Semantic Classification[1]

Anastasia Krithara[2]
Digital Media Systems Laboratory
HP Laboratories Bristol

HPL-2004-182
October 20, 2004*

semantic web, machine learning, document classification

A key challenge in the semantic web is the mapping between different concepts. Many techniques for such mapping exist, but most of them induce a one-to-one mapping, which does not seem to correspond to real world problems. This project proposes a new approach, which tries to use the power of machine learning, and in particular classification algorithms, to solve the mapping task. It introduces a new semantic similarity metric which is used with semantic metadata and classification algorithms. The approach is tested in a real world dataset. Pre-processing of the dataset took place, and in particular feature selection, extraction and representation was implemented, for both content-based and semantic features. The documents of the dataset were classified using the content-based features, the semantic ones, and their combination. The results were compared and they gave us an insight of how semantic features can affect classifiers and traditional features.

* Internal Accession Date Only
[1]MSc dissertation submitted to the University of Bristol
[2]University of Bristol, Bristol, UK

University of BRISTOL

DEPARTMENT OF COMPUTER SCIENCE

Semantic Classification

Anastasia Krithara

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering

September 2004 | CSMSC-04

Abstract

A key challenge in the semantic web is the mapping between different concepts. Many techniques for such mapping exist, but most of them induce a one-to-one mapping, which does not seem to correspond to real world problems. This project proposes a new approach, which tries to use the power of machine learning, and in particular classification algorithms, to solve the mapping task. It introduces a new semantic similarity metric which is used with semantic metadata and classification algorithms. The approach is tested in a real world dataset. Pre-processing of the dataset took place, and in particular feature selection, extraction and representation was implemented, for both content-based and semantic features. The documents of the dataset were classified using the content-based features, the semantic ones, and their combination. The results were compared and they gave as an insight of how semantic features can affect classifiers and traditional features.

Acknowledgements

At this point, I would like to thank all the people who provided me their support during the last 6 months. Firstly, I want to thank my family for their invaluable support. I would like to thank BICAS (Biologically Inspired Complex Adaptive Systems) group, in HP Labs in Bristol, headed by Dr Dave Cliff, who gave me this great opportunity. Special thanks go to Dr. Steve Cayzer who was a great supervisor and was always there to answer all my questions. His help was proved indispensable.

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Anastasia Krithara, September 2004

Tables of Contents

Chapter 1

Introduction

Nowadays, an explosion of information has taken place, which opens up new horizons in the acquisition of knowledge in a wide variety of subjects. However, the huge amount of available data is very difficult to handle. A representative example is the World Wide Web where the search of useful information in a particular subject is very difficult. This is the reason why, several ways to solve this problem have been attempted. One of them, which seems to gain ground, is the use of *ontologies [29]*.

Ontologies are used to describe the semantics of descriptive data (*metadata).* In other words, ontologies are collections of information. They generally have a taxonomy and a set of inference rules. "The taxonomy defines classes of objects and relations among them." "Inference rules in ontologies supply further power" [29]. One problem that occurs is that more than one ontology may be used to describe the same domain, and it is difficult, if not impossible, to make everyone to agree on a particular ontology. So, a way must be found in order to map the ontologies that are semantically related. This procedure is known as *ontology mapping*. The problem of ontology mapping is a key research area for the *semantic web*. The latter hopes to be an "extension of the current web in which information is given well defined meaning, better enabling computers and people to work in cooperation" [17].

Although many techniques exist for inducing a one to one mapping between concepts, the correspondence is unlikely to be this clear cut for real world data. Machine Learning techniques are well suited to dealing with semantic fuzziness, yet typically use mainly content based features, and not, for example, semantic metadata. In this project an investigation of a hybrid approach takes place, constructing and using semantic features as rich inputs to a machine learner. This approach is tested on a real world dataset.

The question we want to answer is if a similarity measure would help in the classification process. We want to find out if fuzziness is better than 1:1 matching. The objectives of this project are to try semantic features to a number of machine learners and then evaluate the results. More than one similarity measure will be tested in order to check their performance.

In other words, we want to achieve three goals:

1. Provide a novel and useful semantic similarity metric for use with semantic metadata and machine learning algorithms.
2. Provide some insight as how semantic similarity metrics can be useful, how they affect machine learners, and how to combine them with traditional features.

3. Use our approach to solve a real world problem.

This thesis is constructed as follows: *Charter 2 gives* the background information that was needed in order to implement the particular project. More specifically, in the first section a description of classification is given and some of the most known classifiers are presented. In the second section, a brief introduction in Semantic web is given, and then in last section, some of the current approaches of schema matching are presented. *Chapter 3* outlines all the necessary details for the implementation procedure. The experimental results are reported in *Chapter 4* and in *Chapter 5* a discussion of the results takes place. The latter also includes the future work that could extend the particular project. Finally, *Chapter 7* includes the conclusion of the project. In the Appendixes there are some parts of the code which was implemented for the purposes of this project and some tables with the achieved results.

Chapter 2

Literature Review

Before describing the work that has been done for the project, in order to understand the key concepts of the project and the proposed solutions, it is necessary to present the work that already has been done in some particular research areas. More specifically, the concepts of machine learning and classification will be presented. Some of the well known classifiers will be explained in more details. Also, the concept of schema mapping will be introduced and its current approaches will be explained.

2.1 Machine Learning and Classification

From the invention of computers, a high interest was showed by scientists, in constructing computer programs that can “automatically improve with experience”. Since then, many algorithms have been invented in this field of computer science which is known as *Machine Learning*. Classification is a subfield of machine learning and therefore, the techniques that are used in the latter are a subset of the general techniques of machine learning tasks.

Classification can be described as a function that maps (classifies) a data item into one of the several predefined classes [1]. A well-defined set of classes and a training set of pre-classified examples characterize the classification. So, the classification process is a 2-step procedure:

- *Training*. In this step a model is being constructed, describing a predefined set of data classes. The training data are analyzed by a classification algorithm in order the model to be constructed.

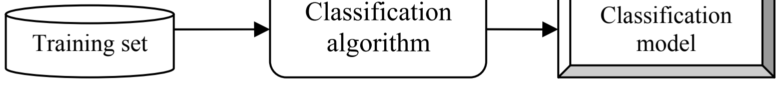

Figure 1: *Training*

- *Classification.* In this step testing data are used in order to calculate the model accuracy. There are several methods to estimate the classifier accuracy. The test data are selected randomly and they are independent from each other. The model classifies the test data and then the known class label is compared with the model prediction about the class. The model accuracy in a specified test dataset is the percentage of the test dataset which has been classified correctly by the learning model.

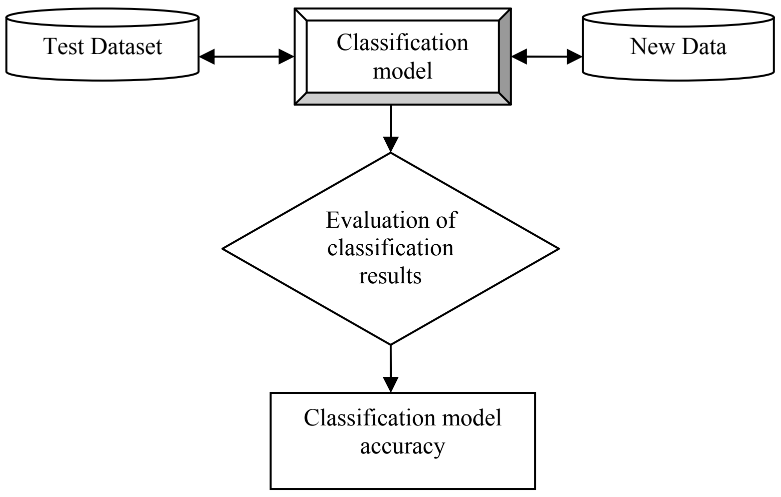
Test Dataset
Classification model
New Data
Evaluation of classification results
Classification model accuracy

Figure 2: *Classification*

If the accuracy of the model is acceptable, the model can then be used to classify other data items. So, we could say that the goal of the classification is to induce a model that can be used to classify future data items whose classification is unknown [1].

2.2 Classification techniques

A number of classification techniques have been developed. We now review some of these techniques, such as: Bayesian classifier, Decision Trees, AIRS (Resource Limited Immune Classifier System), k-Nearest Neighbor and Support Vector Machines. All of them are used in the project in conjunction with our semantic similarity metrics.

2.2.1 Bayesian Classifier

Bayesian Classifier is among the most effective known algorithms for text document classification. It is based on Bayesian statistical classification theory. The aim is to classify a sample *x* in one of the known classes C_1, $C_{2\ldots}$ Cn, using a probability model defined according to Bayes theory. Each category is characterized by a prior probability of observing the category C_i. Also, we assume that a given sample *x* belongs to a category C_i with the conditional probability density function $p(x|C_i) \in [0,1]$. Then, using the above definitions and based on Bayes formula, we define the posterior probability:

$$p(C_i|x) = \frac{p(x|C_i)p(C_i)}{p(x)}$$

An input pattern is classified into a category with the highest posterior probability [2].

The simplest Bayesian Classifier is the *Naïve Bayes Classifier*. "The *Naïve Bayes Classifier* is based on the simplifying assumption that the attribute values are conditionally independent given the target value. The assumption is that given the target value of the instance, the probability of observing the conjunction $\alpha_1, \alpha_2 ... \alpha_n$ is just the product of the probabilities for the individual attributes: $p(\alpha_1, \alpha_2 ... \alpha_n | u_j) = \prod_i p(\alpha_1|u_j)$ " [2]. So, the target value output by Naïve Bayes Classifier (u_{NB}) is:

$$u_{NB} = \underset{u_j \in V}{\arg\max}\ p(u_j) \prod_i p(\alpha_i | u_j)$$

Theoretically, Bayesian Classifiers have the lowest error percentage in comparison with all classifiers. In practice, however, this is not always true, because of the conditional independence assumption which has been made and the lack of available data for the accurate calculation of the conditional probabilities. Nevertheless, researches have shown that Naïve Bayes Classifier is competitive with other well known classifiers, such as Decision Trees and Neural Networks [3].

2.2.2 Decision Trees

Decision Trees are one of the widely used techniques for classification and prediction. A number of popular classifiers construct decision trees to generate classification models.

A decision tree is constructed based on a training set of pre-classified data. Each internal node of the decision tree specifies a test of an attribute of the instance and each branch descending of that node corresponds to one of the possible values for this attribute. Also, each leaf corresponds to one of the defined classes. The procedure to classify a new instance using a decision tree is as follows: starting at the root of the tree and testing the attribute specified by this node, successive internal nodes are visited until a leaf is reached. At each internal node, the test of the node is applied to the instance. The outcome of this test at an internal node determines the branch traversed and the next node visited. The class for the instance is the class of the final leaf node [2].

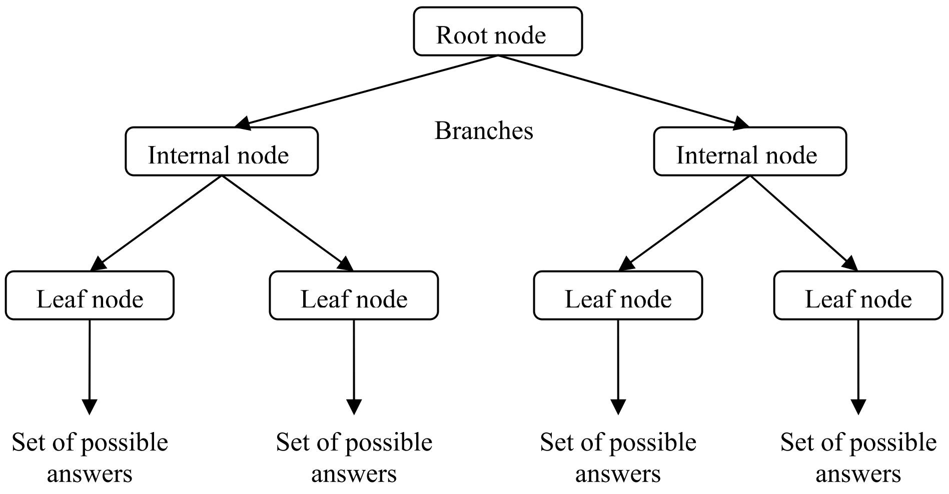

Figure 3: *Decision Tree*

Several algorithms for constructing decision trees have been developed. Some of the most widely known algorithms are: ID3 [2], C4.5 [4], SPRINT [5] etc. In general terms, most of the algorithms have two distinct phases, a *building phase* and a *pruning phase* [2]. In the building phase, the training data set is recursively partitioned until all the instances in a partition have the same class. The result of this procedure is a tree that classifies every data item from the training set. However, the tree constructed may be sensitive to statistical irregularities of the training set. Thus, most of the algorithms perform a pruning phase after a building phase, in which nodes are pruned to prevent overfitting and to obtain a tree with higher accuracy.

The algorithms ID3 [2] and C4.5 [4] are based on a statistical property, called *information gain,* in order to select the attribute to be tested at each node in the tree. The measure definition is based on entropy used in information theory, “which characterizes the (im)purity of an arbitrary collection of examples” [2].

2.2.3 Support Vector Machines (SVMs)

Support Vector Machines provide a powerful methodology for solving problems in nonlinear classification.The theory of Support Vector Machines (SVMs) was first introduced by Vapnik and was developed from the theory of Structural Risk Minimization [7]. “SVMs learn the boundary regions between samples belonging to two classes by mapping the input samples into a high dimensional space, and seeking a separating hyperplane in this space. The separating hyperplane is chosen in such a way as to maximize its distance from the closest training samples” [8]. SVMs have been proved very effective for text categorization because of their property of learning independently of the dimensionality of the feature space [9]. Furthermore, Kivinen et al. [10] have proved both in theory and in practice that SVMs can handle problems with dense concepts and sparse instances, which are often seen in documents vectors.

In addition, SVMs do not need parameter tuning, as they have an automatic way to find good parameters.

2.2.4 k-Nearest Neighbor Learning (kNN)

All the above learning methods are called *eager methods* because they give an explicit description of the target function on the whole training set. On the other hand there are the instance-based methods which are called lazy methods, because the classification model is not built a priori. In their learning procedure they just store all the training instances and in the classification procedure they assign the target function to a new instance. The k-Nearest Neighbor learning (kNN) is the most basic of the instance-based methods. The kNN algorithm stores all available examples and classifies new instances of the example language based on similarity measure. The classification of an instance is most similar to the classification of other instances that are nearby in the vector space, usually in terms of the Euclidean distance [2].

2.2.5 AIRS - A Resource Limited Immune Classifier

AIRS (Artificial Immune Recognition System) is a particular type of artificial immune system, specifically designed for supervised classification tasks. Artificial Immune Systems are inspired by natural Immune Systems. In other words, natural Immune Systems serve as metaphors in Computational Systems [11]. Further details about which aspects of natural Immune Systems are used as metaphors for the Artificial ones can be find in the work of Timmis et al. [13].

AIRS, which is based on kNN, has some properties which make it attractive as a classification system. Firstly, it does not expect from the user to find an appropriate architecture for the system, because it can use its resources according to the data that have been given to it. Secondly, it is capable or generalization. A very important property of AIRS is that the occurring classification system after the training can be significantly smaller than the training set. In addition, it has a number of parameters which can be adjusted by the user according to a specific problem, but as Watkins [14] claims, in most cases the default parameter values give very good results and this can be seen as an advantage of AIRS as it is not necessary for the user to try to find out the best values for them.

All the above properties make the AIRS a very good classifier, which can be compared with many well-known classifiers [11], for example the ones that have been mentioned earlier.

2.3 Information Extraction

In order to apply the above classification methods, we must first extract the information needed from the data we want to classify (XML documents in our case).

This is a very important part of the classification process, because if the extracted information is poor and it does not represent the content of the document fairly well, the classification accuracy results will be also poor. The extracted information is generally represented by a *feature vector*.

We first have to select the features we will extract from the documents. This requires a text processing, depending on the format of the document. In the case of XML documents, the text processing includes the removal of tags, and of some, irrelevant with the content, information that may include. There are also some stemming algorithms, which reduce the number of extracting words if there are two or more that have the same root. For example, if in the documents there are the words, “describe”, “descriptive” and “description”, the stemming algorithm will reduce these words to only one: “descri” which is the root of all the above three.

After the extraction of the features, we have to choose which of them we are going to use in the feature vector. In general, the number of the extracted features is very big and also some of them are not informative at all. So, we have to find a way to select the most representative ones. There are several methods which perform this task. One of them, which was used for the project, is called *information gain*. Given the features, using the information gain we can choose the most useful features. More details about information gain are given in *Chapter 3*.

Finally, a way to represent the chosen features has to be decided. A common one, which is also used in this project, is the use of a boolean vector, where each feature has the value *zero* if the particular feature exists in the document, *one* otherwise.

