has to learn concepts because the interacting people usually do not know anything about the background of the GUIs. So the computer has to know about concepts through examples and background knowledge. So it needs a knowledge representation model which both the computer and its users can understand. AI researchers has defined several models to satisfy these needs. Here are some examples:

Zero order logic (ZOL), or propositional calculus: the knowledge is represented as conjunction of Boolean constraints that stand for the individual features. For example here is a simple representation of an English person:

$$EnglishPerson \Leftarrow ComesFromUK \wedge SpeaksEnglish$$

This representation method has low descriptive power because it is difficult to capture in this way complex concepts encountered in daily life.

Attributional logic: This type of representation is only an extension of ZOL's notation so attribute logic has low expressiveness like ZOL. The boolean constraints of the ZOL become multi value variables which make the knowledge clearer than the simpler ones in the ZOL. Examples are often presented in a table where each row represents an example and column stands for an attribute. See table 2.1 on page 10 as an example.

Predicate logic based techniques are other representation methods which has more descriptive power for example which are based on predicate logic such as horn clauses like clauses in the *Prolog* language. These techniques are the base of second order logic and explicitly constrained languages which sometimes enables recursion in the predicates. With these possibilities the concepts can be easily described for the computer, but using these rules could be expensive.

Alternative representations: The most popular alternative non logic based representation method is using *Minsky's frames*. But in special cases abstract mathematical structures like *grammars* and *finite automata* could lead towards the target on a fast way.

If the information stored on one of this ways, the writer of the program could develop a system which satisfies the *ultra-strong criterion* of machine learning applications (see section 2.2 on page 6).

2.3 Classification

[4] If somebody wants to decide which group is the best for an object then it is a classification problem. Classification is everywhere you can find it for instance in the biology - as Council, Monsieur Arronax's best colleague, categorise species around the newly explored depths of the world on the deck of the Nautilus - or in the chemistry. Classification is not a new concept. Lot of mathematicians work on it in the whole twentieth century and before. But it is forgotten time to time so each rediscovery gives to it a new notation system. But today computers can be used for best results in complex problems. The classification problems could be solved by artificial intelligence techniques mostly which are based on a compound of many mathematical theories for example statistics.

Usual classification methods based on *attributional logic* (see section 2.2.1 on page 7), so the definition which is described below based on this fact. Here are the definitions of classification's basic principles:

[1] An *attribute* ($a_i \in A$ where $i = 1, 2 \dots W$) is a measurable property of an entity. Every attribute has a possible value set ($v_{a_i,j} \in V_{a_i}$), so an *entity* ($\overrightarrow{\epsilon_k} = (v_{a_1,k}, \dots v_{a_W,k})$) is a vector in the attribute space. A *class* ($o_l \in O$) is a user defined concept for example in this paper we used such classes as politics and economics. With these classes we can define a new vector space: Ω which shows the attended classes so the l^{th} dimension of this space should be 1 if o_l is present or 0 else.

With these basic states we can define a classifier or a classification rule as the following function(d):

$$\overrightarrow{R_k} = d(\overrightarrow{\epsilon_k}) \quad where \quad \overrightarrow{R_k} = (r_{k,1}, \dots r_{k,C}) \in \Omega \tag{2.1}$$

Usually the classification rules derived from examples ($\Xi = \{\xi_1, \dots, \xi_E\}$), these are entities with a special attribute which shows the man made classification values for its owner: $\xi_m = \{\overrightarrow{\epsilon_m}, \overrightarrow{R_{k,MM}}\}$

2.3.1 Adaptability

After the learning phase of the classification rules it could be necessary to extend the base of the learning. The usual classifiers are for learning a constant base, so if it change the costly learning has to be started again from the beginning. Thus there is a need for adaptability or *incremental learning* capability in these methods. There are two methods of extending the learning set, the first acts immediately when a misclassification occurs. The other uses a threshold value for the misclassification rate and acts after the misclassification rate risen above this vale.

If a classifier has used like this it easily can lead to a messy classification rule, because it contains some rules from the early stages of its life which should be thrown out when becomes too old or misleading. After throwing out some misleading examples the classifier has to be relearned. So it causes the same problem described above. Which could be solved the inverse method of that.

Each classification algorithms has specialities to solve the incremental learning problem, so these will be discussed on their sections.

2.3.2 Bayesian Classifiers

[4] The Bayesian classifiers use simply probabilities to estimate $\overrightarrow{R_k}$. The concept of these routines is to reduce the total error of classification. So the program has to choose the most probable $\overrightarrow{R_k}$ if it has to classify $\overrightarrow{\epsilon_k}$. So let start with a simple example:

An Example

There are two horses (Sunray and Steamer) in a race whose has a chance to win, and their chances are: $P(Sunraywins) = 0.7$ and $P(Steamerwins) = 0.3$. If we has to classify this case, and we has the next class: $o_1 =$ "Horse wins", then our classifier must have to give this answer when $A = \{Horse\}$ and $\overrightarrow{\epsilon_k} = \{Sunray\}$: $\overrightarrow{R_k} = (1)$. But we know that Sunray has problems on rainy days so its winning chance will be the next: $P(Sunraywins|rainy) = 0.2$. When the attribute

space contains the weather variable the decisions which has to be made by the classifier should be described like in table 2.1.

Case	Horse	Weather	Horse wins
#1	Sunray	Sunny	true
#2	Sunray	Rainy	false
#3	Steamer	Sunny	false
#4	Steamer	Rainy	true

Table 2.1: Attributional logic, theoretical output of the horse race

The Classification Rule

In table 2.1 the Bayes rule showed its form: the classifier should choose the class with the biggest conditional probability depending on the attributes of the actual entity.

$$r_{k,l} = \begin{cases} 1 & \text{if } P(o_l|\overrightarrow{\epsilon_k}) = \max_j P(o_j|\overrightarrow{\epsilon_k}) \\ 0 & else \end{cases} \tag{2.2}$$

But this rule is a dead end if we have a big attribute space which makes much bigger k values than the computers can evaluate this expression. So the $\overrightarrow{\epsilon_k}$ has to be thrown out from the condition, for this reason we need to use the *Bayes theorem*:

$$P(o_l|\overrightarrow{\epsilon_k}) = \frac{P(\overrightarrow{\epsilon_k}|o_l) \cdot P(o_l)}{P(\overrightarrow{\epsilon_k})}$$

To get the easiest form of the classifier this assumption has to be made about the independence of the attributes (Before this assumption this kind of classifiers are the best because the maximisation of the conditional probabilities, see section 1.5 of [1].):

$$P(\overrightarrow{\epsilon_k}|o_l) = \prod_{i=1}^{W} P(v_{a_i,k}|o_l)$$

And the final form of the *Naive Bayesian classifier*:

$$\begin{aligned} \mathrm{NBCR}(k,l) &= \prod_{i=1}^{W} P(v_{a_i,k}|o_l) \cdot P(o_l) \\ \mathrm{MNBCR}(k) &= \max_j \mathrm{NBCR}(k,j) \\ r_{k,l} &= \begin{cases} 1 & \text{if } NBCR(k,l) = MNBCR(k) \\ 0 & else \end{cases} \end{aligned} \tag{2.3}$$

Results With Multiple Classes

Text documents usually has more class memberships because the class definitions are overlapped, so the result has to be altered to show this:

$$r_{k,l,NB} = \begin{cases} 1 & \text{if } NBCR(k,l) \geq MNBCR(k) \cdot \mu \\ 0 & else \end{cases} \tag{2.4}$$

In equation 2.4 the $\mu \in (0,1]$ is used to make a limit for the acceptable probability difference of the classes from the class with the maximal probability. As you can see we can't extract any information from this classifier which could help the people because the rules which they have to apply are pretty much. Thus these rules have to contain every a_i,o_j combination which means there are $C \cdot W$ rules.

Adaptability

The Bayesian classifiers could be up-to-date easily, because only some of the probabilities, which is bothered by the newcomer, ought to be recalculated.

Applications

The applications of this classification method are widespread, but there are some examples in [4] which show how to use Bayesian classifiers when the distribution of the examples are known or can be effectively guessed. These methods using Fisher's iris data to categorise the species of iris, and for practical purposes transistor gain and noise information to decide which freshly produced transistor is good. In this fields the Bayesian classifiers are really usable, because the examples are in normal distribution.

2.3.3 Decision Trees

[3] In the early 1960s the computer scientists formed a new tree based learning system with the basis of cognitive psychology. This system is based on a the

divide and conquer method similar to the simplification and concept detection technique of a human mind so it can be easily understandable by everybody not like the other classification techniques. Then Hunt Marine and Stone defined their *Concept Learning System* (CLS) as the implementation of the newly invented technologies. The algorithm consists of three simply defined different steps:

1. Pick a criterion which separates the set on the best way.
2. Make the separation with this criterion.
3. Go back to the 1st step with each separated set until this set becomes an unique definition.

The techniques which are applying the CLS algorithm usually based on attribute logic, and the criterions are about to separate the sets by the attribute values. Thus this paper contains information only about the attribute based concept learning systems.

The decision tree is a cycle free graph which has nodes to support decisions. A branch of the tree represents a precedence relationship between the connected nodes. The weight of a branch is an element of the attribute value set (V_{a_i}) of origin node of the branch. In the tree the attributes are represented as nodes with as many branches originating from this as the cardinality ($|V_{a_i}|$) of the value set of the actual attribute. The last building block of the tree is the class node which is leaf in every context and shows the result of the classification. When the trees have drawn the attributes are in rounded boxes, the branches are arrows with the actual criterion and the classes are simple boxes.

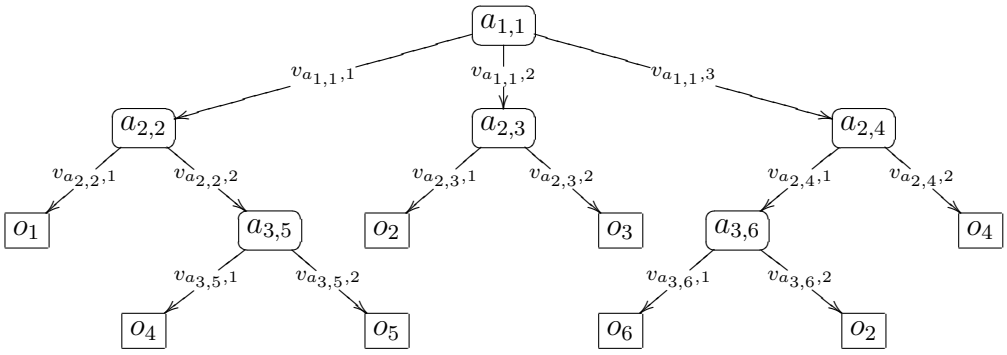

Figure 2.1: Simple decision tree

Usage

[8] There are two possibilities to use this tree to classify a newcomer entity, these are listed here:

The top-down method: The computer could use the tree as a person. To do this it should go through steps shown by figure 2.2.

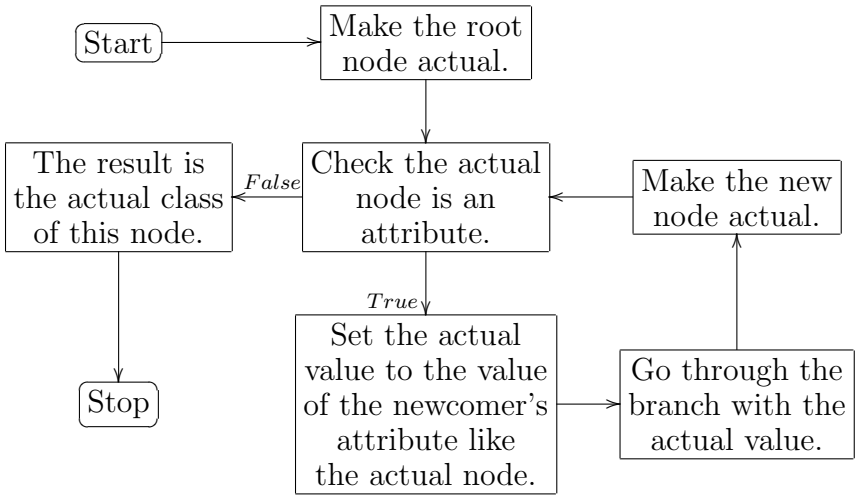

Figure 2.2: Top-down usage of decision trees

The bottom-up method: Logical expressions could be made from a tree, and when a newcomer arrives the classifier just need to check which expression is true and this is the result. An expression have to be built with each path from the root to a leaf. This has to contain all nodes and values which was touched until the leaf. The decision tree on figure 2.1 could be described like table 2.2 which shows 2 different methods of making

o_1	$\Leftarrow$	$a_{1,1} = v_{a_{1,1},1} \wedge a_{2,2} = v_{a_{2,2},1}$
o_2	$\Leftarrow$	$a_{1,1} = v_{a_{1,1},2} \wedge a_{2,3} = v_{a_{2,3},1}$
o_2	$\Leftarrow$	$a_{1,1} = v_{a_{1,1},3} \wedge a_{2,4} = v_{a_{2,4},1} \wedge a_{3,6} = v_{a_{3,6},2}$
o_3	$\Leftarrow$	$a_{1,1} = v_{a_{1,1},2} \wedge a_{2,3} = v_{a_{2,3},2}$
o_4	$\Leftarrow$	$(a_{1,1} = v_{a_{1,1},3} \wedge a_{2,4} = v_{a_{2,4},2}) \vee$ $(a_{1,1} = v_{a_{1,1},1} \wedge a_{2,2} = v_{a_{2,2},2} \wedge a_{3,5} = v_{a_{3,5},1})$
o_5	$\Leftarrow$	$a_{1,1} = v_{a_{1,1},1} \wedge a_{2,2} = v_{a_{2,2},2} \wedge a_{3,5} = v_{a_{3,5},2}$
o_6	$\Leftarrow$	$a_{1,1} = v_{a_{1,1},3} \wedge a_{2,4} = v_{a_{2,4},1} \wedge a_{3,6} = v_{a_{3,6},1}$

Table 2.2: Figure 2.1 represented by logical expressions

expressions: Because o_2 and o_4 have multiple appearances in the tree, and these appearances might join with a logical "or" operator otherwise the leaves with same classes are in different expressions.

