Inducing Conceptual User Models

by
Martin Eric Müller
M.A. (University of Osnabrück) 1995

Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

of the University of Osnabrück

Submitted June 2001
Revised version for publication, March 2002.

Inducing Conceptual User Models

Dissertation zur Erlangung des philosophischen Doktorgrades
am Fachbereich Sprach– und Literaturwissenschaft
der Universität Osnabrück

vorgelegt von
Martin Eric Müller
aus München

Erstgutachten:	Prof. Dr. Claus-Rainer Rollinger Univ. Osnabrück
Zweitgutachten:	Priv.-Doz. Dr. habil. Helmar Gust Univ. Osnabrück
Drittgutachten:	Prof. Dr. Ivo Düntsch Univ. Ulster, Jordanstown, NI.

Osnabrück, im Juni 2001
überarbeitete Version, März 2002

Erklärung[1]
Ich versichere hiermit an Eides statt, daß ich die Dissertation mit dem Titel

Inducing Conceptual User Models

selbstständig und ohne unerlaubte Hilfe verfaßt und die benutzten Hilfsmittel vollständig angegeben habe.
Ich habe die Dissertation noch nicht als Prüfungsarbeit für eine andere wissenschaftliche Prüfung eingereicht und mit dieser Abhandlung noch keinen Doktorgrad erworben.

Martin Eric Müller

[1] gemäß der Promotionsordnung des Fachbereichs Sprach- und Literaturwissenschaft der Universität Osnabrück vom 25.8.1984, §2 (3) a Abs. 2 sowie §2 (3) f.

Abstract

User Modeling and Machine Learning for User Modeling have both become important research topics and key techniques in recent adaptive systems.

One of the most intriguing problems in the 'information age' is how to filter relevant information from the huge amount of available data.

This problem is tackled by using models of the user's interest in order to increase precision and discriminate interesting information from un–interesting data.

However, any user modeling approach suffers from several major drawbacks: User models built by the system need to be inspectable and understandable by the user himself. Secondly, users in general are not willing to give feedback concerning user satisfaction by the delivered results. Without any evidence for the user's interest, it is hard to induce a hypothetical user model at all. Finally, most current systems do not draw a line of distinction between domain knowledge and user model which makes the adequacy of a user model hard to determine.

This thesis presents the novel approach of conceptual user models.

Conceptual user models are easy to inspect and understand and allow for the system to explain its actions to the user.

It is shown, that ILP can be applied for the task of inducing user models from feedback, and a method for using mutual feedback for sample enlargement is introduced.

Results are evaluated independently of domain knowledge within a clear machine learning problem definition.

The whole concept presented is realized in a meta web search engine called OySTER.

Contents

About this Thesis

The thesis as submitted in June, 2001, has been revised for publication during Spring 2002.

- Many typesetting errors, both in spelling and, for example, wrong indices in formulas, have been removed.
- The original thesis has been critizised as not being 'easy to read'. I included some material from my lectures on user modeling and machine learning and rearranged some paragraphs to increase comprehensability. Still, this thesis is not an introductory textbook. Nevertheless, we hope to get the basic idea across to both the textbook–reader and advanced researchers.
- The last version before submission was prepared in December 2000. While awaiting comments, the final version was prepared and then submitted in late June, 2001. Reviews were published in December 2001.
  It is clear that a thesis which is related to a highly dynamic research area is prone to be outdated by the date of publication. I tried to catch up with the last 12 months.
- Since this is an on–line publication, PDF seems to be the appropriate document format while the submitted version was prepared using PostScript. As a consequence, all images included in this thesis had to be redrawn in or converted to PDF–compatible formats. Sadly, the quality of some screenshots suffered from the conversion because PDF cannot include the high resolution PostScript images.

Paragraphs or footnotes that were added or heavily changed during the revision are marked with a superscript star: *.

Structure of Thesis

In the first part, the core concepts needed for the thesis are introduced. We give a brief overview of the contributing disciplines, User Modeling and Machine Learning, and their intersection, Machine Learning for User Modeling.
The first chapter of the second part provides the reader with a more user– and application–oriented overview of the approach of conceptual user models. It can be seen as a less formal but more concrete introductionary part. The rest of part two is the core of the thesis. The basic idea behind the approach is introduced, formally described and the framework for induction of conceptual user models is developed in detail. The part closes with an evaluation of the presented methods.
The final part contains a summary, conclusions and prospects.
During the course of the this thesis many different problems had to be solved which were not directly related to the core idea of the thesis. Therefore, many of those aspects—which are referred to in the text—are documented in the appendix. Most of them concern implementational issues.

Reading Recommendations

The reader who is not familiar with machine learning or user modeling should first read the according sections in part one, chapter two in order to get the basic ideas that are needed for the further understanding of the text.

Core of thesis. Assuming the reader to be familiar with the discipline of machine learning for user modeling, jump to part two, chapter 4 directly. You might need to resolve a few backreferences.

Application oriented introduction. To get a first rough picture of the underlying idea, read section 3.2. Section 3.1.2 relates this idea to our formalization of user adaptive systems. The realization of the idea is described in section 3.2.2 and the rest of chapter 3.