2.4 Semantic Web

Up to this point, we have given an overview of the classification process and we have introduced some well-known machine learners. An important point about current machine learners is the fact that they are able to deal only with content based features. In order to understand why the latter could be a problem we should first introduce the idea of Semantic Web.

The idea of Semantic Web is “an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation” [15]. In order Semantic Web to implement that, it gives structure to data and it uses ontologies to describe the semantics of the data. Semantic Web is a promising idea but it also has many difficult challenges. A key challenge in creating Semantic Web is the semantic mapping among the ontologies [16]. Trying to implement Semantic Web, different ontologies will be created, and some of them may describe similar domains but use different terminologies. So, we must find a way to know the semantic correspondences between their elements [15, 18].

Semantic Web introduces the problem of ontology mapping. There are many techniques which try to solve the above problem by implementing one-to-one mapping between concepts, but these solutions do not seem to be applicable in real world data, as it is rather rare to have exactly one-to-one mapping. On the other hand, machine learner, as mentioned earlier, may seem a good solution to the problem but they do not deal with semantic metadata.

In the following chapters, we give a more detailed presentation of the schema mapping problem, we introduce a number of solutions that have taken place and we present our own approach to this problem.

2.5 Schema matching

"An ontology specifies a conceptualization of a domain in terms of concepts, attributes, and relations. The concepts provided model entities of interest in the domain. They are typically organized into a taxonomy tree where each node represents a concept and each concept is a specialization of its parent" [16]. In other words, ontology can be seen as the schema of the particular concept. So, instead of the term *ontology mapping* we can use the term *schema matching* which is more widely used. In the particular project, we concentrate in taxonomies and not in whole ontologies.

Schema matching could be described as the process "which takes two schemas as input and produces a mapping between elements of the two schemas that correspond semantically to each other" [16].

On this point, we present the reason why schema matching is important for this project and also the several domains that it is needed in order to understand its importance.

Schema matching gives us the possibility to have different views of the same data. It would be very useful to have the option to choose how the data are going to be presented. Everybody is used to a particular view of some data, so it would be very helpful to have the possibility to change the way some data are represented to the way someone is used to. In order to succeed it, we must know the schema in which the data are represented, the schema we want to be represented and of course have the possibility to match the two schemas. This possibility is the one that has inspired the particular project.

Some other domains where schema matching is helpful are also worth mentioning. A very important problem is schema integration: Given a set of independently developed schemas, construct a global view [17]. Because the schemas are developed independently, it is obvious that they have different structure and terminology. So, on this point schema matching is needed so that the relationship between the schemas will be found and then the integration will take place. Also, schema matching is very useful in data warehouses. "Data warehouse is a decision support database that it is extracted from a set of data sources" [16]. So, we need a way to transform data from the data source format into the data warehouse format. Another domain where schema matching is useful is E-commerce. The participants to a transaction exchange messages between each other. It is almost impossible everyone to use the same message schema. So, schema matching is useful in order to give to the participants the possibility to communicate. Finally, it would be very useful for a user to have the possibility to specify the output of a query and the system can find out how this output could be produced. So, the system should be able to map the user's output to schema elements and then find out the qualification that gives the semantics of the mapping. In order to succeed this, we need, as in the previous cases, schema matching.

From all the above, we can conclude that schema matching is really important and solutions for automatic schema matching should be developed.

2.5.1 Current Approaches of Schema Matching

A review of current approaches and their evaluation can be found in both the works of Rahm and Bernstein [18] and Hong-Hai, Rahm and Melnik [24]. Below, we present an approach that employs machine learning techniques and contrast our proposal with it.

AnHai Doan and colleagues [16] have created a system named GLUE which applies machine learning techniques to semi-automatically create semantic mapping. In the above figure (figure 6) we can see the architecture of GLUE system.

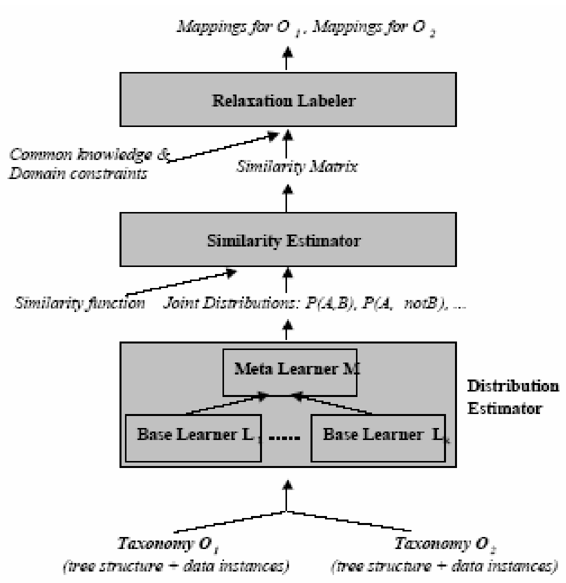
Mappings for O 1, Mappings for O 2
Relaxation Labeler
Common knowledge & Domain constraints
Similarity Matrix
Similarity Estimator
Similarity function
Joint Distributions: P(A,B), P(A, notB), ...
Meta Learner M
Base Learner L1 ..... Base Learner Lk
Distribution Estimator
Taxonomy O 1 (tree structure + data instances)
Taxonomy O 2 (tree structure + data instances)

Figure 4: *The GLUE architecture*
From [16, Figure 2]

As we can see, it takes as an input two taxonomies. A number of machine learning techniques are then applied to compute the joint probability distributions of every pair of concepts (A ϵ O_1, B ϵ O_2). A meta-learner is used in order to combine the predictions of the different learners. The reason for using a meta-learner instead of a single one is that the majority of the learners are good for particular tasks. So, the use of a meta-learner extends GLUE capability.

The computed joint probability distributions are used as an input in the second step of GLUE, the similarity estimator. In this step, the user has the possibility to choose which similarity measure wants to use. On the one hand, the used similarity measure should be able to be computed using only the joint probability distributions as this is the input it has. This restricts the number of the similarity measures that the user has the possibility to use. On the other hand, the user does have the possibility to choose between a number (even if it is restricted) of similarity measures, which is very useful, because the applicability of every measure depends on the occasion and there is one that could be considered as the best for every occasion. So, the Similarity Estimator gives as an output a similarity matrix between the concepts in the two taxonomies.

The last step of the GLUE architecture is the Relaxation Labeler, which takes as input the similarity matrix and exploits several constraint and heuristics to improve matching. The output of this step is a 1:1 mapping between the two taxonomies. A more detailed description of the above system can be found in [16]. This approach is different comparing to ours, because in our case, we want to examine not a 1:1 mapping between two different schemas but a classification of the documents in the one schema, given information about the classification in the other.

There are several other approaches for 1:1 schema matching. In the figure below (figure 7) a classification of the current approaches is given. The general procedure that is common for all of them is the following: They take two schemas as input and they give as output a 1:1 mapping between the elements of these schemas. The mapping is implemented according to the particular matching each approach uses. A survey of such approaches can be found in Rahm's work [18].

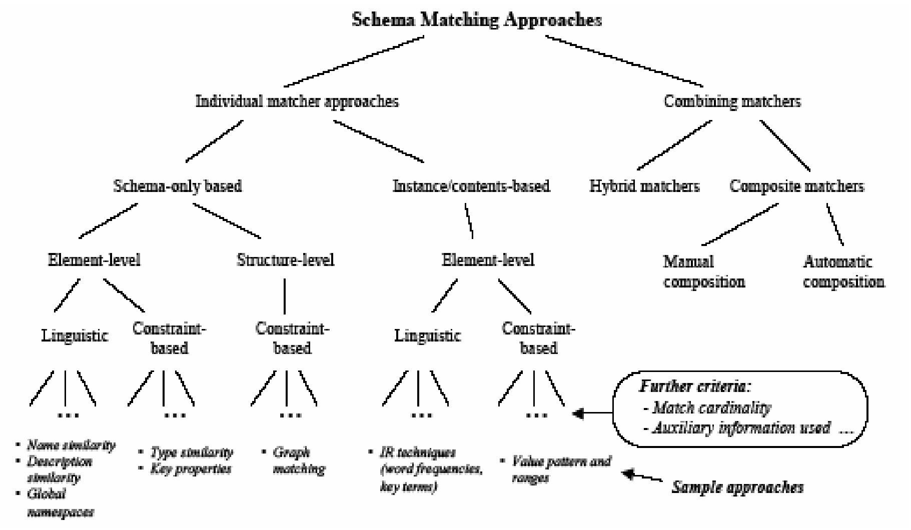

Figure 5: *Classification of schema matching approaches*
From [18, figure 2]

Another interesting work is that of Fausto Giunchiglia, Pavel Shvaiko et al. [26, 27, 28]. They present a new approach for schema matching, called semantic matching. More particularly, their method takes into account not only the labels of the nodes of the schemas they want to match, but also "the semantic relations between the concepts assigned to nodes" [27]. For example, some concepts can be equivalent, or one can be more general than the other.

As mentioned earlier, these approaches map each element of the one schema to the element of the other schema with the highest similarity, which results a 1:1 mapping [16] and this is not applicable in the most of the cases of real world data, as it is difficult for the data to have exactly one-to-one mapping.

Worth mentioning also is the work of Sarawagi, Chakrabarti and Godbole [25]. They present a new approach, called cross-training, which use sample documents from one taxonomy in order to improve the classification in another taxonomy. They assume that the taxonomy is a flat set of class labels. They present two algorithms. One is probabilistic and it is based on EM (Expectation Maximization) and the second is discriminative and it is based on SVMs (Support Vector Machines). Their results show that these methods are better than the baseline classifiers.

The latter approach is closer to the one we examine. The main difference is that, as mentioned earlier, they do not take into account the hierarchy and they consider the taxonomy as flat. In contrary, in our approach we use two-level hierarchies.

In this project, the aim is to find a fuzzy, context sensitive mapping between two schemas by combining semantic features and context-based features and using them as an input in a machine learner.

2.5.2 Semantic similarity measures

In order to match two schemata and create the semantic features, we need a notion of similarity. So, we are interested in semantic similarity measures. There are several similarity measures that can be used. Before we discuss them, it is worth mentioning some principles that we must take into consideration when constructing a tree similarity measure.

Principles for Tree Similarity Measure Construction

According to Tom Morrison [22], the most important principles that we must consider before construct a tree similarity measure are:

1. Matching at categories lower down the tree structure should count more to the measure than matching higher up, because this indicates a more precise match.
2. Matches at the top level of the tree should not count at all.
3. Matching counting should be reduced when they are not exactly on the same category and this reduction should be proportional to the generational distance.
4. The matching metric should be normalized so that it ranges from 0 to 1.
5. The matching metric should take into account all possible matches between the two schemata.

Approaches of semantic similarity measures

As mentioned earlier, there are several semantic similarity measures that can be used as a notion for similarity. Three of them have been used for the purposes of the project.

The first one is our own similarity measure. The idea is the following: for each concept on the tree, we calculate the distance from any other concept of the tree. More particular, we count the number of edges from each of the other concepts and we construct a sort of a taxonomic vector. This vector is normalized before is used.

The inspiration of the second similarity measure was given by the work of P. W. Lord et Al. [23]. The idea is to use the *probability of the least subsumer.* Last subsumer is the node that subsumes A and B (where A and B are nodes on the tree) and there is no other node lower down which also subsumes A and B. Each node on the tree has a probability. So the similarity of A and B depends on the probability of the node that least subsumes both A and B. Like in the previous case, we normalize the vector before we use it.

As a third similarity measure, a simple method is followed. All nodes are represented as a boolean vector. The node to which the document belongs has value 1; all other nodes have the value 0. Using these values we create the feature vector.

More details about the above approaches are given in *Chapter 3(3.3.2)* of the present report.

Chapter 3

Implementation

In this chapter all the details of the implementation are given. The chapter is organised as follows: At first, a description of the dataset is given. Then, the one-to-one schema matching that was implemented is presented. The procedure for the content-based classification is described. More particularly, we present the extraction of the features, their selection and their representation. Also, the tools and the classifiers used are presented. The description of the semantic features is following, in other words, the similarity measures used and the representation of the features. Then, the combination of content-based with the semantic features is taking place and all the details about this procedure are given. Finally, the "semantic" error metric is explained, a measure used for the evaluation of the results.

3.1 Dataset

The dataset which was used is about problem and solution descriptions for HP and Compaq products. The dataset consists of 1864 XML documents. Each is tagged with category data from a number of simple 2-level hierarchies. An example of a part of a hierarchy in which the documents are categorized is given below (Figure 6):

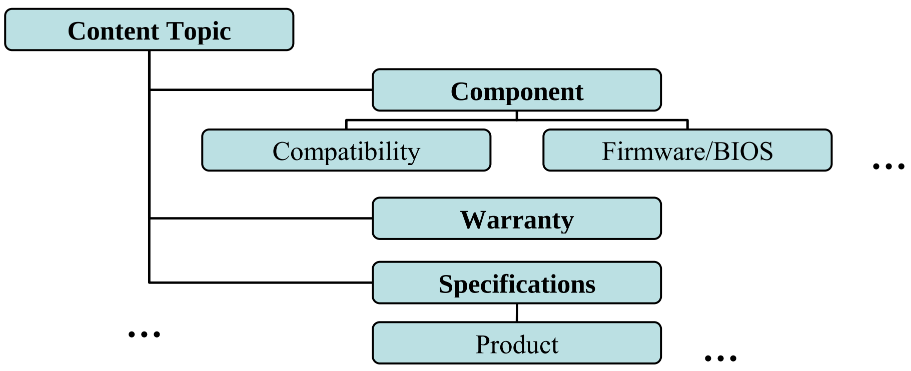

Figure 6: *Some of the values of the hierarchy "Content Topic"*

In the following table, a description of all the hierarchies is given:

Categorization Metadata	**Description**
content_topic	The main topic of the content; it is the parent of <content_topic_details>. This property is single-valued and is required. A question to ask that may help determine what value should be selected is: *What one topic of the product is this content about?*
Environments	The user's environment. In addition to software, this property can apply to technology and applications. This property is multi-valued.
Main_component	The primary component of the product. A component value should only be selected when a component of the product directly relates to the topic. This property is multi-valued.
Minor_component1	See *main_component*.
Minor_component2	See *main_component*.
Product_function	The function of the product discussed in the content – whether the function is operating properly or part of a fix problem document. This property is the parent of <product_function_details> and is single-valued. It is only relevant if the content describes the function, or requires the function to occur.
Software_topic	Software applications being described in the content. This property is single-valued. It does not apply to operating systems, hardware, or drivers.
Symptom	The symptom that led to the problem discussed in the content. This property is only available if *fix problem* is selected as the <user_task>. This property is single-valued.
User_task	The major task that best represents the intent of the document, and the reason why a customer would be interested in reading the document. (Usually the task is part of the title.) The property <user_task> is single-valued, which means only one value can be assigned. It is common in a support document to discuss more than one task, but use this property to assign

Table 1: *schemas description*

Each of these categories has a number of values. Further details about these values of the above hierarchies can be found in the Appendix A.

Having a document classified to a category of one of the above hierarchies (hierarchy A in *figure 6*), we want to determine if the use of that information plus the content-based features can give us better results in the classification of the document in another one of the above hierarchies (hierarchy B in *figure 6*) than using only content-based features. In other words, we want to classify the document in hierarchy B, using information about its position in hierarchy A and content-based features.

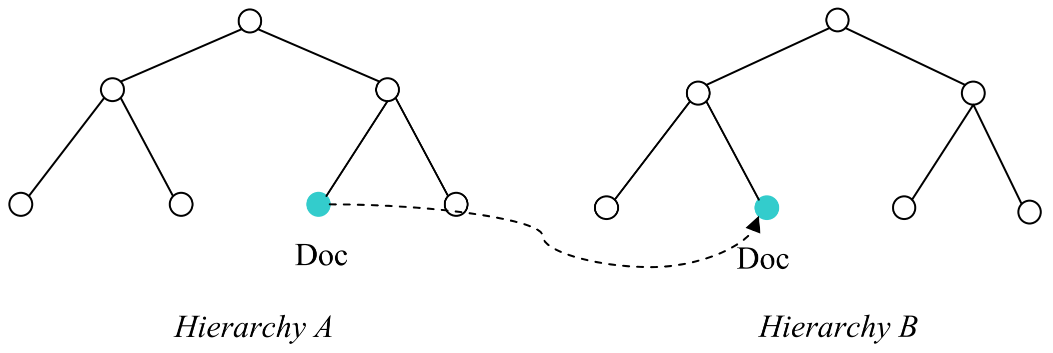

Figure 7: *Having a document classified in hierarchy A try to correctly classify it in hierarchy B*

3.2 Matching between different schemas in the dataset

In order to determine if the dataset described earlier was suitable for the purposes of the particular project, a one-to-one mapping between the hierarchies mentioned above took place. In particular, this mapping gave us the possibility to examine two characteristics. Firstly, the possibility to determine if there is an exact one-to-one mapping, in which case the dataset will be proven unsuitable, as the particular project, as mentioned earlier, focuses in the cases where one-to-one mapping is not enough. Secondly, the possibility to discover if any decent mapping between two particular schemas exists. If there is no connection at all between two schemas then our approach cannot be performed in the particular dataset.