The expression list, which contains only one row per class, is the Classification Rule ($d(\overrightarrow{\epsilon_k})$) of the decision tree based classifiers.

If a man has the graph or the logical expressions he could easily classify a newcomer, thus this classification technique fulfils the ultra-strong criteria of the machine learning (see section 2.2 on page 6). Although tree created by a machine could be very large one, so the trees must be prepared while the tree are growing. (See page 16 for further information.)

Construction

[1] To implement the CLS (see on page 12) some extra steps needed which are exactly defined in the TDIDT (*Top Down Induction of Decision Trees*) or ID3 (*Itemised Dichotomizer 3*) algorithm. Each separation in the CLS make a series of nodes which are connected to the actual attribute with it's values. The node with unique definition becomes a class descriptor ($\boxed{o_l}$) others implicate another decisions thus they are attributes ($\boxed{a_i}$). The separation criterion says that attribute (a_c) should be chosen to make the disjoint set of examples which has the same value on the chosen attribute $\Xi_{v_{ac,j}} = \{\xi_m | v_{ac,m} = v_{ac,j}\}$ (see figure 2.3 on page 15). The rule works as a function ($\Phi(a_i, \Xi)$) with two parameters, the first is the attribute what is supposed to chose, the second is the example set which should be separated. It has to satisfy these criteria ([8]):

1. The function reaches its maximum when all subsets are homogeneous. In this case the information about the attribute value is sufficient to decide whether the example is positive or negative.

2. The function reaches its minimum when 50% of the examples in each of the subsets are positive and 50% are negative.

3. The function should be steep when close to the extremes and flat when in the 50%-50% region.

So the separation criterion of the CLS could be this: pick the attribute with the biggest separation value ($a_c = \{a_j | \Phi(a_j, \Xi) = \max_{a_i} \Phi(a_i, \Xi)\}$). At last here is a possible definition of the separation value on the basis of entropy ($E(\Xi) = -\sum_l p_{o_l} \log p_{o_l}$, where $p_{o_l} = \sum_{k=1}^{|\Xi|} r_{k,l,MM}/|\Xi|$):

$$\Phi(a_i, \Xi) = E(\Xi) - \sum_{j=1}^{|V_{a_i}|} \frac{|\Xi_{v_{a_i,j}}|}{|\Xi|} \cdot E(\Xi_{v_{a_i,j}}) \tag{2.5}$$

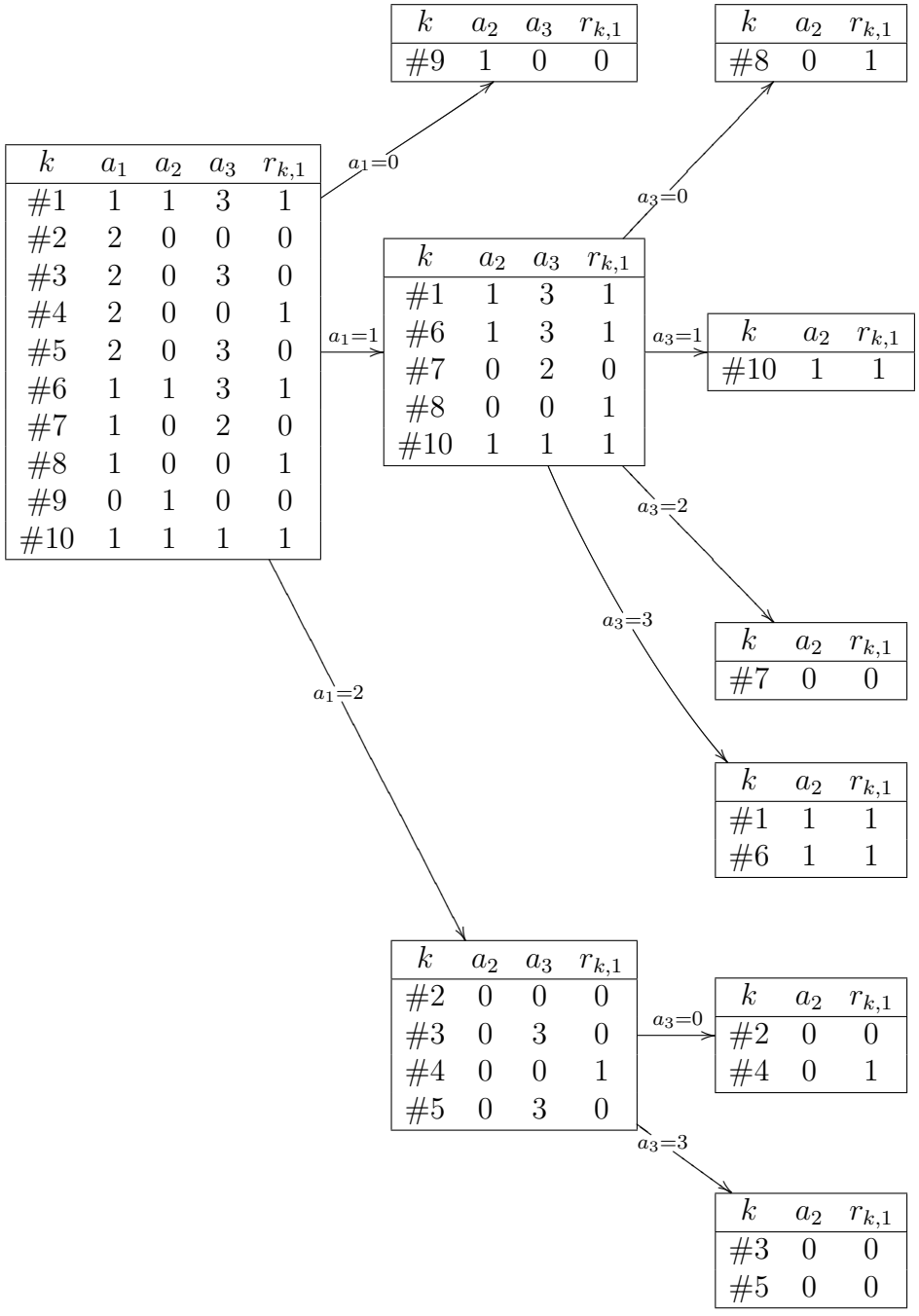

Figure 2.3: Separation of an example set

Specialities For Build Up

The tree has almost done, but there are some problems which has to be worked out. First of all the algorithm has to have scenarios for special cases.

- When the decision tree is growing there is a problem which appear on a regular basis: If there are no more attributes which can be used for separation, the tree should end with a class (o_l) which has the biggest probability (p_{o_l}) in the actual example set. There is an example in figure 2.3 on the branch of $a_1 = 2$ and $a_3 = 0$ for this problem. The other solution for this trouble is because the definition of the classification rule enables more than one returning classes, with the usage of μ what was defined in equation 2.4, the class node should store more than one class which have bigger probabilities than $\mu \cdot \max_l p(o_l)$.

- The other not too usual problem is here: there are some cases when there are more than one attribute ($A_S = \{a_{c1}, \ldots, a_{cS}\}$) with the biggest Φ value on a certain example set. In that occasion the tree could be built on two ways and when it finished one of the ways which gives smaller accuracy when tested with the whole example set should be thrown out. But it needs more work to do, so the implementation could be simpler if a randomly chosen attribute from A_S will be the separator.

The tree has finished, but it could be messy because the decisions made near the leaves are based only some examples which could be noisy. Thus the unlimited growing has to be stopped with threshold values which are determined by a data engineer. These threshold values could be for example: the minimum value of separation Φ_{min} or the minimal cardinality of the example set. When the watched values decreasing below these numbers the growth of the tree must stopped and the rule with no attributes should be used which is defined above these lines. There are other methods of tree pruning which are not discussed here because these are transforming the tree after the CLS finished, but there are a lot of examples in chapter 3 of [1].

Results With Multiple Classes

However the basic classifier may give a vector form Ω (see formula 2.1 on page 9) with more ones, this effect will appear when the builder of the tree was not certain about the outcome of the classification (see the second paragraph of the section before). If Φ reaches 0 then it means all of the examples about the actual decision has the same man made classification vector, thus the tree is not needed to grow bigger on this branch. This will stop the meaningless growing

of the tree but the information amount which could be gathered from the tree is the same.

There is an other solution which can provide more accurate results on a costly way: there should be a tree for every class and its negation in the Ω.

Adaptability

The decision tree based classifiers has much bigger problems to implement incremental learning, but if the splits of the example sets are stored then the newcomer just easily can stored in these sets and for the correct trees the splitting rule should be calculated on the new sets. This could cause the rebuilding of the whole tree but the recalculation only needed on big depths usually. There is a rough but fast method too that stores the newcomer in the affected parts of the tree and on the class node it just changes the probabilities of the classes so it could cause a new outcome of the classification.

Applications

The applications of this classification technique usually concentrating on the fact that the tree could be usable by a human being, so they usually using a few attributes to build the tree. The usage of these trees started by the American army to recognise six ship classes using their radar range profiles, this pattern recognition works like the number recognition problems which demonstrates the pruning techniques in the book [1].

The 5^{th} part of [8] describes some interesting usage of the decision trees and other classification techniques which helps the medical practice. These effective methods of medication are not accepted by the most of the health experts because they are not too flexible, and the decision trees can't express to much information because of the pruning techniques used to make it human readable.

2.3.4 AQ Learning

This classification method is the opposite of the decision tree based classifiers, because these begin with generalised rules and specialise them until it fits all of the examples. This was designed in 1969 by Michalski who is one of the editors of [8]. The goal of this method to make decision rules like in table 2.2 on page 13 thus the usage of the rules could be the same like the bottom-up method of the decision trees.

Rule finding

1. Divide all examples into the subsets (Ξ_i, $1 \leq i < C^C$ where $\Xi_i \neq \emptyset$) with same $\overrightarrow{R_{j,MM}}$.
2. Chose one example (ξ_k) from Ξ_i.
3. Generalise its characteristics until the generalisation does not affects any other $\overrightarrow{R_{j,MM}}$. Calculate all of the independent generalisations which are called together the *star*.
4. According to some *preference criterion* select the best rule in the star. The preference criterion could select the rule which covers the maximal number of examples from Ξ_i, or the most general rule in the star but each classification problem should have its own.
5. If this rule jointly with all previously generated rules covers all examples then stop, otherwise chose an other example from one of the subsets and repeat the steps from 3.

With a noisy example learning set this method will stop too early so the rules will be too specialised which shows the bottleneck of this technique. To solve the problem there is a second tier designed on the top of the AQ learner which drops the not too important rules.

Adaptability

With this method the adaptability can be solved easily because the newcomer has to be passed through the previously defined steps, and these rules, which generalised from them, have to be joined to the actual rule set.

2.3.5 Accuracy

[1] [4] Before a classification rule (d) becomes applied it needs to pass some simple tests about its accuracy. But what is accuracy in the field of classification? The accuracy rating (θ_k) of $\overrightarrow{\epsilon_k}$'s classification should be calculated one of on these ways:

$$\begin{aligned} O_{k,d,F} &= \{o_l | r_{k,l,d} = r_{k,l,MM}\} \\ \theta_{k,d,0} &= |O_{k,d,F}|/C \end{aligned} \tag{2.6}$$

$$\begin{aligned} O_{k,d} &= \{o_l | r_{k,l,d} = 1\} \\ O_{k,MM} &= \{o_l | r_{k,l,MM} = 1\} \\ O_{k,d,S} &= O_{k,d} \cap O_{k,MM} \\ \theta_{k,d,1} &= \begin{cases} |O_{k,d,S}|/|O_{k,MM}| & \text{if} \quad |O_{k,MM}| > |O_{k,d}| \\ |O_{k,d,S}|/|O_{k,d}| & \text{else} \end{cases} \end{aligned} \tag{2.7}$$

The second rating is much more strict than the first because the first one understates the errors when there are a lot of classes ($\theta_{k,0} \geq \theta_{k,1}$.)

Then to qualify the classifier several $\overrightarrow{\epsilon_k}$ is needed, but these entities must satisfy some basic rules, for example they should be independent from the example set, but they should be with the same probability distribution. The solution is far from ideal, because an independent set of testing entities with infinite cardinality is needed. Usually collecting examples is very costly because the classification made by a specialist. Thus the example set, which is provided for learning, is used for these tests too. But another techniques needed to make the example set independent from the tests.

