TUHH
Technische Universität Hamburg-Harburg

STS
Software
Technology
Systems

Master Thesis

A Machine Learning Approach for Fusing Semantic Annotations of Multimodal Data

submitted by
Thorben Ole Heins

supervised by
Prof. Dr. rer.-nat. habil. Ralf Möller
Prof. Dr. Karl-Heinz Zimmermann
Dipl.-Ing. Kamil Sokolski

Institute for Software Systems
Hamburg University of Technology
Hamburg, Germany

Acknowledgments

First of all I want to thank Prof. Möller for giving me the opportunity to work on such an interesting topic and persuading Prof. Zimmermann to be my second supervisor. As predicted, it was rather an exciting rafting tour than a calm and relaxing canoing trip.

Secondly I want to thank Kamil Sokolski for all those inspiring discussions. I also want to thank him for commenting on all the drafts, which was a very valuable input during the end phase of this work.

Also Michael Wessel has to be mentioned here, as he helped me a lot, working with RacerPro and always provided me with the most recent releases.

Special thanks to everyone who helped me during the corrections process

I also want to thank my family, for supporting me in any imaginable way throughout all my studies.

Last but not least I want to thank my girlfriend for everything!

Declaration

I hereby confirm that I have authored my master thesis with the title

> *A Machine Learning Approach for Fusing Semantic Annotations of Multimodal Data*

independently and without use of others than the indicated resources. All passages taken out of publications or other sources are marked as such.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich meine Masterthesis mit dem Thema

> *A Machine Learning Approach for Fusing Semantic Annotations of Multimodal Data*

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, October 29, 2009

Thorben Ole Heins

Abstract

In information retrieval systems, structures are built up that represent (multimodal) documents (e.g. web pages) and describe their contents. These structures are called semantic annotations. One challenge is to combine the semantic annotations of the different modalities of one document in a way that individuals that are described in annotations of more than one modality, are identified as the same individual. This is called fusion and is until now realized by the use of logical reasoning techniques. In this work we developed a machine learning based approach for fusion, implemented a prototype using this approach, tested it with kNN and SMO classifiers and evaluated its performance.

Contents

List of Figures

List of Tables

Chapter 1

Motivation

"You don't know why it works. You don't know how it works. You just push a button - and it works!"[1]

(Scott Collins about the user view on software.)

1.1 Introduction

As stated in this citation, the user in general does not care about the actual algorithmic methods that are used to present a result (of whichever kind) to the user. As long as the user is presented with an *acceptable* result in a *reasonable* time, the user does not care which approach the software developer used to solve a problem. The scientific view of course is another. The researcher tries to enhance the two factors of *time of response* and *quality of the result* that is presented to the user. One area in which this is particularly important is the area of information retrieval (IR).
Today the need for information retrieval in the world wide web is becoming more and more important every day. Individual persons as well as companies, such as news agencies for example, are using the means to search for content on the web every day on the whole planet. In order to be able to present good search results to the users, IR systems build up a structure describing the content of a (multimodal) document. One approach to build up such a structure is known as semantic annotation.
Multimodal documents, like web pages, may contain elements like text, images or audio visual parts. When semantically annotating the content of such multimodal documents, IR systems want to exploit the fact, that there are correlations between the semantic annotations of the different parts of the document. One approach that exploits these fact is known as fusion. Until now by the use of logic based rules, the semantic annotations of a multimodal document are fused in a way, so that new knowledge about the multimodal document as a whole can be gained. For example one can think of semantic annotations of an image and a text part of a multimodal document. If the annotations for both parts of the document contain knowledge about the fact, that in each of the document parts (modalities) a person is present, it might be possible that these

[1] From the documentary *Code Rush* about the beginnings of the Mozilla project.

annotations describe the same actual person in reality. The challenge to identify those annotations that describe the same object is approached by fusion. This approach has already been implemented by Atila Kaya and is described in his doctoral thesis [8]. On the basis of the results of this implementation we now want to develop a machine learning based approach for fusion.
One of the potential benefits of a machine learning approach is that the performance regarding the time fusion takes might be decreased.
Until now, the approach implemented by Kaya also is specifically tailored to the application domain, regarding the logical rules, on which basis the fusion takes place. We want to research, whether there is a more general machine learning based approach that can be used for fusion in any kind of domain.
Another reason to use a machine learning based approach instead of a logic based one is that machine learning is known to be able to operate on large scales of data which is the kind of data IR systems operate on, while logic based fusion can run into problems doing so [14].
Of course using machine learning for fusion also does have disadvantages compared to the use of logic based fusion. Machine learning is, as opposed to the logic used in Kaya's implementation, not exact. So when using machine learning, one has to deal with "false" knowledge. Coming back to the aforementioned example, there might be a person in image and text, who are identified to be the same person, but it is plain wrong. Anyhow this might be negligible, as in in the typical IR scenario the user will himself make a selection among the results he is presented. Of course if there are "too many" false results presented to the user, he will most likely use another IR system in future. It is also the task of this work to figure out whether machine learning for fusion has the potential to help to not deliver "too many incorrect" results to the user.

1.2 Problem Definition

In this thesis we want to find out whether a better performance for fusion can be achieved by the use of selected machine learning algorithms than by the use of a logic reasoner. In order to do so, we at first have to develop an approach for fusion with machine learning. Then we have to select some machine learning algorithms that we want to run our tests with. Having done that, we then have to find a machine learning tool, that supports those algorithms. In the following implementation period we have to develop a tool that creates input data for the selected machine learning tool, so that we can afterwards run tests on the approach that was developed beforehand. In the evaluation of the run tests, we have to find a measure to judge the approach's and algorithm's performance on the test data.

1.3 Structure of this work

This work is divided into seven chapters. In this one we presented the topic of this work. In chapter 2 we introduce the theory that stands behind the description logic we used in this work. Then in chapter 3 we present the general idea behind machine learning and present the particular algorithms and the tool we used in this work. Afterwards in chapter 4 we at first present the logic

based approach for fusion that is used until now, before we present the machine learning based approach for fusion that is developed in this work. The prototypical implementation including architecture and test description is introduced in chapter 5. In chapter 6 we describe the test data and evaluation method we used. We also evaluate the test results, that were gathered during the tests. In the final chapter 7 we summarize the achievements before giving an outlook on the future research.

Chapter 2

Description Logic

2.1 Introduction

As the test data that is used in this work is based on Description Logic (DL) [2], it is necessary to introduce the basic ideas standing behind it.
In order to formalize the terminology of an application domain, a knowledge representation language is needed. Such a language needs to be able to express things like "concepts", "roles", "individuals" and "axioms". DLs are one approach to fulfill the need for such a language.
A Description Logic System consists of four parts. First of all there is the description language which has the ability to build complex concepts based on atomic concepts and roles. Such a language does have formal and logic based semantics. In section 2.2 we introduce such a language.
In the center of such a DL system there is the knowledge base (kb), consisting of two parts: t-box and a-box. The t-box (terminological box) defines the terminology of the application domain as a finite set of axioms. It defines the vocabulary that is necessary to build the a-box (assertion box) which contains facts about the world of the application domain. The a-box is a non-empty finite set of assertions.
The fourth component of a DL system is the reasoning component (reasoner) which is able to derive implicitly represented knowledge. Such a reasoner bases its algorithms on well-defined basic inference procedures, here are some examples: subsumption and the instance problem. The subsumption allows to order the terminology of an application domain in a subsumption hierarchy providing useful information about the connections between different concepts of the terminology.

2.2 $\mathcal{ALC}$

DLs are a family of logic-based knowledge representation languages. The application areas for DLs are manifold, some usage scenarios are databases, natural language processing or bio-medical ontologies [3]. Depending on the application scenario the ontology engineer has to choose among the various languages that are offered. The test data that we used in this work uses the language $\mathcal{SHN}(\mathcal{D})$,

but for the way we used DL in this work another, less expressive language was sufficient: $\mathcal{ALC}$.

In DL languages there are two kinds of *atomic symbols* being *atomic concepts* and *atomic roles*. We use the letters A and B for atomic concepts an R for atomic roles. The letters C and D are used for arbitrary *concept descriptions*. By the use of so called *concept* and *role constructors* complex descriptions can be built using atomic symbols. The letters S and T are used for arbitrary role descriptions.

The syntax of $\mathcal{ALC}$ is defined as:

Concept syntax:

C, D	$\longrightarrow$	$A \mid$	(atomic concept)
		$\top \mid$	(universal concept)
		$\bot \mid$	(bottom concept)
		$\neg A \mid$	(atomic negation)
		$C \sqcap D \mid$	(intersection)
		$\forall R.C \mid$	(value restriction)
		$\exists R.\top \mid$	(limited existential quantification)
		$C \sqcup D \mid$	(union)
		$\exists R.C$	(full existential quantification).

Role syntax:

S, T	$\longrightarrow$	$R \mid$	(atomic role)
		$R \sqcup S \mid$	(union)

Having this syntax, one now still needs the definition of the semantics of the language, in order to be able to use the language for knowledge representation. In DLs there is an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ that consists of a non-empty domain $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$. This interpretation function assigns to every atomic concept A a set $A^I \subseteq \Delta^I$ and also to every atomic role R a binary relation $R^I \subseteq \Delta^I \times \Delta^I$. By the following inductive definitions the interpretation function is extended to concept descriptions:

Concept semantics:

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg A)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \backslash A^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \left\{ a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}} \right\} \\
(\exists R.\top)^{\mathcal{I}} &= \left\{ a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \right\} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &= \left\{ a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}} \right\}.
\end{aligned}
$$

Role semantics:

$$
(S \sqcup T)^{\mathcal{I}} = S^{\mathcal{I}} \cup T^{\mathcal{I}}.
$$

$$
\begin{aligned}
\mathsf{Woman} &\equiv \mathsf{Person} \sqcap \mathsf{Female} \\
\mathsf{Man} &\equiv \mathsf{Person} \sqcap \neg\mathsf{Woman} \\
\mathsf{Mother} &\equiv \mathsf{Woman} \sqcap \exists\mathsf{hasChild}.\mathsf{Person} \\
\mathsf{Father} &\equiv \mathsf{Man} \sqcap \exists\mathsf{hasChild}.\mathsf{Person} \\
\mathsf{Parent} &\equiv \mathsf{Father} \sqcup \mathsf{Mother} \\
\mathsf{Grandmother} &\equiv \mathsf{Mother} \sqcap \exists\mathsf{hasChild}.\mathsf{Parent} \\
\mathsf{Wife} &\equiv \mathsf{Woman} \sqcap \exists\mathsf{hasHusband}.\mathsf{Man}
\end{aligned}
$$

Figure 2.1: Example of a terminology (t-box) with concepts about family relationships. (Excerpt from [2])

2.2.1 Terminologies

The first step when it comes to applying DLs is to create the t-box, consisting of a set of general concept inclusion axioms (GCIs). In the most general form GCIs look like this: $C \sqsubseteq D$ ($R \sqsubseteq S$) or $C \equiv D$ ($R \equiv S$). C and D are concepts, while R and S are roles. The axioms containing this relational symbol $\sqsubseteq$ are called *inclusions*, while the others are called *equalities*. In the following only axioms including concepts are dealt with, for the sake of simplicity.
The semantics of the axioms is defined as follows: An inclusion $C \sqsubseteq D$ is *satisfied* by an interpretation $\mathcal{I}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Also an equality $C \equiv D$ is satisfied by $\mathcal{I}$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$. Given $\mathcal{T}$ as a set of axioms, then $\mathcal{T}$ is satisfied by $\mathcal{I}$ iff each element of $\mathcal{T}$ is satisfied by $\mathcal{I}$. If an axiom (respectively a set of axioms) is satisfied by $\mathcal{I}$, then we say that it is a *model* of this axiom (respectively a set of axioms). If two axioms (respectively two sets of axioms) have the same models, they are *equivalent*. [2]
We call an equality that has an atomic concept on its left-hand side a *definition*. In order to introduce *symbolic names* for complex descriptions, definitions are used. As an example there is this axiom from figure 2.1: $\mathsf{Father} \equiv \mathsf{Man} \sqcap \exists\mathsf{hasChild}.\mathsf{Person}$. Here we associate the description on the right-hand side with the name Father. The symbolic names stated in such a way can then be used as abbreviations in other axioms. An example of this can also be found in figure 2.1: $\mathsf{Parent} \equiv \mathsf{Father} \sqcup \mathsf{Mother}$. If a finite set of definitions $\mathcal{T}$ is distinct respective the definitions of symbolic names, we call it a *terminology* or *t-box*.