3.2.1 Pre-process of the dataset

As mentioned above, the data was available in XML documents, so some pre-process had to take place in order to implement the mapping.

A bash script was implemented in order to extract the information needed from the dataset (the code can be found in the Appendix C.1). The output of the script was a file with the following format:

```
DocID,Content Topic,Content Topic Detail,Environment,Symptom,…

CN0093W,software,software_operating system,NA,error message,…
CN0082W,component,component_firmware/BIOS,NA,NA,…
BU021216_EW01,security,NA,Unix,performance,…
OT011107_EW02,component,NA,NA,performance,…
…
```

The first line contains the names of all the categories of the schemas. Each of the rest lines corresponds to a document and contains the values of the categories, respectively with the first line (e.g. the first value is the DocID, the second one is the Content Topic, the third one is the Content Topic Detail and so forth).

This file was then given as an input to a java program. The latter, created a similarity table between the categories of two particular schemas (the user can choose any two schemas among the available ones). The values in the table represent the number of documents that belong in both the respective categories of the two schemas.

This table is then used in order to find an optimal mapping between the chosen schemas. The program maps each category of the one schema with a category of the second schema according to the values on the table. In order to make that more clear, we present below the pseudocode of the algorithm:

```
FOR (index = 0; index<cats1.length; index++)
        FIND cats2 [j]
        WHERE similarity (cats1 [index], cats2 [j]) is maximum
ENDFOR
```

Where *cats1 []* and *cats2 []* are the two schemas and *similarity (cats1 [i], cats2 [j])* is the value in the similarity table which represents the number of documents classified in category i from the first schema and in category j from the second one.

The above allows the mapping of one category from the first schema with a category of the second schema that may have already been mapped with another one, in other words it allows many-to-one mapping. This allows mapping where the number of categories between the different schemas is not the same, so some categories would stay unmatched.

At first, another attempt for one-to-one mapping (see Appendix B.1 for details) was made but it was thrust aside for two reasons: firstly, it was computationally expensive and secondly the output was restricted to one-to-one mapping (in opposition to the algorithm described earlier). In other words, if a category from the one schema has been matched with a category from the other schema, then it cannot be match with any other category. But in our approach there isn't any restriction of this kind.

3.2.2 Results of matching

The most representative results of the mapping can be found in Appendix B.2. Experiments have taken place for all the possible combinations of the available schemas.

The number and the percentage of the correctly and incorrectly classified documents are given. In addition, the correspondence between the categories of the two schemas is given. To be more specific, the number of the categories from the first schemas that corresponds to each category of the second schema is given. This information is important, because we want to know if the correspondences are well spread (we don't want, for example, all the categories from the first schema to be corresponded in only one category of the second one).

From the above results, considering the characteristics we want the mapping to have as mentioned earlier, we can conclude that: Firstly, there is no exact one-to-one mapping between any pair of the different schemas. Secondly, we can remark some good correspondences between some pairs of schemas. This means that the dataset seems to be suitable for the purposes of the project.

Having determined that the dataset is suitable, we had then to choose also the suitable pairs of schemas in which our approach was tested. The total number of available pair of schemas comes up to 72 and as a result is not possible (in terms of time restrictions, as the whole procedure followed in order to test our approach is time-consuming) to test them all. This is why we had to select the ones that were given us the most suitable matching results.

The pairs of schemas which were chosen are presented in the table below (in the table, *Schema A* → *Schema B* means that we have already the documents classified in *Schema A* and we want to classify them in *Schema B*):

Schema A	→	**Schema B**
Minor Component 1	→	Content Topic
Content Topic	→	Minor Component 1
Minor Component 1	→	Symptom
Symptom	→	Minor Component 1
Content Topic	→	Minor Component 2
Minor Component 2	→	Content Topic

Table 2: *Selected pairs of schemas*

3.3 Classifiers and Classification procedure

Using the selected pairs of schemas, the classification experiments took place, using two tools: WEKA and AIRS classifier.

"The Waikato Environment for Knowledge Analysis (WEKA) is a comprehensive suite of Java class libraries that implement many state-of-the-art machine learning and data mining algorithms" [19]. It is available on the World-Wide-Web. It contains tools for data pre-processing, classification, regression, clustering, association rules and visualization. We are going to use WEKA for classification using the classification models mentioned in chapter 2 (except from AIRS, for which WEKA has no algorithm).

For the AIRS classifier, the source code of Andrew Watkins [14, 20, and 21] will be used.

The experiments that took place for this project can be divided in three main steps. In the first, only content-based features were used for the classification. In the second, semantic features were developed and used and in the third a combination of content-based and semantic features took place. In the following sections a detailed description of the above steps is given.

3.3.1 Classification using content-based features

As mentioned above, at first classification took place using only content-based features. In the diagram below, the procedure which was followed is described.

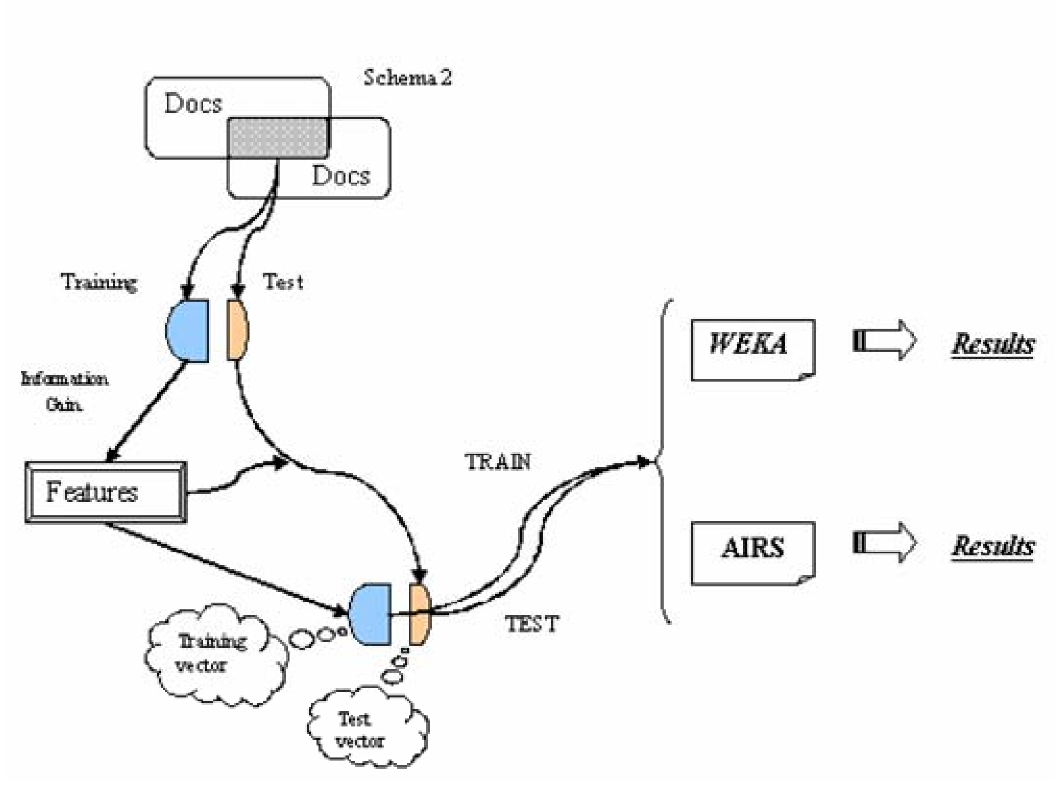

Figure 8: *The classification process*

We first have to select the documents which are categorized in both the selected schemas. As we use a real world dataset, there are many NA (Not Available) values, so not all the documents are categorized in all schemas. Nevertheless, from the matching we performed, described in section 3.2, we found out that the pairs of schemas we selected have a decent number of documents. More particular, the number of documents for the pair of schemas we have selected is approximately between 80 and 420, which seems to be a reasonable amount of documents for an accurate classification.

After selecting the documents, we divided them in training and test set. In order to have more accurate results, we use 5 times a 10-fold validation procedure. According to the latter, for each of the 5 times we do the following: we shuffle the order of the documents in the dataset and we divide it in 10 portions. The 9 are used as training set and the remaining one as test set. The portion which was used as a test set is rotated so that at the end all the 10 portions have been used as test set. To make it clear, in figure 9 a diagram of the process is given. In order to calculate the final result, we average all the predictions produced during the above procedure. In particular, we take the average of the 50 produced results (5 times * 10-fold validation).

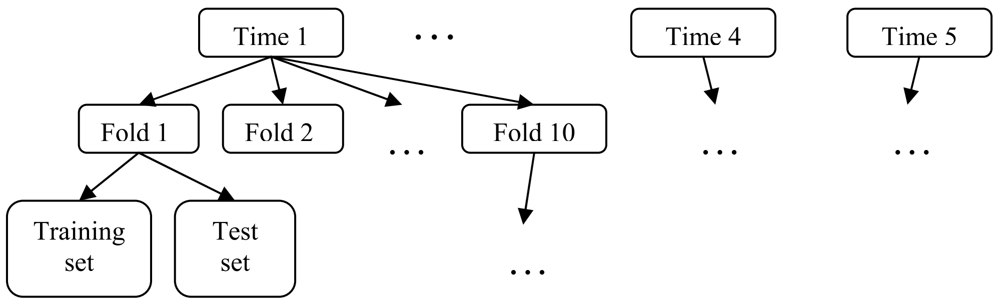

Figure 9: *Structure of files*

From the training set we select the features we want. A detailed description of the selection and representation of the features is given in the next paragraphs. The reason we use only the training set is to avoid information bleed. In other words, we want the feature vector to be different in each fold of the 10-fold validation for more accurate results (if we were using also the test set-which is changing in each fold- the extracted features would always be the same) .Using these features, we create the features vectors for both the training and the test set.

In order to give them as input to the classifiers, we have to create the files in the format that WEKA and AIRS accept them. WEKA uses arff files (one for the test and one for the training set) in which first the attributes (features) and their values are written and then the data vectors. Below, an example is presented.

```
@RELATION ContentTopic--MinComp1

@ATTRIBUTE upgrad {0,1}
@ATTRIBUTE firmwar {0,1}
@ATTRIBUTE insight {0,1}
@ATTRIBUTE manag {0,1}
@ATTRIBUTE server {0,1}
@ATTRIBUTE download {0,1}
…
@ATTRIBUTE class {component,component_compatibility,…}

@DATA
0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0,0,0,component
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,component
0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,component
0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,component
…
0,0,0,1,1,1,0,0,0,0,0,0,0,0,1,1,0,0,1,1,software_compatibility
```

```
0,0,0,0,0,0,1,0,1,0,0,0,1,0,1,0,0,0,0,1,software_compatibility
0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,software_compatibility
0,0,1,0,0,1,1,0,1,0,1,0,1,1,1,1,1,1,0,1,software_compatibility
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,software_compatibility
…
```

AIRS accepts two files (one for the test and one for the training set) with just the data vectors, in the following format:

```
0 0 0 1 0 0 0 0 0 1 0
1 0 0 0 1 1 1 0 0 0 0
1 1 1 0 1 1 1 0 1 0 2
1 1 0 1 1 1 1 1 0 0 2
0 0 1 1 1 0 1 0 0 0 9
1 0 1 1 1 1 1 1 0 0 9
1 0 1 0 1 1 0 0 0 0 10
```

As we can notice, AIRS does not accept the name of the classes as strings, so we have to correlate each value of the class with a number.

It also needs a configuration file in which information about the parameters of AIRS is given. An example of the configuration files that was used in the project can be found in Appendix C.2.

Features extraction, selection and representation

Having selected the training set, we have to extract the information we want from these XML documents, in order to use it for the creation of the feature vector. It would be possible to use all the words contained in all of the documents as features, however it is more sensible to perform some process of features selection for two reasons: On the one hand, many words in the documents are present for purposes of XML tags, so they are useless in sense of the classification process. On the other hand, in the documents there is also the information for the schemas and the categories they belong. Using all the available words of the documents, we would use also this information, but as mentioned before, the purposes of this part of classification is to use only the content-based features.

For the above reasons, we decided to extract only the words which are contained in the body of the XML documents, excluding all the special characters. A Java program and a script were implemented in order to extract these features. The output of this program is a file for each document, which contains the list of words which exist in the body of the particular XML document. In order to enhance the classification accuracy, a stemming algorithm was also used. In particular a porter stemming algorithm was used, implemented by John Keyes, fully described in [30].

Using all the words extracted by the above procedure, the performance of the classifiers is not expected to be good enough, as they are too many, and in addition to that, we have dimensionality limitations. So, a selection of the most informative words is recommended. The concept used in the particular project for the reduction of the number of features is *information gain*. The latter is a concept which is based in the entropy measure which is "a measure of the impurity in a collection of training examples" [2]. Information gain gives us a measure of how informative an attribute is, in classifying the training data. In other words, what information gain aims is the reduction of the entropy which is caused by classifying the examples according to this attribute [2].

The following example will make the concept of information gain more understandable. Imagine we have 4 documents which are classified in a set of classes and they contain a number of words, as shown below:

	Hardware	**Memory**	**Disk**	**Problem**	**CLASS**
Doc 1	1	0	1	1	A
Doc 2	1	1	0	1	A
Doc 3	0	0	0	1	B

The word "*Hardware*" exists in both *Doc 1* and *Doc 2* and not in *Doc 3*, which means that is specific for class A (in which only *Doc 1* and *Doc 2* belong), so it has a high information gain. In contrast, the word "*Problem*" exists in all three *Docs*, so it has a low information gain, as no information about the classification in classes A and B can be derived. The words "*Memory*" and "*Disk*" have intermediate values of information gain.

More precisely, the formula with which we can calculate information gain of a word A in a collection of documents S is the following:

$$Gain(S, A) \equiv Entropy(S) - \sum_{u \in Values(A)} \frac{|S_u|}{|S|} Entropy(S_u)$$

and $$Entropy(S) \equiv \sum_{i=1}^{c} - p_i \log_2 p_i$$

where Values(A) is the set of all the possible values of A and $S_u = \{s \in S \mid A(s) = u\}$.

For the calculation of the information gain, 2 scripts and one program implemented by Julie GreenSmith [32] were modified for the specific dataset. The program uses Laplace modifier to avoid arithmetic overflow (in the case we have $\log_2 0$).

After having calculated the information gain for all the words extracted from the documents, we then selected the *k* most informative ones. Different values for *k* were tested in order to find the one which gives us the better results. Finally, the number 50 seems to give the more representative results, so this is the one that was used. Using the selected words we created the feature vectors. Each word represents a feature and has two values: 1 if the document contains that word, 0 otherwise. After creating the suitable files for the classifiers, as described earlier in this section, everything is set to go. The above procedure is taking place, as mention earlier 5 *10 times (5 times *10-fold validation). The average of these results was calculated. The results are discussed in chapter 4.

3.3.2 Classification using semantic features

In this section a description of the procedure followed for the classification using semantic features will be given. In this case, as we have mentioned earlier, we want, using the information of the categorization in one schema, to classify the documents in another schema. In order to represent the features according to the classification in one of the schemas we need a notion of similarity. There are several similarity measures that can be used. We are interesting in semantic similarity measures. Below, the three semantic similarity measures, mentioned in *Chapter 2 (2.5.2)*, that was used for the purposes of the project are described.

The procedure is similar to the one followed for the content-based features, in terms of using 5 times 10-fold validation and of using the same classifiers. The difference consists in the selection of the features, as in the present case we do not care about the content, but only about the categorization to the schemas.

"Counting edges" approach

Our first attempt was to use our own similarity measure. For each concept on the tree, we count the number of edges from each of the other concepts and we construct a sort of a taxonomic vector.