The first measure of the accuracy (Θ) could be misleading with a little example set, because after the learning the accuracy is estimated by the classification of the example set thus the average accuracy of this estimate could be larger than it would be with a real testing set. This technique usually called *re-substitution estimate*:

$$\Theta^R(d) = \sum_{i=1}^{E} \theta_{i,d}/E \tag{2.8}$$

The obvious way of checking the accuracy is to separate the example set into a larger and a smaller subset and learn with the larger one, then check with the smaller. This usually called the *test sample estimate* (Θ^{ts}). But both sets have to have the same probability distribution, so the elements of each set are picked from the example set randomly.

$$\Theta^{ts}(d) = \sum_{i=1}^{K} \theta_{i,d_{E-K}}/K \tag{2.9}$$

The extremity of the test sample estimate has only one test entity. This leads to the *leave-one-out estimate* (Θ^{lvo}), which says learn the example set without only one entity, and check its accuracy rating, then do this again with all of the entities of the example set, and at last calculate the average of the accuracy ratings. This accuracy value costs more, but with small example sets this is the simply way to get closer to the real one:

$$\Theta^{lvo}(d) = \sum_{i=1}^{E} \theta_{i,d_{\Xi \setminus \xi_i}} / E \tag{2.10}$$

2.4 Attribute reduction

[6] Prior to using a classification algorithm in document classification, its inputs must be prepared. This preparation creates a small attribute set from the large data mine of the example set with full of possible attributes ($A_{\max}, \quad |A_{\max}| \approx \infty$), for example terms. The number or the instance of the terms are easily collectable, but it is hard to manage the huge number of terms, so the available terms must be filtered out before they become attributes. This preventive filtering is called relevance analysis. Thus an importance value ($\rho(a_i)$) is added to every term , this value depends on the actual term's class or the example set. Then only those terms will be used which have greater relevance value than a minimal reference ($\rho_{\min}$).

Now we can define the attribute reduction as follows:

$$A = \{a_i | (\rho(a_i) > \rho_{\min}) \wedge (a_i \in A_{\max})\} \tag{2.11}$$

These methods need statistically representative occurrence numbers for the processed terms. Every term can be transformed to fulfil this condition with a thesaurus. A term with a relatively rare occurrence can be generalised using the thesaurus, so it could be relevant with other generalised terms on the same field.

2.4.1 Entity Based

This sort of importance value calculation uses the fact that a term is present in the actual entity or not. The calculation using the relative frequencies of the appeared terms per class ($p_l(a_i|o_l)$). The calculated value should be the indicator of the importance, so the relative frequencies are not about a simple term. An aggregation has to make the value available for a term. This aggregation could be an average function, but a dispersion function could show the differences, thus this can show that a term could be a really good separator of the classes. Here is the formula of the entity based relevance value:

$$\begin{aligned} PS_{ij} &= \{a_i | (v_{a_i,k} > 0) \wedge (r_{k,j,MM} = 1)\} \\ p(a_i|o_j) &= \frac{|PS_{i,j}|}{\sum_{k=1}^{C} |PS_{i,k}|} \\ \rho_E(a_i) &= D_l(p(a_i|o_l)) \end{aligned} \tag{2.12}$$

2.4.2 Frequency Based

The method above is more widely usable than these and below, because these are for attributes like document's terms which have an occurrence value as attribute value. This means these methods only usable when the attributes' values from $\mathbb{N}^+$. This type of relevance calculation has 3 genres, which are the next:

Class based: This calculation works similar to the entity based, because it uses the relative frequencies of the occurrence of a term by class ($PS_{ij} \sim N_{ij}$).

$$N_{ij} = \sum_{r_{k,j,MM}=1} v_{a_i,k}$$
$$\rho_C(a_i) = \max_{k=1...C} \frac{N_{ik}}{\sum_{j=1}^{C} N_{ij}} \tag{2.13}$$

Term based: This calculation is the most simple of all because it has an independent value, the number of all terms in all documents, from a_i at the denominator. And it uses the N_{ij} defined in equation 2.13:

$$\rho_T(a_i) = \frac{\sum_{j=1}^{C} N_{ij}}{\sum_{k=1}^{W} \sum_{l=1}^{C} N_{kl}} \tag{2.14}$$

Term Frequency Inverse Document Frequency: [5] This method applied generally for relevance calculation because its flexibility. The idea behind this technique is to keep down the attributes with extremely high occurrences. The two frequency based techniques defined before are the victims of these attributes, because they make these terms unbeatable, so some really useless terms (like "a" or "the" or a more complex one: "of the") will lead the list of relevant attributes.

This technique is a product of two measures. The first is the term frequency (TF_i) and this should transform the occurrence values of a term into a number. The second is the inverse document frequency (IDF_i) which is in the product to lower the first member's effect when the term is common in other entities too.

Both members could be computed on multiple ways. For example the TF_i may be the one of the frequency dependent calculations defined above. This paper will use simply solutions, like these:

$$\begin{aligned} TF_i &= \max_j v_{a_i,j} \\ IDF_i &= \log(1 + \frac{E}{\sum_{j=1}^{C} |PS_{ij}|}) \\ \rho_{TI}(a_i) &= TF_i \cdot IDF_i \end{aligned} \tag{2.15}$$

2.4.3 Applications

These techniques seem like the techniques just used with others, but sometimes these could be enough because the provided relevance values are usable to weight entities without using more technique. These techniques was the base of the WebWatcher application which provides a keyword based search engine for a website and it is described in [8].

When these techniques become servants the user of them not just a classifier it could be for example an associative rule finder too. This was the main idea of the FACT system ([8]) which can effectively explore associations in text documents which are described with relevant keywords provided by some attribute reduction technique.

Chapter 3

Design and Implementation

3.1 The Program

The problems described in the previous chapter were solved with a program which has 3 main parts. These parts have unique interfaces which suit to each phase of data mining. Thus a newly developed method can be added to it later easily.

1. part is the loader. This part is the service of the example and testing entities thus it is the source of the information.

2. part is for the dimension reduction of $A_{\max}$. It uses the relevancy values to rank the terms in the example set. Although these routines can calculate the importance values for terms, the algorithm to find terms in the documents is more complicated, therefore it is not detailed in this paper. In these routines only words will be used. For example the a-priori algorithm can be used for detecting the frequent word sets.

3. part is the classifier. To reach its aim it needs a helper class which describes the examples with the entities and with their man made classification. The two classifier which was chosen for implementation is the ID3, and the Bayesian. The ID3 was chosen because the human readable rule set which could be generated from it. The Bayesian classifier was chosen because this could be the base of the comparison of accuracy, and speed. The other methods usually not used for text categorisation so this is the main reason of the decision about them.

Figure 3.1 on page 24 shows the layered structure of the project based on the parts described above and the relations between the specific layers. On this

figure you can see that adaptability is only a little extension of the classifiers because an adaptive classifier uses the recycled parts of their predecessors.

The project was implemented in Java because of the inhomogeneity of the possible machines to run, so a little knowledge about this language needed to understand this chapter of the paper, but it usually based on standard object oriented programming techniques.

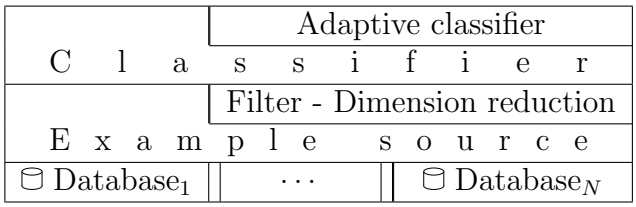

Figure 3.1: The layered structure of the project

3.2 Example source

The *DataService* is an abstract Java class which provides access to an example set in the memory, thus the extension of the class is just about the constructor which can conjoin the memory with the origin of the examples. Nowadays there are computers with a big amount of memory, but in some cases the program could run out of the memory space. It could be the bottleneck of the data mine so the extension should be more radical if it happens. With this extension only some of the examples could be loaded in the same time and the others could be on the disk or in the database until they needed.

The mining techniques need to go through all of the examples as mentioned in section 2.1.2 on page 5, thus if it is possible the data should be stored in the memory for faster learning times. After the learning only the classification rule ($d()$) is needed and the other ones like the dataservice could be released. Therefore the learning time could be a secondary option after accuracy which usually depends on the number of the correctly assorted learning examples. But the service usually just load the examples and do not do anything to correct the distribution of these data.

Before the data is stored in the memory the loader in the constructor of a descendant has to make a decision about the actual example. If the example could be misleading for the classifiers it should be skipped. But these skips need precaution, because the learning will begin after this operation and a bad decision here may lead a dead end to find the best classification rule.

The database, where the examples are stored originally, has different structure from this class needs, because the data which the program use has another goals too and usually these exist before the users decided to find out the classification rule about this. Thus the constructor has to transform the data and this takes long time. When searching for the best classinfication rule the dataservice should used several times which could be in different runs, so there is a need for faster loading times. This could be achieved by transforming the database to fit our needs immediately, or the program can make a mirror about the sources locally and this will stored as its need. The Java gives the best solution for this, so the *DataService* was implemented on this way. The solution is the *serialization* capability of Java. Each extension of this Java class should be serializable to reduce the loading times with the *ObjectInputStream*.

3.2.1 Representation of the examples

The set implemented as an indexed list so each example (ξ_k) has the index in the program like on this sheet (k). An example have to get its description about the attributes and the man made classifications. In this phase the attributes come from the $A_{\max}$, and each entity has different dimensions, because some of the attributes are missing from the actual example.

The man made classifications can make a fork in the implementation, because there are more than one possibilities to describe these. The simply solution gives exactly one man made classification per example, therefore the example which belongs multiple classes need a workaround to describe. This could be made on two ways. The first way clones the example several times to make one for each class where its belongs. This way has the next disadvantages: It makes the learning set of the classifier confusing because the same entity has multiple class definitions; and an example with many man made classifications keep down the other ones which has only one or two. The second way solves these problems, but on an expensive way. Namely this technique holds only one man made classification per class but it has to decide which one. The disadvantages of this method are the next: it reduces the examples for the classes; and it is usually chose a random class form the man made classification list to categorize the example, meanwhile it should make the class distribution like before to get correct results from the classifier which would learn the examples generated with this method.

The only satisfactory solution is to have a list of the man made classifications for each example. But it has disadvantages, because usually the classification techniques defined to use or return only one class per example, so these need some work on it to support multiple classes for their input and output. The previous sections gives some hints about results with multiple classes see pages

11 and 16 to know how these methods work.

All of the class description techniques implemented and replaced by the next one during the incremental development of the *DataService* class, now the *Document* class, which is defined in the *DataService* to represent the examples, has three properties: the *Name* which identifies it, the *ClassList* ($\overrightarrow{R_{k,MM}}$) which stored for every examples as it defined in the previous paragraph, and the *TermOccurrences* which shows the occurrance numbers of certain attributes from $A_{\max}$ in the actual document. See figure 3.7 on page 42 for more information about these classes.

An instance of the Document class will occupy about 2 times bigger amount of memory than the entity is copied to the memory without separating it to attributes. Before using these classes it is needed to check that the entities would fit into the memory else none of the constructors defined below could do its job.

3.2.2 The descendants and their constructors

These classes has only functions to implement their constructors easier. Therefore these are not necessary to understand the concepts of the system's bulding blocks. There is 2 implementations available this time which are different only the way of they are fetching the data from its source.

Usually the data sources contains information in little pieces, and the example retreival only available on their basis. These pieces are smaller than a document and contains information only about one man made classification but they have an identifier about the document which has them. The *Name* value of the *Document* is used to store this identifier, and later all of the information pieces come to its document through that. Thus the *Name* have to be a unique identifier in all of the example sources because they could have set operations which uses these values too. The *Name* value can be interpreted easily in both implementation.

Unfortunately most of the attribute reduction techniques has problems with frequent words[1]. There is a simple technique which significantly increases the preciseness of the classification's results but its need more help from the users. When the data is prepared and ready to load to the system its need some steps before the learning. The database has to be loaded and then a filtering technique has to be used to find the best attributes which describes the diversity of the examples. The technique usually fails first because it is bothered some really frequent word which does not mean anything in the current context. The

[1]These constructors load only words as terms and do not contain any solution to find frequent word sets to achive better results in filtering and classification

user can collect the stowaways and make a set of terms from them. After that the whole procedure from the beginning could be repeated, but the constructor have to know about these terms through the paramater which called *MissMe* and *MissList*, because during the loading they can be filtered out. This should be repeated until none of the actual language's frequent terms appearing in the list of the relevant terms. Usually every language has some really frequent word these can be collected and stored in separate files. Then if a new data mine has to be extracted the user just simply gives its language and the filter should do its work on the right way.

The two descendants reveal different ways of gathering information, the first one is based on filesystem managed by the operating system. This one easily can be extended to gather information from the Internet for example follow links and analyse hyper text files. The second one is much closer to OLAP systems because it use SQL to retrieve the examples from a DBMS software.

DirsToFacts

There are two ways to initialize a DataService through a filesystem these are different only in the notation system in their parameters. When the DirsToFacts is constructed the file names and the classes comes from different places but the background are the same in both cases. The first parameter of the two different constructors are described here:

- The first constructor gets only a string on this parameter, and this string describes a directory which is the root of the directory structure where the documents are categorized by directories. The root directory has to get as many directory as many classes are in the O set. These directories should contain only files (as entities ($\overrightarrow{\epsilon_k}$)), because the subdirectories of the directories, which represent classes, are ignored. These files has the name which will stored in the *Name* field of their *Document* class. If a there are files with the same name in other subdirectories of the root dir, then the second and more appearence of the file are ignored and only the class name, which is the name of the subdirectory where it was found, will be stored.

- The second constructor has nothing to discover just do the dirty work because it has in its parameter a map which shows the entities, as keys of the map, and its class set, as values, too. So it just load the file what is described in the key, then makes the document with the attribute vector comes from the file and copies the class set which it has had jet from the value. Here the *Name* of the *Document* will be the filename described in the key without its pathname if it contains one.

The class contains a static servant function which could be usable for the other parts of the data mine. This function loads all of the words of its parameter's file and returns with their occurrence values as they would be attributes. The name of this function is *SzoSzamol* and it could be used for loading the word list to miss for the needed language.