2.2.2 Reasoning

We now introduce formal notions for the aforementioned inference or reasoning procedures. The problem of *concept satisfiability* is to check whether a model for a concept description (axiom) exists. Further the *t-box satisfiability* problem is to check whether a model for this t-box exists. We already mentioned the *concept subsumption* problem, which is to check whether $C \sqsubseteq D$ holds in all models of the t-box.
To test whether an individual i is an instance of a concept description C s.t. an a-box and a t-box, is another problem called *instance test* or *instance problem*. To determine whether there exists a model of an a-box $\mathcal{A}$, that is also model of the t-box, is called *a-box consistency problem*.
The *instance retrieval* problem is the problem to find all the individuals i among the assertions in an a-box that are an instance of a concept description C [6].

$$\mathsf{Woman}(\mathsf{Alice})$$
$$\mathsf{Man}(\mathsf{Bob})$$
$$\mathsf{Woman}(\mathsf{Carol})$$
$$\mathsf{hasChild}(\mathsf{Alice}, \mathsf{Bob})$$
$$\mathsf{hasChild}(\mathsf{Bob}, \mathsf{Carol})$$

Figure 2.2: Example a-box using the axioms from the t-box in figure 2.1.

It is the task of the so called *reasoner* to solve those problems and therefore offer inference services to do so. The reasoner we used in this work is RacerPro from Racer Systems.

Open World Assumption

There often is an analogy established between databases and description logic knowledge bases. Here the t-box is compared to the schema of a database, while the a-box is compared to the instances that are stored in the database. Anyhow this comparison is misleading, because in databases only explicitly stored data is considered as "true" data, while in an a-box also implicit knowledge is considered as valid knowledge. This notion is referred to as the *open world assumption* [2]. Let us shortly illustrate this using the example a-box depicted in figure 2.2.
This a-box only contains few concept and role assertions, but with the notion of open world assumption and reasoning, one can find more implicit knowledge. We only want to give one example. When we send this request to the reasoner: (?x $\mathsf{Grandmother}$), as a result the reasoner will present us $\mathsf{Grandmother}(\mathsf{Alice})$ w.r.t a-box and t-box. This is possible, because description logic knowledge bases also allow implicit knowledge and hence allow reasoning for it.

Chapter 3

Machine Learning

In order to find out what machine learning (ML) is, we will first try to find out what learning is. By textbook definition learning is: "modification of a behavioral tendency by experience (as exposure to conditioning)" [11]. ML research as well as biological learning research which deals with the learning behavior of humans or other animals, benefit from each other [13].
As stated in the citation from the dictionary, learning leads to a modification in the future behavior. In ML various kinds of *learning algorithms* are used, so that a computer can base its future behavior on the knowledge it acquired by applying those algorithms to a finite set of *data* [19]. Typically this set of data is very big, so that ML techniques are used to examine the dataset. During the examination of the dataset the learning algorithms are used to find a structure in the data. The finding of this structure by the use of algorithms is the learning process. There are three different types of algorithms: *supervised learning*, *unsupervised learning* and *reinforcement learning* [17].

3.1 Types of Machine Learning

3.1.1 Supervised Learning

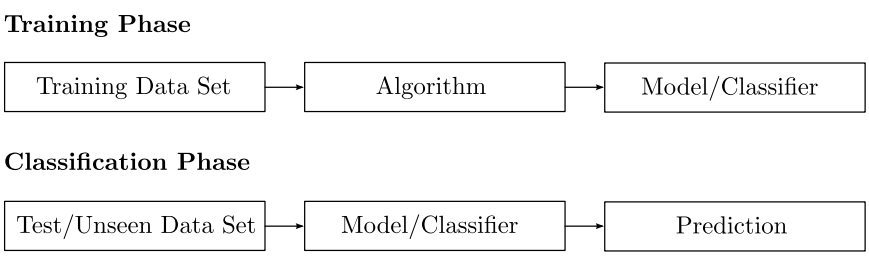

Figure 3.1: The supervised learning scheme. [10]

Supervised learning, is basically learning a function from examples of its inputs and respective outputs. This can for example be a classification problem where a vector of input values is mapped to an output class. One famous example is that one has a vector of weather input data like outlook, temperature, humidity,

windiness and a class of feature that determines, whether to play football or not (like the example in figure 3.2). Here the weather data is recorded over a certain time and then through supervised learning techniques a *classifier* is learned that can afterwards be used to classify new input vectors of weather data and therefore predict whether the football game should/will take place or not.
The general supervised learning scheme is depicted in figure 3.1, where it is parted in the two phases of training and classification. The algorithms that are used in the training phase can vary. In this work we used two different algorithms that are introduced later in this chapter in subsection 3.2.4. We used supervised learning algorithms, because the test data that we used enabled us to do supervised learning.

3.1.2 Unsupervised Learning

Unsupervised learning does not have the output it could learn from, like it is in supervised learning, but it tries to find a way to organize the data [7]. It is then the task of a system engineer to find respective names for the classes of data that the unsupervised learning algorithm found. One famous example of unsupervised learning is known as *clustering*. Given a set of input data, clustering algorithms find *clusters* in the data that in some way might correlate.

3.1.3 Reinforcement Learning

The last type of ML algorithms, reinforcement learning, is classically used in *agents* that have to perform tasks. An agent could be a robot for example. The agent is not presented any direction from a "teacher" but it rather has to perform a task. During the performance of that task the agent observes its *environment* and also the *feedback* it gets. This feedback is the basis for the reinforcement learning process. If the feedback is good, the agent will "remember" that its behavior in beforehand was "good" and will most likely try to behave in such a way in the future. [18, 17]

3.2 Tools

3.2.1 Weka

There are various machine learning tools on the market. One among those is developed at the The University of Waikato in New Zeeland and called *WEKA* (Waikato Environment for Knowledge Analysis). This tool was developed in order to make ML techniques generally available [22]. WEKA is an open source software and gives the opportunity to get hands-on experience in ML, with a vast amount of available algorithms coming along with it.

3.2.2 Input

All the algorithms implemented in WEKA use a relational table in the *ARFF* (Attribute-Relation File Format) format as input. This format can either be read from a (in advanced generated) file, or be generated by a database query. In figure 3.2 one can see an example of such an ARFF file. Lines that begin with

```
% ARFF file for the weather data with numeric values
%
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
@attribute humidity real
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
%
% 14 instances
%
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
rainy,79,96,FALSE,yes
rainy,68,80,FALSE,yes
rainy,65,70,TRUE,no
overcast,64,65,TRUE,yes
sunny,72,95,FALSE,no
sunny,69,70,FALSE,yes
rainy,75,80,FALSE,yes
sunny,75,70,TRUE,yes
overcast,72,90,TRUE,yes
overcast,81,75,FALSE,yes
rainy,71,91,TRUE,no
```

Figure 3.2: ARFF file for weather data, that can be used to train a classifier, that then predicts whether to play football or not. ([22], figure 2.2)

a `%` sign are comments. Right after the first two lines of comments the relation name is defined (`weather`). The next block is the definition of the attributes (`outlook, temperature, humidtity, windy, play?`). Attributes can have four different kinds of types: nominal, numeric, string and date. In the case of a nominal attribute, like `outlook`, the possible values (here: `sunny, overcast, rainy`) of this attribute are defined in curly brackets after the attribute name. Many of the offered algorithms are not capable of working with string attributes. In this work only nominal attributes are used, so we will not further describe the details of the other types of attributes. If there is an attribute value unknown, it is represented with a "`?`".

3.2.3 Interfaces

When it comes to using WEKA, it offers a couple of interfaces to the user. There is a graphical user interface (gui), that allows the user to interactively select the dataset to be handled, the algorithms and their parameters. After a classifier has been built the user is able to evaluate the outcome and apply the newly created classification model[1] to datasets that have to be classified. The gui also offers some visualization methods.
As an alternative to the gui, there is the command line interface (cli) , that allows WEKA to be used without having to use the gui. Logically there is no visualization available in the cli. Apart from that, the cli offers all the

[1]The models that we are talking about in the WEKA context, are other models than the DL models, we introduced in chapter 2. In the remainder of this work, when we talk of models, we are talking of classifier models in the WEKA context.

functionality that the gui offers. So classification models can be built with all the algorithms available. Those models can then also be used to classify datasets, that the user provides.

The third way to use WEKA is to directly embed it into your own java code. WEKA is completely implemented in java and offers the possibility to use all the functionality directly from your java code. This also gives the opportunity to program your own classifier.

In this work we decided to use the first two interfaces. As the test data that was used during the evaluation of the method lead to a huge memory requirement, the tests could not be performed on a single workstation, so the direct embedding of WEKA into the code was not an option.

3.2.4 Algorithms

In this work only supervised learning algorithms were used. We chose the two algorithms k-Nearest Neighbor (kNN) and Support Vector Machines (SVM) and applied them to the test data. In the following we describe each one of them. The algorithms were chosen because they use vectors of values (feature vectors) as input which is the format in which we represent the data in this work. Furthermore in the BOEMIE paper those algorithms were used to handle data, that has similar properties in a different setting. Further details about how the data is represented can be found in section 4.2.

k-Nearest-Neighbor

The k-nearest neighbor algorithm (kNN) is a simple machine learning algorithm. Another name for it is instance-based learning algorithm [1]. In this work anyhow, we will use the term k-nearest neighbor algorithm or kNN. It is used to classify objects based on a training set. The objects that have to be classified are in a way compared to the objects in the so called feature space. The feature space is an n-dimensional space that contains each training example as an n-dimensional vector representing a point in the n-dimensional space. n is the number of features that describe a training example. Classification is achieved by comparing the object to be classified to the feature space and then finding the k closest neighbors. The class of objects that is most common amongst the k nearest neighbors is chosen as the class for the object to be classified. k is always a positive integer. In the case that $k = 1$ the object to be classified is just classified as the class of its nearest neighbor. The use of a k value greater than one, can be motivated easily. When one is in bad luck, and the one nearest neighbor is just of the wrong class, while the two next closest neighbors would be of the correct class. In this case a value of $k = 3$ would have lead to the desired result. In order to be able to identify the nearest neighbors, a distance metric is needed. As we deal with a binary set of possible values for all the n features, that describe their location in the feature space, we can use an euclidean distance metric [17]. The euclidean distance is defined as following: $d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$ with $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ being two points in the n-dimensional feature space.

Sequential Minimal Optimization

Sequential Minimal Optimization (SMO) [15] is a training algorithm for Support Vector Machines (SVM) [20] that is performing well on large datasets. SVM or kernel machines [17] are a machine learning methodology that performs well on a broad selection of applications [4]. They are able to deal with high-dimensional input data [9], as we have it in our case. SVM is based on the theoretical foundations of statistical learning theory [9] and has well founded mathematical base, that is also geometrically intuitive [4]. The basic concept of SVM is to find a so called *maximum margin hyperplane* h that separates the training space into two parts (in the case of binary classification, like we have it in this work). Figure 3.3 depicts such a case, with two classes of examples (+,−).