Modern search engines. Basic concepts from information retrieval are introduced in section 2.3.1. A brief comparison on search indices is shown in figure 2.6. Index based search engines and meta search engines are described in sections 3.1 and 3.3. Section 2.3.3.1 presents related work in the field of recommender systems and chapter 3.2 finally introduces the adaptive meta search engine OySTER.

A selective overview. The idea behind machine learning for user modeling is illucidated in section 2.3.2. Paragraph 3.2.1.1 gives a very intuitive introduction in the general idea behind OySTER and conceptual user models. Section 3.2 gives a more application oriented introduction. The idea behind conceptual user models is described and formalized in chapter 4. Section 5.1 focuses on representational issues in conceptual user models. The new user modeling problem is described in section 5.3. In section 5.2, we describe how to generate large samples from few feedback data. The approach is evaluated in section 6.3.
Related work is discussed in sections 2.3.3 and 3.3.

Acknowledgements

It must be stressed, that one motivation for this thesis was to develop an abstract formal framework for describing and evaluating adaptive user interfaces which make use of machine learning methods in user modeling. Therefore, a certain degree of formalism is required.
It has been stated by one of the reviewers, that such an approach is prone to criticism, since a clear picture of the idea allows for detection of fallacies and drawbacks. However, it is my opinion that a clear result offers a more valuable and promising contribution to the scientific community than a self–satisfied and vague presentation of apparent impressive results.

I want to thank numerous anonymous reviewers for their comments on earlier paper publications related to my work and the reviewers for my dissertation, Prof. Dr. Rollinger, PD Dr. Gust and Prof. Dr. Düntsch.
I am grateful to all researchers involved in the machine learning for user modeling community who by their detailed criticism helped to improve the work as presented in this thesis.
I also want to thank all my friends and colleagues for creating a fruitful and supporting environment.

I

MACHINE LEARNING FOR USER MODELING

In this part, we give a brief introduction into the fields of user modeling and machine learning.
User modeling (UM) is one technique that is used in adaptive user interfaces (AUIs) which allows for a better human–computer interaction (HCI). It takes a considerable amount of intelligent techniques to enable a system to adapt to a user autonomously. Therefore, such interfaces are also referred to as intelligent user interfaces (IUI).

In machine learning (ML), a system induces a theory (or rather, a hypothesis) which shall explain and predict phenomena that are being observed based on a limited set of such observations.
Depending on the paradigm, theory–driven approaches like decision tree induction (TDIDT) or inductive logic programming (ILP) require a sophisticated domain model but deliver a clear, logic based hypothesis. Other approaches—like artificial neural networks (ANN) including self–organizing maps—are easier to start with, but do not guarantee to deliver a understandable theory.

We show, that ML is just the right technique that is needed in AUIs to automatically learn about the user and thus, to adapt to his or her individual needs.

Chapter 1

INTRODUCTION

1.1 Motivation*

Nowadays, many communication and manipulation interactions are interactions between humans and computer controlled devices. Human computer interaction (Hci) ranges from electronic switchboards to information retrieval systems—from electronic ticket machines and internet last–minute booking services to tactical head up displays in military aircraft.
The opportunities offered by modern computer technology promised a better every day life, where robots carry out unpleasant tasks, machines are easier to control and intelligent computers would provide us with any information we are looking for. Actually, we are overwhelmed by a multitude of buttons, wheels and switches on control panels; instead of wondering whether to find information, we struggle with the task of how to find relevant information only and quite often we are simply unable to read the desired information from a variety of screens that display information in different languages using different graphical user interfaces, designs and presentation methods.

1.1.1 User Modeling and Adaptive User Interfaces

The terms 'user modeling' (Um) and 'adaptive user interface' (Aui) are sometimes used interchangeably. Actually, user modeling has older roots than adaptive user interfaces: In the 70's, user modeling was mainly concerned with explicit modeling of dialog partners in the natural language processing Ai community[1]. Then, with the growing impact of computer science, issues of Hci were discussed throughout all disciplines. At this point, experience from user modeling entered the discourse to form the discipline of Aui.
In this work, we will use the term 'user model' or 'user modeling' for a distinct part or technology used within adaptive user interface systems. 'User modeling' is the process

[1] Alfred Kobsa, who worked on belief models and partner modeling in natural language dialogs, [Kobsa and Trost, 1984], published a paper on 'user modeling' in a natural language system in 1986 [Kobsa, 1986].

of representing and refining user characteristics in a 'user model' in order to achieve an adaptive behavior of the interface. An AUI is the whole system by which a user interacts with a system.
Intelligent user interfaces, IUIs, adapt themselves to the user by reasoning about the user and refining their internal model of the user's needs. The refinement of the user model is a sequence of inferences based upon several observations of user interactions, commonly known as feedback (already introduced into the information retrieval community in [van Rijsbergen, 1979]). This yields an abstract architecture as shown in figure 1.1. A user's information *request* is translated into a system *query* by taking into account