SQLToFacts

This class uses Oracle Net8 component and the Java's JDBC driver for this component to reach the local copy of the Reuters news agency's database at the University of Miskolc. This database has the ER modell which has shown on the figure 3.2. When it is initiated the next query will run automatically on the server:

$$\pi_{\text{description,title,word,occurrence}} \\ (\text{words\_in} \bowtie_{\text{words\_in.itemid=doc.itemid}} \\ (\text{doc\_bip} \bowtie_{\text{(doc\_bip.itemid=docuemt.itemid)} \wedge \text{(doc\_bip.code=bip.code)}} \\ (\text{document} \bowtie_{\text{type}='\text{T}'} (\text{bip}))))$$

The database has 2586 (E) documents categorised into 123 (C) classes, so it is usual that an entity has more than one man made classification. The number of appeared words in the whole example set is 24527 ($|A_{\max}|$).

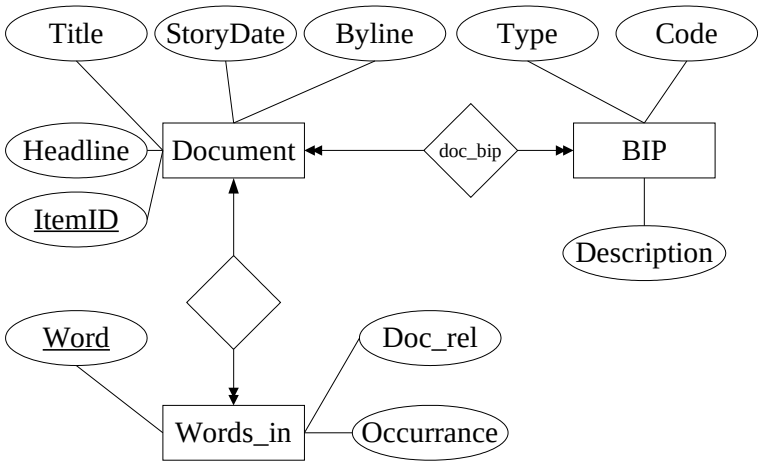

Figure 3.2: The ER model of the database

3.2.3 The ways of gathering examples from the source

Figure 3.1 on page 24 shows two ways to reach the examples. The first way is usually used by the filtering class which needs the whole example set to search the relevant terms so these classes will use the raw example set. The second way is a little bit different because the filtering has effects only on the entities, and its man made classifications will be in the same form as they are loaded. Thus the *DataService* has three methods to do these things:

getRawDoc(k): With this function the filter program will simply have the k^{th} example represented by an instance of the *Document* class.

myClass(k): With this function the classifier or the accuracy checker could get the $\overrightarrow{R_{k,MM}}$.

myAttribs(k,A): This function has the most complex source code, because it transforms the stored entities from the attribute space $A_{\max}$ to A which should come from the filter. Here each attribute in A has a limitation about its value which helps the program to return attribute values like the classifier awaits for. Its a really rarely used function because it is needed only when the A set was defined outside of a filtering method, but this is usual when the attribute value sets were designed by the user.

3.2.4 Set operations with more sources

Each classifier has a learning example set which is represented by a *DataService* object. To get the optimal classification rule this object should be as big as it can, but before it becomes confusing. But a *DataService* object mainly for only one source, so there is a need for join the examples from different sources in one service. And here comes the role of the *Name* property of the *Document* class, because two *DataService* objects will have a union where no examples with the same identifier.

The test sample accuracy estimate which is described on page 19 needs to separate the example set to a learning and a test sample part. To support this estimate the *DataService* class has a special function which initialises a new one with some of its elements. The new size (E_1) of the original example set has to be defined. So the new one's size will be the next: $E_2 = E - E_1$. After the operation each set must have the probability distribution like the original one, to fulfil this criterion the program need to select the elements of the new set at least randomly from the original set. To compare two classifiers with Θ^{ts} each running of this procedure has to use the same random seed.

The leave one out estimate (Θ^{lvo}) needs every example to be separated from the example set once while it is on progress. To do it fast the system needs a function to separate only one but randomly accessed element from the example set. This function should be used when a newly arrived example is learned by an adaptive classification method because if something change (for example the filter) the set will be up to date during the relearning.

3.3 Filtering

The scenario of the dimension reduction of the entities has two phases. The first phase generates the list of the relevant terms (or attributes). This list will contain all of the attributes from $A_{\max}$ so this needs to calculate relevance values ($\rho(a_i)$) for every element of $A_{\max}$ and then the attributes in the list should be ordered by these (ρ) values. After the initial phase, which is solved by the constructors and the *relevanciaSzamitas()* function, an example could be asked from the filter, and the answer will be an entity from the space of the N_L most important attributes which are on the head of the list.

These are the main objectives of the *RelevanceCalculator* class. But it should be extended because it is not contain any calculation methods, because it is the frame for their descendants which usually implement the relevance calculation method only in the *relevanciaSzamitas()* function. Thus it is only for operating with the list of the attributes:

getAttribs(): This function will retrieve the first N_L attributes from the list. This could be useful when the user want to decide which terms should be throwed out during the loading of the example set. Because if the list contains terms, which seem not too important just too frequent, could be collected and passed to a *DataService*.

getFilteredDoc(): This function has the same goal like the one which was defined in *DataService* and called *myAttribs()*, but this is much faster because it assumes the maximal value of the attributes and it uses the N_L long head of the list of the relevant attributes so it has some limitations and some advantages. This is usually called by a classifier not a user created program because the classifier needs filtered learning examples to finish the building of the classification rule as fast as possible, and it can not build any attribute list to increase its accuracy.

getLevel(), setLevel(): These functions are to handle the $N_L = |A| = W$ value.

toString(): This will produce a string with a comma separated list of the ordered attributes from $A_{\max}$. This is useful when the calculation method should be checked, it provides a basic method to do it. But the string which comes from this function could be extremely long because of the big cardinality of $A_{\max}$.

3.3.1 Connection between the filter and the classifiers

Classifiers highly depends on the A, and the Filtering classes could easily change that. It should be ambiguous output when a classifier gets entities from a different attribute space than it learned from. So this should be stopped. There are some techniques doing this, but this paper will focus a basic and an advanced method only. Here they are:

Multi-threaded listener: First of all with this method the system needs a new thread for checking the filters. If a filter changes its state the classifiers — based on it — should rebuild their classification rule. But if the filter changed nobody should access the classifier, because it could be faulty, thus the program needs synchronisation techniques to stop using the rule until its becomes good. This technique spreads the needed code to the whole system, and it makes the code heavily maintainable. And the biggest disadvantage of this technique is the next: it consumes the CPU for checking sets every time, when it is not needed too.

Event driven listener: The modern operating systems are event driven usually, because this brings the solution near to its cause, so its concentrating the code around the changes. In this case the only thing which can change the A is the function called *setLevel()*. Thus it is needed to inform all of the involved objects. Each object must say if its involved when it begins to use the A provided by the filter which will change the behaviour of the *setLevel()* function because it will be costly if many classifiers using its filter. But this is the only way to anticipate the messy classification results.

The easiest method to defend the classifier is to write the documentation about the *setLevel()* more precise. But if the user forgot the explicitly needed learning when it applies *setLevel()* then it would cause unexpected results.

Therefore the implementation uses the event driven listener technique, which needs the ability to tell the filter about the newly involved classifiers. This ability has implemented in the function named *addReLearnListener()* which has a pair to remove the not classifiers which are not depending from the filter anymore, this function is called *removeReLearnListener()*.

3.3.2 The descendants

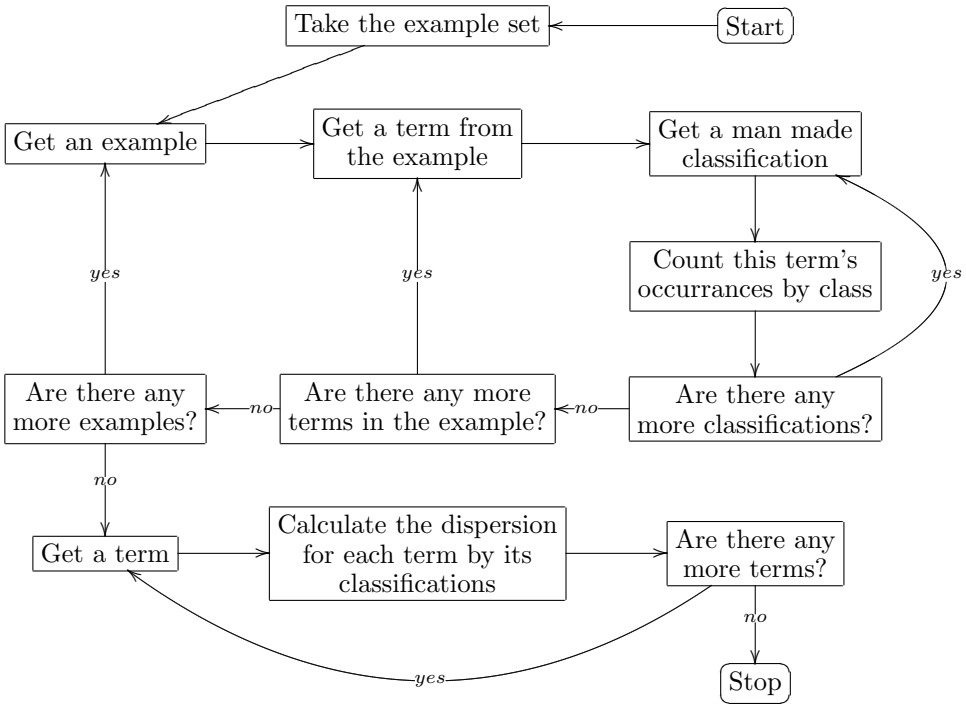

Figure 3.3: The flowchart of the Entity based filtering

All descendants of the *RelevanceCalculator* class have to implement the *relevanciaSzamitas()* method which should rank all of the terms of the learning example set while the constructor is running. Table 3.1 shows the implemented filtering methods costs, their efficiency will be discussed later in section 4.2.1 on page 4.2.1. To see how attribute reduction works read section 2.4 starts on page 20. To see how these connected to each other and the other parts of the project check figure 3.7 on page 42. The cost functions was calculated from the flowcharts of the actual descendant, the figure 3.3 shows flowchart of the most involute filtering method.

Relevance calculation method	Class of its implementation	Cost function
Entity based (ρ_E)	*FileBasedRelevance*	$O(\|A_{\max}\| \cdot (E+3) \cdot C)$
Class based (ρ_C)	*LocalRelevance*	$O(\|A_{\max}\| \cdot E \cdot C)$
Term based (ρ_T)	*GlobalRelevance*	$O(\|A_{\max}\| \cdot E)$
TF-IDF (ρ_{TI})	*TF_IDFRelevance*	$O(\|A_{\max}\| \cdot E \cdot 2)$

Table 3.1: Relevance calculation methods and their cost functions

3.4 The classifiers

3.4.1 Basics

The classifiers have many relatives, but everything begins with two interfaces. The first one defines the classifier and the other one provides the listener which is needed to use the event driven (see section 3.3.1 on page 31) programming thechnique.

The *Classifier* interface

All of the classifiers has the example set in a special form which is described in the classifiers basic interface. The elements of this representation are *Fact* objects which has entities from A. The classifier has the right to hold this set to make its work easier, but usually its just memory waste after the learning finished.

The interface called *Classifier* specifies two functions too. The first one is for the main aim of this paper because it is for applying the classification rule defined in section 2.1 on page 9. This is the product of the learning implemented in the descendants. The output of the classification differs a little from the definition, because it will provide only the names of the classes which seems to be appeared in the newcomer.

The other function defined because the ultra-strong criterion of Mitchie (see section 2.2 on page 6). The classification rule should be printable if a person could use it so it has to be implemented to help the user make its own classifications with the computer generated rule.

The *ReLearnable* interface

The interface called *ReLearnable* is the other basic for classifiers. This is not needed for every classifier, but it is for comfortable use of the implementation.

This interface has only one function which could ensure with the usage of *addReLearnListener()* that the filter is the same like the one during the learning.

The *AbstractClassifier* class

On the basis of these two interfaces here comes some real program, which will do something. The *AbstractClassifier* class do a couple of things to make the writing of a classifier easier. This will make the dirty work about the *ReLearnable* interface, because it tells the filter, which comes as a parameter in the constructor, that the newly created object is involved about its changes.

It contains a function to generate the learning example set with *Fact* objects. Here it contacts the filter and gathers the entities in the A attribute space and it also contacts with the example source to collect the man made classifications of these entities. The examples without any relevant terms[2] in their entity, will be excluded from the learning set.

This contains a get and *setMargin* function to operate the values which was defined in section 2.3.2 on page 11 as μ. This is a commonly used parameter to get multiclass results. It is usable in Bayesian classifiers and of course in decision tree based classifiers too.

At last there is a method which helps to check the accuracy of the classifier with calculating of the accuracy rating (see formula 2.7 on page 19) of the recently classified entity. The implemented accuracy rating calculation method applies the function with $\theta_{k,d,1}$. This function do not do any classification so the user has to provide the result of the classification rule ($O_{k,d}$) and the man made classification ($O_{k,MM}$) for the k^{th} entity.

3.4.2 Bayesian classifiers

The implementation of this classifier could be done on several ways, the fastest way is to store all of the conditional probabilities needed by the classification rule in the memory. But to enable adaptability these probabilities has to be changed thus extra information has to be stored about the examples. The original solution should be replaced by a new one which is not about the conditional probabilities but the values to calculate them. Every time when an attribute (a_i) appears the program will note ($N_i = \sum_{k=1}^{E} v_{a_i,k}$) it with an additional information about which classes(o_l) were active ($N_{i,l} = \sum_{k=1\ldots E, r_{k,l,MM}\equiv 1} v_{a_i,k}$) when the term appeared. All examples will be counted too on two different ways with their classes ($N_l \sum_{k=1}^{E} r_{k,l,MM}$) and without them (this means E).