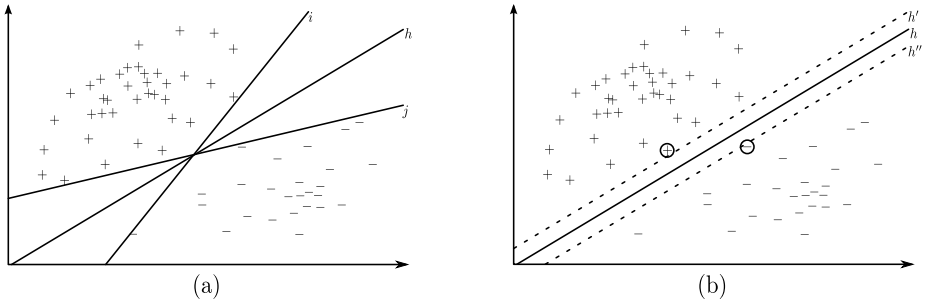

Figure 3.3: SVM examples for hyperplanes that separate the training set, with the optimal hyperplane h.

In figure 3.3 (a) one can see three different hyperplanes h, i and j that all separate the training feature space into two parts. But as can be seen in figure 3.3 (b), only the hyperplane h provides maximum margin to the two closest examples of the training set that are marked with circles (Which are the support vectors that give this method its name). The dashed hyperplanes h' and h'' are only drawn to show that the margin to each of the closest examples in the training set is maximal. The training process of finding h can be done using different approaches. The one that we use in this work is SMO.
The following description of SVMs and SMO is based on Platt's report[15].
In the simplest (linear) case a SVM is a hyperplane h separating the positive from the negative examples with a maximum margin. The margin is defined as the distance of h to the closest positive and negative examples. The output of a linear SVM is defined as:

$$u = \mathbf{w} \cdot \mathbf{x} - b \tag{3.1}$$

Here $\mathbf{w}$ is the normal vector to the hyperplane h and $\mathbf{x}$ is the input vector. The plane $u = 0$ is the separating hyperplane. The planes that lie on the closest positive examples are located at $u = \pm 1$. Therefore the margin m is defined as:

$$m = \frac{1}{\|\mathbf{w}\|_2} \tag{3.2}$$

Maximization of the margin is expressed as the following optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2 \text{ s.t. } y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1, \forall i \tag{3.3}$$

Here $\mathbf{x_i}$ is the ith training example and y_i is the class of the training example, that is also the output of the SVM. For positive examples y_i has the value $+1$ and for negative example -1.
Using the Lagrange function for the principle of duality [5], this optimization problem is converted into a dual form, that is a quadratic programming (QP), where the objective function F is only depending on a set of Lagrange multipliers α_i:

$$\min_{\alpha} F(\alpha) = \min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \alpha_i \alpha_j - \sum_{i=1}^{N} \alpha_i, \tag{3.4}$$

subject to the inequality constraints,

$$\alpha_i \geq 0, \forall i, \tag{3.5}$$

and the linear equality constraint,

$$\sum_{i=1}^{N} y_i \alpha_i = 0, \tag{3.6}$$

where N is the number of training examples.
For each training example there is a corresponding Lagrange multiplier. By using the previously determined Lagrange multipliers, the normal vector $\mathbf{w}$ and the threshold b can be computed using:

$$\mathbf{w} = \sum_{i=1}^{N} y_i \alpha_i \mathbf{x}_i, b = \mathbf{w} \cdot \mathbf{x}_k - y_k \text{ for some } \alpha_k > 0. \tag{3.7}$$

Because $\mathbf{w}$ can be computed by using equation 3.7 from the training dataset in advance, the computational power needed to later classify new objects using the linear SVM is constant subject to the number of non-zero support vectors.
For some datasets no hyperplane h can be found that splits the training instances into two parts of positive and negative examples, thus it is not linearly separable. For this non-linear cases a modified version of equation 3.3 that allows margin failures is defined as:

$$\min_{\mathbf{w},b,\mathbf{z}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{N} z_i \text{ s.t. } y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 - z_i, \forall i. \tag{3.8}$$

Here z_i are slack variables that permit margin failure and C is used to trade off a wide margin with a small number of margin failures. By again applying the principle of duality, the constraints 3.5 are transformed to boxed constraints of the form:

$$0 \leq \alpha_i \leq C, \forall i. \tag{3.9}$$

By further generalizing SVMs to non-linear classifiers, the output of a non-linear SVM, explicitly computed from the Lagrange multipliers looks like this:

$$u = \sum_{j=1}^{N} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}) - b, \tag{3.10}$$

where K is a kernel function, that measures the distance between the stored training vector $\mathbf{x}_j$ and the input vector $\mathbf{x}$. The SMO implementation in WEKA uses polynomial and Gaussian kernel functions to train the SVM classifier [22]. The Lagrange multipliers are still computed by solving the quadratic programming problem. As the objective function F is still quadratic in α:

$$\begin{aligned} \min_{\alpha} F(\alpha) = \min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \alpha_i \alpha_j - \sum_{i=1}^{N} \alpha_i, \\ 0 \leq \alpha_i \leq C, \forall i, \\ \sum_{i=1}^{N} y_i \alpha_i = 0. \end{aligned} \tag{3.11}$$

This QP problem (3.11) is the one that the SMO will solve. SMO decomposes the QP problem into QP sub-problems. On every step of decomposition SMO chooses to smallest possible optimization problem to be solved. Each time two Lagrange multipliers are chosen and then jointly optimized by finding their optimal values. These values are then updated in the SVM, so that they are taken into account for further computations.
After all the SMO algorithm is very fast, also for large and high-dimensional datasets. This is the reason why the algorithm was used in this work.

Chapter 4

Fusion

Having introduced the theoretical basics in chapter 2 on which logical fusion through reasoning is founded, we will now present the state of the art approach that was developed by Kaya in his doctoral thesis [8]. Afterwards we present the new approach for fusion that facilitates the machine learning techniques introduced in chapter 3. But first of all we have to define what fusion means in general.

4.1 Fusion through reasoning

In his doctoral thesis [8] Kaya developed a method to fuse semantic annotations from different modalities in a reasoning process. This method was developed in the context of a project called BOEMIE which is a short term for *Bootstrapping Ontology Evolution with Multimedia Information Extraction*. According to the web page [16] of the project, BOEMIE is best described as following:

"*BOEMIE was an ambitious large-scale research effort that has advanced considerably the state of the art in multimedia content analysis. In order to make multimedia content like videos or images searchable the data must be meaningfully annotated. This is commonly done by humans, but it is a hard and expensive task. Using sophisticated algorithms to extract semantics from multimedia content, BOEMIE annotates content with semantics automatically and provides valuable knowledge for both, content providers and content consumers.*

BOEMIE technology fuses knowledge that is automatically extracted by most of the popular media types (audio, video, images and text). In addition to making the content richer, the extracted knowledge is used to expand our understanding of the domain (for example athletics) and extract even more useful knowledge. This knowledge takes the form of ontologies and is represented in a machine-readable format. The synergetic process of extracting semantics from content and enriching the domain knowledge is the fundamental idea of bootstrapping that BOEMIE has pioneered."

The example domain in the BOEMIE project is that of sports news webpages.

We will stick to this domain in our examples.

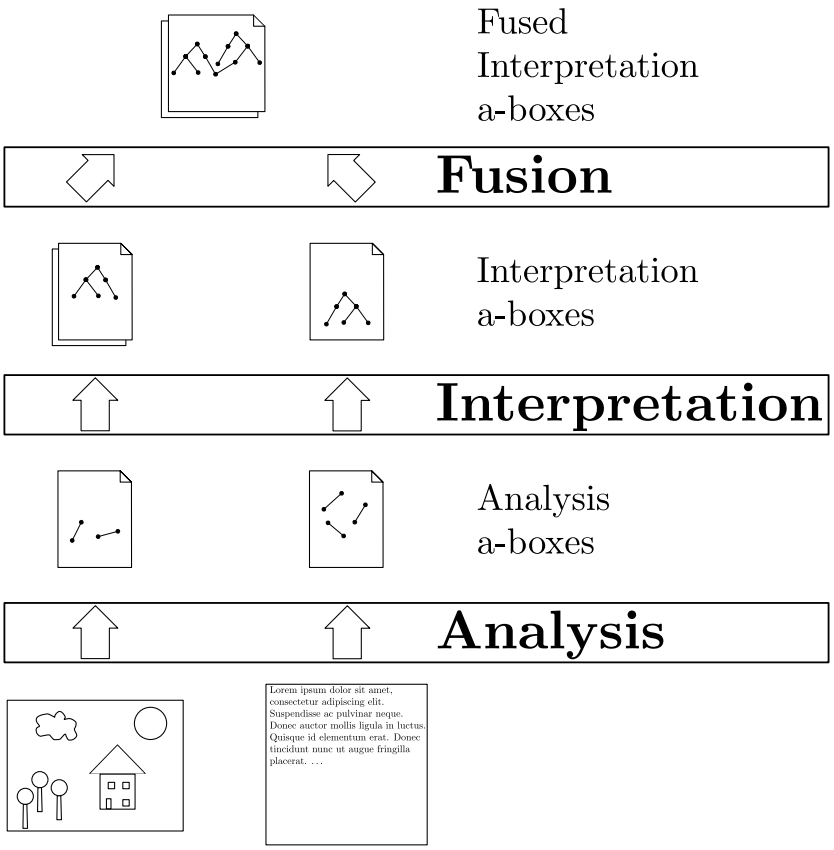

Figure 4.1: Workflow in the BOEMIE project, including analysis, interpretation and fusion steps [8].

The workflow that was used in the BOEMIE project is depicted in figure 4.1. The three steps *analysis*, *interpretation* and *fusion* are intended to enrich the knowledge about the content, that is handled at that time. In the whole workflow all the elements that are handled do get unique identifiers, so that they can be tracked at all time. If we think of the example, that can be seen in figure 4.1, there are two parts of one multimedia document, an image and a text. Those parts are individually and modality specifically *analyzed*. The knowledge about the individual parts of the multimedia documents are stored in *analysis a-boxes*. Here only basic knowledge is represented in the form of so called *mid-level concept (MLC)* assertions.

In the next step the modality specific analysis a-boxes are *interpreted* which has *interpretation a-boxes* as outcome. The interpretation is, roughly put, achieved through a reasoning step called abduction [8], which we will not further describe in this work, as it is not important for the outcome of this work. The only thing that we should mention here, is that there are domain specific rules used in the interpretation process. In the interpretation step there are *high-level*

concepts (HLC)[1] assertions added to the kb which are basically new relations between the already existing MLC assertions. For one analysis a-box there can be multiple interpretation a-boxes, as the interpretations do not have to be distinct. Assuming that the logic interpretation rules are formulated in a way that make it impossible to decide whether there is a high jump or pole vault in one image, there would be for example two interpretation aboxes for one image.[2] The fusion step now tries to combine the interpretation a-boxes from the different modalities, so that, with the help of some background knowledge about the domain, individuals from the different a-boxes are identified as the same individuals. The individuals in the different interpretation a-boxes always do have unique names. An example of a case where such a *same-as* assertion can be added to the *fused interpretation a-box* is $abox_{fused1}$, when there is one person depicted in an image, that has an assertion $person(ind_1)$ in the interpretation a-box $abox_{img1}$ and there is also a person in the text that has an assertion $person(ind_2)$ in the interpretation a-box $abox_{txt1}$. Following some rules that are not important for this work and therefore will not be further described, in the fusion step an assertion of the form $same-as(ind_1, ind_2)$ is added to the knowledge base.