For example, imagine the following 6 node hierarchy, where d_1 to d_4 are classified documents (figure 10):

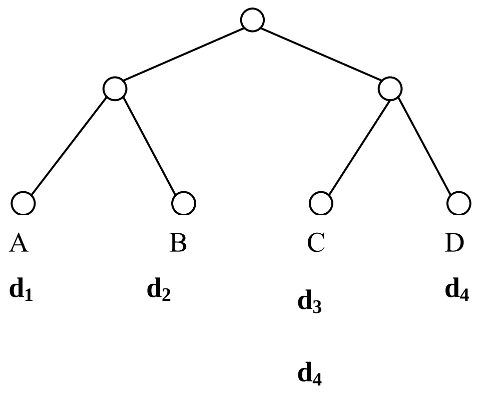

Figure 10: *Example of the naïve approach*

We create then a vector {(distance from A), (distance from B), (distance from C), (distance from D)}. So, for each of the above nodes, we count the number of edges and we have the results:

A = (0, 2, 4, 4)
B = (2, 0, 4, 4)
C = (4, 4, 0, 2)
D = (4, 4, 2, 0)

Using these results, we create the following vector (*Table 3*), which is then normalized. The suitable files required from the classifiers, described earlier, are created according to this vector and are given as input to them:

	A	**B**	**C**	**D**
$\mathbf{d_1}$	0	2	4	4
$\mathbf{d_2}$	2	0	4	4
$\mathbf{d_3}$	4	4	0	2
$\mathbf{d_4}$	4	4	0	2
$\mathbf{d_4}$	4	4	2	0

Normalize $\Longrightarrow$

	A	**B**	**C**	**D**
$\mathbf{d_1}$	0	0.5	1	1
$\mathbf{d_2}$	0.5	0	1	1
$\mathbf{d_3}$	1	1	0	0.5
$\mathbf{d_4}$	1	1	0	0.5
$\mathbf{d_4}$	1	1	0.5	0

Table 3: *The vector created using the method of counting edges before and after it was normalized*

As we can see from the table above (*Table 3*) we assume that if a document is classified in more than one category of one schema, we have multiple entries of this document in the vector (respectively to the number of categories it is classified). This assumption was applied to all three methods, described in the present section.

Probability of the least Subsumer

In addition to the method described earlier (counting edges), a more refined similarity measure was used. The idea was inspired by the work of P. W. Lord et Al. [23]. They used the concept of the probability of the least subsumer.

All the measurements are taking place using the information content, which is defined as “the number of times each term, or any child term, occurs in the corpus” [26]. Each node of the tree has a probability. The lower down on the tree structure, the less information content a node has. The root’s information content equals to 1. In the example in figure 11, if we want to find the similarity between A and B, we have to find the node that least subsumes both nodes. Least subsumer is the node that subsumes A and B and there is no other node lower down which also subsumes A and B. So, in the above example, the red node subsumes A and B but it is not the least subsumer. The black node is the least subsumer. So, the similarity of A and B depends on the probability of the black node. A low probability of the black node implies a high similarity of the concepts (A and B).

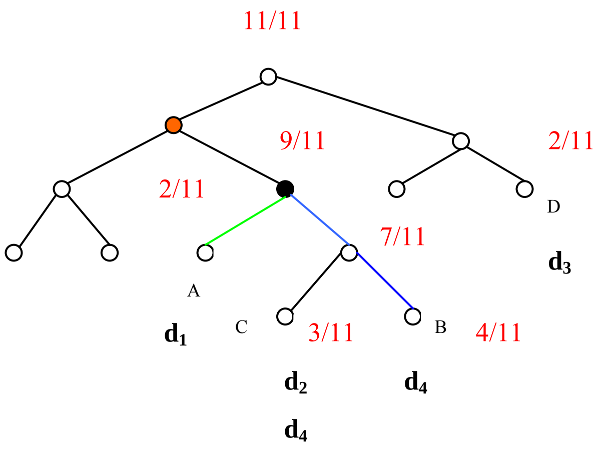

***Figure 11:** Example of Probability of the least subsumer*

As probability for each node on the tree, we used the number of documents classified in the particular node (category) or below divided by the total number of documents in the tree (# *(docs in the node or below) / (total #docs)*).

The distance between A and B is calculated by the following equation ($p_{ms}(A,B)$ is the probability of the least subsumer of A and B):

$$dist(A,B) = 2\ln p_{ms}(A,B) - (\ln p(A) + \ln p(B))$$

$$\text{where} \quad p_{ms}(A,B) = \min_{c \in S(A,B)} \{p(C)\}$$

Imagine in the example of *figure 11* the total number of documents is 11. The numbers in red in the figure are the probabilities of each node. Using the equations described above we can create the following vector (after, the table is normalized as in the previous approach):

	A	**B**	**C**	**D**
d_1	0	2.315007613	2.602689685	3.409496184
d_2	2.602689685	1.406913648	0	3.004031076
d_3	3.409496184	2.716349004	3.00403107	0
d_4	2.315007613	0	1.406913648	2.716349004
d_4	2.602689685	1.406913648	0	3.004031076

***Table 4:** The vector created using the method of probability of the least subsumer*

"Simple" approach

The third approach we used is the simplest one. The idea is to represent the nodes as a boolean vector. The node that the document belongs takes the value 1; all the other nodes take the value 0. For example, the vector that is created from the tree in figure 12 is the following:

	A	B	C	D
d_1	1	0	0	0
d_2	0	1	0	0
d_3	0	0	1	0
d_4	0	0	1	0
d_4	0	0	0	1

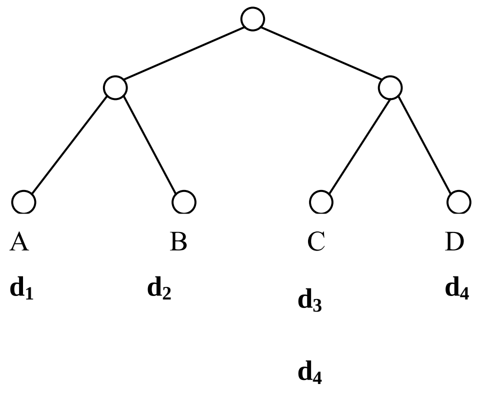

Figure 12: *Example of the Simple approach*

This approach, does not take into consideration the hierarchy. Nevertheless, we wanted to compare this method with the previous two, in order to find out how the information about the hierarchy can help the classification process. The representation of the features using this method is very close to the format that the classifiers are used to accept the features. More particular, the format follows the same concept as in the content-based features: 1 if the documents belong to that node and o otherwise, while in content-based features we have 1 if the word exists in the document, o otherwise.

In order to implement these three approaches we just discussed, three java programs were written which create the respective vector for each approach and use it for creating the suitable files, which was then given as input to the classifiers.

3.3.3 Combination of content-based and semantic features

After having classified the documents using only content-based and only semantic features, our goal was to combine the above features and try to find out if this could improve the accuracy of the classifiers.

Imagine we have the following table:

	X	Y	Z	W		A	B	C	D
d_1	1	0	1	1		0	0.5	1	1
d_2	0	1	1	1		0.5	0	1	1
d_3	1	1	0	0		1	1	0	0.5
d_4	1	0	0	1		1	1	0	0.5
d_4	1	0	0	1		1	1	0.5	0

Table 5: *Combination of content-based and semantic features*

The first part of the table is a standard keyword vector where 1 indicates that d_i has, for example, a particular word (X, Y, Z or W in the current example) and 0 otherwise. The second part of the table is the same (*Table 3*) used in the example described in *figure 10*, for the counting edges approach.

Current approaches indicate that a machine learner gets as input only the first part of the table (the keyword vector). This is what we did in the classification using content-based features. On the other hand, if we use only the second part of the table (semantic vector), the best results we can obtain is the optimal 1:1 mapping. This is what the three approaches described in 3.3.2 did.

What we did next, was to use both keyword and semantic vectors. In order to combine them we gave as input to the classifiers a long vector as the one in table 5, which concatenates all the features.

In order to examine how the different kind of features used (content-based and semantic) affect the performance of the classifiers, we used different weighting of the features. Except using the features as they were, we did experiments using 100% of the content-based features and 50% of the semantic ones and vice versa. Using 50% of the features means that we multiplied the features with the respective percentage (with 50/100 in this case). In the next chapter, where all the results are presented and explained, the affect of different weighting in the performance of the classifiers is also explained.

For the creation of the vectors and the files used for the classification, a Java program was implemented. Different percentages for the weighting can be chosen form the user.

3.4 "Semantic" Error metric

As we mentioned in the beginning of the chapter, in this section we present a measure which was used in order to evaluate the performance of the classifiers using the different features.

The "semantic" error metric gives us an insight on "how wrong" a misclassified example is. In order to make this clear, imagine we have the following schema (*figure 13*):

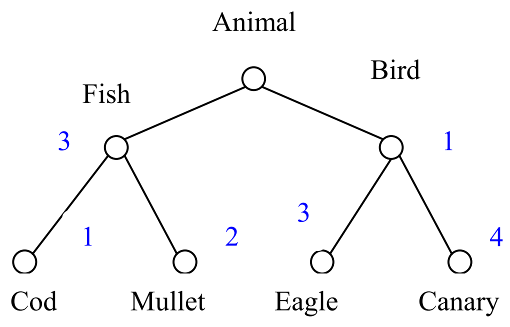

Figure 13: *example schema*

Imagine we have a document that should be categorized in the node *"Eagle"* but it is classified in the node *"Canary"*. We can say that this was misclassification into only a sibling class which is better than being classified in the node "Cod" which is completely irrelevant. In other words, we can say that classified a document in a sibling node is better (and "less" wrong) than classify it in a completely irrelevant class.

Trying to apply this idea, in order to see if combination of content-based and semantic features can help in the reduction of the "semantic" error, we implement the following method:

Suppose *table 6* is a confusion matrix produced by WEKA for the example described above (*figure 13)*. In *table 7* suppose we have the semantic vector created by, for example, the "counting edges" method.

	Fish	Cod	Mullet	Bird	Eagle	Canary
Fish	2	1	0	0	0	0
Cod	0	1	0	0	0	0
Mullet	0	1	1	0	0	0
Bird	0	0	1	0	0	0
Eagle	0	0	0	1	1	1
Canary	1	0	0	0	0	3

Table 6: *Confusion Matrix*

	Fish	Cod	Mullet	Bird	Eagle	Canary
Fish	0	1	1	2	3	3
Cod	1	0	2	3	4	4
Mullet	1	2	0	3	4	4
Bird	2	3	3	0	1	1
Eagle	3	4	4	1	0	2
Canary	3	4	4	1	2	0

Table 7: *Semantic Vector (counting edges)*

Having these two tables, we calculate then the accuracy of the classifier (considering the "semantic" error) as followed:

- We first calculate the distance using the following formula:

  $A_{11}*B_{11} + A_{12}*B_{12} +\ldots+ A_{1n}*B_{1n} = X_1$

  $A_{21}*B_{21} + A_{22}*B_{22} +\ldots+ A_{2n}*B_{2n} = X_2$

  …

  $A_{n1}*B_{n1} + A_{n2}*B_{n2} +\ldots+ A_{nn}*B_{nn} = X_n$

where A_{ij} and B_{ij} is the jth column of the ith row of table A and B respectively.

- We then add the results of the calculated distances:
  $X_1+X_2+\ldots+X_n = Y$

- Finally we calculate the accuracy as followed:

  Accuracy = 1 – (Y / Max Distance), where Max Distance is the maximum total distance we can have. In other words, it is the total number of documents multiplied by the maximum distance between the nodes of the tree (in the particular case 14 documents * 4 = 56)

If we apply this method to the above example we have the following results:

2*0 + 1*1 + 0*1 + 0*2 +0*3 + 0*3 = 1
0*1 + 1*0 + 0*2 + 0*3 + 0*4 + 0*4 = 0
0*1 + 1*2 + 1*0 + 0*3 + 0*4 + 0*4 = 2
0*2 + 0*3 + 1*3 + 0*0 + 0*1 + 0*1 = 3
0*3 + 0*4 + 0*4 + 1*1 + 1*0 + 1*2 = 3
1*3 + 0*4 + 0*4 + 0*1 + 0*2 + 3*0 = 3

Y = 1+0+2+3+3+3 = 12

Max Distance = 14 * 4 =56

Then the accuracy equals to: Acc = 1 – 12/56 = **0.7857**

The normal accuracy (#correctly classified docs/ total #docs) equals to:
9/14 = **0.6428571**

The procedure to calculate the accuracy, considering the "semantic" error, for the "probability of the least subsumer" method is similar, with the only difference that the Maximum distance equals to the number of documents (because in this case the maximum distance between the nodes is always 1).

For the "simple" method there is no need to implement this metric, as it will have exactly the same results as the accuracy. If we recall the method, described earlier in this chapter (3.3.2) we will notice that applying the "semantic" error metric will end up with the number of the correctly classified documents divided by the total number of documents, which is nothing else but the accuracy.

Having implemented this method in Java we run the experiments and we calculate the accuracy considering the "semantic" error. Unfortunately, AIRS classifier does not return the confusion matrix so we could not apply the above idea for it. We applied it only in the WEKA classifiers.

Some parts of the code used for the implementation of all the programs mentioned in this chapter are displayed in the Appendix C.3.

Chapter 4

Results and Analysis

The purposes of the current chapter are to present the results obtained and analyse the performance of the classifiers using our approach discussed in the previous chapter. From the analysis of the results we wanted to reach the following conclusions: firstly, if using the content-based features alone the classifiers have better performance than using the semantic features alone. Secondly, if combining the content-based and the semantic features can give us better results than using each of them alone and thirdly, if the weighting of the features when we combine them makes any difference to the accuracy of the classifiers.

In order to reach the above conclusions, we used two measures: on the one hand, we used the *accuracy* which is the number of correctly classified documents divided by the total number of documents. We checked the results using Wilcoxon Mann-Whitney ranking test [31] which is non-parametric. The particular test does not assume normal distribution. This is the reason why the standard error was not calculated and it is not presented in the diagrams (standard error assume normal distribution). On the other hand, we used *"semantic" error metric*, which in contrast to accuracy, can tell us not only if a document was incorrectly classified, but also gives us a notion of "how wrong" it was misclassified. This measure was explained in detail in section 3.4.

4.1 Accuracy results

The accuracy was calculated as an average of the 50 runs (5 times * 10-fold validation). Both WEKA and AIRS return as result the accuracy, which is the number of correct classified documents, divided by the total number of documents.

Initially, we needed a comparison of the performance of different classifiers used in the project. Using only content-based features, the difference in the performance between the classifiers was quite small, so the Wilcoxon Mann-Whitney ranking test was used in order to find out if this difference is statistically significant or not. Experiments took place using different number of features, in order to find which the one with the better accuracy. In particular, we did experiments using 5, 10, 20, 50, 100, 150, 200 and 500 features. The achieved results showed that when we used a large number of features (more than 100), the classifiers have significantly better results in the training set but no difference in the test set, in comparison with the use of smaller number of features (50 and less). Our suspicion was that a large number of features may lead the classifiers to overfit. Also, when we used less than 50 features the results was a bit worse than using 50 features. This is why the use of 50 features was finally preferred. Also, the results showed (*table 8* for example – the results for the other pairs were similar) that the performance of Support Vector Machines and K-Nearest Neighbour (for K=1 and K=4) was a bit better than the others and AIRS was worse than the others, but as we can also see in *figure 14*, this was not always the case as the results are depended in the dataset we used.

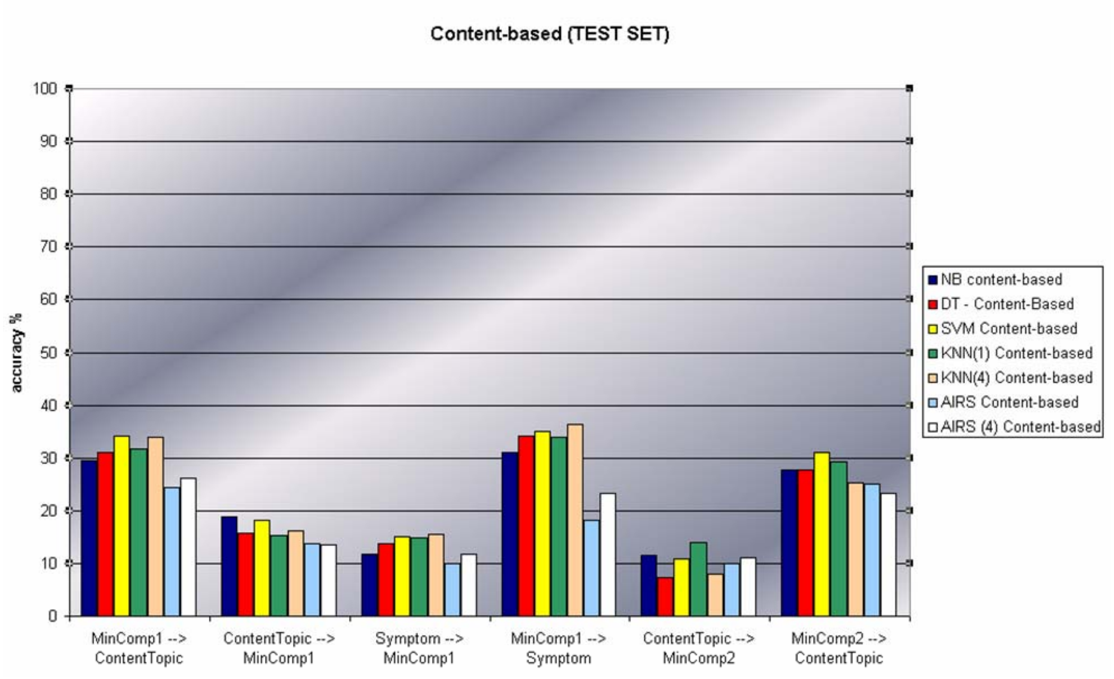

Figure 14: *A comparison of the classifiers performance using content-based features, showing small differences between their performance*

MinComp1 -- Symptom	TEST SET			
CONTENT-BASED	NB	DT	SVM	AIRS
KNN	Better (9.779e-06)	Better (0.006333)	Not Significant (0.9464)	Better (3.632e-08)
AIRS	Worse (0.02067)	Worse (1.351e-05)	Worse (3.151e-07)	
SVM	Better (7.905e-05)	Better (0.02278)		

Table 8: *Statistical significance of the results for content-based features for the pair of schema Minor Component 1 – Symptom*[*]

Respectively, a comparison of the classifiers' performance using only semantic features took place, in order to find out if in this case the results were

[*] The table contains the results from Wilcoxon test. It tells us if the classifier in the Y axis is better/worse from the one in the X axis. For example it tells us that KNN is better that Naïve Bayes and that AIRS is worse than Decision Trees.