[2]These terms are stored in A after filtering $A_{\max}$

The implemented constructor's cost function will be the next:

$$O(EW + W + EWC + EC + WC) \rightarrow O(EWC) \tag{3.1}$$

The way of the constructor's implementation determines the technique to follow when the classification rule has to be applied. This could be a simple calculation based on the conditional probabilities which were defined in equation 2.4. Or it should calculate the needed conditional probabilities every time when it has to classify a newcomer. This calculation based on the values made by the constructor:

$$NBCR(k, l) = E^{-1} N_l \prod_{i, v_{a_i,k} > 0} N_{i,l}/N_i \tag{3.2}$$

The cost function of the classification is the next:

$$O(W + WC + C) \rightarrow O(WC) \tag{3.3}$$

The only function which has not appeared yet is the toString. Writing out the conditional probabilities of every class and attribute pair is a waste of time because the person who has to use it would be frustrated with the big amount of data. The output should be readable by somebody so only the values N_i, $N_{i,l}$, and N_l will be displayed, and with small attribute and class numbers it could be usable.

Adaptability

The stored values makes it easy to implement the adaptive classifier based on the Bayes rule because the implementor need to adjust the N_i, $N_{i,l}$ and N_l values correctly. Adaptation of these values to the new environment could change the results of the classification because of the equation 3.2. The cost function of learning a newcomer is the next: $O(1)$.

The interface which shows this classifier is an adaptive one has the name *CanGrow*. If this is implemented the class will have a new function to *learn* a new example.

3.4.3 Decision tree based classifiers

All of these classifiers using decision trees so they have got common parts to manage the tree which parts are defined in the basic *ID3* class. If the tree had built the result would have used on the same way independently from the actual implementation. These were the common design patterns during the set up of these classifiers.

These classifiers could handle non binary attributes, and with these the generated trees could be easier to read and use by a person, but these are equivalent in knowledge representation so the program asks the filter to deliver only binary attributes.

[1] The decision trees usually apply attributes with finite value set. If there are some attributes with infinite value possibilities then the builder has to find out how to separate their value sets to finite subsets and the tree's decisions will about which subset contains the actual value (see figure 3.4). There could be an extension like this one and the tree should be usable after this extension builds up the tree.

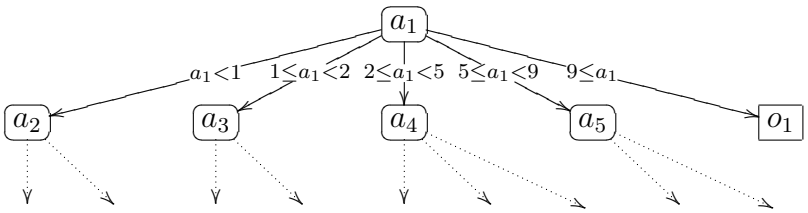

Figure 3.4: Possible usage of an infinite value set

The building blocks of the tree

A node from the decision tree represents an attribute or a class distribution. These will be stored in a *NodeDescriptor* object as a string or an occurrence map. To describe a tree the branches has to be stored somewhere. There are two ways to describe a branch of the tree. The first make a set of pairs with nodes these pairs defines a branch between the affected nodes. This representation is hardly useful because the tree should be accessed through the nodes and in this representation one node will be available as many times as many connections it has. To avoid this problem all of the branches begins from a node should be described locally thus if a node represents an attribute it has to contain branches to the next attributes in the tree too. With this storage method each node should be accessed through another except the root which has to be stored in a special place because every method using the tree start with it.

These are the main concepts during the design of the *NodeDescriptor* class. This class has an attribute (a_i) storage called *Name* which is null if the node has to represent class distribution. The class distribution stored in the *Occurrences* member of the class which is a *TreeMap* with Class Names (o_l) as keys and occurrence numbers (N_l) as values. At last the branches are stored in the *Children* member which has the attribute values (v_{a_i}) as keys and nodes with the same value in its examples as values.

To implement adaptability now only the class distribution could be changed when a wrongly classified newcomer has to be learned. The [1] describes the nodes of decision trees as a group of examples with the same value on specific attributes. So these examples should be stored also in the node and then the changes has to be made on these sets which make it possible to rebuild the wrong parts of the tree. This will be stored as a *Vector* of *Fact*s named *ActualDocs*. Check figure 2.3 on page 15 to know how it could be represented.

Usage of the tree

The usage with the nodedescriptor's actual implementation is really simple because if a node contains an attribute then the program has to check the entity which has to be classified to identify the actual value of the attribute and go to the next node which stored with that key. And after that the program has to be recurse with this function until it find a class distribution to become the result. The cost function of this recurse function will be the next:

$$O(W + W + C + C) \rightarrow O(W) \tag{3.4}$$

The other application of the decision tree is to gather the classification rule in a human readable form. The program could be draw the tree to the display but it is not too usable, so it should be transformed to if-then rules for every class. To get this rule set the tree has to be used backwards from the leaves to the root because every leaf has to be used to produce the correct set. The recursion function defined above should be applied to reach each leaf of the tree. And the decisions needed to reach a leaf should be written to the output. The decision trees made by an extension of this class should be well pruned to be useful for humans.

Cost function of the construction

The constructor make the decision tree during its operation. Every recursion of the CLS produces a node (to see how check section 2.3.3 on page 12). To calculate the cost function of the basic ID3 algorithm there are some measures which are used only here, these are the next:

- The E_i is the cardinality of the example set of the actual (i^{th}) iteration.
- The W_i is the cardinality of the attribute set of the actual iteration.
- The $V_{\max}$ is the average cardinality of the attribute value sets. ($V_{\max} = \frac{1}{W}\sum_{j=1}^{W}|V_{a_j}|$)

All of the cost functions holds the next measures and it has usual relations like this: $V_{\max} \ll C \ll W \ll E$ (therefore $V_{\max}$ and C could be neglected if they are not multipliers of the others). Any iteration of the ID3 has the same recursive cost function where R_i is the i^{th} iterative step: $R_i = E_iC + W_iV_{\max} + W_iE_i + W_iC(2E_i + V_{\max}) + E_i + V_{\max}R_{i+1}$. The whole cost of the iterations could be calculated as follows: $O(R_0)$. This cost function has two limits, which are these:

The best case: The builder find a split which separates the examples into homogeneous subsets in the first iteration. If that happens the second iteration becomes formal because it would identify the case that all of its examples belongs to only one class which will transform the R_1 into the next degenerate form: $R_1 = E/V_{\max}C$. This case has the next cost function:

$$O(2EC + WV_{\max} + WE + WC(2E + V_{\max}) + V_{\max} + E) \rightarrow O(EWC) \quad (3.5)$$

So the best case of the ID3 is the usual case of the Bayes (3.1). This case means that the attribute which is chosen first is the class identifier itself with another name, so it is not too usual. There is a real vestige of this function which usually appears when the actual node becomes a class distribution, this has the cost function like R_1 and its called R^*. This can be evolved from R_1 using the example set of the actual iteration instead of the whole.

The worst case: The tree could be built with all of the possible nodes, which means all of the attributes used from the root to the leaves ($R^* \rightarrow 0$). The cost of this case will be calculated below.

In each iteration the values of our measures of the actual steps can be defined like this: $W_i = W - i$, $E_i = E/(V_{\max})^i$. After the K^{th} step all of the attributes has built in the tree thus $W - K = 0 \Rightarrow W = K$. So every attribute value has to get at least an example which leads to the following equation: $E/(V_{\max})^K \geq 1 \Rightarrow K = \log_{V_{\max}} E$. With these assumptions the cost of the iterations of the worst case could be calculated as follows:

$$\begin{aligned} R_i &= EC/(V_{\max})^i + (W-i)V_{\max} + \\ &\quad (W-i)E/(V_{\max})^i + C(W-i)(2E/(V_{\max})^i + V_{\max}) + \\ &\quad + V_{\max} + E/(V_{\max})^i + V_{\max}R_{i+1} \rightarrow \\ &\quad \rightarrow (C + (W-i)(1+C) + 1)E(V_{\max})^{-i} + V_{\max}R_{i+1} \\ R_0 &\approx \sum_{i=0}^{K}(W-i)EC = EC\sum_{i=0}^{W} W - i \end{aligned}$$

And at last here is the cost function of the worst case:

$$O(R_0) = O(EW^2C) \quad (3.6)$$

Adaptability

The probability based classification (*ID3withProbability* class) uses the rough method of making adaptability for decision tree based classifiers which method was defined in section 2.3.3 on page 17. Thus the newcomer has to be classified to know which node contains the class distribution for it, after that the distribution has to be regenerated with the new example to modify the class distribution of the branch used during the classification. Therefore the cost function of newcomer's the learning is the same like the classification's cost: $O(W)$. But usually the ID3 generates trees with few examples on the leaves, so the relative frequencies of the classes could be far from their probabilities. To solve this problem the basic ID3 contains conditions about to make a leaf immediately when they become true (these conditions evolved from the CLS itself for example when the system run out of attributes a new leaf has to be created). These conditions can be modified in the extensions, thus this extension has to change them before the tree has built. The simply extension of these conditions has to stop the growing of the tree when there are not enough examples to calculate the proper relative frequencies. See section 2.3.3 on page 16 for more information about statistically few examples. The *setStatisticallyFew()* and *getStatisticallyFew()* methods are to control the required cardinality of the example set ($E_{i,\min}$) to check for a new attribute node.

The probability based method will build a tree like the original ID3 if the $E_{i,\min} = 1$ but any other cases ($E_{i,\min} > 1$) the tree could be different. This method will leave the structure of the tree untouched, so no new nodes will be defined, only the leaves will change. The construction of the tree with $E_{i,\min}$ will stop the growing of the tree after $L = K - \log_{V_{\max}} E_{i,\min}$ iterations which means the cost function of this extensions' construction will be the next: $O(EWLC)$.

The other way holds the tree in the same condition like the original ID3 does. This means it has to change the structure of the tree whenever it is needed. This extension of the original ID3 called *ID3withReconstructTree*, and it extends the class described above to use the limitation of the example sets. With this technique the newly arrived example will be stored in all of their affected separations and if a separation changes the attribute which has to be used for the next one then the tree have to be rebuilt from that separation. The cost function of the construction an object extended like this is the same as the probability based one, because it is not bothered. But the learning of a newcomer becomes more costly than that, because this starts at $O(W)$ and it could be like the construction of the whole tree if the attribute of the root node changes. Usually it is near to its smaller bound.