4.2 Fusion through machine learning

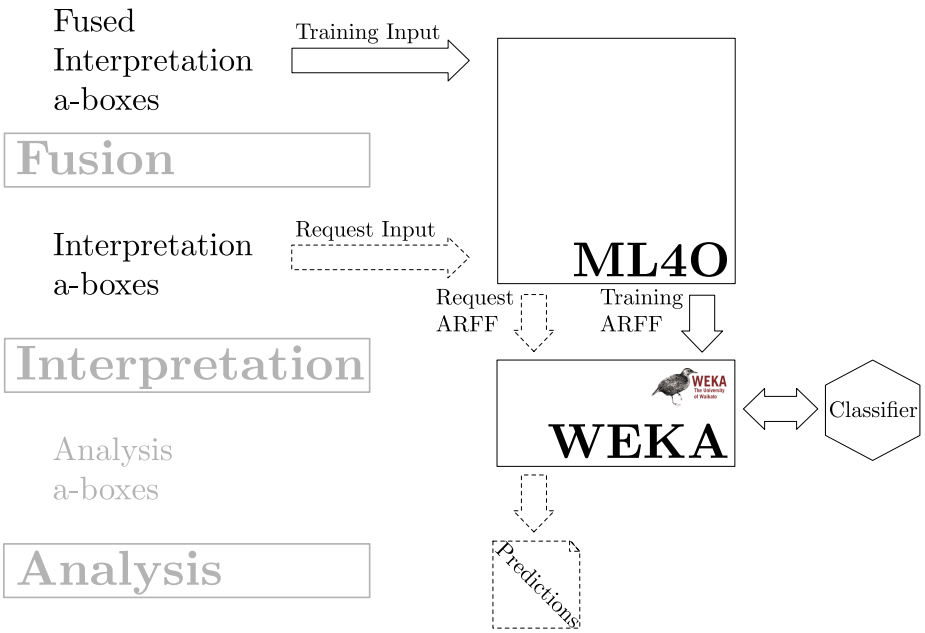

Figure 4.2: Workflow of Fusion through machine learning using the input from the BOEMIE project

[1]The terms mid-level and high-level concept stem from the system architecture and do not have any semantical meaning.

[2]For more information about multiple interpretation results please consult Kaya's doctoral thesis [8].

The fusion approach that we developed in this work, was inspired by a BOEMIE internal deliverable [14]. Here machine learning approaches to replace the reasoning processes of interpretation and fusion have been presented. Details about any implementational details were spared, so that in this work we developed an architecture from scratch.
In figure 4.2 a first overview of the workflow in this architecture is sketched. One can see that the fused interpretation a-boxes are used as training input for the Machine Learning for Ontologies (ML4O) module. This then creates ARFF files that are used to train one or several classifiers, also called models, with the machine learning framework WEKA. The interpretation a-boxes are used as input for request generation by the ML4O module, that puts out request ARFF files. Those request ARFF files can then be classified using WEKA in combination with the prior generated models. The output of this classification process is called prediction.

4.2.1 Generating the Input

After having given a small overview of the workflow in the architecture, we now present the approach for creating a classifier that can predict $same-as$ assertions. In subsection 3.2.2 on page 10 we saw that WEKA uses relational tables as input. Those tables can also be thought of as matrices. Such a matrix has n lines, one for each example (feature vector) that is considered, and $m+1$ columns, m for the features, that describe the example and one for the class of the example. Such a matrix for training the classifier is called T and is depicted in equation 4.1.

$$T = \begin{array}{c} \\ example_1 \\ example_2 \\ \vdots \\ example_n \end{array} \begin{array}{c} \begin{array}{ccccc} feature_1 & feature_2 & \dots & feature_m & same-as \end{array} \\ \begin{pmatrix} 1 & 0 & \dots & 0 & 1 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix} \end{array} \tag{4.1}$$

As one can see, the possible values for the features are 1 and 0 meaning that if a feature vector contains a 1 entry for a feature in some way is described by this feature. Knowing this we now have to find a way to represent the a-boxes in such a way. We choose to view a-boxes as undirected graphs, as the test data set that was used in this work only uses undirected roles. Generally a-boxes can also contain directed roles that would result in the use of directed graphs as an alternative representation for a-boxes. In the BOEMIE deliverable [14] a similar approach was used in another setting (interpretation through machine learning). Representing a-boxes as graphs can be done by mapping each individual in an a-box to one node in a graph and also mapping each role assertion between two individuals in an a-box to an edge between the respective nodes in that graph. One such example can be found in figure 4.3
Now for creating the training data for the machine learning algorithms, those graphs have to be represented as feature vectors. We use the neighborhoods of two individuals to represent the individuals as a feature vector. This is achieved by representing the neighborhood of an individual as a list of paths of the form

$$concept_1 \overset{role}{\text{——}} concept_2. \tag{4.2}$$

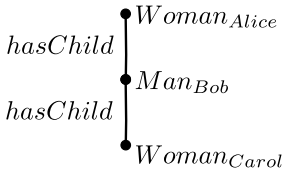

Figure 4.3: Graph representation of the a-box in figure 2.2 on page 8.

In this work we used a path length of one step, meaning that the neighborhood is only examined in depth "1" to build the features. A higher depth for the feature building has to be investigated in future research. In order to build the features, the names of the individuals are substituted by their direct types. This means for one edge in the graph like this one $individual_1 \xrightarrow{role_1} individual_2$ with these properties $directTypes(individual_1) = A, B$ and $directTypes(individual_2) = C$ there are two paths, looking like these $A \xrightarrow{role_1} C$ and $B \xrightarrow{role_1} C$. These paths are the describing features of the training matrix T, as well as the request matrix R, that is is used in the classification phase,

$$R = \begin{array}{c} \\ example_1 \\ example_2 \\ \vdots \\ example_{n'} \end{array} \begin{array}{c} \begin{array}{cccccc} feature_1 & feature_2 & \dots & feature_m & same-as \end{array} \\ \begin{pmatrix} 1 & 0 & \dots & 0 & ? \\ 0 & 1 & \dots & 0 & ? \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & ? \end{pmatrix} \end{array} \tag{4.3}$$

that looks very similar to T. There are two differences anyhow, the last column has unknown values and the number of lines is n'. The size of n' depends on the total number of individuals that are present in the interpretation a-boxes that have to be classified and can be calculated as

$$n' = \sum_{i=1}^{N} i - U. \tag{4.4}$$

Here N is the number of individuals in all interpretation a-boxes for one document. The constant U is the number of examples that could not be represented by the features that are available in this model/classifier, while R is generated. Here one has to remember that all the examples that could not be represented by the features are potential $same-as$ candidates. So if one is in bad luck, the model just cannot handle those examples, that would have to be classified as $same-as$.
Another way to build the features is to include the individuals' modalities. Then the features look like this

$$_{modality_1}concept_1 \xrightarrow{role} {}_{modality_2}concept_2. \tag{4.5}$$

On the one hand this enriches the features in a way, that also the modalities of the individuals that are currently handled is integrated into the learning process, without the learning algorithm having to be aware of it. On the other hand it also leads to a higher dimensionality in the feature vector space thus leading to higher computational complexity. Another thing one has to remember is, that

including the modalities can lead to more examples that cannot be represented by the features, thus leading to a bigger constant U. One example can be constructed without a problem. Given four individuals with their direct types and modalities:

$$\begin{aligned} i_1, directTypes(i_1) = A, modality(i_1) = m_1 \\ i_2, directTypes(i_2) = B, modality(i_2) = m_2 \\ i_3, directTypes(i_3) = A, modality(i_3) = m_1 \\ i_4, directTypes(i_4) = B, modality(i_4) = m_3. \end{aligned} \tag{4.6}$$

We assume that i_1 and i_2 were used during training the classifier and no individual like i_4 concerning the modality concept combination was used during the training. We also assume that there was a role assertion with $role_1$ present during training that eventually lead to the building of this feature using modalities

$$_{m_1}A\overline{\quad role_1 \quad}_{m_2}B. \tag{4.7}$$

In the building phase of request matrix R now individual i_3 is handled and its neighborhood is examined. There is one role assertion that looks like this

$$_{m_1}i_3\overline{\quad role_1 \quad}_{m_3}i_4, \tag{4.8}$$

which by substitution is transformed into the potential feature

$$_{m_1}A\overline{\quad role_1 \quad}_{m_3}B. \tag{4.9}$$

This potential feature cannot be found in the set of features, as during the learning phase there was no such example containing this particular kind of neighborhood. If now two individuals that cover this case are considered that only have role assertions that lead to unknown potential features, the example cannot be classified. In the case where the modalities are not used during training and classification, this example would not be unclassifiable. As during the training this feature would have been built

$$A\overline{\quad role_1 \quad}B, \tag{4.10}$$

and during the generation of R the potential feature of equation 4.9 would also look like this

$$A\overline{\quad role_1 \quad}B, \tag{4.11}$$

and thereby match the feature from equation 4.10.

Chapter 5

The Prototype

Having developed an approach for fusion facilitating the machine learning framework WEKA, we now have to develop a system architecture that generates the training and request input for WEKA described in subsubsection 4.2.1 on page 20 that creates the classifier models and uses those in combination with WEKA to create predictions for the request input.

5.1 Architecture

In figure 5.1 the architecture we developed is depicted (Note the key on the lower right). The center of this architecture is the ML4O tool that has three different modes of operation. The first one is the the "training" mode. It takes a training sub set of the fused interpretation a-boxes as input and generates a training ARFF file with the help of Racer which is the DL reasoner we used (It is described in more detail in subsection 5.1.1). This training ARFF file is then passed to the machine learning component WEKA that is used to create two classifier models (SMO & kNN).

Later these models are used to classify new instances, but first a request file has to be generated. This is done in the second mode of operation of the ML4O tool which takes all the interpretation a-boxes of one multimodal document as input and generates a request ARFF file as output. Using the request ARFF combined with the previously created models, the machine learning component is used to classify the new examples in the request ARFF. The output of these classification processes is then put into prediction files.

The third step is only used to test the performance of the classifier model, before it is actually used. In this evaluation step, the beforehand created predictions are evaluated using one of the fused interpretation a-boxes that is analog to the interpretation a-boxes (ground truth abox) that were used to generate the request ARFF. This can be done, because for all the interpretation a-boxes that were used in this work, a fused interpretation a-box already existed. After the evaluation is done, the result is written into a result file.