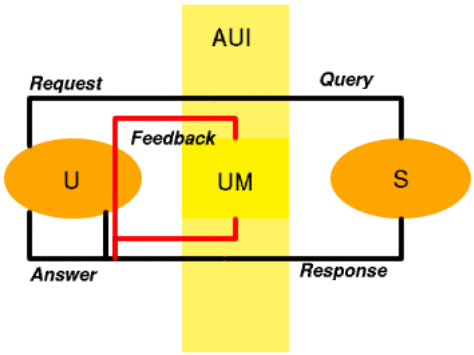


Figure 1.1: An abstract AUI*

knowledge about the domain (i.e. the system S) and knowledge about the user (taken from the internal user model UM). The system's *response* is transformed (by filtering, arrangement, aggregation etc.) again taking into account the user model UM into an *answer* that is presented to the user. The user interactions (including explicit *feedback*) are used to refine the user model in order to be able to deliver more precise reuslts next time.

1.1.2 Machine Learning for User Modeling

In machine learning (ML), we deal with artificial systems which learn how to perform better through experience. By observing examples from a *sample*, the learning algorithm LA tries to induce a hypothesis H which approximates a general, unknown law that explains the nature of all objects of the domain (both observed and unknown). This inductive inference is supported by a set of background knowldege BGK. As long as the hypothesis and according predictions disagree with observations, the hypothesis needs to be refined. This process is depicted in figure 1.2. In user modeling, we are concerned with building artificial systems that behave differently for different users. It

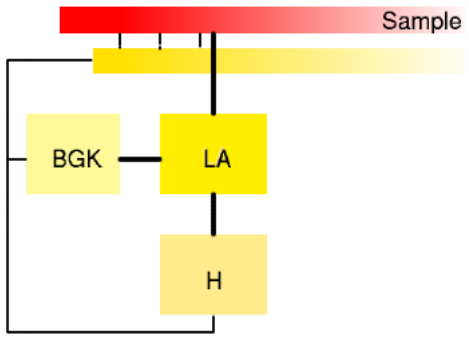


Figure 1.2: An abstract ML system*

is a straightforward idea to allow a system to learn how to adapt to different individual users.

Both Machine Learning and User Modeling have been hot topics from the early advent of artificial intelligence —since both try to make the computer a more 'intelligent', more user and problem adapted system. Similarly, both disciplines are (again) the focus of current research interest: in an expanding data society, the search for information has become a difficult task. Machine learning methods are used to extract knowledge out of huge data sets (e.g. in the data mining community), and user modeling tries to extract user dependent relevant information out of huge information systems.

How can one build an appropriate user model? In general, user actions have to be interpreted in order to construct and subsequently refine the user model autonomously. Now, given a user model and user feedback on fallacies of this model, we have to carry out a model refinement procedure in order to improve performance.[2] This already suggests a strong resemblance to machine learning tasks.

The parallels of user modeling and machine learning are depicted in figure 1.3. The left hand side depicts an abstract machine learning system.

The input is a sequence of labeled data. The learning algorithm then tries to find a hypothesis that matches the target function, i.e. the labeling function. This is the same setting for all machine learning approaches.

On the right hand side, we have sketched an adaptive user interface system. Real world data (as delivered from the information source) is presented via a user interface that incorporates a user model. The user now gives feedback on whether the data delivered was relevant with respect to his needs. This feedback is used in an adaptive algorithm to change the user model in the user interface—thus, hopefully, producing a better result to the user's query.

[2] 'Improvement of performance' here means to 'perform better', not just 'faster'.