All the tables of this section should be read in a similar way.

different. The results showed (*figure 15* – "counting edges" method) that Naïve Bayes did not have good results using only semantic features. Also, AIRS proved in general to lack in performance (*Table 9*) in comparison with the other classifiers, which seemed to have more or less the some results (not statistically significant difference). The results using the "probability of the least subsumer" method for the creation of the semantic features were similar.

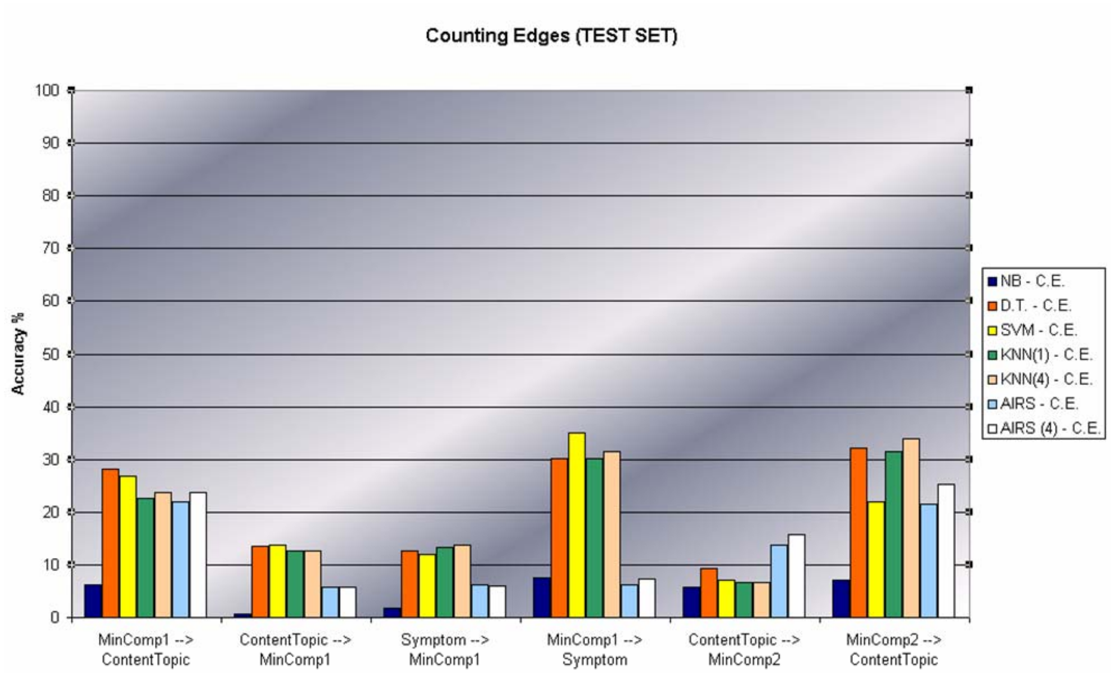

Figure 15: *A comparison of the classifiers performance using semantic features (counting edges)*

MinComp1 -- Symptom	TEST SET			
Semantic Features (counting edges)	NB	DT	SVM	AIRS
KNN	Better (8.475e-09)	Better (0.006104)	Worse (0.03348)	Better (6.304e-07)
AIRS	Better (2.814e-07)	Worse (5.927e-06)	Worse (4.271e-06)	
SVM	Better (2.738e-09)	Not Significant (0.6242)		

Table 9: *Statistical significance of the results for semantic features (using counting edges method) for the pair of schema Minor Component – Symptom*

For the third method used for creation of semantic features, the "Simple" method, the results are quite interesting. For the majority of the classifiers the performance is similar to the ones achieved using the others methods for the semantic features. But we can see a better performance of Naïve Bayes classifier (*figure* . The possible reasons for that are discussed in the next chapter.

Then, a comparison of content-based and semantic features took place. The results showed that as an overall the use of just semantic features is in most cases almost as accurate as using only content-based features, except for Naïve Bayes which is discussed later. However, in the training set, if we use only content-based features the results are better than using only semantic ones (*figure 16* and *Table 10*).

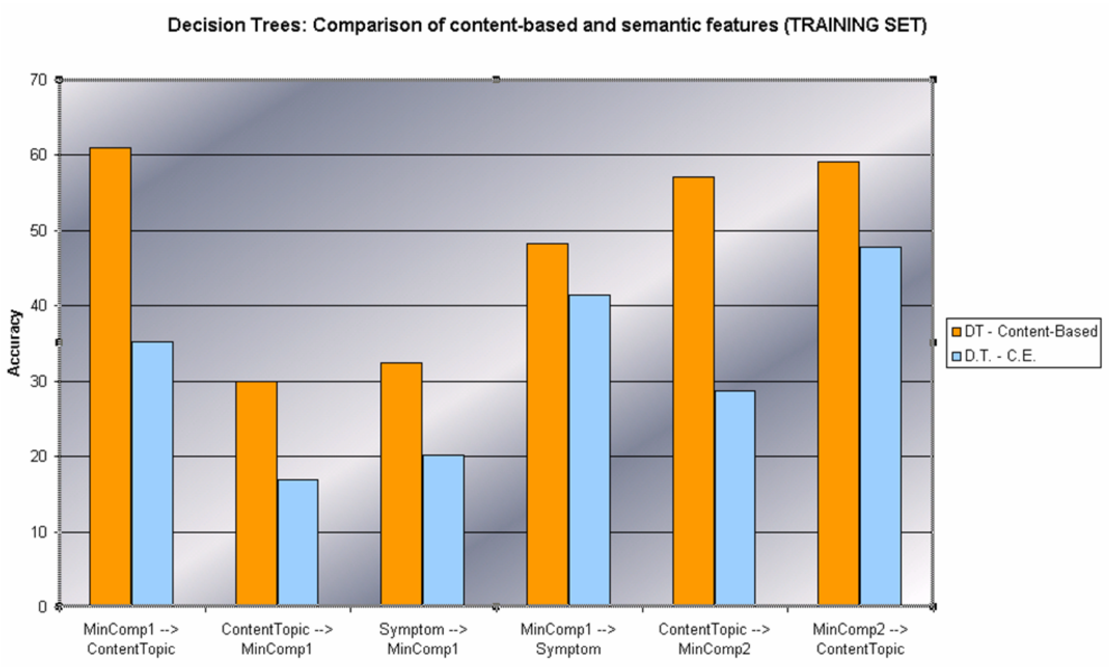

Figure 16: *Training set: A comparison of content-based and semantic features (for Decision Trees) Content-based features are proved better in the training set*

Content Topic-MinComp1	TRAINING SET
Decision Trees	SEM-FEATURES
CONTENT-BASED FEATURES	better (7.821e-10)

Symptom -- MinComp1	TRAINING SET
Decision Trees	SEM-FEATURES
CONTENT-BASED FEATURES	better (7.821e-10)

Table 10: *Statistical significance of the results for the training set in the comparison of content-based and semantic features. Content-based features are proved better*

Nevertheless, in the test set (which is actually the one that we are interested in) the performance of the classifiers using either content-based or semantic features is almost the same. In some cases the content-based features seems to have a bit better results (for example, in figure 17, the first column), as in others semantic features seems to be more accurate (for example, in figure 17, the last column). As an overall, we can conclude that the performance of the classifiers using semantic features is almost as good as content-based features.

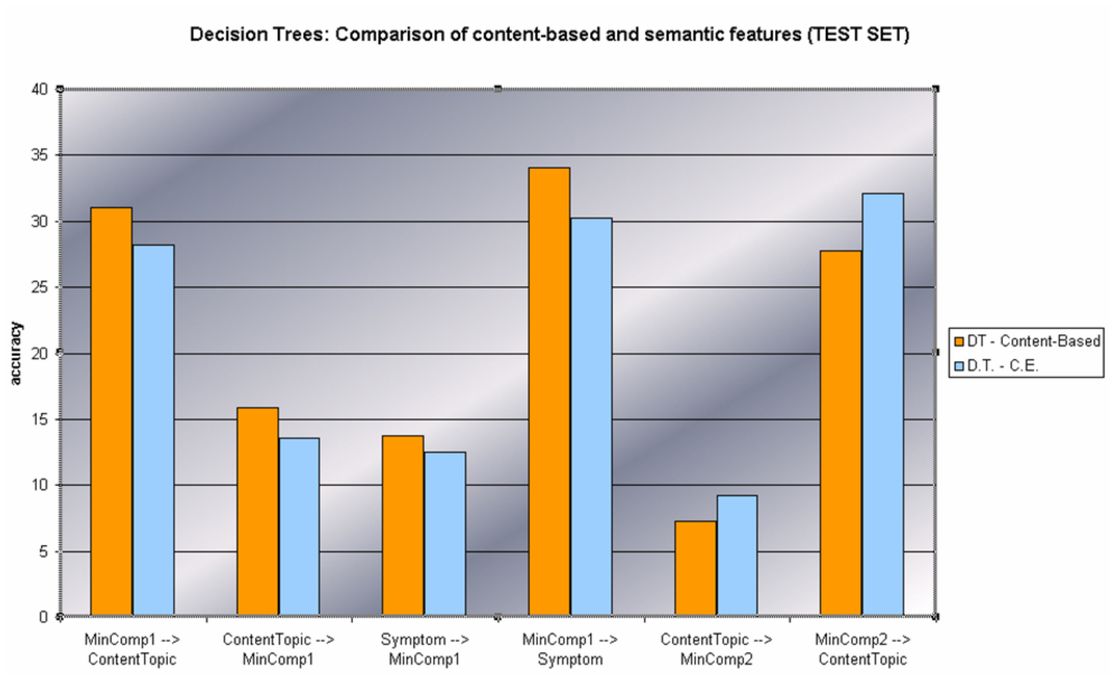

Figure 17: *Test set: A comparison of content-based and semantic features (for Decision Trees) No significant difference between them*

Content Topic -MinComp1	TEST SET
Decision Trees	SEM-FEATURES
CONTENT-BASED FEATURES	Not Significant (0.1201)

Symptom -- MinComp1	TEST SET
Decision Trees	SEM-FEATURES
CONTENT-BASED FEATURES	Not Significant (0.1917)

Table 11: *Statistical significance of the results for the test set in the comparison of content-based and semantic features (Decision Trees). No significant difference in most cases.*

As mentioned earlier, Naïve Bayes does not seem to have good performance with the semantic features created from either "counting edges" or "probability of the least subsumer" method, as it seems to prefer content-based features (*figure 18* and *Table 12*).

Symptom -- MinComp1	TRAINING SET
Naïve Bayes	SEM-FEATURES (Counting edges)
CONTENT-BASED	better (1.149e-09)

Table 12: *Statistical significance of the results of Naïve Bayes classifier for content-based and semantic features ("counting edges" method). Content-based features perform better.*

In contrast, using the "Simple" method, even though the results for most of the classifiers are similar to the results of the other two methods, when it comes to Naïve Bayes, things are a bit different. More particular, the Naïve Bayes classifier does not seem to have so bad performance. Content-based features may have better performance than the semantic ones (as in the cases where the "counting edges" and the "Probability of the least subsumer" method were used), but in this case the difference is apparently smaller (*figure 18* and *Table 13*).

Symptom -- MinComp1	TEST SET
Naïve Bayes	SEM-FEATURES (Simple)
CONTENT-BASED	better (0.04167)

Table 13: *Statistical significance of the results of Naïve Bayes classifier for content-based and semantic features (Simple method). Content-based features perform better, but the difference is significant smaller in comparison with the other methods*

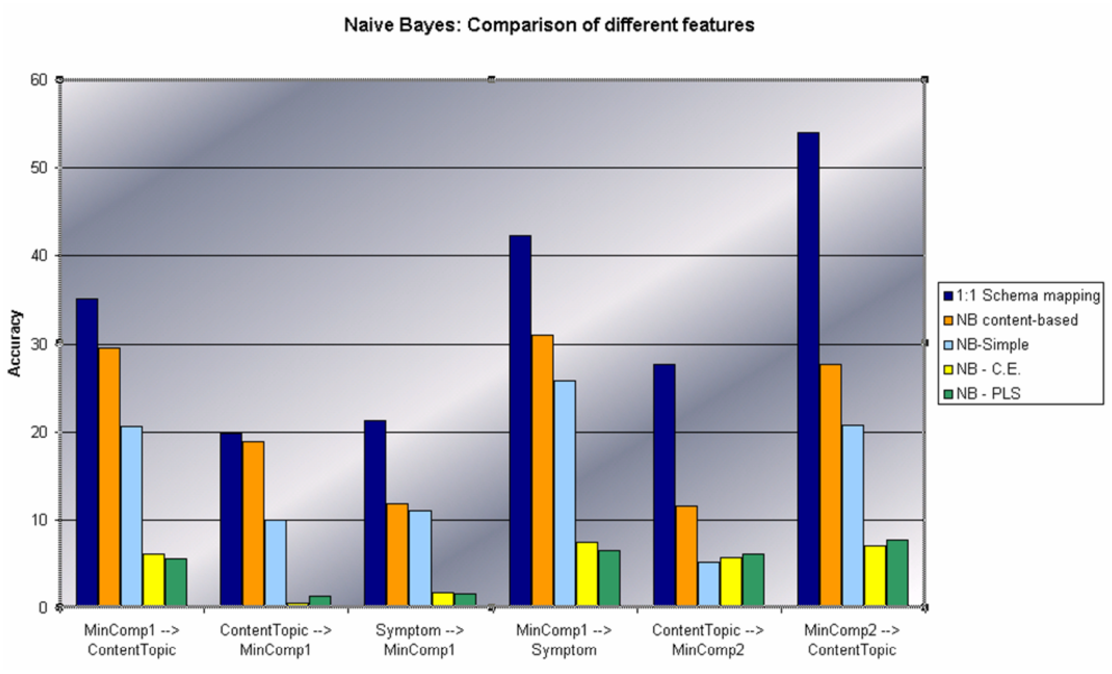

Figure 18: *Test set: A comparison of 1:1 mapping, content-based and semantic features using the all the three methods (for Naïve Bayes classifier). "Simple" method performs better than the others*

After comparing content-based and semantic features, their combination followed. As mentioned in the previous chapter (section 3.3.3), different weighting of the features was used in order to see how this affects the performance of the classifiers. Experiments took place and below some diagrams are presented that show, for the selected schema pairs, how classifiers perform with content based features alone (left hand column in each group), category based (semantic) features alone (right hand side) and some mixture (middle columns).

The diagram below (*figure 19*) is an example of the results we got from the experiments. It shows the results of the performance of Support Vector Machine. As we can see, the weighting did not seem to have any effect in this particular classifier. In fact, we had the same result for all the classifiers in WEKA. AIRS differs (*figure 20*), but as we can see, the difference is very small and it has been proved statistically insignificant (using again Wilcoxon tests).

The main question we want to answer is if the combination of content-based and semantic features can improve the performance of the classifiers comparing with the performance using either content-based or semantic features.