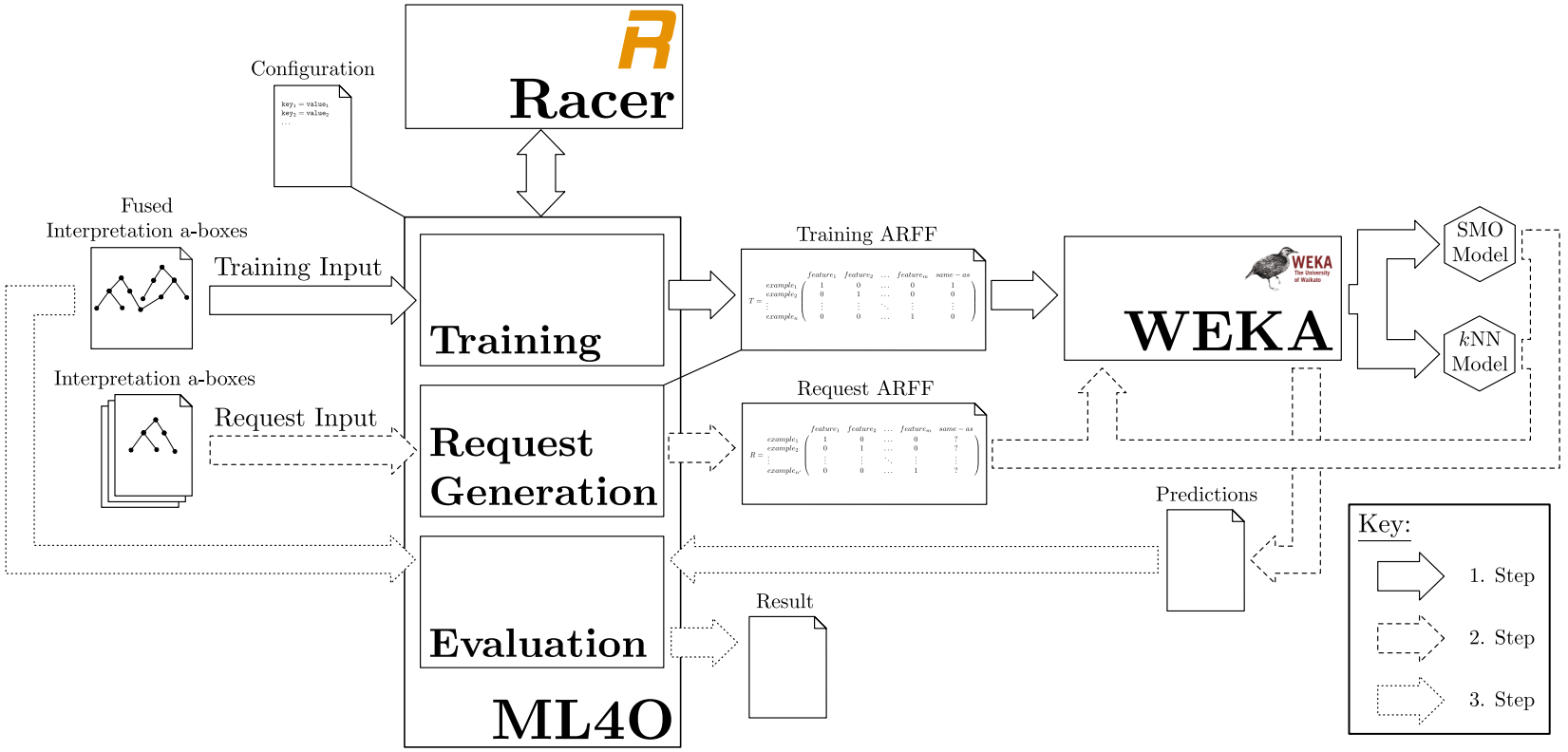

Figure 5.1: Architecture of the system developed in this work

5.1.1 3rd Party Components

As already mentioned in chapter 3, the machine learning framework WEKA, developed at the the University of Waikato in New Zeeland, was chosen as the machine learning component in the architecture. We used the most recent stable version (3.6.1) available at October 12th, 2009. As the description logic reasoner we chose to use RacerPro. RacerPro is developed by Racer Systems in Hamburg, Germany, and can be used freely for research purposes, after applying for a license. Due to the fact that we wanted to use the recently released java interface to RacerPro, called JRacer 2.0, we also had to use the RacerPro 2.0 preview, more precisely RacerPro Version 2.0 2009-09-18, that is freely available to everyone.

5.2 Implementation

As a language for implementing the prototype we chose Java. It has the advantage that it can be used on nearly any imaginable platform. Also the fact that RacerPro offers an Java API was one reason to choose Java as programming language.

5.2.1 ML4O

The ML4O tool was the main part that had to be implemented in this work. It is in a prototypical state and does not claim to be implemented using software engineering facilities. The tool is for the biggest part contained in one executable class `de.thorbenheins.mt.MachineLearningForOntologies`. This class is structured into five parts, being training part, testing part, evaluation part, shared part and utility part. The three parts of training, testing and evaluation contain the functions that were implemented for the respective mode of operation of the ML4O tool. The shared part contains functions that are used by more then one mode of operation of the ML4O tool.
In the following we describe the general implementational decisions, without going into technical details. We chose to not include any source code in this work, but of course one can find the code of the tool on the cd that comes with this work here: `/ml4o-src.tgz`. The java documentation to the tool which describes in more detail, which functions were created and what those functions do, as well as where the potential for future improvements can be found, are also included in this file.

Training

The training mode basically consists of two steps which each iterate over all the a-boxes in the training set:

1. Determine how the features for the training matrix T look like regarding the current training set

2. Fill the training matrix T with feature vectors derived from the training set

For both steps we describe what is done with each a-box of the training set. In the first step the individuals of the a-box are sorted into modality specific lists. This can be done, because the test data from the BOEMIE project that we used contains fused interpretation a-boxes that are structured as depicted in figure 5.2 (b). In order to then find out, which individuals belong to which modality originally, we introduced a transitive role that is parent of all other roles. The code segments that are presented in the following, are written in the Lisp, which is used by RacerPro:

```
1  (define-primitive-role super-role :transitive t)
2
3  (evaluate
4   (let*
5    (
6     (roles (all-roles))
7     (roles (remove-if
8      (lambda (x)
9       (or (consp x)
10       (role-used-as-datatype-property-p x (current-tbox))))
11     roles))
12   )
13   (dolist (role roles)
14    (role-has-parent role 'super-role (current-tbox)))
15  ) ;end of let*
16 )
```

Here in line 1, the transitive role `super-role` is introduced, using the axiom `define-primitive-role` that is a built in feature of RacerPro, which defines a primitive role. Then in the lines 3 to 16 the role is integrated into the existing t-box. In line 6 the set of `roles` is defined by getting all the current roles from the t-box, by executing the query `all-roles`. Then from line 8 to line 11 the size of the `roles` set is reduced by removing all roles that are no symbols (see line 9), as well as by removing all roles that are used as datatype property (see line 10). We will not go into further details regarding the choice of these factors. Then in the lines 14 and 15 all the roles that are left in the `roles` list are made children of the super-role.
Having done this we now can query for all individuals that are "children" of one of the modality nodes in the a-box. In the following we drop the t-box prefixes for the respective concept names, for the sake of brevity and readability. An example of a query that returns the the individuals of one modality looks like this:

```
(retrieve (?x ?y)
 (and (?y ?x super-role)
  (?y Text)
  (or (?x MLC)
      (?x HLC))))
```

In this example a query for all the tuples (`?x`, `?y`) that are related through the previously introduced super-role, where `?y` is of type `Text` and `?x` is of type `MLC` or `HLC`. By exchanging the value of the concept name `Text` by the concept name of the other modalities, all the lists of concepts that belong to those modalities respectively can be retrieved.
After all the lists of individuals sorted by the modalities have been created, the next step to take, in order to create the features for the training matrix T, is to explore the neighborhood of each individual in those lists. This is done by at first getting all the role assertions for the individual using the built in RacerPro capabilities:

```
(all-role-assertions-for-individual-in-domain individualName)
```

```
(all-role-assertions-for-individual-in-range individualName)
```

As already mentioned in subsection 4.2.1 on page 20, in order to create the features, we do not use the individual names, but the direct type(s) of the individuals. This is done using yet another RacerPro function:

```
(individual-direct-types individualName)
```

With the data that was acquired so far, the features now can be built. The features look like this:

```
Modality1__ConceptName1__Role__Modality2__ConceptName2
```

If the features are being built without using the modalities, the features look very similar:

```
null__ConceptName1__Role__null__ConceptName2
```

As one can see, the modality information is just omitted, before storing the feature in the list of features. The last feature is always the class of the example, stating if an example is "$same - as$" or not.
Note that for the test data we used the separator symbol (__) was fitting, while it may be not fitting for other test data sets. If necessary it can be changed centrally in the class `ML4OConstants`.
Also note that when using this approach to create the features for a training set, the dimensionality is minimal regarding the training set. One could also think of another approach to build the features. One approach might be to not only take the actual assertions into account, but *all* possible combinations of concept names and roles of the ontology. This would lead to a higher dimensional feature space, that on the one hand would have higher computational costs, but on the other hand also would cover every possible feature one could think of. This would have the benefit, that all future examples to be classified, could be represented with the given feature space. This approach was not used in this work and it is up to future research to evaluate the usability of such an approach.
Having built all the features for the current training set, in the second training step the training matrix T has to be filled with feature vectors. In this step it is again iterated over all the a-boxes in the training set. At first the $same - as$ assertions for this a-box are stored in a list using this RacerPro function:

```
all-same-as-assertions
```

And like in the feature building step the individuals of an a-box are sorted in modality specific lists. Afterwards for *all* combinations of two individuals the respective neighborhoods are examined and stored in a feature vector, by setting the value of a feature to "1" if one of the two individual's neighborhoods contains a role assertion that looks like the respective feature. The class of the current example is then set, after searching for the combination of those two individuals in the list of $same - as$ assertions.

Request Generation

For each set of interpretation a-boxes one request file has to be generated. We describe the process for one a-box. The implementation anyhow, allows to generate the request files for a bunch of interpretation a-boxes from multiple documents. This is implemented training set specific and would have to be

adapted to another training set. In the following the general idea of how to create a request file is described.
The features for the request matrix R are the same as the features of the training matrix T. So the first step is to copy the features from T. After that the feature vectors for R have to be created. This is done very similarly to the way it is done for the T, with two necessary differences: First the class of the examples is naturally unknown, as the examples have to be classified. This means that the value of the class feature will be set to "`?`" for each feature vector. Second there is no single a-box that contains all the individuals that have to be combined, but there are a couple of interpretation a-boxes for the different modalities. In order to overcome the lack of such an a-box, we decided to merge all the interpretation a-boxes into one big a-box. This can be very simply achieved using these RacerPro capabilities:

```
1  (owl-read-file "abox-file" :kb-name mergedkb)
2  (owl-read-file "abox-file" :kb-name mergedkb :init nil)
```

In line 1 the parameter `:init` is not used, as its default value is "`t`", meaning that the knowledge base with the name `mergedkb` is created empty and then filled with the contents of the `abox-file`. This command is only used for the first of the a-boxes to be merged into a big a-box. For the following a-boxes, the command from line 2 is used, so that the assertions that are currently in the knowledge base are preserved while the assertions from the new a-box "`abox-file`" are added to the knowledge base with the name `mergedkb`. In this way in the end an a-box called `mergedkb` can be used to query for neighborhood information of individuals. Anyhow, this can only be done, when the `mergedkb` a-box is consistent. So before using this a-box, this has to be checked. In this prototype, we just dropped an a-box if it was inconsistent after merging all the interpretation a-boxes. Note that there might be better approaches to generate the request files, that are subject to future research.

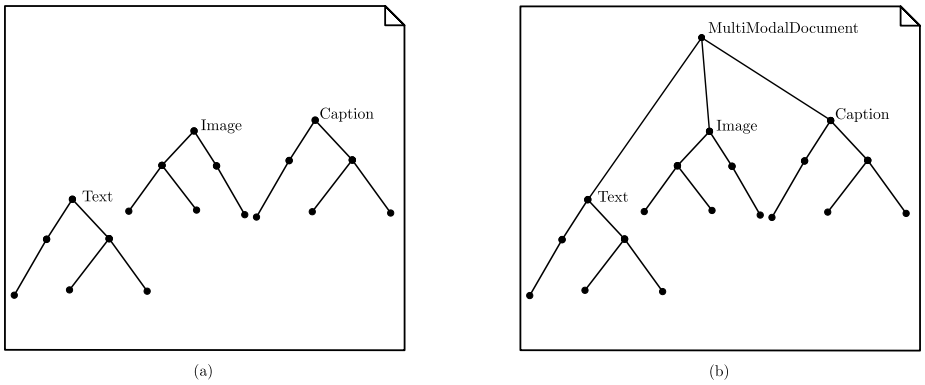

Figure 5.2: (a) Simplified structure of a merged a-box, that contains interpretation a-boxes from various modalities. (b) Sketch of a simplified structure of a fused interpretation a-box that contains a MultiModalDocument node on top of the interpretation a-boxes.