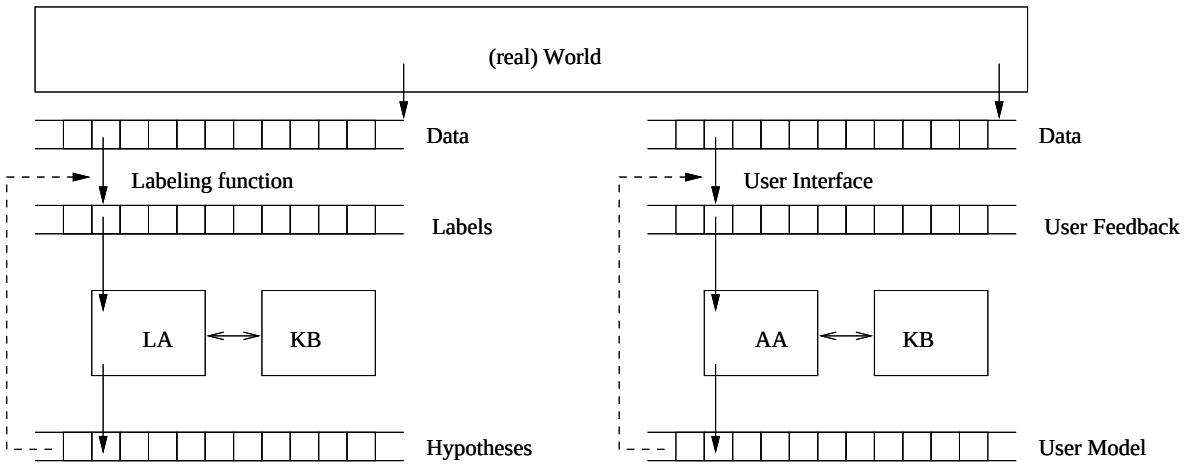


Figure 1.3: The parallels between machine learning and adaptive user interfaces

1.2 OySTER: Searching for information pearls.

Dealing with machine learning, one depends on data. Data is domain dependent. Furthermore, the algorithms are sensitive to data availability, representation, quality and so on. Thus, the question arises which domain to choose. It is quite popular to choose a 'chique' domain that both attracts people and provides data 'en masse'. Prototypically this is the world wide web (Www).
Searching for information on the Www is becoming a more and more difficult task: With the advent of huge search engines the question of being able to find information has turned into the question of how to find only relevant information. Both precision and recall on a search of over one billion documents is pretty poor. Therefore, it seems desirable to have a search mechanism at one's disposal which guarantees higher precision. This problem is the subject of several research areas, including natural language interfaces[3] and information retrieval, [van Rijsbergen, 1979]. A recent approach is the use of user adaptive systems which, in the context of web search, allows for a personalized query refinement and response filtering.
Both above mentioned observations lead to the same conclusion: The paradigm of applying machine learning techniques in the context of user modeling in order to learn a user's interest. This thesis introduces a new aspect of machine learning for user modeling; namely the use of conceptual user models. The use of conceptual user models introduces

[3]Here, the system tries to anticipate what the user exactly is looking for by analyzing and 'understanding' the query—based on the assumption that natural language is the language which allows for the most precise query formulation from the user's point of view. Recent research also takes into account the whole interaction discourse by interpreting query–response sequences as a dialogue which helps to further specify the user's needs, see [Ronthaler, 2000].

a new unified method of describing a user's interest with respect to an underlying category, and the ability to explain both a model, and the reason why recommendations were given by the system to a user. Furthermore, the problem posed by the usage of conceptual expressions as user models gives rise to the usage of inductive logic programming, which is a fairly unknown and underestimated method within machine learning for user modeling.

In the course of this thesis, we developed an adaptive meta search engine for the World Wide Web called OySTER. The idea behind implementing OySTER was to create a testbed for different user modeling techniques in a large domain. The need for more precise search engines is evident—and users searching for information on the web provide us with a large amount of data that can be used for machine learning processes.

Some ideas presented in this thesis are realized within OySTER. Additionally, the system has been developed according to the needs of a real 'workbench': It is a highly modularized multi agent system and allows for easy replacement of different agent families which carry out the different tasks of wrapping, classification, user modeling, content based filtering and information presentation. The user's view on the system is presented in chapter three; a more in–depth description can be found in the appendices.

Chapter 2

MACHINE LEARNING FOR USER MODELING

In this chapter we will introduce the most important ideas of user modeling and machine learning which contributed to the new and still growing discipline of machine learning for user modeling (ML4UM). The discipline of ML4UM had its first official appearance in a workshop at the 6th International Conference on User Modeling in 1997, [Jameson et al., 1997], where the special interest group ML4UM was founded. Ever since then, the community grew and met frequently for meetings and workshops in several national and international surroundings. At this year's 9th International Conference for User Modeling, 16 of 100 submissions were directly related to machine learning (plus a considerable amount of submissions in the field of user modeling, information retrieval and machine learning).

Due to its impact in both parent disciplines it is hard to categorize research advances. This holds especially for the machine learning part: Utilized techniques are chosen with respect to the problem, and since the data available (i.e. user feedback) suffers from a great amount of inherent noise, only few machine learning techniques are commonly used.

This thesis is based on a machine learning approach which has not been used frequently within user modeling, namely inductive logic programming (ILP). Thus we will have to include a few 'extra' pages about this special machine learning approach in this chapter.

First, we will start with an introduction into the field of user modeling.

2.1 User modeling

User modeling has become a synonym for more user friendly systems that adapt to the user's behavior or preferences step by step. They actually cover only a fraction of all systems that are involved in a human–computer–interaction.