3.5 System wide diagrams

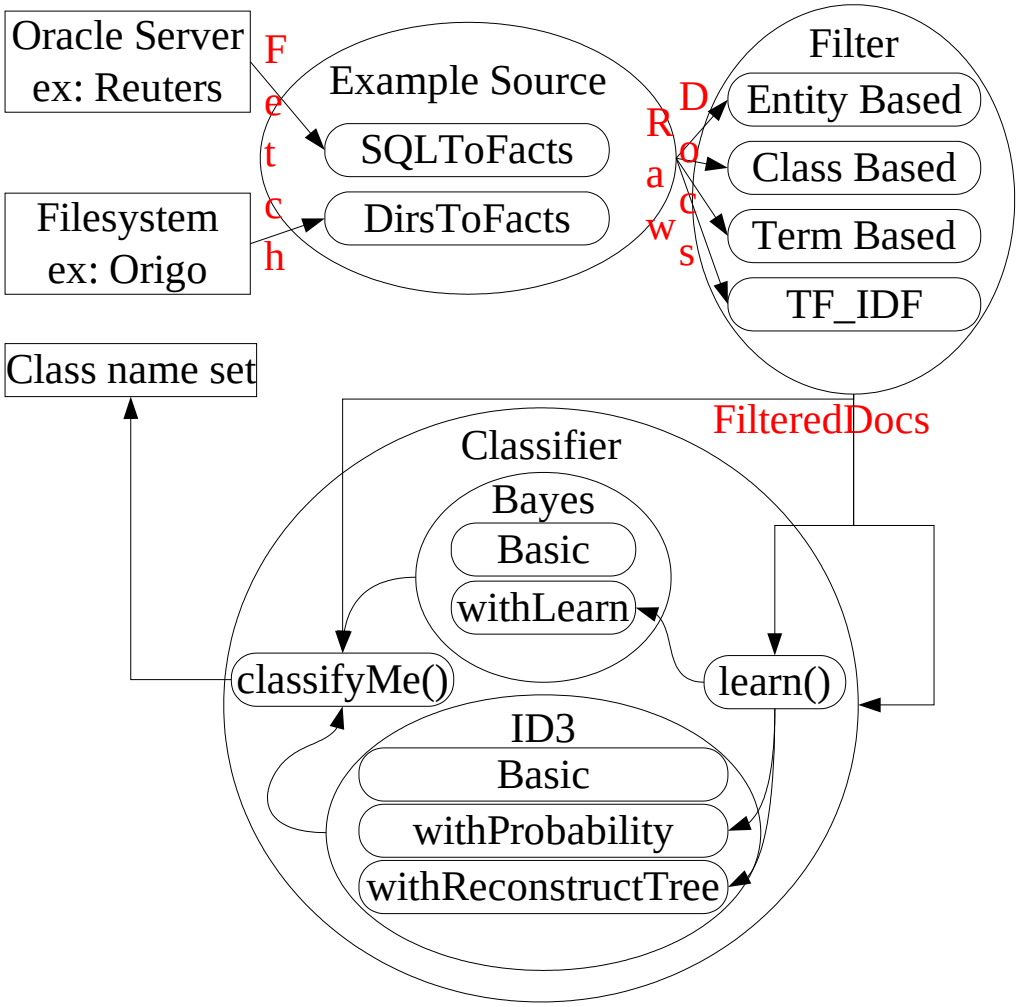

Figure 3.5: The data flow diagram of the project

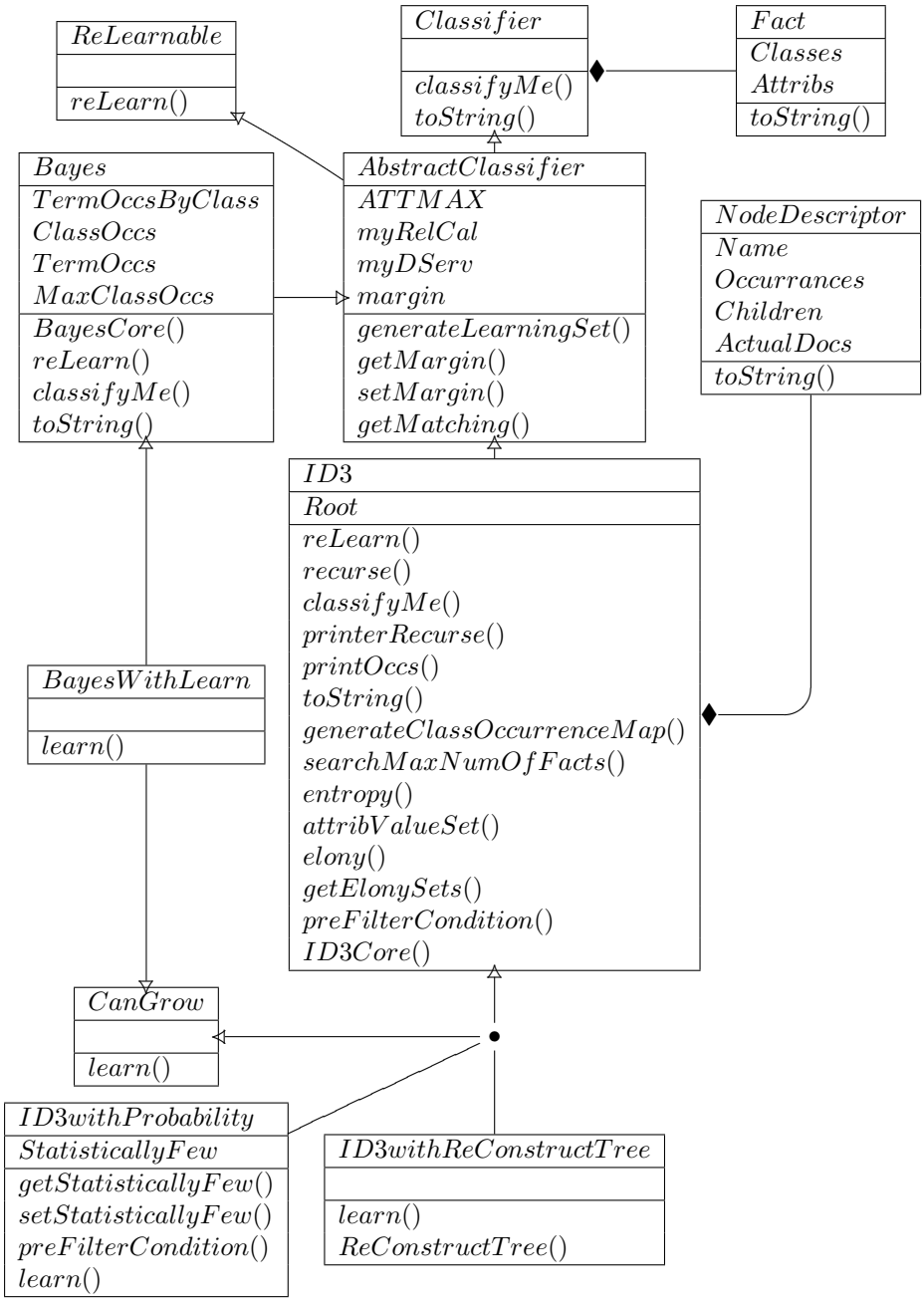

Figure 3.6: The Classifier and its descendants

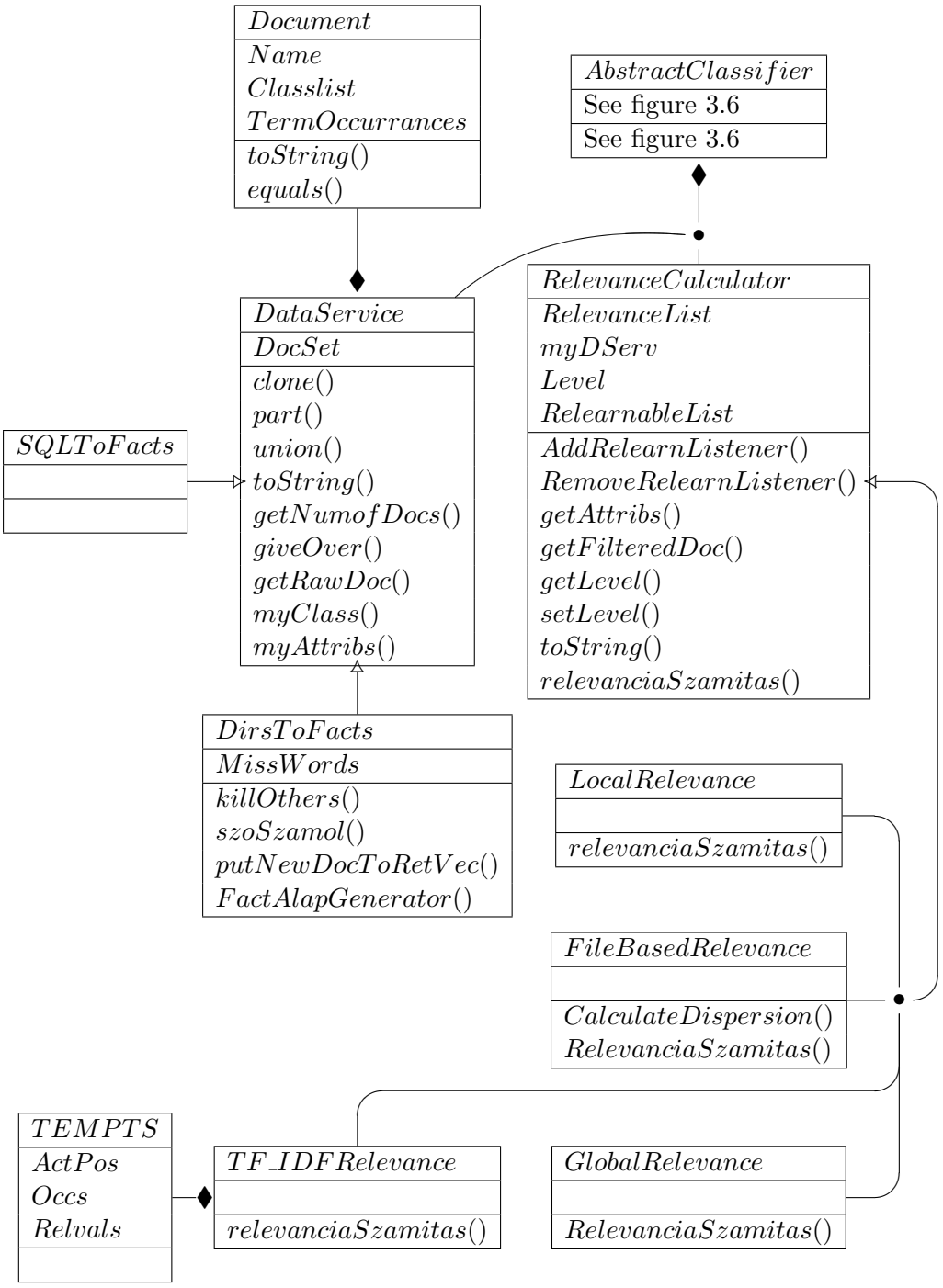

Figure 3.7: The connection between the classifier and the filter/loader

Chapter 4

Testing and Results

4.1 Testing

After each implemented classifier there was two or three tests on it. The filter was replaced with a class which provides 4 words from the example set, and these words was chosen by hand. The decision trees and the conditional probabilities was calculated by hand with this 4 attribute for a little example from section 11.4 in [2] and for a smaller and a bigger database. And the last step was the comparison of the decision trees made by hand and by the computer. If these are the same then its passed on the test and it can be used on the comparison.

4.1.1 Origo

The small database was stored on a file-system to demonstrate the abilities of the DirsToFacts, and it contains 180 articles from the Hungarian web portal named [Origo]. These articles has gathered in two part the first was collected in march 2003 on the same day, and the second in may 2003 on the same day too. The different time of the collecting try to ensure the independence of the data. In these examples the entities has only one man made classification ($|\overrightarrow{R_{k,MM}}| = 1$). The categories was the most interested categories by the portal's visitors based on the page impression data published by the site:

- Crime
- Economy
- Entertainment
- European Union

- News from Hungary
- Politics
- Sport
- Technology
- World news

4.1.2 Skipped Words

The first runs of the filtering method was used for to collect the highly frequent words of the databases which have to be missed. These words are the most frequent words of a language usually. The next words was skipped during the loading phase in the results section:

Hungarian a, az, azonban, azt, be, csak, ez, ezt, mint, még, sem, de, ki, olyan, kell, volt, mert, több, után, fel.

English a, about, after, also, all, an, and, are, as, at, be, because, been, between, but, by, can, could, did, down, for, four, from, had, has, have, he, her, his, if, in, into, is, it, its, just, m, may, more, much, next, no, not, on, one, only, or, other, out, over, s, she, since, some, such, than, that, there, three, their, they, this, to, two, u, up, was, we, well, were, when, where, which, while, who, will, with, would.

Some of the frequent words are not too frequent in these examples to fox the filters so these are not listed here.

4.1.3 Fighting against the cost function

E learning phase required to calculate the leave-one-out accuracy estimate, and one learning phase could be finished after 31 seconds. There are two ways to make this method faster. The first way is easy because less attributes have to be used, but it leads inaccurate results so to calculate this estimate each phase should run on a different machine because these are independent phases (The University of Miskolc has eleven Sun Blade 100 machines to reach the Oracle server which provides the Reuters database and these machines could be the base of the execution). Thus the program needs parallelisation which can be solved on different ways, and here is a list about three of them:

Handwork: This technique is the easiest, because the processes are started by a man and the running parameters provided when the process is started can separate the results of the system.

NFS based: This technique use the capability of the network file system which makes available the files written locally. The parameters are continuously updated by the actual process which changes its state, and to provide the separate results it should occupy a slice of the work.

RMI based: This technique are based on the Java Remote Method Invocation algorithm which makes the file-system based communication unnecessary, and with a correctly designed service the clients can gather a job, which leads a process-farm structure.

Parallel system tool: This technique mixes the first two method and a tool has to be used to make the processes automatically and gather their results to the host machine.

The program has an implementation about the NFS based method which calculates the independent iterations on different computers and each computer has the right to change the central file which stores the identifier of the last job. The results will be collected in an other file.

4.2 Results

The Origo database contains only one man made classification per entity, and there are nine classes in it. So the accuracy of the random generator ($d_R()$) should be the next:

$$\Theta(d_R) = \binom{9}{1}^{-1} = 0.\dot{1}$$

The Reuters database contains 3 man made classifications per entity on average, and these were chosen from 123 different class. So the accuracy of the random generator should be the next:

$$\Theta(d_R) \in \left[\binom{123}{3}^{-1}, 123^{-1} \cdot 3^{-1}\right] = [3.3 \cdot 10^{-6}, 2710 \cdot 10^{-6}]$$

The [Origo] database was collected for testing purposes because the Reuters database is too large. Thus the two best filtering methods, the differences between accuracy ratings and the best margin (μ) value should be selected before the comparison of the classifiers started on the Reuters database.

4.2.1 Deciding the test bed for the Reuters DB

Comparison of the Filtering methods

The effectiveness of filtering methods was checked with both of the base classifiers on the [Origo] database, and the best two filtering method should be the Entity based and the TF-IDF which shows unmatched face of its powers because it could be near in accuracy both the Entity based and the Term based filtering methods depends on the classifier and the accuracy checker technique. The figure 4.1 shows the general differences with the most precise accuracy Θ^{lvo} which was defined in equation 2.10.

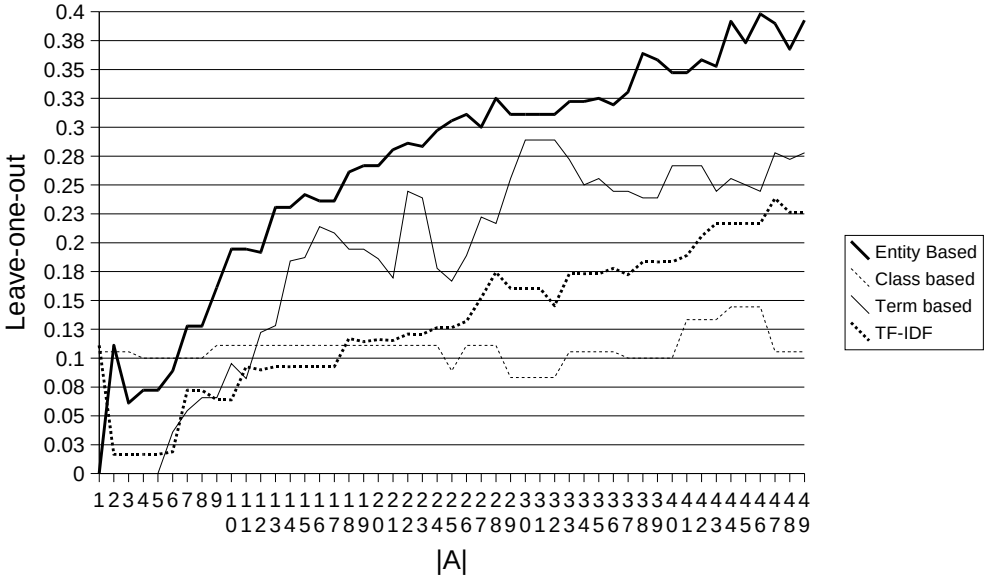

Figure 4.1: The difference between the filtering methods

The other frequency based filtering method have problems not just about their efficiency, but they cause big decision trees too which takes more time to understand. The size of the decision tree could be characterised by the number of its leaves. The next table shows an example where the ID3 has the [Origo] database to learn and twenty attributes was chosen by the filters.