This was one important step towards generating the request file. Another important step is to put the individuals of the merged a-box into modality specific lists. As the structure of the previously created merged a-box (see figure 5.2

(a)) of interpretation a-boxes is different to the structure of a fused interpretation a-box (compare figure 5.2 (b)), we could not create the lists of individuals sorted by modality in the same way as during the training. We rather used a straightforward approach, that simply takes all the interpretation a-boxes from one modality and puts all the individuals from those a-boxes into a modality specific list. Anyhow, this approach is training set dependent and would have to be changed for any other training set.
After preparing everything, now the feature vectors for all the combinations of two individuals can be created like in the training. The only difference is, that the class of the example is unknown and therefore the value of the class feature is set to "?".

Evaluation

The evaluation mode of the ML4O tool is used to generate statistical data about the performance of the classification done by WEKA. It is in the current state dependent on the test data set and has to be changed for each other test data set. The general ideas of how to acquire the statistics about the classification step are generally applicable and can therefore be adopted in future scenarios.
The basic functionality is to create a list of values for attributes that are interesting for evaluating the performance of the classification model. Such a list is created for each request ARFF file, or to be more precise, for every prediction file that was the outcome of a classification of a request ARFF file. We chose the following attributes for the evaluation method, used in this work:

1. Name of the request ARFF file
2. Name of the ground truth a-box file that contains the $same-as$ assertions generated during the fusion through reasoning
3. Name of the machine learning method that was used for classification
4. Name of the model file that contains the classifier
5. Name of the file that contains the prediction that is currently evaluated
6. A boolean value indicating whether modalities where used for feature generation
7. Number of fused interpretation a-boxes in the training set
8. Number of features used
9. Number of $same-as$ assertions in the ground truth a-box that was generated during the fusion through reasoning
10. Number of predicted $same-as$ assertions
11. Number of correctly classified examples
12. Number of false negatives
13. Number of false negatives that were not int the request ARFF file
14. Number of false positives

15. Number of instances/examples in the request ARFF file

16. Number of seconds needed for classification

The values for the attributes 1-7 are simply acquired by parsing the name of the prediction file, as a naming convention was used (manually) during the whole classification process. The number of features (8) is fetched from the training ARFF file. The number of $same-as$ assertions (9) is gotten from the ground truth a-box using RacerPro and the `all-same-as-assertions` function. The number of predicted $same-as$ assertions (10) is got by counting the predicted $same-as$ assertions in the prediction file. For the values of the attributes 11-14 the actual $same-as$ assertions, including the individual names, are got from the ground truth a-box. As the individuals of each example are included in the request ARFF file as a comment, it is easy to cross-compare the list of ground truth $same-as$ assertions to the list of predicted $same-as$ assertions. By doing so, the values for the attributes 11, 13 and 14 can be simply counted. In order to get the number of false negatives that were not in the request ARFF file (12) one has to look for all false negatives in the request ARFF file. The number of those false negatives that could not be found in the request ARFF file makes up the value for 12. The number of instances in the request ARFF file (15) can be counted. The number of seconds that were needed to classify (16) can be calculated, because the scripts that were used to automate the classification process (see appendix A) put timestamps in the prediction files. In chapter 6 the evaluation based on those values can be found.

Configuration

The configuration of the ML4O tool is done completely, using a configuration file that contains key-value pairs. This means, that there are no command line parameters evaluated by the tool. The configuration file can be found in this place:

```
~/ml4o/conf/settings.properties
```

A complete reference of how to use this configuration file can be found in appendix B.

5.3 Testing

After having described the implementational part of the ML4O tool we now describe how the tests were run and describe the test environment in which we tested the architecture depicted in figure 5.1. At first we describe the testing environments that were used to carry out the tasks. Basically the tests were split into five parts:

1. Training data generation

2. Training the classifier

3. Request data generation

4. Classification of request data

5. Evaluation of the classification results/predictions

As one can see, the parts 1, 3 and 5 are performed by the ML4O tool, while the parts 2 and 4 are performed by WEKA. As WEKA turned out to be resource consuming (see chapter 6), operating on the data provided by the ML4O tool, we decided to use two different test platforms for the two different tools.

5.3.1 Platforms

The platforms specifications looked like this:

- CPU: Intel Core 2 Duo T9400 @ 2.53 GHz
- Main Memory: 3GB
- Operating System: Windows Vista Business 32-Bit, Service Pack 2
- Java Virtual Machine: JDK 1.6.0_14
- RacerPro Version 2.0 2009-09-18

which was used to run 1, 3 and 5.

- CPU: Intel Xeon Quadcore X3320 @ 2.5GHz
- Main Memory: 8GB
- Operating System: Ubuntu 8.04.3 LTS
- Java Virtual Machine: OpenJDK 64-Bit Server VM (build 1.6.0_0-b11, mixed mode) and Java HotSpot(TM) 64-Bit Server VM (build 14.2-b01, mixed mode)
- Weka 3.6.1 2009-06-05

which was used to run 2 and 4. The second platform has two different java virtual machines, because two computers with the same specifications, apart from the java virtual machine, were used.

5.3.2 Test Execution

For all parts of the test execution that use the ML4O tool the RacerPro has to run.

ML4O Tool

In order to run the ML4O tool, one has to unpack the tarball from the cd that comes with this work to a destination folder of choice (we assume `~/` is chosen as destination folder). The tarball can be found under this path on the cd:

```
/ml4o.tgz
```

After this has been done, the tool can be invoked, using this command:

```
java -jar ~/ml4o/ml4o.jar
```

But before this can be done the configuration file

```
~/ml4o/conf/settings.properties
```

has to be set up properly. For more information about how to set up the configuration file properly, please consult the documentation that can be found in appendix B. Most of the keys can be set once at the beginning. Only the values of the keys `modusOperandi` and `useModalities` have to be changed during the later execution of tasks. We describe one example run for one training set from beginning to end that uses modalities. So after having set up the configuration file, and especially these two values:

```
modusOperandi=fusionTraining
useModalities=yes
```

one now can start the tool and it will create the `wekaArffTrainingFile` as output. With this file, now classifier models of various sorts can be created, using the WEKA tool. Anyhow, in this work two kinds of classifier models were used, as described in subsection 3.2.4. For this example run let us create a SMO classifier using this command (Note: Take care that the classpath contains the weka.jar):

```
java -Xmx8G weka.classifiers.functions.SMO -t ~/ml4o/data/ml4o.arff -d ~/ml4o
    /models/ml4o-SMO.model
```

This creates a classifier model here:

```
~/ml4o/models/ml4o-SMO.model
```

When training a classifier, WEKA offers optional configuration of the classifier, which we did not use in this work. The different classifiers offer different parameters. A (detailed) description to all the classifiers and their respective parameters can be found in the WEKA documentation [21].
In order to now predict $same - as$ assertions for a set of interpretation a-boxes the configuration has to be changed to:

```
modusOperandi=fusionRequestGeneration
```

By invoking the ML4O tool again a request ARFF file will be created as output in the `wekaArffRequestFolder` (assuming that there was a `toBeClassified.did` file, that contains the document id that can be used to resolve a reference to a subfolder of the `aboxTestingRepo`!):

```
~/ml4o/data/toClassify/toBeClassified.arff
```

In order to now get predictions for this request ARFF file, one changes into the `wekaArffRequestFolder` and then triggers the classification process by invoking one of the scripts from appendix A (Note that the script takes *three* parameters!):

```
~/classifySMO.sh ~/ml4o/models/ ml4o-SMO.model ~/ml4o/data/ml4o.arff
```

This basically invokes WEKA in classification mode which could also manually be done by invoking WEKA manually like this:

```
java -Xmx8G weka.classifiers.functions.SMO -p 412 -l ~/ml4o/models/ml4o-SMO.
    model -T ~/ml4o/data/toClassify/toBeClassified.arff
```

Here it is assumed that the machine, the classification is taking place on, has a main memory of 8GB, thus the parameter `Xmx8G` of the java virtual machine. It is also assumed, that the feature space has the dimensionality `412`. As the highest dimension of the feature vector is always the one that has to be predicted, indicated by the usage of the parameter `p` of WEKA. The script that was used

above automatically extracts the dimensionality of the feature space from the training ARFF file and would trigger classification for all the request ARFF files that are in the current directory.
Apart from doing this, the script also creates a file that contains the prediction result as output which otherwise would only be displayed in the console. This file can be found in the `predictionFolder`:

```
~/ml4o/predictions/log-ml4o-SMO.model-toBeClassified.arff.txt
```

This is particularly important for the next step being the evaluation. In order to do this, one has to change the configuration file like this:

```
modusOperandi=fusionResultEvaluation
```

By invoking the ML4O tool the evaluation process is triggered which is described in more detail in chapter 6. The only thing we want to mention here is that after the evaluation is over, there will be an evaluation result file like this:

```
~/ml4o/evaluationResults/evaluation.result.of-log-ml4o-SMO.
    model-toBeClassified.arff.txt
```

Note that the evaluation will take place for all prediction results that are found in the `predictionFolder`.
There is a tutorial on classification using WEKA from Bamshad Mobasher that gives a step by step introduction into the classification using WEKA [12].

Chapter 6

Evaluation

In this chapter we present the results of the tests we ran. But before doing so we describe which evaluation/test data we used and how we built up the statistics that can be found in this chapter.

6.1 Evaluation Data

As the basis for our tests we used the data from the BOEMIE project. For our purposes, the domain of the test data set does not matter, so we will not describe the ontology in more detail.
All the data we used during this evaluation can be found on the cd in the file `/aboxes.tgz` that comes with this work or also on the internet. For the fused interpretation a-boxes/groundtruth a-boxes (`*.abox`) and `*.did` files one has to look here: `http://repository.boemie.org/ontology_repository_abox/deployment/abox/1/`. The interpretation a-boxes that are used to generate the request ARFF files can be found here: `http://repository.boemie.org/BoemieRepository/Analysis/`[1]. Each folder corresponds to one multimodal document, that has been analyzed and interpreted. The folder name corresponds to an id, that can be found in the `*.did` files from the first url. In the subfolder `/RMDFv1/_aux/interpretation/` the folders for the modalities can be found, that contain the interpretation a-boxes for this multimodal document.
In table 6.1 the test data is described in the matter of number of a-boxes, used for each part of the evaluation. As one can see, we used only a-boxes for training and testing that were smaller than 300kb. An a-box file from this ontology of this file size most of the time contains about 500 individuals, about 700 concept assertions and 1100 role assertions (see table 6.4). These numbers have a direct effect on the dimensionality of the feature space and the number of feature vectors that are contained in an training ARFF file. On the one hand the time to generate the ARFF files grows with those factors and, even more importantly, on the other hand the need for main memory grows for classifier training and classification with those factors. To proof our point, we ran a test in which only the biggest of the $same - as$ containing ground truth a-boxes

[1] Note that a big portion of the OWL files in this repository lack the first < character. We wrote a tool, that fixes this problem for a given tree of folders. It can be found on the cd under `/add-on/xmlFixer.tgz`.

Number of a-boxes total	616
Number of a-boxes containing $same - as$	325
Number of a-boxes containing $same - as$ and $<$ 300kb	162
Number of a-boxes used for training	40
Number of a-boxes that are testing candidates	122
Number of a-boxes that are working candidates	20
Number of a-boxes used for testing	6

Table 6.1: Quantity-wise description of the test data set.

Number of training a-boxes	Number of example feature vectors in training ARFF file	Highest measured main memory consumption during classifier training (SMO)
10	43120	1.4GB
20	107708	2.7GB
30	144380	6.9GB
40	169036	>8GB

Table 6.2: Number of example feature vectors in training ARFF file s.t. main memory consumption during classifier training and number of training a-boxes.

Number of training a-boxes	Dimensionality of feature space	
	With modalities	No modalities
10	380	335
20	589	517
30	684	590
40	802	668

Table 6.3: Dimensionality of the feature spaces s.t. number of training a-boxes, classifier type and modality usage.