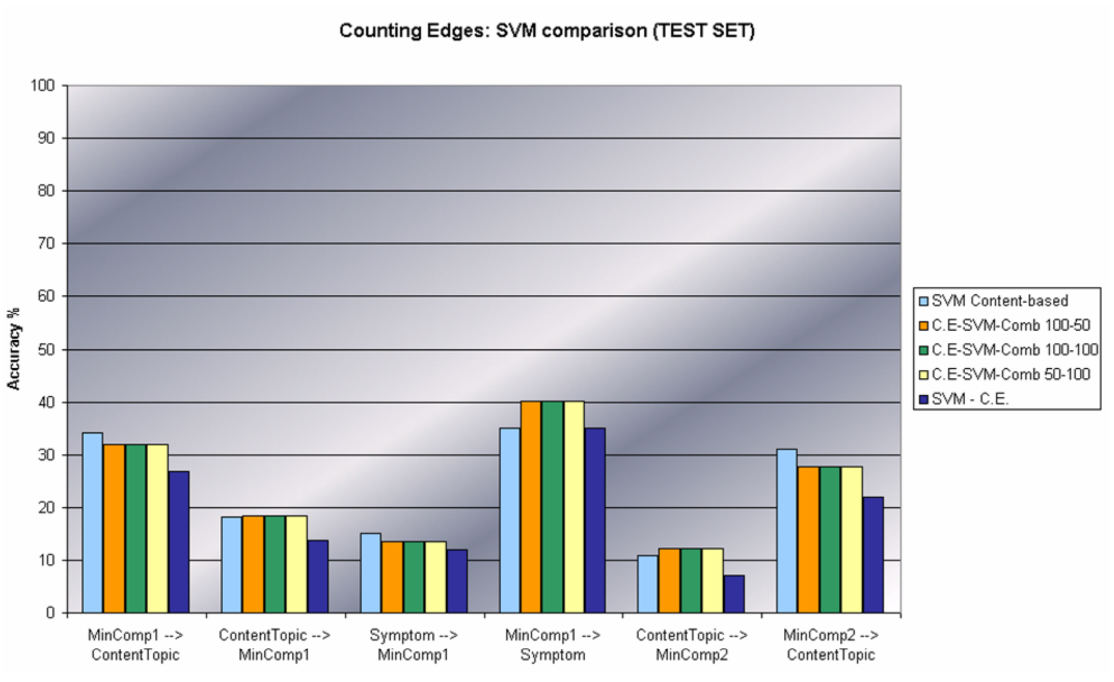

Figure 19: *Test set: A comparison of content-based, semantic features and their combination for SVM classifier*

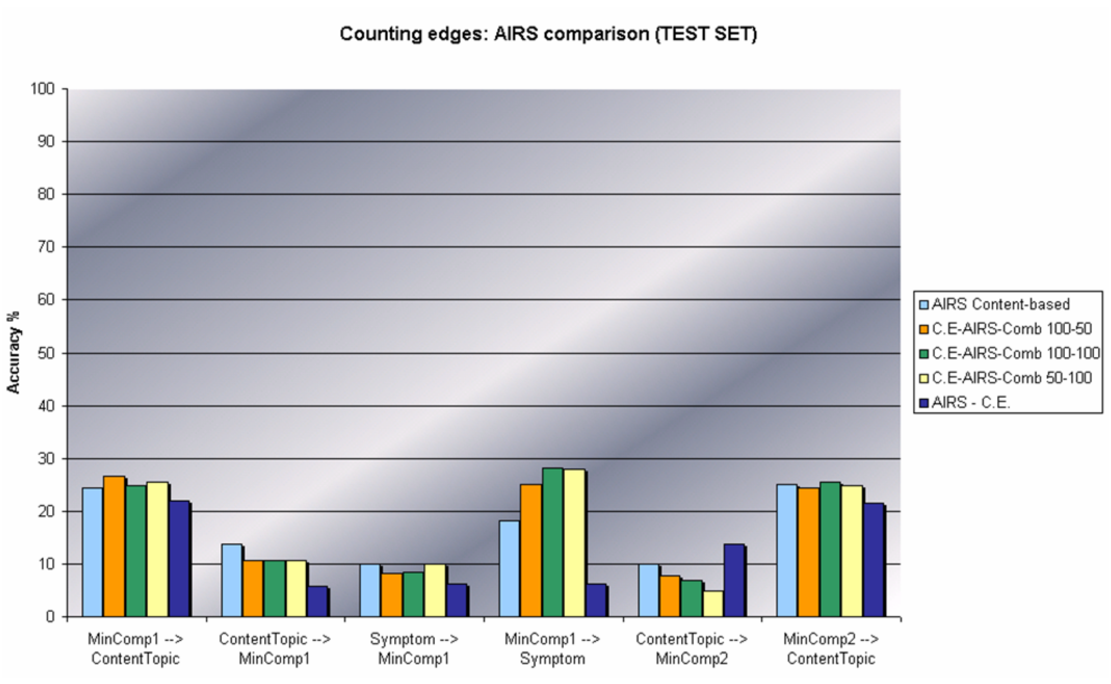

Figure 20: *Test set: A comparison of content-based, semantic features and their combination for AIRS classifier*

As we can see and in the diagrams above (*figure 19* and *20*), the performance in most cases is similar to the one using only content-based features. In both the diagrams, one can notice a better performance of the classifiers in the combined features for the Minor Component – Symptom pair of schemas, but as we can see in *table 14* below, no significant difference found between them.

MinComp1-Symptom		TEST SET
SVM	COMBINATION	SEM-FEATURES
CONTENT-BASED	Not significant (0.1574)	Not significant (0.9364)
SEM-FEATURES	Not significant (0.1311)	

MinComp1-Symptom		TEST SET
AIRS (100-100)	COMBINATION	SEM-FEATURES
CONTENT-BASED	Not Significant (0.09364)	Not Significant (0.3389)
SEM-FEATURES	Not Significant (0.2046)	

Table 14: *Statistical significance of the results of AIRS and SVM classifier for content-based, semantic features and their combination. No significant difference in the results*

The results have showed that in general there is not significant improvement. In some cases (as the one below, *figure 21*) in the training set the combination of the features seems to perform better (and the difference is statistically significant), but in the test set (*figure 19*) no improvement is noticed. In other words, the combination suggests an overfitting of the classifier (definition according to [2]: “Given a hypothesis space H, a hypothesis $h \in H$ is said to *overfit* the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' the training examples, but h' has a smaller error than h over the entire distribution of instances”). Possible reasons for that are discussed in the next chapter.

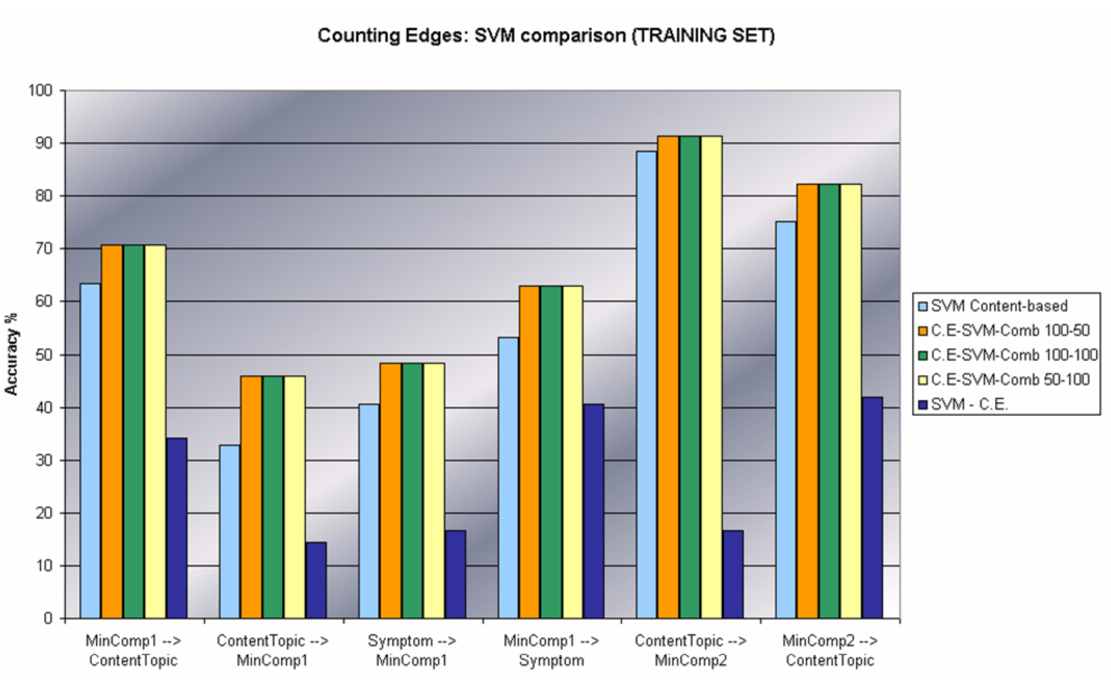

Figure 21: *Training set: A comparison of content-based, semantic features and their combination for SVM classifier*

4.2 “Semantic” Error metric results

As mentioned in the beginning of this chapter, as well as the accuracy, the “semantic” error metric was calculated for the semantic features created with the “counting edges” and the “probability of the least subsumer” approaches.

The results leaded to two conclusions. On the one hand, the accuracy calculating taking into account the semantic error, seems to be similar for content-based and semantic features in most cases.

On the other hand, comparing the accuracy for the combination of content-based and semantic features with either the content-based alone or the semantic features alone, calculated by the method described earlier, we can notice (for example in *figures 22* and *23*) that there is no statistically significant difference. These results are similar to all the other experiments we performed for all the classifiers.

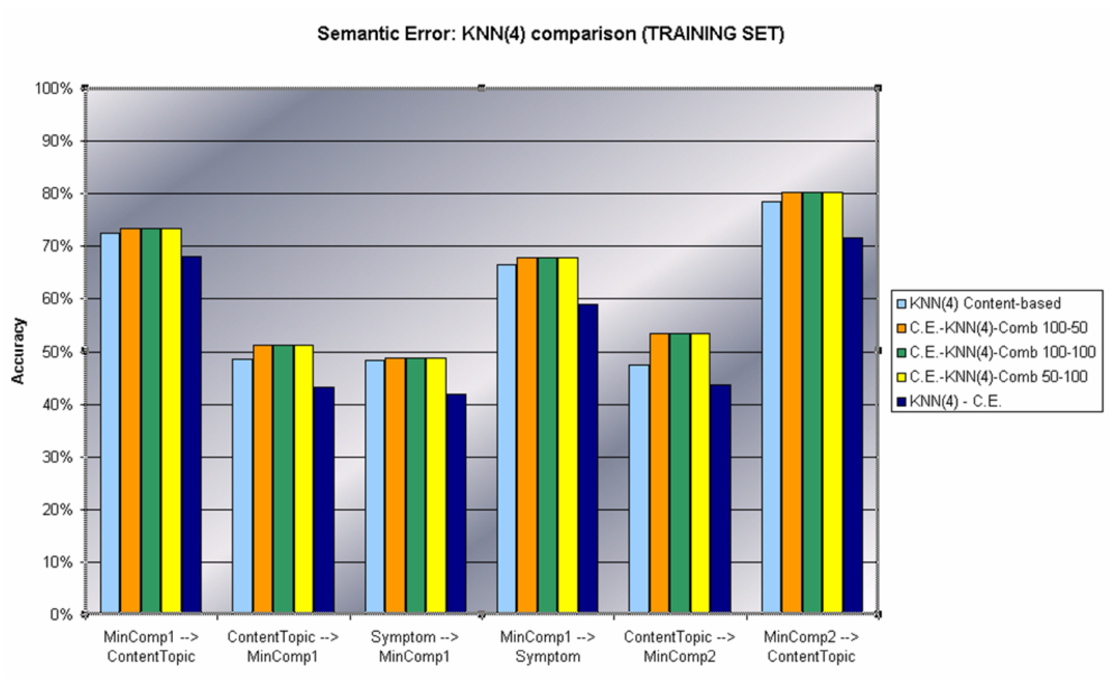

Figure 22: *Training set: Accuracy, considering the "semantic" error*

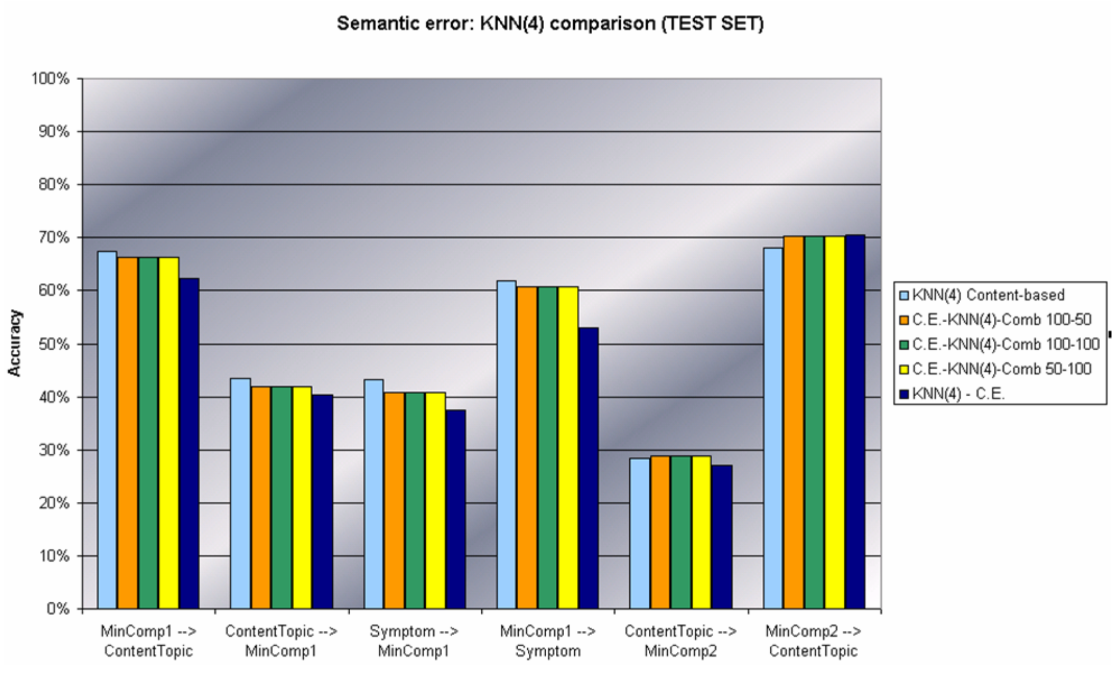

Figure 23: *Test set: Accuracy, considering the "semantic" error*

In the next chapter, we have a deeper discussion on the results. We summarise our conclusions and discuss the possible causes which led to these results.

Chapter 5

Discussion and Future Directions

The purpose of this chapter is to review and discuss the results presented in the previous chapter. Also, some future directions will be discussed. The goals of the project were reached, but there is still a number of different avenues related to the present project which have not still been explored and they look promising.

5.1 Discussion

Summarizing the results presented in the previous chapter, we reached several conclusions. Initially, we examined the behaviour of the different classifier used for the purposes of the project in the same features. We concluded that for content-based features the behaviour of all the classifiers was similar. Some differences in the performance of the classifiers between the different schema pairs were noticed, but they proved statistically insignificant. At this point we should mention that, as we can notice, the obtained results were quite poor (the accuracy is around 30-40%), in comparison with the accuracy that the classifiers used can generally obtain. Nevertheless, the results are not as "bad" as they look. In particular, the performance of the classifiers may not be high but it is better than a random classification. All the schemas which were used have at least 16 categories. Some of them have much more (around 45). So, a random classification would have an accuracy of probably around 10% (and maybe smaller) which is worst than the results obtained. A possible reason for having such low performance may be the fact that we used naïve content. If we had used for example some rules the results would probably be better. As our goal was not to improve the content-based results, but to test our own approach using semantic features, we though that it was advisable to spend our time for the benefit of our approach rather than the improvement of content-based results.

The results of optimal one-to-one mapping were discussed in chapter 3 (in section 3.3.2). Comparing them with the results of the classifiers we can see that in general they are better as expected (as the mapping is optimal). However, in few cases, classifiers seem to do slightly better. This can be explained by a "lucky break" of training and test sets (1:1 algorithm was tested on the entire dataset).

When semantic features were used, we noticed some interesting results. On the one hand, most of the classifiers seem to have similar performance when we compared the performance of the content-based features. Naïve Bayes classifier does not seem to perform well with semantic features created by the first two methods we used ("counting edges" and "probability of the least subsumer"). On the other hand, when we used the "Simple" method to create the semantic features, even though the results for most of the classifiers were similar to the ones achieved using the other methods, in the case of Naïve Bayes classifier, thing seems to be different. More particular, the latter performed much better than the other methods and the results were equivalent to the content-based features. This result enforces our conclusion mentioned earlier, according to which Naïve Bayes classifier prefers features in the format of the content-based ones. As we have mentioned in *chapter 3* (section *3.3.2*) the format of the features which is used from the "Simple" method is very close to the format of the content-based features and is the same with the one the classifiers are used to. This means, that even though most of the classifiers seems to adapt with the use of different format of the features, Naïve Bayes does not. The reason for that could be the fact that Naïve Bayes classifier is based on the simplifying assumption of conditional independency among the features. But the semantic features created using the "counting edges" and the "probability of the least subsumer" approaches are distance-based, which means that they are dependent to each other. So, Naïve Bayes seems not to be able to deal with them. At this point we should mention that for the same reason we expected decision trees to perform quite bad, but the latter seems to be able to handle them.

Another conclusion we can reach using the results is that generally the performance of the classifier using only semantic features is just as good as the one which use only the content-based features. They both seem to be useful sources, but imperfect. Thus, one would believe that combining these features (content-based and semantic) would improve its performance.