2.1.1 Human computer interaction

Human Computer Interaction can only be accomplished with the use of interfaces. These are, mainly, visual information displayed on screens and the keyboard.
Thinking one step further, interfaces are all kinds of devices that are used by computers, robots or machines to display (or output) information and all devices we use to control the machine (like touch screens, joysticks, eye-trackers, voice, and so on). The interface is not defined by the mere term 'screen' but of course by 'presentation'. There are numerous graphical user interfaces (GUI)—generic GUIs for operating systems and proprietary graphical interfaces for special applications (like flight control, life function monitoring, etc.).
For all GUI standards and application domains there exist different guidelines for the graphical realization, layout or functionality—but one interface for all users forces people to learn and to adapt to the system. This might include learning different (query) languages or domain models which differ and leads to different views and expectations on both the problem and query formulation as well as on the interpretation of results.
'Good' HCI, however, does not neccessarily mean that one needs an adaptive interface. Any system, that can be controlled by a human with a minimum cognitive effort can be regarded to as a system with good HCI. The most important characteristics of such a system is *visibility* and *affordance* of the interface components. In other words, the controls themselves must be 'easy' to manipulate and both appearance and mode of interaction should be 'intuitevly' mapped onto the function of the control element, [Preece, 1994]. As a metaphoric example, one might compare a human–computer interaction situation with the problem of getting from the ouside into a building. The doorway corresponds to an interface: if there is no, there is no way to enter the building at all. With invention of the door, many new problems arised: lock and unlock, push or pull, access restrictions and so on. A door with a 'good' human–door interaction needs not to be an automatic slide door: it might be appropriate to have a simple swing–door. Good affordance means, that from the design of the door it is immedietlay clear, whether we have to push or pull the door (or wheter both is possible). However, automagic slide doors with motion sensors provide some kind of adaptivity and are—in most cases—much easier to use: you just walk through them and the door will open itself. Of course, there are bad examples as well: many automatic doors in hospitals have push–buttons instead of sensors; and the buttons are located at the wall in a reasonable distance from the door (in order to allow nurses to both push a bed and open doors). But for a visitor this mechanism is rather opaque. Another bad example are rotating doors as you might find them at the entrance to shopping malls: They rotate at a constant speed (to slow for visitors in a hurry and too fast for the elderly) and the latency period to re–start rotating after a timeout is rather long.
As a conclusion, it seems, that in certain situations adaptivity is a good method to increase HCI—but it is not the one and only method.

2.1.2 Adaptivity*

Adaption of user interfaces to individual user needs has several facets based on user characteristics which do not belong to a user model (said to contain information about the user's interest) in the strong sense.
Thus, a system can adapt to different physical or mental abilities or disabilities. Examples for such user ability restrictions are blind or visually disabled people, lacking motory skills, or cognitively impaired users. Impairment does not necessarily imply physical or mental disability but also low cognitive or physical capacity due to context—like bad vision in dark or foggy environments, lacking motory skills when wearing heavy protective garments, and low cognitive capacity due to overload, stress, risk or hazardous situations.
Taking into account such contextual information, this could help to improve information presentation on displays (including ergonomic considerations with respect to readability) and interaction (like button arrangement, default selections and menu organization).
In general, an adaptive user interface tries to increase the quality of HCI by:

1. building a model of relevant aspects of the user
2. by drawing non–trivial conclusions from
3. observed user interactions
4. and applies the hypothesis about the user characteristics in order to
5. support the user in completing his task

Starting at item (3), one collects evidence describing the user and induces in a process of upward inferencing (2) a model of the user (3). This model is applied in a downward inferencing procedure to predict the user's behavior (4) or support the user in another way. This yields a function diagram describing an abstract adaptive user interface as shown in figure 5.[1*]

Example: Wearable computers for crisis management. Consider a wearable computer designed for the co-ordination of individuals in a rescue team: If the 'wearable' recognizes a hazardous situation in a foggy environment, the user should be alarmed acoustically instead of visually. In another, harmless situation the user might ask the system about how to proceed if he encountered a tank leakage with an unknown fluid. Then, the interface might guide the person step by step through a rescue plan, each time asking the user to acknowledge the last step. If then for example, the system detects a hazardous situation, it would switch to short directives (instead of guidance through a procedure),

[1] Adapted from a tutorial on user modeling by kind permission of Anthony Jameson.

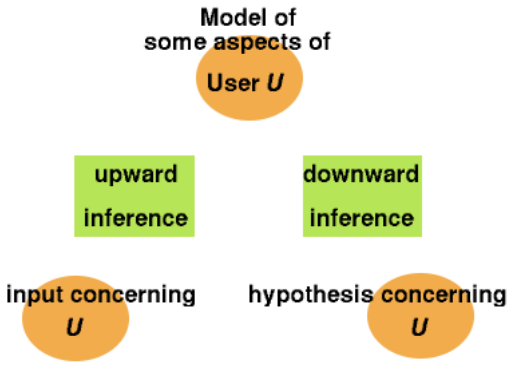


Figure 2.1: Human–Computer Interaction with adaptive interfaces*

then to an acoustic interface (provided there is no noise) with rigid time delays between the single steps and finally maybe even to just a short message like 'Get out of here!'.

This scenario has not been described in the literature so far, but there are many similar projects: for example, [Specht and Kobsa, 1999] describe an adaptive museum guide. [Petrelli et al., 1999] describe the influence of social context in the same context, while [Berthold and Jameson, 1999] focus on cognitive aspects.

Example: A user adaptive web shop. As another example, imagine a web–based catalog of an online shopping service. In general, the same considerations as above apply to this scenario as well. Here, the information presented is chosen with respect to some idea about the user's interest. The presentation itself adapts to user preferences (like more textual or graphical representation) and to the properties of the communication channel such as display properties, sound capability or bandwidth, [Jörding, 1999]. Similarly, special knowledge about the user's abilities or disabilities should be taken into account as well (motory or cognitive skills), [Trewin and Pain, 1997, Spooner and Edwards, 1997]. In contrast to a guide or control system, a user adaptive interface in an information transfer—i.e. communication situation—must be a subordinate communication partner: it must not control the user but the user must be able to control the system. In some cases, the acceptance of a system can be assisted by the use of humanoid interface elements, [André and Rist, 2000].