(`247.abox`, 873kb, 1218 individuals) was used to create a training ARFF file. The creation of this ARFF file took about *fifty* hours. The feature space of this example had the dimensionality 227 and there were 551164 examples feature vectors in this training ARFF file. As this is a very high number of example

a-box file-name	Number of $same-as$ assertions	Number of individ-uals	Number of concept assertions	Number of role asseritons	Filesize in kb
`2.abox`	1	318	494	759	203
`61.abox`	1	237	371	556	149
`120.abox`	1	323	499	786	207
`142.abox`	1	250	384	559	151
`170.abox`	41	451	725	1097	291
`171.abox`	1	452	717	1099	291
$\sum$	46				

Table 6.4: Ground truth a-boxes used for testing the classifiers.

feature vectors compared to the data in tables 6.2 and 6.3, where a training data set of forty a-boxes produced about one third of example feature vectors and had a dimensionality of 668 which resulted in a memory consumption beyond the capacities of the test machine, we decided to limit the size of the a-boxes that were used for evaluation to 300kb.
Furthermore we decided to use the forty *smallest* a-boxes, containing $same-as$ assertions as training set. As stated in table 6.2 the memory consumption for building a classifier based on a training set of forty a-boxes exceeded the availability of main memory. This was the case for the SMO classifier not using modalities. As there could be made no comparisons, building only the kNN classifier, we were forced to use an even smaller training set. We ran tests for training set sizes of ten, twenty and thirty ground truth a-boxes. For each of the training set sizes, tests with the kNN as well as the SMO classifier were performed, both with and without using modalities to train the classifier. This makes a total of twelve different test rounds. The statistics about the results of those classifications can be found in the tables 6.5, 6.7 and 6.6. The raw classification and evaluation data can also be found on the cd in the files `/classification-results.tgz` and `/evaluation-results.tgz`.
As stated in table 6.1 we used six ground truth a-boxes to test the different classifiers in each test round. Information about the a-boxes we chose can be found in table 6.4. Note that there are forty-six $same-as$ assertions to be found. From the set of twenty working a-box candidates, we chose two small, two medium and two big a-boxes regarding their file sizes. Due to time limitations not all twenty working candidates were tested. Anyhow the results that stem from the total number of seventy-two tests show some hints on how well the approach performs, that was developed in chapter 4.

6.2 Evaluation Method

Now the question is, what do we want to evaluate? The answer is simple: The performance of the classification of the machine learning approach developed in chapter 4 and implemented in chapter 5. We chose a straightforward way how to measure the performance. As we have a set of ground truth a-boxes described in table 6.4, we can classify feature vectors for those a-boxes, that were generated

by the ML4O tool in `fusionRequestGeneration` mode. After the classification has finished we can compare the set of predicted $same-as$ assertions in the prediction file, like the one in listing 6.1, to the set of $same-as$ assertions that can be found in the respective ground truth a-box (`2.abox` in this particular case). The feature vectors that were predicted to have the class $same-as$ can be identified by looking at the column `predicted` in the prediction file. In this case in line 13 the `inst#` or feature vector 15 is classified as $same-as$.
The first character in the `predicted` column is the index of the so called label, starting with 1. After the colon the value of that label, and therefore the predicted class, can be found. In our case the label set looks like this same-as $= \{1, 0\}$. So for feature vector number 15 in line 13 of the listing the predicted column states that this feature vector is classified as $same-as$. In another case like in line 9 of the listing, the predicted column states 2:0, which means it is not $same-as$.
Now we can use the `inst#` value to find this feature vector in the request file. After each feature vector in the request file, there are the two names of the individuals that were used to generate this feature vector are stored in a comment. Knowing the names of the individuals and all the $same-as$ assertions from the ground truth a-box, it is easy to count the false positives, false negatives, correctly classified, number of predicted $same-as$ assertions and number of $same-as$ assertions in the ground truth a-box that can be found in the evaluation file in listing 6.2 in the lines 8, 9, 11, 14 and 15.

```
1  START:1255699758
2  15:29:18
3  command: java -Xmx8G weka.classifiers.functions.SMO -p 590 -l ~/models/30
       smallaboxes-no-modalities-SMO.model -T ~/ml4O/arff/30smallaboxes/no-
       modalities/toClassify/2.arff
4
5
6  === Predictions on test data ===
7
8   inst#     actual  predicted error prediction ()
9       1        1:?        2:0   +   1
10      2        1:?        2:0   +   1
11    ....       ...        ...   .   .
12    ....       ...        ...   .   .
13      15       1:?        1:1       1
14    ....       ...        ...   .   .
15    ....       ...        ...   .   .
16  28203        1:?        2:0   +   1
17
18 END:1255699800
19 15:30:00
```

Listing 6.1: Classification result in form of a prediction file, produced by one of the scripts in appendix A.

The values for the attributes in the lines 1-7 are simply parsed from the file name of the prediction file. It would also be possible to count the number of training a-boxes (line 7), in the second line of the training ARFF file. The number of false negatives that were not in the request ARFF file can also easily be counted. We just take the set of false negatives, and look for the individual combinations of these feature vectors in the request ARFF file. If a combination of individuals cannot be found in the request ARFF file, the counter is increased by one.
The number of feature vectors in the request ARFF file (line 13) is counted in the request ARFF file. The number of features (line 12) is parsed from the

training ARFF file, but could also be counted in the training or request ARFF file. As the prediction file contains the two timestamps in the lines 1 and 18, the number of seconds that were needed for classification (line 16 in the evaluation file) can be calculated by subtracting the start from the end time.

```
1  arffFileName: 2.arff
2  groundTruthAboxFilename: 2.abox
3  mlMethod: SMO
4  modelFilename: smallaboxes-no-modalities-SMO.model
5  predictionFilename: D:\MasterThesis\predictions\log-30smallaboxes-no-
       modalities-SMO.model-2.arff.txt
6  modalitiesUsed: false
7  numOfAboxesInTrainingSet: 30
8  numOfCorrectlyClassifiedInstances: 1
9  numOfFalseNegatives: 0
10 numOfFalseNegativesNotInRequestFile: 0
11 numOfFalsePositives: 164
12 numOfFeatures: 590
13 numOfInstancesInArffFile: 28203
14 numOfPredictedSameAsAssertions: 165
15 numOfSameAsAssertionsInGroundTruth: 1
16 numOfSecondsToClassify: 42
```

Listing 6.2: Evaluation result file, created by the ML4O tool in `fusionResultEvaluation` mode.

6.3 Evaluation Results

Number of training a-boxes	Number of false positives				Number of false negatives				Number of false negatives not in request ARFF file			
	*k*NN		SMO		*k*NN		SMO		*k*NN		SMO	
modality	*y*	*n*	*y*	*n*	*y*	*n*	*y*	*n*	*y*	*n*	*y*	*n*
10	0	10	0	3	46	36	46	40	5	4	5	3
20	0	12	20733	270	46	23	30	**9**	5	3	2	**0**
30	0	24	**47998**	1094	**46**	22	30	**5**	5	3	2	**0**

Table 6.5: Numbers of false positives, false negatives and false negatives that were not in the request ARFF file s.t. number of training a-boxes, classifier type and modality usage. The *modality* line containing *y* and *n* determines, whether for theses tests, modalities were used for feature building.

In table 6.5 one can see a statistic about all the feature vectors that were classified as false positives or false negatives. One can see that the *k*NN classifiers produced a lot less false positives in all test rounds, while the SMO classifier produced large amounts of false positives especially in the test rounds with twenty and thirty training a-boxes and modalities. The performance regarding false positive production overall decreases throughout all tests.
On the other hand the SMO classifier has the best classification results looking at the classifications without modalities and twenty and thirty training a-boxes. This can also be seen in table 6.6, where one can see that these classifiers correctly classified 80% and 89% of the $same - as$ assertions in the test set.

Number of training a-boxes	Correctly classified $same-as$ assertions in %				Correctly classified total in %			
	kNN		SMO		kNN		SMO	
modality	y	n	y	n	y	n	y	n
10	0	24	0	14	99.99	99.99	99.99	99.99
20	0	53	36	**80**	99.98	99.98	92.26	99.89
30	0	56	36	**89**	99.98	99.98	**82.11**	99.59

Table 6.6: Percentages of correctly classified $same-as$ assertions and correctly classified instances total s.t. number of training a-boxes, classifier type and modality usage. The *modality* line containing y and n determines, whether for theses tests, modalities were used for feature building.

Another interesting point is that the numbers of false negatives, not in the request ARFF file, vary from kNN to SMO. This does not make any sense, as for both classifier types the same request ARFF files were used. There is either a bug in the evaluation code, or the classification of the WEKA tool is not working correctly. The reasons for this irregularities in the data were not further examined due to time limitations.
The major surprise anyhow was to see that all classifiers using modalities for training and classification performed worse than the ones, not using modalities. This is also valid for the amount of time that is consumed to classify one feature vector as can be seen in table 6.7. This stems from the fact that the training feature spaces that use modalities are always of higher dimension than those, that do not use modalities, as can be seen in table 6.3.

Number of training a-boxes	Number of feature vectors in request ARFF files		Number of seconds for classification			
			kNN		SMO	
modality	y	n	y	n	y	n
10	267475	269047	37209	25228	97	91
number of seconds per instance			*0.139*	*0.093*	*0.0003*	*0.0003*
20	268378	269308	92653	58606	217	220
number of seconds per instance			*0.345*	*0.218*	*0.0008*	*0.0008*
30	268439	269308	155890	106754	657	279
number of seconds per instance			***0.581***	***0.396***	***0.0024***	***0.001***

Table 6.7: Numbers of instances in all request ARFF files, total seconds of classification and seconds needed for classification of one instance s.t. number of training a-boxes, classifier type and modality usage. The *modality* line containing y and n determines, whether for theses tests, modalities were used for feature building.

In table 6.7 one can also see that the performance regarding classification speed of the kNN, as a lazy classifier, is far beyond the performance of the SMO

classifiers. The higher the dimension of the feature space, the longer one will have to wait for classification results.

Chapter 7

Conclusion

7.1 Innovations

After having given an introduction into information retrieval and one of its approaches to handle fusion of semantic annotations of a multimodal document with description logic reasoning, we introduced the idea of using machine learning for this task and pointed out what the objectives of using machine learning, instead of logical reasoning are.

Afterwards we developed a general applicable approach in which the feature vectors are built up by examining the neighborhoods of the two individuals that are to be classified.

Followed by the development of the new approach, it was implemented as a prototype.

Finally tests were run with this prototype that enabled us to evaluate the performance of our approach on the BOEMIE test data set. The results of the classification tests render the expectations that we had before running the tests, with one exception: The performance of the classifiers that used modalities in the features, was by far worse than the performance of the classifiers without modalities. The prior assumption was that, by enriching the feature building process through including modalities, the classifier would be more precise, but the contrary could be observed. We could especially observe an increasing number of false positives using modalities and the SMO classifier, where the number of false positives literally exploded.

Overall one can say, that the approach we developed in this work cannot be used as is. Anyhow it can be seen as a basis, on which further research can develop better performing approaches and implementations for fusion through machine learning.

7.2 Future Work

Even though the approach that we developed and the test we ran using it were partly successful, there is a big potential for future research to dig deeper into the details of this particular approach and potentially improve the performance.

7.2.1 Altering the approach

One possibility to improve the overall performance of the approach might be to change it in a way, that different levels of classification could be used to get better results. For the given test results one could think of an approach where two different machine learning algorithms are combined in a way, that the classification results from one classifier are re-classified, thus using benefits of both algorithms and minimizing their disadvantage. For the tests, that were run, we propose that at first a classification with the SMO classifier is performed. As there are a lot of false positives (especially in high dimensional feature spaces) using SMO classification, the positive ($same-as$) classified feature vectors could be re-classified by the kNN classifier, which produces less false positives. This would have two advantages leading to a synergetic effect:

1. The classification result becomes more exact than just using the SMO classifier

2. The runtime of the kNN classifier is reduced, by presenting only possible positive examples.

In this way, also higher dimensional spaces could be classified with the kNN classifier, in spite of its laziness.
Another idea focuses on the reduction of the dimensionality of the feature space. When examining the training matrix, one could look for selected features that occur rather seldom, maybe 1% of all example feature vectors contain this feature. Then this feature could be omitted, thus reducing the dimensionality of the feature space and therefore leading to less computational complexity.