The results we achieved, in contrary, do not seem to agree with the above concept. In particular, we had two main results: either the combination did not seem to make any difference in the performance of the classifier or, in some cases, the combination seems to suggest overfitting. For the first case, a possible hypothesis could be that content-based features provide enough information such that the classifier gets very little benefit form a few extra bit of information that the semantic feature gives. In other words, even though when using only semantic features they seems to be quite informative, in reality when they are combined with content-based features probably the do not offer much more information than that we already have. Probably because they actually offer the same "sort" of information. For the second case, where the classifier gives some signs of overfitting when we combine the features, a possible hypothesis could be that the combination confuses the classifier rather than helping it. The different representation of semantic features, in comparison with the representation of the content-based ones, may cause a degradation of feature handling in the classifiers.

In this point, is worth mentioning that our idea to use different weighting on the features did not make any difference, especially in the WEKA classifiers, where the results were identical. This result, made us believe that WEKA internally weights the features to give the best results, this is why our weighting did not have any effect. The small difference in the results of AIRS classifier, which is not included in WEKA, strengthens this belief.

Concerning the "semantic" error, the combination did not seem to improve the results we have using only content-based features. A small difference can be noticed when using the “Simple” method, but mostly in the training set and less in the test one.

In short, this project has proved that the combination of semantic features with the content-based ones is not as easy as it might look. Even if, intuitively one would believe that the combination could improve the performance of the classifiers, it is not seem to be the case. Of course, this is the first attempt to this direction and there are many possibilities and variation to be examined. In the following section, some of them are presented as ideas for future work.

5.2 Future Directions

As we mentioned earlier, this project has achieved an insight of how semantic features can deal with content-based features and with a number of classifiers. However, there are many areas related to this project yet to be explored. In this section some ideas of how this project could be extended will be given.

Firstly, we could perform classification for content-based and semantic features separately, and then try to weight their prediction. As we have seen from the produced results, semantic features alone seem to be quite informative but when they are combined with content-based features they do not improve the classifiers as expected and instead they seem to confuse them. We first have to examine if the content-based and the semantic features make the same “sort” of errors. In other words, we want to know if they face the same difficulties during the classification. If they do not, then a combination of their prediction would look promising. So, it seems worth trying to discover if they misclassify documents for the same reasons and then try to weight their prediction, in a meta-classifier for example, and see if that could improve the performance of the classifiers.

Secondly, it would be interesting to perform the approach we introduce in this project in different datasets and see their results. It is common that classification is a dataset-depended procedure. This is why it would useful to have a more general idea of the results of the presented approach, using different datasets.

Thirdly, we could use a richer feature representation based for example on the word frequency and the information gain values and compare the results with he current ones. Also, different representation of the semantic features could be performed. As well as the three methods we introduce in this project, several other could be used.

These are some of the ideas for extending the current project and produce new results which can then be compared with the ones presented in the present work.

Chapter 6

Conclusions

As this thesis comes to an end, it is time to review the content of the previous chapters. *Chapter 1* outlined the idea and the purposes of the project. It presented the extent of the problem of the data overload and it suggested our approach: the use of semantic features as a rich input to classifiers. *Chapter 2* introduced the relevant research area around the project and provided the necessary information in order to understand all the method used in the present project. *Chapter 3* outlined the implementation procedure followed in order to create the semantic features and use them in the classification procedure. *Chapter 4* presented the results of all the experiments that took place during the project and in *Chapter 5* a discussion of the achieved results was presented. In particular, we found that the combination of semantic and content-based features is not as simple as it may look. The combination of the features does not seem to improve the performance of the classifiers. In the same Chapter, some future suggestions that could extend the project were presented.

At this point the objectives of the project should be mentioned again and the outcome of the work should be evaluated. As mentioned in *Chapter 1*, the objective of the project was to implement a new approach, according to which semantic features are used, in addition to the traditional ones, in the classification procedure of a dataset. The latter gave us an insight of how these two different kinds of features can deal with each other and if their combination can improve the classifiers' performance. As we mentioned in the previous paragraph, the combination of the features was not as easy as it looks. The classifiers did not seem to can easily deal with the combination of different kind of features, even if they have satisfying results when they use the different features separately.

The outcome of the project is highly significant, not only because it opened the door to a new direction, but also because it gave an insight of how classifiers are dealing with semantic features and how the latter can be combined with the traditional ones.

References

1. Usama M. Fayyad, Gregory Piatesky-Shapiro, Padhraic Smuth and Ramasamy Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI Press 1996

2. T. Mitchell. *Machine Learning.* McGraw-Hill, 1997

3. D. Michie, D. J. Spiegelhalter & C. C. Taylor. *Machine Learning, neural and statistical classification,* (edited collection). New York: Ellis Horwood, 1994

4. J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufman, 1993

5. J. Shafer, R. Agrawal, M. Mehta. "*SPRINT: A scalable parallel classifier for data mining".* In Proc. Of the VLDB Conference, Bombay, India, September 1996

6. M. Berry, G. Linoff. *Data Mining Techniques For marketing, Sales and Customer support.* John Willey & Sons, Inc, 1996

7. V. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, 1995

8. P. Clarkson & P. J. Moreno. *On the Use of Support Vector Machines for Phonetic classification,* 1999

9. T. Joachims. *Text categorization with support vector machines: Learning with many relevant features.* In C. Nedellec and C. Rouveirol, editors, Proc. European Conference in Machine learning, pages 137-142, Chemitz, Germany, 1998, Springer

10. J. Kivinen, M. Warmuth, and P. Auer. *The perceptron algorithm vs. winnow: Linear vs. logarithmic mistake bounds when few input variables are relevant.* In Conference on Computational Learning Theory, 1995

11. Andrew B. Watkins and Lois C. Boggess. *A Resource Limited Artificial Immune Classifier.* In Proceedings of Congress on Evolutionary Computation, Pages 926-931. IEEE, May 2002. Part of the 2002 IEEE World Congress on Computational Intelligence held in Honolulu, HI, USA, May 12-17, 2002

12. Jerome Carter, *"The immune system as a model for pattern recognition and classification,"* J American Medical Informatics Association (JAMIA), vol. 7, no. 1, pp. 28-41, 2000

13. Jon Timmis, Mark Neal, and John Hunt, *"An artificial immune system for data analysis,"* Biosystems, vol. 55, no. 1/3, pp. 143-150, 2000

14. Andrew B. Watkins, *"AIRS: A resource limited artificial immune classifier,"* M.S. thesis, Mississippi State University, 2001

15. Tim Berners-Lee, James Hendler, and Ora Lassila. *The Semantic Web.* Scientific American, 284(5):34-43, May 2001

16. AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. *Learning to Map between Ontologies on the Semantic Web.* WWW2002, Honolulu, Hawaii, USA, May 7-11, 2002

17. Parent C, Spaccapietra S. *Issues and approaches of database integration.* CACM 41(5):166-178, 1998

18. Erhard Rahm, Philip Bernstein *.A survey of approaches to automatic schema matching.* P. Scheuermann (Editor). Springer, 2001

19. Ian H. Witten, Eibe Frank, Len Trigg, Mark Hall, Geoffrey Holmes, and Sally Jo Cunningham. *Weka: Practical Machine Learning Tools and Techniques with Java Implementations.* New Zealand, 1999

20. Watkins, Andrew. *Artificial Immune Recognition System (AIRS) C++ source code*, 2002

21. Watkins, Andrew and Jon Timmis. *Artificial Immune Recognition System (AIRS): Revisions and Refinements*. In Proceedings of 1st International Conference on Artificial Immune Systems (ICARIS), pages 173-181. University of Kent at Canterbury, September 9-11 2002

22. Tom Morrison. *Similarity Measure Building for Website Recommendation within an Artificial Immune System*. MSc thesis. University of West of England, 2002

23. P. W. Lord, R. D. Stevens, A. Brass and C. A. Goble. *Semantic similarity measures as tools for exploring the Gene Ontology.* University of Manchester, 2003

24. Hong-Hai, Sergey Melnik and Erhard Rahm. *Comparison of Schema Matching Evaluations*. University of Leipzig. August 2002

25. Sunita Sarawagi, Soumen Chakrabarti and Shantanu Godbole. *Cross-training: Learning probabilistic mappings between topics*. IIT Bombay. SIGKDD 2003

26. Fausto Giunchiglia and Pavel Shvaiko. *Semantic Matching.* University of Trento. April 2003

27. Fausto Giunchiglia, Pavel Shvaiko and Mikalai Yatskevish. *S-MATCH: An algorithm and an implementation of semantic matching.* University of Trento. February 2004

28. Deborah L. McGuinness, Pavel Shvaiko, Fausto Giunchiglia and Paulo Pinheiro da Silva. *Towards Explaining Semantic Matching.* University of Trento. February 2004

29. T. Berners-Lee, J. Hendler, and O. Lassila. *The Semantic Web*. Scientific American, 279, 2001

30. M. F. Porter. *An algorithm for suffix stripping.* _Program_, Vol. 14, No. 3, pp.130-137, 1980

31. Wilcoxon Mann-Whitney Test http://www.fon.hum.uva.nl/Service/Statistics/Signed_Rank_Test.html

32. Julie Greensmith. *New Frontiers For An Artificial Immune System*. University of Leeds. MSc Thesis. September 2003

Appendix A

The categories of each schema

```
===== CONTENT TOPIC =====
0: component
1: component_compatibility
2: component_firmware/BIOS
3: component_manageability
4: component_power management
5: security
6: software
7: software_compatibility
8: software_diagnostic
9: software_driver
10: software_hp
11: software_manageability
12: software_operating system
13: software_utility
14: specifications
15: specifications_product
16: warranty

====== ENVIRONMENT =====
0: DOS
1: Linux
2: Novell
3: OpenVMS
4: Unix
5: Windows-English
6: Windows-localized
7: cluster
8: internet
9: network
10: wireless

====== SYMPTOM =====
0: error message
1: error message_beep codes
2: error message_displayed on
control panel
3: error message_displayed on
front panel
4: error message_displayed on
pc
5: error message_installing
6: error message_system
7: feeding issue
8: feeding issue_jam
9: output
10: output_color quality
11: output_unexpected
12: performance
13: performance_freeze
14: performance_no response
15: performance_noise
16: performance_slow

====== USER TASK =====
0: compare
1: fix problem
2: fix problem_configure
3: fix problem_download
4: fix problem_obtain service
5: fix problem_order
6: fix problem_reinstall
7: fix problem_replace
8: fix problem_setup
9: fix problem_shutdown
10: fix problem_startup
11: fix problem_uninstall
12: fix problem_upgrade
13: maintain
14: maintain_download
15: maintain_replace
16: maintain_upgrade
17: obtain service
18: order
19: setup
20: setup_configure
21: setup_loading
22: setup_upgrade
23: uninstall
24: uninstall_reinstall
25: use product
26: use product_choosing
27: use product_configure
28: use product_loading
29: use product_shutdown
30: use product_startup

====== MAIN COMPONENT =====
0: CPU
1: LCD
2: RAID
3: cables
4: chassis
5: control panel
6: controller
7: docking station
8: drive
9: drive_CD
10: drive_CD-RW
11: drive_DVD
12: drive_IDE
13: drive_SCSI
14: drive_floppy
15: drive_hard
```

```
16: drive_library
17: drive_removable
18: drive_tape
19: drive_zip
20: enclosures/racks
21: expansion card
22: expansion card_modem
23: expansion card_multi-
function
24: expansion card_network
25: expansion card_remote
control
26: expansion card_sound
27: expansion card_video
28: fan
29: memory
30: motherboard
31: network device
32: network device_hub
33: network device_router
34: network device_switch
35: peripheral
36: peripheral_camera
37: peripheral_joystick
38: peripheral_keyboard
39: peripheral_microphone
40: peripheral_monitor
41: peripheral_mouse
42: peripheral_printer
43: peripheral_scanner
44: peripheral_speakers
45: port
46: port_PS/2
47: port_SCSI
48: port_USB
49: port_infrared
50: port_parallel
51: port_serial
52: power
53: power_battery
54: power_power cord
55: power_power module
56: power_power protection
57: slot
58: slot_PCI slot
59: supplies
60: supplies_optical disc

====== MINOR COMPONENT 1 =====
0: CPU
1: LCD
2: RAID
3: cables
4: chassis
5: control panel
6: controller
7: docking station
8: drive
9: drive_CD
10: drive_DVD
11: drive_IDE
12: drive_SCSI
13: drive_floppy
14: drive_hard
15: drive_library
16: drive_removable
17: drive_tape
18: drive_zip
19: enclosures/racks
20: expansion card
21: expansion card_modem
22: expansion card_network
23: expansion card_sound
24: expansion card_video
25: fan
26: front panel
27: memory
28: motherboard
29: network device
30: network device_router
31: network device_switch
32: peripheral
33: peripheral_fax
34: peripheral_keyboard
35: peripheral_monitor
36: peripheral_mouse
37: port
38: port_PS/2
39: port_SCSI
40: port_USB
41: port_serial
42: power
43: power_battery
44: power_power cord
45: power_power module
46: power_power protection
47: slot
48: slot_PCI slot
49: supplies
50: supplies_floppy disk

====== MINOR COMPONENT 2 =====
0: CPU
1: RAID
2: chassis
3: controller
4: drive
5: drive_DVD
6: drive_floppy
7: drive_hard
8: drive_tape
9: expansion card
10: expansion card_modem
11: expansion card_video
12: memory
13: motherboard
14: network device
15: network device_router
16: peripheral
17: peripheral_fax
18: peripheral_keyboard
19: peripheral_monitor
```

```
20: peripheral_speakers
21: port
22: port_AGP
23: port_SCSI
24: port_USB
25: port_serial
26: power
27: power_battery
28: power_power module
29: slot
30: slot_PCI slot
31: supplies
32: supplies_floppy disk

====== PRODUCT FUNCTION =====
0: backing up
1: backing up_audio
2: backing up_data
3: connecting
4: copying
5: faxing
6: printing
7: restoring
8: sending

====== SOFTWARE TOPIC =====
0: business productivity
1: development tools
2: entertainment
3: financial
4: personal productivity
5: presentations
6: spreadsheet
7: synchronization
8: virus protection
9: word processing
```

Appendix B

One-to-one schema mapping: algorithms, results

B.1 Algorithm for Optimal 1:1 Mapping

An exhaustive algorithm of optimal 1:1 mapping is implemented. Below, a brief description of the algorithm is given: Imagine we have the above similarity matrix (table 2), where {A, B, C, …} and {X, Y, Z, …} are two schemata.

F	**X**	**Y**	**Z**
A	**0.5**	**0.7**	**0.4**
B	**0.1**	**0.9**	**0.2**
C	**0.2**	**0.8**	**0.1**

Then, an algorithm which finds the optimal 1:1 mapping between the two schemata is the following:

```
bestMatch (cats1, cats2, sumSoFar, matches)
FOR (index = 0; index < cats2.length; index++)
   newSum=sumSoFar +similarity(cats1[0], cats2[index])
  matches = matches + {cats1[0], cats2[index]}
  newCats1 = cats1 - cats1[0]
 newCats2 = cats2 - cats2[index]
 IF(newCats2 IS empty) THEN
        IF (newSum > maxSum) THEN
            match = matches
            maxSum = newSum
      ENDIF
 ELSE
        bestMatch(newCats1, newCats2, newSum, match)
        ENDIF
NEXT index
```

B.2 Results of 1:1 mapping (the most representative ones)

schema1 - schema2	correct	error	% correct	correspondences: schema1 -->schema2
TOPIC - ENVIRONMENT	1099	520	67.88140828	3 --> Unix
				14 --> Windows-English
TOPIC - SYMPTOM	517	1133	31.33333333	8 --> error message_displayed on pc
				7 --> performance
				2 --> performance_no response
TOPIC - USERTASK	757	1638	31.60751566	13 --> fix problem
				2 --> fix problem_upgrade
				1 --> fix problem_configure
				1 --> fix problem_setup
TOPIC - MAIN COMPONENT	235	1273	15.58355438	1 --> memory
				1 --> drive_CD
				5 --> controller
				4 --> expansion card_video
				1 --> LCD
				1 --> drive
				1 --> power
				1 --> drive_DVD
				1 --> drive_hard
				1 --> expansion card_remote control

TOPIC - MINOR COMPONENT 1	84	341	19.76470588	4 --> motherboard
				3 --> peripheral_monitor
				4 --> controller
				1 --> power
SYMPTOM - MINOR COMPONENT 1	66	244	21.29032258	3 --> motherboard
				1 --> port_USB
				3 --> peripheral_monitor
				5 --> controller
				1 --> RAID
				2 --> expansion card_video
				1 --> expansion card_network
SYMPTOM - MINOR COMPONENT 2	20	39	33.89830508	6 --> CPU
				2 --> peripheral_monitor
				2 --> port_USB
				4 --> controller
				1 --> network device_router
				1 --> RAID
SYMPTOM - PRODUCT FUNCTION	46	31	59.74025974	12 --> connecting
				4 --> backing up
SYMPTOM - SOFTWARE	35	50	41.17647059	7 --> personal productivity
				3 --> business productivity
				4 --> entertainment
				1 --> development tools
				1 --> virus protection