Most applications we deal with in user modeling are similar to the scenario described in the second example. Furthermore, there is evidence, that user control over the system is a requirement in user guidance systems as well, see section 2.1.4.

2.1.3 User modeling

Crafting adaptive user interfaces that change their behavior with respect to the user's profile, preferences and interest, requires building user models that contain information about exactly those characteristics of the user. The user model can then be used to adapt to the user by means of non–trivial inferences and thus by reasoning about the user. AUIs with UM can help in any domains; as stated earlier we will present some examples in section 2.3.3. We will now briefly discuss, what kind of knowledge we need to store in a user model, how it can be retrieved and which methods we can use in order to adapt to the user.

2.1.3.1 What's in a user model?

Of course, this depends on the domain. In general, every user model consists of two parts: a domain dependent part and a domain independent part.

User profiles. The latter characteristics of a user are often referred to as a user 'profile' that can be altered in the course of some 'registration' process. Nevertheless, this data is important for more informative user models as well. The data might—among other data—include information about the age, gender and education of an individual and maybe his address. Together with his address, it could also contain demographic information, for example collected or derived in the course of data mining processes on a large customer database. Domain independent user data for the user profile or user preferences is usually obtained by explicit user interaction. This process is time consuming and bothers the user. Thus, profile data should be acquired only once and then distributed under different applications.[2] Furthermore, demographic information can be used in order to pre–define some aspects of the user profile.

User preferences. Another type of knowledge about the user is information about his 'preferences' which are usually defined through a 'customization' process. This part of the user model does not fit in the strict dual discrimination of domain dependency and independency but rather combines aspects of both: It is rather application dependent information about preferred presentation modes, font sizes, update intervals and so on. This part of the user model also contains information about the user's abilities as described in section 2.1.1. Furthermore it might contain information about the context—which could be environmental information in the case of a wearable computing scenario or hard– and software specific information as for example bandwidth in a web based information retrieval scenario. A bit more application dependent is information about the current session—as user attention decreases over time due to the user becoming tired. These phenomena are all user–, task– and domain dependent.

[2]Sharing customer user profiles is not to be confused with customer profile exchange, CPEX.

User models. Domain dependent user model information finally contains information about the user's individual attitude to the domain—for example his interest with respect to the domain in an information retrieval scenario, his knowledge level with respect to course materials in a tutoring scenario, his expertise in an interactive help system and so on.
Domain dependent user models are a crucial problem within the user modeling process: On the one hand, they need to be accurate and should provide a complete and enclosed image of the user in order to be able to adapt in the right way. On the other hand, it is hard to obtain all this information. Asking the user to fill in questionnaires reveals two drawbacks: First, the user is likely to be unwilling to answer lots of questions. Second, if the user actually tells the system about himself, the information is not very reliable. This again is for three reasons: First, users want to leave a good impression—accordingly they sometimes give a user model that is not really adequate with respect to the actual interests, needs or capabilities of the user. Second, many users only pretend to be willing to give feedback. A similar situation arises, if the user is forced into a questionnaire: He will only click some buttons at random just to give any feedback and to get rid of the form as soon as possible. Finally, there is the inherent problem of misunderstanding:
As an example image a user would have to rank himself as a novice, intermediate or expert. Boundaries between novice and intermediate are defined differently for different users—like modest or self confident people. Furthermore, if a tutoring system for computer science asks the student for his current knowledge level, the student, the system designer and the lecturer might have had different ideas about the relevant knowledge (applied computer science, programming skills, theoretical computer science, maths, ...) and its level (basic, intermediate and expert level) that is needed for this course.
This explains the principal problem of communicating the meaning of a user model to the user himself. In other words: both the user and the systems may have different models of the domain and they may have different models about each other. This leads to the investigation of some aspects of cognitive science in the user modeling community (see 2.1.4).

2.1.3.2 Different methods of user modeling

User modeling techniques are divided into two categories: First, a user can be described in terms of other user's properties. Thus a user is assigned a class of users who share a majority of common features. Since in this approach the growing number of users together forms classes of user types, this approach is commonly known as *collaborative user modeling*. The second approach to user modeling tries to describe a user by discriminating between individual preferences. Thus, one might call this approach *individual user modeling* as opposed to collaborative modeling. Since the user's individual interest is mostly based on (information) content, and content can be used for the effective *filtering* of relevant information. Therefore, this approach is also called *content based user*

modeling.[3]

Collaborative user modeling. In this approach, user models contain user actions which are based on feature representations. Such actions can be explicit feedback or simply any interaction that can be recorded. The user model is then built up by matching the user data against all other user's data; similarly, recommendation is performed by choosing unknown, yet 'near' objects from the domains of matching user models. In other words, collaborative user modeling can be described as clustering in a high dimensional space. Clusters represent user groups and using a distance measure, 'interest' can be defined by hyperboles.
Due to its principle, collaborative user modeling can be realized relatively quickly and easily. One of the big advantages of this technique is, that it does not need any structural or meta–knowledge about the objects. Furthermore, this approach is able to generalize easily by taking into account all other user's models (i.e. by extending the hyperbole with respect to the next cluster's centroid).
On the other hand, collaborative user modeling cannot cope with 'extremely individual', say, 'singular' user models. As a result, it performs badly with few users or little data per user.
One of the most prominent examples for collaborative recommender systems is the online book and media store Amazon. In the context of web based recommendation systems, the WebWatcher is good example (see 2.3.3.1).