7.2.2 Implementation

Also on the implementation level a couple of potential improvements can be found. The one that would probably have the biggest impact on the precision of the classifier, aims at the generation of the features. In this work the neighborhoods were only examined in a depth of one. If the neighborhoods would be examined deeper, the feature space would become higher dimensional, but the description of the neighborhood would at the same time become more and more precise, the deeper the neighborhood is examined.
The generation of the feature vectors for the training and request matrices is not programmed very efficiently. A suggestion for improving the performance in this area is, to let Racer do the work internally, instead of sending each small request to it over the network interface. This should boost the speed of the generation of the matrices drastically, as there is a big communicational overhead, when sending a large amount of requests to Racer.
One obvious future improvement is that currently the predictions from WEKA are not added to the knowledge base. It was not the intention of this work to achieve this, but it would be the logical next step.
When generating the request matrix, we right now just merge *all* interpretation a-boxes of one multimodal document into one big a-box. This often leads to inconsistencies in the merged a-box, so that it could not be used for further testing. A less straightforward approach when merging the interpretation a-boxes might lead to better results, when generating the request matrices. Kaya

used one approach in his PhD thesis [8] that introduces the notion of *fusion paths*, which might be a good starting point for improvements.

7.2.3 Testing

In this work we only used two machine learning algorithms with default configurations to build classifiers. There is much potential to increase the classifier performance, by tweaking their parameters, that was completely left out in this work. One example would be, to choose another value than one for the kNN classifier. And of course there are other machine learning algorithms, like Naïve Bayes or LogitBoost that tests could be run with.
Also tests with bigger a-boxes for both, the generation of training and request matrices is to be researched and will offer deeper insight into the performance of this fusion approach.
In order to decide whether the machine learning approach is better or worse than the logic based approach, one would have to run performance tests with the logic based fusion. Afterwards one can compare the results from this work to the new test results and decide which approach outperforms the other one.

Nomenclature

kNN	k-Nearest Neighbor
a-box	Assertion Box
ARFF	Attribute-Relation File Format
BOEMIE	Bootstrapping Ontology Evolution with Multimedia Information Extraction
cli	command line interface
DL	Description Logic
gui	graphical user interface
HLC	High-Level Concept
IR	Information Retrieval
kb	Knowledge Base
ML	Machine Learning
ML4O	Machine Learning for Ontologies
MLC	Mid-Level Concept
SMO	Sequential Minimal Optimization
SVM	Support Vector Machines
t-box	Terminological Box

Appendix A

Classification Scripts

```
#USAGE in the directory with the request arff files : ~/classify.sh /path/to/
    current/model/ current.model /path/to/training.arff
FILES="*"
export CLASSPATH=/usr/lib/jvm/java-1.5.0-sun/lib:~/weka/weka-3-6-1/weka.jar
CURRENTDIR=`pwd`
TMPLINE=`grep "%%%% Number of attributes: " $3`
COLONIDX=`expr index "$TMPLINE" :`
NUMOFATTS=${TMPLINE:COLONIDX}
for f in $FILES
do
 MYCOMMAND="java -Xmx8G weka.classifiers.functions.SMO -p$NUMOFATTS -l $1$2 -
     T $CURRENTDIR/$f"
 START=`date "+%s"`
 echo START:$START >> ~/ml4o/predictions/log-$2-$f.txt
 date "+%H:%M:%S" >> ~/ml4o/predictions/log-$2-$f.txt
 echo "command: $MYCOMMAND" >> ~/ml4o/predictions/log-$2-$f.txt
 $MYCOMMAND >> ~/ml4o/predictions/log-$2-$f.txt
 END=`date "+%s"`
 echo END:$END >> ~/ml4o/predictions/log-$2-$f.txt
 date "+%H:%M:%S" >> ~/ml4o/predictions/log-$2-$f.txt
done
```

Listing A.1: Bash script that was used to automatically classify a list of request ARFF files in the current directory using the SMO classifier.

```
#USAGE in the directory with the request arff files : ~/classify.sh /path/to/
    current/model/ current.model /path/to/training.arff
FILES="*"
export CLASSPATH=/usr/lib/jvm/java-1.5.0-sun/lib:~/weka/weka-3-6-1/weka.jar
CURRENTDIR=`pwd`
TMPLINE=`grep "%%%% Number of attributes: " $3`
COLONIDX=`expr index "$TMPLINE" :`
NUMOFATTS=${TMPLINE:COLONIDX}
for f in $FILES
do
 MYCOMMAND="java -Xmx8G weka.classifiers.lazy.IBk -p$NUMOFATTS -l $1$2 -T
     $CURRENTDIR/$f"
 START=`date "+%s"`
 echo START:$START >> ~/ml4o/predictions/log-$2-$f.txt
 date "+%H:%M:%S" >> ~/ml4o/predictions/log-$2-$f.txt
 echo "command: $MYCOMMAND" >> ~/ml4o/predictions/log-$2-$f.txt
 $MYCOMMAND >> ~/ml4o/predictions/log-$2-$f.txt
 END=`date "+%s"`
 echo END:$END >> ~/ml4o/predictions/log-$2-$f.txt
 date "+%H:%M:%S" >> ~/ml4o/predictions/log-$2-$f.txt
done
```

Listing A.2: Bash script that was used to automatically classify a list of request ARFF files in the current directory using the kNN classifier.

Appendix B

Configuration File of the ML4O tool

settings.properties		
key	**value**	**description**
racerHostname	any hostname or ip	Hostname of the machine RacerPro is running on.
racerPort	0 ... 65535,	Portnumber of RacerPro.
modusOperandi	fusionTraining fusionRequestGeneration fusionResultEvaluation	Sets the mode of operation of the ML4O tool.
useModalities	yes, no	Sets whether modality information should be used to train the classifier or generate the request matrix.
modalityTypes	Text, Image, ...	Concept names of the modalities in the ontology.
wekaArffTrainingFile	e.g. data/ml4o.arff	The wekaArffTraining file is the used for fusionTraining as the reference, to where the training arff file should be written, as well as for the fusionRequestGeneration, so that the columns of training and test files are the same.
wekaArffRequestFolder	e.g. data/toClassify/	used to specify, in which the fusionRequestGeneration should put its request ARFF files.

`aboxTrainingFolder`	e.g. /aboxes/training	Specifies the folder, in which the fusionTraining is looking for a-boxes to train with.
`aboxToBeTestedFolder`*	e.g. /aboxes/did-files/	Specifies the folder in which the files with the a-box ids can be found. In BOEMIE the *.did files contain those ids.
`aboxTestingRepo`*	e.g. /aboxes/urRepo/	Specifies the directory in which the interpretation a-boxes can be found. In this directory, there are folders, that are named as the a-box ids from the *.did files in the `aboxToBeTestedFolder` (We used a mirror of this folder: `http://repository.boemie.org/BoemieRepository/Analysis/`)
`aboxSubStructureFolder`*	e.g. /path/to/int-aboxes/	Specifies the subfolder in each a-box id folder, that contains the folders for the different modalities of a document after interpretation.
`predictionFolder`	e.g. /ml4o/predictions/	Specifies the folder, in which the prediction output produced with the scripts of appendix A can be found, so that the fusionResultEvaluation can find them.
`fusedInterpretation-AboxFolder`*	e.g. /aboxes/fused/	Specifies the folder, in which the ground truth or fused interpretation a-boxes can be found, so that the evaluation can use them.
`evaluationResultFolder`	e.g. /ml4o/evaluationResults/	Specifies the folder, in which the ML4O tool stores the evaluation results, for the predictions in the predictionFolder.

`arffFileRoot`*	e.g. /arff/	Specifies the folder, in which the evaluation mode of the ML4O tool can find the ARFF files in a folder structure that matches the implementation of the evaluation part of the ML4O tool.

* The use of this option is test data set dependent.

Bibliography

[1] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991. 3.2.4

[2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, Cambridge, 2007. (document), 2.1, 2.2.1, 2.2.2

[3] F. Baader and F. Distel. Exploring finite models in the description logic $\mathcal{EL}_{\text{gfp}}$. Technical report, Theoretical Computer Science, TU Dresden, Germany, 2009. 2.2

[4] K. P. Bennett and C. Campbell. Support vector machines: Hype or hallelujah. *SIGKDD Explorations*, 2(2):1–13, 2000. 3.2.4

[5] I. Bronstein, K. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 2001. 3.2.4

[6] S. Castano, S. Espinosa, A. Ferrara, V. Karkaletsis, A. Kaya, R. Möller, S. Montanelli, G. Petasis, and M. Wessel. Multimedia interpretation for dynamic ontology evolution. Technical report, National Centre for Scientific Research "Demokritos" (NCSR), University of Milano (UniMi), Hamburg University of Technology (TUHH), 2007. 2.2.2

[7] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. Wiley, second edition, 2001. 3.1.2

[8] A. Kaya. *A Logic-Based Approach to Multimedia Interpretation*. PhD thesis, Hamburg University of Technology, 2009. (document), 1.1, 4, 4.1, 2, 7.2.2

[9] V. Kecman. *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models (Complex Adaptive Systems)*. The MIT Press, 2001. 3.2.4

[10] M. Listiani. Support vector regression analysis for price prediction in a car leasing application. Master thesis, TU Hamburg-Harburg, Mar. 2009. (document), 3.1

[11] Merriam-Webster. Merriam-webster online dictionary. `http://www.merriam-webster.com/dictionary/learning`, Date checked: 2009-09-10. 3

[12] B. Mobasher. Classification via decision trees in weka. http://maya.cs.depaul.edu/~classes/ect584/WEKA/classify.html, Date checked: 2009-10-21. 5.3.2

[13] N. J. Nilsson. *Introduction to Machine Learning.* Computer Science Department, Standford University, USA, 1996. http://robotics.stanford.edu/people/nilsson/MLDraftBook/MLBOOK.pdf, Date checked: 2009-09-10. 3

[14] S. Perantonis, R. Möller, S. Petridis, N. Tsapatsoulis, D. Kosmopoulos, M. Anthimopoulos, B. Gatos, E. Iosif, G. Petasis, V. Karkaletsis, G. Stoilos, W. Hesseler, K. Biatov, M. Wessel, A. Kaya, and K. Sokolski. D2.9 semantics extraction from fused multimedia content. Technical report, National Centre for Scientific Research "Demokritos" (NCSR), Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V. (FHG/IMK), University of Milano (UniMi), Centre for Research and Technology Hellas (CERTH), Hamburg University of Technology (TUHH), Tele Atlas (TA), 2009. 1.1, 4.2, 4.2.1

[15] J. C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Microsoft Research, 1998. 3.2.4

[16] B. Project. Intorduction - boemie. http://www.boemie.org/, Date checked: 2009-10-06. 4.1

[17] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach.* Alan R. Apt, second edition, 2003. 3, 3.1.3, 3.2.4

[18] R. S. Sutton and A. G. Barto. *Reinforcement Learning, An Introduction.* MIT Press, 2002. 3.1.3

[19] C. Thornton. *Techniques in Computational Learning, An Introduction.* Chapman & Hall Computing, 1992. 3

[20] V. N. Vapnik. *The nature of statistical learning theory.* Springer, second edition, 1998. 3.2.4

[21] WEKA. Weka javadoc. http://weka.sourceforge.net/doc/, Date checked: 2009-10-21. 5.3.2

[22] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, San Francisco, second edition, 2005. (document), 3.2.1, 3.2, 3.2.4