Appendix C

Some parts of the implemented code (Java and Bash Scripts)

C.1 Bash script : extraction of schema information form the XML files

```
#!/bin/bash

# Takes as input the XML files and creates a single file containing all the information
# fo rall schemas and the categories the files belong

printf "DocID,Content Topic,Content Detail,Environment,Symptom,Symptom Detail,User Task,User Task Detail,main
component,main component detail,minor component1,minor component1 detail,minor component2,minor component2
detail,product function,product function detail,software topic\n\n"

for F in *.XML
do
egrep '<original_docid>|<content_topic>|<content_topic_detail>' "$F" |cut -f2 -d\> |cut -f1 -d\< | (while read ID
do
read TOPIC
read DETAIL1
read DETAIL2
read DETAIL3
read DETAIL4
read DETAIL5

if [ -z "$TOPIC" ] || [ "$TOPIC" = "not applicable" ] ; then
```

```
printf "$ID,NA,NA,"
elif [ -z "$DETAIL1" ] || [ "$DETAIL1" = "not applicable" ]; then
printf "$ID,$TOPIC,NA,"
elif [ -z "$DETAIL2" ]; then
printf "$ID,$TOPIC,$TOPIC"_"$DETAIL1,"
elif [ -z "$DETAIL3" ]; then
printf "$ID,$TOPIC,$TOPIC"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,$TOPIC"_"$DETAIL2,"
elif [ -z "$DETAIL4" ]; then
printf "$ID,$TOPIC,$TOPIC"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,$TOPIC"_"$DETAIL2,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,$TOPIC"_"$DETAIL3,"
elif [ -z "$DETAIL5" ]; then
printf "$ID,$TOPIC,$TOPIC"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,$TOPIC"_"$DETAIL2,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,$TOPIC"_"$DETAIL3,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,$TOPIC"_"$DETAIL4,"
else
printf "$ID,$TOPIC,$TOPIC"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,$TOPIC"_"$DETAIL2,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,$TOPIC"_"$DETAIL3,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,$TOPIC"_"$DETAIL4,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,$TOPIC"_"$DETAIL5,"
fi

done);
egrep '<original_docid>|<environment>' "$F" |cut -f2 -d\> |cut -f1 -d\< | (while read ID
do
read ENVIRONMENT1
read ENVIRONMENT2
read ENVIRONMENT3
read ENVIRONMENT4

if [ -z "$ENVIRONMENT1" ] || [ "$ENVIRONMENT1" = "not applicable" ]; then
printf "NA,"
elif [ -z "$ENVIRONMENT2" ]; then
```

```
printf "$ENVIRONMENT1,"
elif [ -z "$ENVIRONMENT3" ]; then
printf "$ENVIRONMENT1,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,$ENVIRONMENT2,"
elif [ -z "$ENVIRONMENT4" ]; then
printf "$ENVIRONMENT1,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,$ENVIRONMENT2,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,$ENVIRONMENT3,"
else
printf "$ENVIRONMENT1,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,$ENVIRONMENT2,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,$ENVIRONMENT3,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,$ENVIRONMENT4,"
fi
done);

egrep '<original_docid>|<symptom>|<symptom_detail>' "$F" |cut -f2 -d\> |cut -f1 -d\< | (while read ID
do
read SYMPTOM
read DETAIL1
read DETAIL2
read DETAIL3


if [ -z "$SYMPTOM" ] || [ "$SYMPTOM" = "not applicable" ] ; then
printf "NA,NA,"
elif [ -z "$DETAIL1" ] || [ "$DETAIL1" = "not applicable" ]; then
printf "$SYMPTOM,NA,"
elif [ -z "$DETAIL2" ]; then
printf "$SYMPTOM,$SYMPTOM"_"$DETAIL1,"
elif [ -z "$DETAIL3" ]; then
printf "$SYMPTOM,$SYMPTOM"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,$SYMPTOM"_"$DETAIL2,"
else
printf "$SYMPTOM,$SYMPTOM"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,$SYMPTOM"_"$DETAIL2,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,$SYMPTOM"_"$DETAIL3,"
fi
done);

egrep '<original_docid>|<user_task>|<user_task_detail>' "$F" |cut -f2 -d\> |cut -f1 -d\< | (while read ID
do
read USERTASK
read DETAIL1
read DETAIL2
read DETAIL3
read DETAIL4
read DETAIL5

if [ -z "$USERTASK" ] || [ "$USERTASK" = "not applicable" ] ; then
printf "NA,NA,"
elif [ -z "$DETAIL1" ] || [ "$DETAIL1" = "not applicable" ]; then
printf "$USERTASK,NA,"
elif [ -z "$DETAIL2" ]; then
printf "$USERTASK,$USERTASK"_"$DETAIL1,"
elif [ -z "$DETAIL3" ]; then
printf "$USERTASK,$USERTASK"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,$USERTASK"_"$DETAIL2,"
elif [ -z "$DETAIL4" ]; then
printf "$USERTASK,$USERTASK"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,$USERTASK"_"$DETAIL2,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,$USERTASK"_"$DETAIL3,"
elif [ -z "$DETAIL5" ]; then
printf "$USERTASK,$USERTASK"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,$USERTASK"_"$DETAIL2,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,$USERTASK"_"$DETAIL3,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,$USERTASK"_"$DETAIL4,"
else
printf "$USERTASK,$USERTASK"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,$USERTASK"_"$DETAIL2,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,$USERTASK"_"$DETAIL3,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,$USERTASK"_"$DETAIL4,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,$USERTASK"_"$DETAIL5,"

fi
done);

egrep '<original_docid>|<main_component>|<main_component_detail>' "$F" |cut -f2 -d\> |cut -f1 -d\< | (while read ID
do
read MAINCOMP
read DETAIL1
read DETAIL2
read DETAIL3
read DETAIL4
read DETAIL5
read DETAIL6


if [ -z "$MAINCOMP" ] || [ "$MAINCOMP" = "not applicable" ] ; then
printf "NA,NA,"
elif [ -z "$DETAIL1" ] || [ "$DETAIL1" = "not applicable" ]; then
printf "$MAINCOMP,NA,"
elif [ -z "$DETAIL2" ]; then
printf "$MAINCOMP,$MAINCOMP"_"$DETAIL1,"
elif [ -z "$DETAIL3" ]; then
printf "$MAINCOMP,$MAINCOMP"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL2,"
elif [ -z "$DETAIL4" ]; then
printf "$MAINCOMP,$MAINCOMP"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL2,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL3,"
elif [ -z "$DETAIL5" ]; then
printf "$MAINCOMP,$MAINCOMP"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL2,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL3,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL4,"
elif [ -z "$DETAIL6" ]; then
printf "$MAINCOMP,$MAINCOMP"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL2,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL3,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL4,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL5,"
else
printf "$MAINCOMP,$MAINCOMP"_"$DETAIL1,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL2,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL3,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL4,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL5,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,$MAINCOMP"_"$DETAIL6,"
fi


done);

egrep '<original_docid>|<minor_component1>|<minor_component1_detail>' "$F" |cut -f2 -d\> |cut -f1 -d\< | (while read
ID
do

read MINORCOMP1
read DETAIL1
read DETAIL2

if [ -z "$MINORCOMP1" ] || [ "$MINORCOMP1" = "not applicable" ] ; then
printf "NA,NA,"
elif [ -z "$DETAIL1" ] || [ "$DETAIL1" = "not applicable" ]; then
printf "$MINORCOMP1,NA,"
elif [ -z "$DETAIL2" ]; then
printf "$MINORCOMP1,$MINORCOMP1"_"$DETAIL1,"
else
printf "$MINORCOMP1,$MINORCOMP1"_"$DETAIL1,NA,NA,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,$MINORCOMP1"_"$DETAIL2,"
fi
done);
```

```
egrep '<original_docid>|<minor_component2>|<minor_component2_detail>' "$F" |cut -f2 -d\> |cut -f1 -d\< | (while read
ID
do

read MINORCOMP2
read DETAIL1
read DETAIL2

if [ -z "$MINORCOMP2" ] || [ "$MINORCOMP2" = "not applicable" ] ; then
printf "NA,NA,"
elif [ -z "$DETAIL1" ] || [ "$DETAIL1" = "not applicable" ]; then
printf "$MINORCOMP2,NA,"
elif [ -z "$DETAIL2" ]; then
printf "$MINORCOMP2,$MINORCOMP2"_"$DETAIL1,"
else
printf "$MINORCOMP2,$MINORCOMP2"_"$DETAIL1,NA,NA,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,$MINORCOMP2"_"$DETAIL2,"
fi
done);

egrep '<original_docid>|<product_function>|<product_function_detail>' "$F" |cut -f2 -d\> |cut -f1 -d\< | (while read
ID
do
read PRODUCT
read DETAIL1
read DETAIL2

if [ -z "$PRODUCT" ] || [ "$PRODUCT" = "not applicable" ] ; then
printf "NA,NA,"
elif [ -z "$DETAIL1" ] || [ "$DETAIL1" = "not applicable" ]; then
printf "$PRODUCT,NA,"
elif [ -z "$DETAIL2" ]; then
printf "$PRODUCT,$PRODUCT"_"$DETAIL1,"
else
printf "$PRODUCT,$PRODUCT"_"$DETAIL1,NA\n"
printf "$ID,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,$PRODUCT"_"$DETAIL2,"
fi
done);

egrep '<original_docid>|<software_topic>' "$F" |cut -f2 -d\> |cut -f1 -d\< | (while read ID
do

read SOFTWARE

if [ -z "$SOFTWARE" ] || [ "$SOFTWARE" = "not applicable" ] ; then
printf "NA\n"
else
printf "$SOFTWARE\n"

fi
done);

done
```

C.2 Configuration file for AIRS

```
#values describing the data

# Training File:
TrainF contentTopic.trn

# Test data File:
TestF contentTopic.tst

# norms file
# the norms file has 2 lines in it with 2 feature vectors
# the first line has the minimum value for each feature
# and the 2nd line as the maximum vale for each feature
NormsFile contentTopic.norms

# where to output the statistics
Stats contentTopic.sts

# where to output the trained system
NetOut contentTopic.net

# where to output the predictions for each vector in the test
# and training set
Pred contentTopic.prd

# the number of features in each vector in the data set
NGenes 4

# the number of classes in the data set
Classes 3

# is there a label for each vector?
# if so, this label must be in the 1st column of the data
labels yes


#various learning parameters

# number of passes through the training data
# has only been experimented, really, with value of 1
Epochs 1

# the clonal rate
CRate 10

# the mutation rate
MRate 0.1

# number of training data items to seed the network
InitSize 0

# the Affinity Threshold Scalar (ATS)
Nat 0.2

# the stimulation threshold
StimT 0.95

# the number of resources allowed in the system
NumRes 200

# the K value used for the systems response to data items
BcellK 7

# this is the hyper mutation rate for Memory Cell offspring
# if it is not set, then the default value is 2
HRate 10.0

# if NumRes is not set, then RRate is used as a scalar for
# determining the number of resources in the system
# numres = RRate * number of training items
#RRate 3.0

# this is the rate at which resources are added to an ARB
# if it is not specified then RAddRate = Clonal Rate
#RAddRate 5

# if InitSize is not set, then TrainP is used to determine
# the number of data items to use to seed the network
# initsize = TrainP * number of training items
#TrainP 0.2
```

C.3 Code: creates the semantic vector for the “counting edges” approach

```
/*
 * Created on 06-Aug-2004
 * Author: Anastasia Krithara
 *
 * Creates a table containing the distances between the
 * categories of a given schema
 * Distance: the number of edges
 */

import java.io.*;
import java.util.*;

public class SemanticVector
{
   public static boolean TRAIN = false;
            public static String infoLine = "";
            public static String category = "";
            public static String categories = "";
            public static String nextToken = "";
            public static String outname = "";
            public static int pos1 = 0;
            public static int pos2 = 0;
            public static String[] classes;
            public static File output;
            public static File docs;
            public static File path;
            public static String line="";
            public static StringTokenizer tokenizer;
            public static String delim = "\n";
            public static boolean EOF = false;
            public static boolean found = false;
            public static boolean exist = false;
            public static boolean child1 = false;
            public static boolean child2 = false;
  public static File[] OUTfiles; //array which holds the names of all out files
  public static File[] CAT;
  public static File[] CATEST;
  public static int len = 0;
 public static int[][] SemVect;
 public static double[][] NormVect;

            public static double[][] createVect(int[][] SemVect,double[][] NormSem,String[] classes) throws IOException
            {
                                        len = classes.length;
                                        SemVect = new int[len][len];

                               for(int w=0;w<len;w++)
                               {
                                        for(int q=0;q<len;q++)
                                        {
                                                    child1 = false;
                                                    child2 = false;
                                                    pos1 = classes[w].length();
                                                    pos2 = classes[q].length();
                                                    char[] ch1 = classes[w].toCharArray();
                                                    char[] ch2 = classes[q].toCharArray();

                                                    for(int a=0;a<ch1.length;a++)
                                                    {
                                                                if(ch1[a] == '_')
                                                                {
                                                                            child1 = true;
                                                                            pos1 = a;
                                                                            break;
                                                                }
                                                    }//end for

                                                    for(int b=0;b<ch2.length;b++)
```

```
                        {
                                    if(ch2[b] == '_')
                                    {
                                                child2 = true;
                                                pos2 = b;
                                                break;
                                    }
                        }//end for


                        if(classes[w].equals(classes[q]))
                                    SemVect[w][q] = 0;

                        else if(classes[w].startsWith(classes[q]+"_") || classes[q].startsWith(classes[w]+"_"))
                                    SemVect[w][q] = 1;

                        else if(classes[w].substring(0,pos1).equals(classes[q].substring(0,pos2)))
                                    SemVect[w][q] = 2;

                        else
                        {
                                    if(child1 && child2)
                                                SemVect[w][q] = 4;

                                    else if((child1 && !child2) || (!child1 && child2))
                                                SemVect[w][q] = 3;

                                    else if(!child1 && !child2)
                                                SemVect[w][q] = 2;
                        }//end else

                        //System.out.println("SemVect["+w+"]["+q+"]: "+SemVect[w][q]);
            }//end for q
}//end for w

NormSem = normalize(SemVect);
```

```
				return NormSem;

}//end createVect

public static double[][] normalize(int[][] vector)
{
		double[][] afterVect = new double[vector.length][vector[0].length];
		double Sum = 0;

		for(int l=0;l<vector.length;l++)
		{
			Sum=0;
			for(int m=0;m<vector[l].length;m++)
			{
				Sum = Sum + vector[l][m];
				//System.out.println("vector["+l+"]["+m+"]: "+vector[l][m]);
			}//end for m

			//System.out.println("Sum: "+Sum);
			for(int m=0;m<vector[l].length;m++)
			{
				afterVect[l][m] = (double)vector[l][m]/Sum;
			}//end for m

		}//end for l

		return afterVect;
}//end normalize


public static String[] findClasses(String FOLDER,int FINAL,int FOLD)
{
		for(int Final=0;Final<FINAL;Final++)
		{
```

```
String str = Integer.toString(Final+1);

for (int Fold=0;Fold<FOLD;Fold++)
{
        len = 0;
        String nbr = Integer.toString(Fold+1);
        String name = "Fold".concat(nbr);
        String PATH = "/home/anakri/Final"+str+"/Fold"+nbr+"/TRAINING/SCHEMAS/"+FOLDER+"/";
        String TESTPATH = "/home/anakri/Final"+str+"/Fold"+nbr+"/TEST/SCHEMAS/"+FOLDER+"/";

path = new File(PATH);
//System.out.println(path);
CAT = path.listFiles();

for(int l=0;l<CAT.length;l++)
{
        if(CAT[l].isDirectory())
                len++;
}

 path = new File(TESTPATH);

   CATEST = path.listFiles();
   for(int l=0;l<CATEST.length;l++)
   {
        exist = false;
        if(CATEST[l].isDirectory())
        {
                for(int d=0;d<CAT.length;d++)
                {
                        if(CATEST[l].getName().equals(CAT[d].getName()))
                                exist = true;
                }
                if(!exist)
                        len++;
        }//end if
```

```
                    }//end for
                  classes = new String[len];
int i=0;
for(int c=0;c<CAT.length;c++)
{
          if(CAT[c].isDirectory())
          {
                  classes[i] = CAT[c].getName();
                  i++;
          }

}//end for


 path = new File(TESTPATH);

  CATEST = path.listFiles();
  for(int c=0;c<CATEST.length;c++)
  {
          exist = false;
          if(CATEST[c].isDirectory())
          {
                  for(int d=0;d<CAT.length;d++)
                  {
                          if(CATEST[c].getName().equals(CAT[d].getName()))
                                  exist = true;
                  }
                  if(!exist)
                  {
                    classes[i] = CATEST[c].getName();
                    i++;
                  }
           }//end if
  }//end for
}//end FOLD
```

```
   }//end FINAL

             return classes;
  }//end findClasses

}//end SemanticVector
```