Individual user modeling. In this approach, one tries to build user models which describe the individual user's interest regardless of knowledge about other users. Accordingly, this method requires a much more elaborate domain description which allows for a detailed description of the user. To pick up the clustering metaphor from the last section, individual user modeling clusters objects instead of users and tries to find entities which are close to the centroid defined by the user model.
Recommendation is performed by matching information against the user model instead of matching different user models. In the context of the Amazon example from above, this means that individual user modeling as content based filtering would recommend books not based on similarity between users but rather on similarity between objects, i.e. books: content based recommendation would offer books whose topics are related to those books already purchased. As a result, individual user modeling tends to require more representational and performance power than collaborative user modeling. Once provided with a suitable representation formalism, content based user modeling allows for an easy inspectability; the system itself can explain to the user why certain recommendations have been made since both user models and adaptive processes can be

[3]It is noteworthy, that the terms were inspired by application domains. Of course, collaborative user modeling can be content based, too.

explained to the user—provided a suitable representation of data[4]. The system behavior can be predicted by the user and thus the user feels 'in charge'.
If not restrained, adaption by machine learning techniques might tend to over–fitting. This may finally lead to an (unwanted) user guidance instead of user support. In general, individual user modeling cannot be applied in domains where 'content is not available'; i.e. where there is no domain theory describing the objects under consideration.
A very popular example for a content based recommendation system is Syskill&Webert (c.f. 2.3.3.1).

2.1.3.3 Acquiring user feedback

In order to revise the user model, we need user feedback. User feedback in general can be divided into two types: explicit and implicit. In the ideal case, a system would interpret natural user actions without needing any further explicit feedback. It is more likely that we will have to ask the user for some explicit feedback.

Implicit user feedback. Implicit feedback is hard to detect and hard to interpret. In our example, domains of user guiding systems, the systems would need to detect the user's cognitive condition by, for example, stress pattern detection in voice analysis. Recently, more and more sensory data about human users has been able to be obtained: Physiological data, eye-tracking, galvanic skin resistance data, GPS. The problem is how to interpret these data.
In the domain of web based information systems, several data can be interpreted as implicit feedback: Clicking a link, time spent on a page, scrolling, printing and so on. Sadly, all those data share common disadvantages: they all are, if at all, positive evidence for a user's interest. Furthermore, they are of decreasing vagueness—a link name can be misleading such that loading does not at all express the user's interest since he expected something completely different. Time spent on a page is unreliable, since the user might have been distracted from inspecting the page. Printing, for example, could have been a result of a stress situation, where the user simply did not have enough time to read the document on the screen and thus sent it to the printer in order to evaluate it at a later date.[5]

Explicit user feedback. Explicit feedback is obtained by asking the user whether he liked something. For a small and static domain description, a first rough picture of a user

[4]Offering 'shortcuts' by predicting user actions from action sequences using an artificial neural network therefore would require the ability to explain the net's prediction in terms of rules.

[5]The same argument can be applied to bookmarking which usually is interpreted as a strong evidence of interest. This emphasizes the fact, that the sample noise in user modeling is rather high. Further sources of implicit feedback (like mouse movements) are not taken into account here, because such techniques require higher functionality at the client side.

model can be derived using questionnaires. Nevertheless, questionnaires have several drawbacks: First of all, humans are not at all keen on filling in forms. This task is carried out only in situations where the effect is immediate. This poses a very difficult problem for user modeling, where from few data with a minimum amount of time a perfect user model shall be inferred. Second, answers to a questionnaire are not as reliable as one might expect (this has already been pointed out in 2.1.3.1).
The introduction of 'hot' and 'cold' buttons reduced a questionnaire to a binary form: If something is rated as a hot topic it is interpreted as positive evidence of the user's interest; if something is rated as 'cold' it is interpreted as negative evidence. Nevertheless, it is a well known fact, that positive feedback is given only sparsely—for a good reason: if the user has found something of interest, he is not willing to co-operate further since his information need is satisfied. The same applies to negative feedback, except that the effect is even stronger due to the fact that a user is still more inclined to give feedback like 'that was good, more of this!' instead of 'that was uninteresting'.[6] In general, situations which provoke negative feedback should be avoided by any means since such situations repulse even willing users. This leads to a dilemma where we must not ask for negative feedback though it is desperately needed.
Aware of this situation one tries to get the maximum out of a situation by means of so called 'Forced–Feedback Back–Buttons'. Here, the functionality of a 'Back Button' which leads from a selected document back to a list of results is enhanced by dividing it into two buttons: 'That was a good one; more from the list!' and 'That was a bad one, let me take another look'. Sadly though, it has been shown that the use of such buttons is also highly unreliable; the choice of the button does not significantly differ from a random choice (see footnote 6).