Filter	Leaves
Entity Based	29
Class Based	13
Term Based	120
TF-IDF	49

The class based filter provides too special terms as attributes because these

terms usually contained only one example and this makes unusable of the classifiers based on this filter.

Comparison of the Accuracy values

With the best filtering method and with the basic ID3 classifier the three accuracy calculating techniques were tested. The test sample technique was used with the two third of the [Origo] database as learning examples, and the other part was the base of the calculation. The figure 4.2 shows how misleading the resubstitute accuracy values are. And the test sample estimate (Θ^{ts} ,see 2.9) seems good enough to test the accuracy of the classifiers learned the Reuters database because it is near the most accurate accuracy value which was defined in section 2.3.5: the leaving-one-out estimate (Θ^{lvo}).

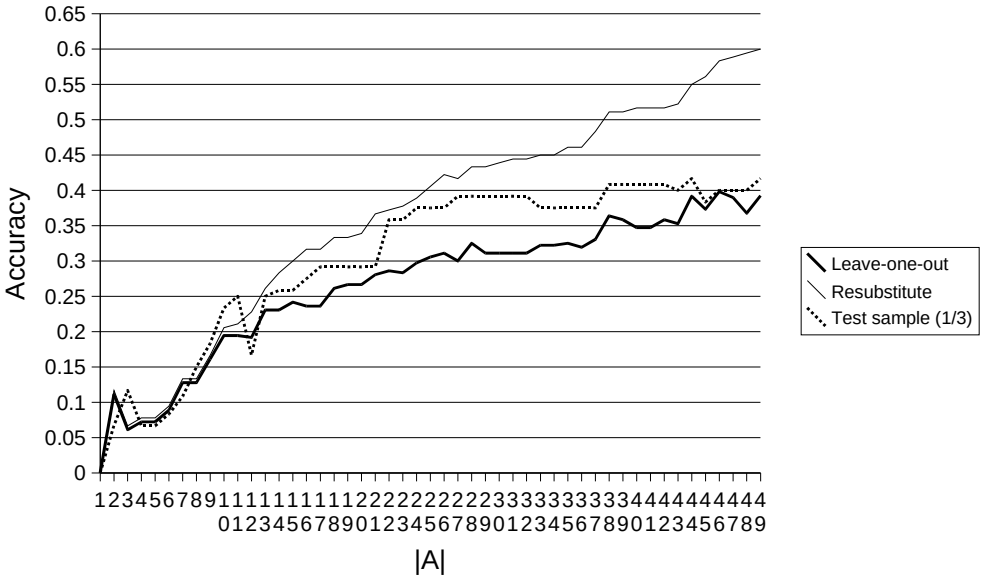

Figure 4.2: The difference between the accuracy values

Deciding the best margin (μ) value

The [Origo] database contains only one man made classification per entity which means this decision can not be made using this source, and the comparison of the classifiers has to contain information about the dependency of the μ too.

4.2.2 Comparison of the classifiers

Here the test sample estimate used with 2/3 of the examples for learning and the others for checking the accuracy. The TF-IDF attribute reduction method

gives the same results as before because it provides averagely 5 percent less accurate results than the Entity based one. The entity based (ρ_E) method was used on figure 4.3

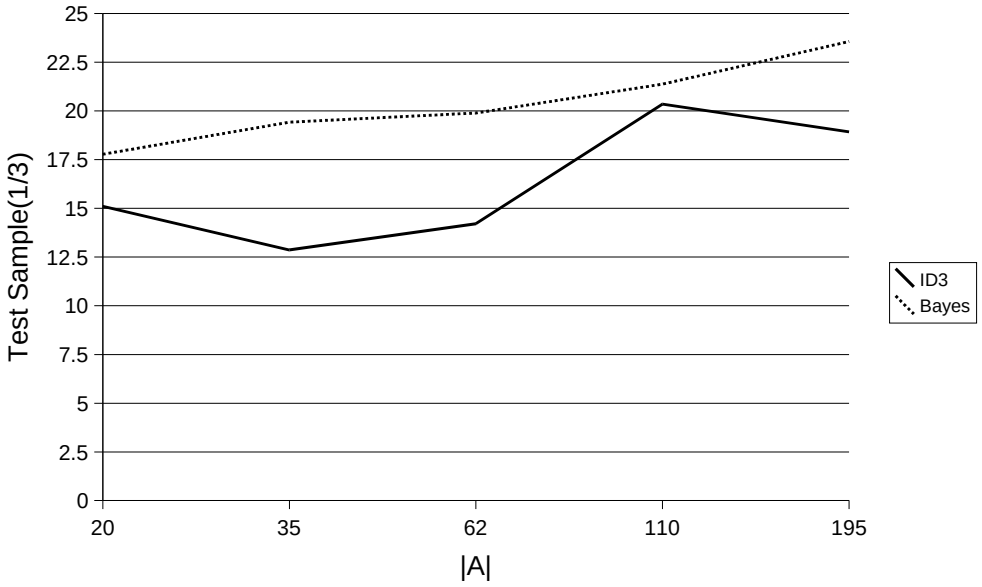

Figure 4.3: Comparison of the classifiers

After that the margin (μ) was subsequently changed from 0 to 1 when the W was 110, to check the dependency of the accuracy. Which shows that the ID3 collects valuable information on the leaves of the tree because if all of the classes used from a leaf the accuracy increased dramatically. The Bayesian classifier has the same effect but usually its results has their maximum when the margin is near to 0.5. The figure 4.4 shows the typical behaviour of these classifiers.

At last the statistically few (see 3.4.3 on page 39) extension of the ID3 gives a simple chance to build the tree faster and make it usable when adaptability comes. If this parameter changes the accuracy falls behind the acceptable levels so it is not usable without the learning ability. See figure 4.5 for the results.

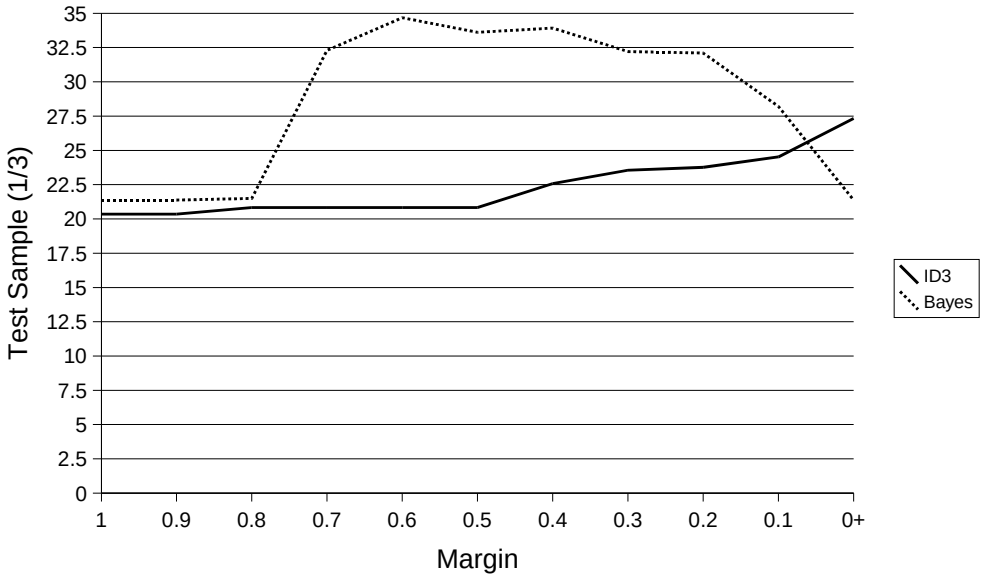

Figure 4.4: Comparison of the classifiers

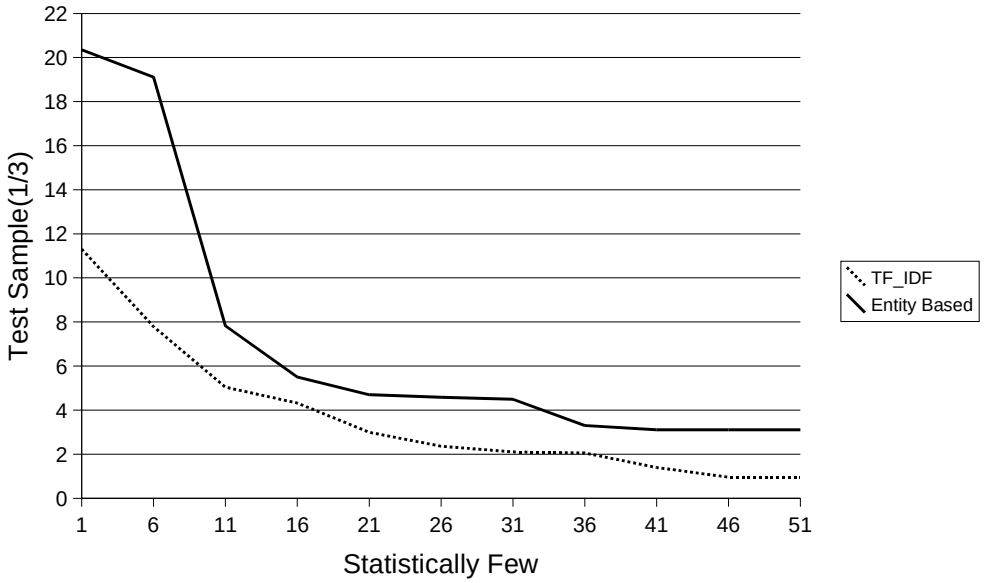

Figure 4.5: The effect of the statistically few extension

4.2.3 Comparison of the adaptive classifiers

The adaptability was defined for faster learning times when the users are not satisfied with the results of the current classifier and they want to add more examples to increase its accuracy. Because the whole learning set has to be learned again which could be a really long procedure. The simulation of this event was really simple, only one third of the examples was used for learning and after that other one third of the examples was used for extending the knowledge of the classifiers. After the extension the last one third of the examples was used to test the accuracy with the usual test sample estimate. The time and the results was checked with $W = 110$, and table 4.1 shows them.

Classifier	Learning time	Accuracy(Θ^{ts})
ID3withProbability	14.7s	7.59%
ID3withReConstructTree	7 hrs	20.5 %
BayesWithLearn	5.4s	21.4%
ID3	2days	20.5%
Bayes	1.4hr	21.4%

Table 4.1: Comparison of adaptive classifiers

These results are based on the fact that the statistically few setting has to be prohibited because the relatively small amount of examples.

Chapter 5

Conclusion and Perspective

5.1 Completed objectives

All of the objectives defined in the abstract solved. And some causative problems had been analysed too.

5.2 Identified problems

Similarly CLS, the Bayesian algorithm throws out the attributes which do not add more information. With a too high number of attributes neither of the implemented algorithms can do the learning in expected time. The CLS based classifiers' learning times are increasing dramatically when some attributes are added confirming the $O(W^2)$ cost function.

The example output in appendix A shows a decision tree transformed to decision rules. On this output the typical problems of the decision trees are appearing. The pruning can be easily done because there are so many branches with join-able parts. The other easily realisable problem is about the source because it provides a lot of words in their inflexioned forms so usually the classifier or the filter notice these words analogy.

5.3 Fields of possible applications

The main application field of this project could be the big categorised document search engines like Yahoo, where this tool could help the people whose made the most of the categorisation process by hand before. But it could be a not too smart search engine too like the WebWatcher.

[7] Nowadays using e-mail could be more and more frustrating because the increasing number of the spam letters. These could be throwed out automatically by learning and using these classifiers which makes easier life. The web browsers could contain a parental lock based on this technique too because the pages which should not be seen by our child contains specific terms.

5.4 Further developments

5.4.1 Parallel Computing

The base theory of CLS makes the ID3 ideal to make parallel programs with it, and its main disadvantage could disappear. Because the separations of the example set could be calculated on different computer so the final tree would be available after each node has finished.

5.4.2 Missing features

- The ID3 could have better results if separate trees would have built for each class.
- The tree could have built with more branches on a simple value and these branches could be weighted like a neuron's input. So the finished learning leads a tree which has branches to strengthen if good classification was made. The tree will work like a neural network with this technique.
- The tree should have the ability to make continuous and non binary attributes. [1]
- To make the tree human readable some off-line pruning method should be implemented which will change the tree after the learning phase. [1]
- With using an attribute thesaurus the terms of the examples could be easily analysable by the filter and this leads better classification results. [6]
- All of the adaptability based on the fact that the attribute set can not change, but usually the attribute set will be the first when the change seems unavoidable. Thus the attribute set should be changeable behind the scenes.
- Visualisation of the decision trees, because these are easier to use, than the rule sets provided now.

Chapter 6

Notation index

a_i	The i^{th} attribute
A	The attribute set
W	The cardinality of A
$v_{a_i,j}$	A possible value of the a_i
V_{a_i}	The value set of the a_i
$\overrightarrow{\epsilon_k}$	An entity. A vector from the attribute space.
o_l	A class identifier
O	The set of classes
Ω	Classification result space
$d()$	A classification rule
$\overrightarrow{R_k}$	An element of Ω
$r_{k,l}$	The classification result for $\overrightarrow{\epsilon_k}$ about the class o_l.
$\overrightarrow{R_{k,MM}}$	A man made classification.
Ξ	An example set
E	The cardinality of the learning example set.
ξ_m	An element of Ξ. This connects $\overrightarrow{\epsilon_k}$ to $\overrightarrow{R_{k,MM}}$
μ	The limit for multi-class result
Φ	Separator function for CLS
θ_k	The accuracy rating of $\overrightarrow{\epsilon_k}$'s the classification
$\Theta(d)$	The accuracy of the d classification rule
Θ^R	The redistribution estimate of the accuracy
Θ^{ts}	The test sample estimate of the accuracy
Θ^{lvo}	The leave-one-out estimate of the accuracy
$A_{\max}$	The set of the possible attributes
$\rho(a_i)$	The importance value of an attribute from $A_{\max}$
ρ_E	Entity based relevance calculation function
ρ_C	Class based relevance calculation function
ρ_T	Term based relevance calculation function
ρ_{TI}	TF-IDF relevance calculation function

Bibliography

[1] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification And Regression Trees*. Chapman & Hall, 1984.

[2] Istvan Futo, editor. *Mesterseges Intelligencia*. Aula, 1999.

[3] Paul Gestwiczki. Classification algorithms. *http://citeseer.nj.nec.com/gestwicki97id.html*, 1997.

[4] Mike James. *Classification Algorithms*. Collins, 1985.

[5] Thorsten Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. *http://citeseer.nj.nec.com/54920.html*, 1997.

[6] Laszlo Kovacs. Document clustering using concept lattice and attribute thesaurus. *http://www.iit.uni-miskolc.hu/~kovacs*, 2002.

[7] Laszlo Kovacs, Tibor Repasi, and Gabor Kecskemeti. Document classification in scope of spam filtering. In *Proceedings of SAMI 2004 (Herlany Slovakia)*, 2004.

[8] Ryszard S. Michalski, Ivan Bratko, and Miroslav Kubat, editors. *Machine Learning and Data Mining: Methods and Applications*. Wiley, 1998.

Appendix A

Example output of the ID3

```
If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=0
   and iss=0 and intermonth=0 and dutroux=0 and soymeal=0
   and bushels=0 and cbot=0 and melissa=0
   then
      P(COMMODITY MARKETS)=0.3333333333333333
      P(MARKETS)=0.3333333333333333
      P(SOFT COMMODITIES)=0.3333333333333333

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=0
   and iss=0 and intermonth=0 and dutroux=0 and soymeal=0
   and bushels=0 and cbot=0 and melissa=1
   then
      P(COMMODITY MARKETS)=0.5
      P(MARKETS)=0.5

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=0
   and iss=0 and intermonth=0 and dutroux=0 and soymeal=0
   and bushels=0 and cbot=1 and cwt=0 and bushel=0
   then
      P(COMMODITY MARKETS)=0.3103448275862069
      P(CORPORATE/INDUSTRIAL)=0.034482758620689655
      P(MARKETS)=0.3103448275862069
      P(REGULATION/POLICY)=0.034482758620689655
      P(SOFT COMMODITIES)=0.3103448275862069

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=0
   and iss=0 and intermonth=0 and dutroux=0 and soymeal=0
   and bushels=0 and cbot=1 and cwt=0 and bushel=1
   then
      P(COMMODITY MARKETS)=0.3333333333333333
```

```
      P(MARKETS)=0.3333333333333333
      P(SOFT COMMODITIES)=0.3333333333333333

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=0
   and iss=0 and intermonth=0 and dutroux=0 and soymeal=0
   and bushels=0 and cbot=1 and cwt=1
   then
      P(CAPACITY/FACILITIES)=0.07142857142857142
      P(COMMODITY MARKETS)=0.2142857142857142
      P(CORPORATE/INDUSTRIAL)=0.07142857142857142
      P(DOMESTIC MARKETS)=0.07142857142857142
      P(MARKETS)=0.21428571428571427
      P(MARKETS/MARKETING)=0.07142857142857142
      P(PRODUCTION/SERVICES)=0.07142857142857142
      P(SOFT COMMODITIES)=0.21428571428571427

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=0
   and iss=0 and intermonth=0 and dutroux=0 and soymeal=0
   and bushels=1 and cbot=0 and bushel=0
   then
      P(CAPACITY/FACILITIES)=0.05128205128205128
      P(COMMODITY MARKETS)=0.28205128205128205
      P(CORPORATE/INDUSTRIAL)=0.07692307692307693
      P(MARKETS)=0.28205128205128205
      P(PRODUCTION/SERVICES)=0.02564102564102564
      P(SOFT COMMODITIES)=0.28205128205128205

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=0
   and iss=0 and intermonth=0 and dutroux=0 and soymeal=0
   and bushels=1 and cbot=0 and bushel=1
   then
      P(COMMODITY MARKETS)=0.3333333333333333
      P(MARKETS)=0.3333333333333333
      P(SOFT COMMODITIES)=0.3333333333333333

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=0
   and iss=0 and intermonth=0 and dutroux=0 and soymeal=0
   and bushels=1 and cbot=1
   then
      P(COMMODITY MARKETS)=0.3333333333333333
      P(MARKETS)=0.3333333333333333
      P(SOFT COMMODITIES)=0.3333333333333333

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=0
```

```
   and iss=0 and intermonth=0 and dutroux=0 and soymeal=1
   and cbot=0
   then
      P(COMMODITY MARKETS)=0.2631578947368421
      P(CORPORATE/INDUSTRIAL)=0.05263157894736842
      P(EXTERNAL MARKETS)=0.05263157894736842
      P(MARKETS)=0.2631578947368421
      P(MARKETS/MARKETING)=0.05263157894736842
      P(PRODUCTION/SERVICES)=0.05263157894736842
      P(SOFT COMMODITIES)=0.2631578947368421

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=0
   and iss=0 and intermonth=0 and dutroux=0 and soymeal=1 and cbot=1
   then
      P(COMMODITY MARKETS)=0.3333333333333333
      P(MARKETS)=0.3333333333333333
      P(SOFT COMMODITIES)=0.3333333333333333

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=0
   and iss=0 and intermonth=0 and dutroux=1
   then
      P(GOVERNMENT/SOCIAL)=0.5
      P(INTERNATIONAL RELATIONS)=0.5

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=0
   and iss=0 and intermonth=1
   then
      P(COMMODITY MARKETS)=0.3333333333333333
      P(ENERGY MARKETS)=0.3333333333333333
      P(MARKETS)=0.3333333333333333

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=0
   and iss=1
   then
      P(BONDS/DEBT ISSUES)=0.14285714285714285
      P(CORPORATE/INDUSTRIAL)=0.2857142857142857
      P(EQUITY MARKETS)=0.14285714285714285
      P(FUNDING/CAPITAL)=0.14285714285714285
      P(MARKETS)=0.14285714285714285
      P(PRODUCTION/SERVICES)=0.14285714285714285

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=0 and olds=1
   then
      P(CRIME, LAW ENFORCEMENT)=0.2
```

```
      P(DOMESTIC POLITICS)=0.2
      P(GOVERNMENT/SOCIAL)=0.4
      P(SPORTS)=0.2

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=1 and dutroux=0
   then
      P(CRIME, LAW ENFORCEMENT)=0.25
      P(EUROPEAN COMMUNITY)=0.25
      P(GOVERNMENT/SOCIAL)=0.5

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=0 and sex=1 and dutroux=1
   then
      P(CRIME, LAW ENFORCEMENT)=0.5
      P(GOVERNMENT/SOCIAL)=0.5

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=1 and bushel=0
   and intermonth=0
   then
      P(COMMODITY MARKETS)=0.3076923076923077
      P(CORPORATE/INDUSTRIAL)=0.019230769230769232
      P(ENERGY MARKETS)=0.28846153846153844
      P(EQUITY MARKETS)=0.019230769230769232
      P(MARKETS)=0.3269230769230769
      P(METALS TRADING)=0.019230769230769232
      P(PRODUCTION/SERVICES)=0.019230769230769232

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=1 and bushel=0
   and intermonth=1
   then
      P(COMMODITY MARKETS)=0.3333333333333333
      P(ENERGY MARKETS)=0.3333333333333333
      P(MARKETS)=0.3333333333333333

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=0 and nymex=1 and bushel=1
   then
      P(COMMODITY MARKETS)=0.2
      P(ENERGY MARKETS)=0.2
      P(MARKETS)=0.2
      P(METALS TRADING)=0.2
      P(SOFT COMMODITIES)=0.2

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=0 and yastrzhembsky=1
   then
```

```
      P(BIOGRAPHIES, PERSONALITIES, PEOPLE)=0.05555555555555555
      P(DOMESTIC POLITICS)=0.4444444444444444
      P(GOVERNMENT/SOCIAL)=0.4444444444444444
      P(WAR, CIVIL WAR)=0.05555555555555555

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=1 and revs=0
   then
      P(ACCOUNTS/EARNINGS)=0.3125
      P(ANNUAL RESULTS)=0.0625
      P(CORPORATE/INDUSTRIAL)=0.3125
      P(PERFORMANCE)=0.3125

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=0
   and shrs=1 and revs=1
   then
      P(ACCOUNTS/EARNINGS)=0.3333333333333333
      P(CORPORATE/INDUSTRIAL)=0.3333333333333333
      P(PERFORMANCE)=0.3333333333333333

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=1
   and sex=0
   then
      P(CRIME, LAW ENFORCEMENT)=0.2857142857142857
      P(DEFENCE)=0.14285714285714285
      P(GOVERNMENT/SOCIAL)=0.42857142857142855
      P(INTERNATIONAL RELATIONS)=0.14285714285714285

If shr=0 and borrower=0 and unaudited=0 and verified=0 and scandal=1
   and sex=1
   then
      P(CRIME, LAW ENFORCEMENT)=0.5
      P(GOVERNMENT/SOCIAL)=0.5

If shr=0 and borrower=0 and unaudited=0 and verified=1 and scandal=0
   then
      P(GOVERNMENT/SOCIAL)=0.9722222222222222
      P(WAR, CIVIL WAR)=0.027777777777777776

If shr=0 and borrower=0 and unaudited=0 and verified=1 and scandal=1
   then
      P(GOVERNMENT/SOCIAL)=1.0

If shr=0 and borrower=0 and unaudited=1 and shrs=0
   then
      P(ACCOUNTS/EARNINGS)=0.29850746268656714
      P(ANNUAL RESULTS)=0.029850746268656716
      P(CORPORATE/INDUSTRIAL)=0.31343283582089554
      P(EQUITY MARKETS)=0.014925373134328358
```

```
      P(FUNDING/CAPITAL)=0.014925373134328358
      P(MARKETS)=0.014925373134328358
      P(PERFORMANCE)=0.2985074626865671 4
      P(SHARE CAPITAL)=0.014925373134328358

If shr=0 and borrower=0 and unaudited=1 and shrs=1
   then
      P(ACCOUNTS/EARNINGS)=0.25
      P(ANNUAL RESULTS)=0.25
      P(CORPORATE/INDUSTRIAL)=0.25
      P(PERFORMANCE)=0.25

If shr=0 and borrower=1 and iss=0
   then
      P(BOND MARKETS)=0.09090909090909091
      P(BONDS/DEBT ISSUES)=0.2727272727272727
      P(CORPORATE/INDUSTRIAL)=0.2727272727272727
      P(FUNDING/CAPITAL)=0.2727272727272727
      P(MARKETS)=0.09090909090909091

If shr=0 and borrower=1 and iss=1
   then
      P(BONDS/DEBT ISSUES)=0.3333333333333333
      P(CORPORATE/INDUSTRIAL)=0.3333333333333333
      P(FUNDING/CAPITAL)=0.3333333333333333

If shr=1 and shrs=0 and revs=0 and unaudited=0
   then
      P(ACCOUNTS/EARNINGS)=0.29365079365079366
      P(ANNUAL RESULTS)=0.14285714285714285
      P(COMMENT/FORECASTS)=0.007936507936507936
      P(CORPORATE/INDUSTRIAL)=0.2777777777777778
      P(PERFORMANCE)=0.2777777777777778

If shr=1 and shrs=0 and revs=0 and unaudited=1
   then
      P(ACCOUNTS/EARNINGS)=0.3333333333333333
      P(CORPORATE/INDUSTRIAL)=0.3333333333333333
      P(PERFORMANCE)=0.3333333333333333

If shr=1 and shrs=0 and revs=1
   then
      P(ACCOUNTS/EARNINGS)=0.3333333333333333
      P(CORPORATE/INDUSTRIAL)=0.3333333333333333
      P(PERFORMANCE)=0.3333333333333333

If shr=1 and shrs=1 and revs=0
   then
      P(ACCOUNTS/EARNINGS)=0.28125
```

```
      P(ANNUAL RESULTS)=0.0625
      P(CORPORATE/INDUSTRIAL)=0.3125
      P(FUNDING/CAPITAL)=0.03125
      P(PERFORMANCE)=0.28125
      P(SHARE CAPITAL)=0.03125
If shr=1 and shrs=1 and revs=1
   then
      P(ACCOUNTS/EARNINGS)=0.30612244897959184
      P(ANNUAL RESULTS)=0.05102040816326531
      P(CORPORATE/INDUSTRIAL)=0.3163265306122449
      P(FUNDING/CAPITAL)=0.01020408163265306
      P(PERFORMANCE)=0.30612244897959184
      P(SHARE CAPITAL)=0.01020408163265306
```