2.1.4 Goals and caveats of user modeling

Designing adaptive intelligent systems, we are faced with cognitive processes from two different points of view: First, being adaptive is a cognitive task which means adapting to a certain situation.[7] Thus, user modeling can be interpreted as a testbed for evaluating hypotheses of cognitive processes. The second aspect works the other way around: Designing an adaptive systems we want to minimize the 'cognitive load' of the user. Thus, being adaptive means to adapt to a cognitive system. This implies the existence of a cognitive model of the user which explains the multitude of different tutoring systems, personal assistants, etc.

[6] Personal communication, Mathias Bauer (DFKI Saarbrücken) at the AAAI Spring Symposium on Adaptive User Interfaces, March 2000; Anthony Jameson, University of Saarbrücken, November 2000.

[7] Of course, 'being adaptive' means being adaptive on a higher knowledge level as, for example, adapting to a user's interest or abilities. Adaption without a model of what the system adapts to is not a cognitive process. Therefore, neither simple mechanical devices like thermostats, 'driver adaptive' automatic gear boxes or interactive but static help systems are addressed here.

Not everything that can be done, should be done. In [Miller, 2000] it is shown that Gricean conversational maxims apply to the user modeling problem as well. Imagine an abstract human–computer interaction situation:

- the system must provide the user with the right ('relevant') information.
- the system must provide the user with the right amount of information.

The first rule implies several non–trivial requirements. First, we need to know, what 'relevant' means. Relevance is both user and context dependent: Different user's are interested in different topics, and different situations ask for different aspects. As a consequence, we need as much information about the user and his environment as possible in order to be able to react in an adequate way.
Similarly, the second rule has major implications. While the concern of the first rule is similar to the question 'What to say' (in the sense of content selection), the second rule deals with the problem of 'how to say it' (in the sense of communicating only the right amount of relevant information in order to avoid 'over–information'). Again, this is both user and context dependent. Different users speak different languages or prefer different modes of presentation, while different situations may force the system to present information in different as well. As a consequence again, both user models and domain knowledge help to react adequately.
As an example, consider an adaptive tour guide. Knowing about the user's interests with respect to arts, the tour guide could offer different routes in an art museum. Similarly, the system could offer different explanations for the same exhibits to different users. Contextual information alters the system's actions again: A visitor who only has a little amount of time, might prefer a quick tour or shorter explanations.
There are numerous projects that are dealing with adaptive tour guides. For example, [Specht and Kobsa, 1999] mainly concentrates on adaptive routes and explanations, while [Petrelli et al., 1999] focuses on the contextual aspects.
[Berthold and Jameson, 1999] present a framework for an adaptive airport guide which tries to approximate the user's cognitive load by detecting stress patterns and thus gives adequate route descriptions.
It remains to be seen, what 'adequate' means: A system that statically models the user's interest and the context would not help or guide but rather force the user into a certain communication situation. This would inherently contradict the first aim of user modeling, namely to adapt to the user in order to make the interaction easier for him. Thus, as a rule of thumb, the system always has to be a subordinate communication partner.
There are much more adaptive systems around, which all have to meet the requirements of the Gricean maxims if they want to be accepted by the user. For example, there are adaptive help systems with pop–up tool tips, adaptive office systems which alter the pull–down menu structure with respect to the user's needs, adaptive route advisors which nav-

igate the driver with respect to his driving preferences ([Göker and Thompson, 2000a]) or artificial fighter pilot's associates which change their information displays with respect to the current tactical situation and tactical systems in operation centers ([Miller, 2000, R.Penner and Steinmetz, 2000]).
From these examples, it becomes clear that all adaptive systems must meet two requirements:

- The user is always in charge and rules the system—not vice versa.
- The effort required of the user to understand and manipulate the interface is much less than the effort that is saved by the adaptive interface (in other words: adaption to the adaptive interface is easier than using a rigid interface).

One of the most important commandments listed in [Miller, 2000] is essential for acceptance of user interfaces:

> Explain what you are doing.

In many cases, it is essential for the user to have the feeling of understanding the system. Thus, the user needs to have an idea about what the systems knows and what the systems assumes about the user. In this way, the user is able to understand *why* the system performed a certain step towards adaption. For today's complex software systems the understanding of such systems becomes more and more difficult. Accordingly, acceptance by every-day users is hard to achieve the more complex and the more powerful such systems are. It is argued, that the reason for this behavior is that humans build models of how such systems work internally. In other words, users construct models of the system. Acceptance then depends on whether the system behaves as the user expects on base of his model of the system. Systems that 'show off' and promise a real 'intelligent' behavior induce a model which the system is not able satisfy. In consequence, modern computer systems should 'behave' in way which makes a user to build an appropriate model of how the systems work. At this point, human computer interaction enters a level of meta–user modeling which can be compared to mutual knowledge in human communication situation ('I think, he knows, that I know, that he knows ...'): Different users with different background and different goals will build different models of the same system. Therefore, the appearance of the system needs to be adapted by the system itself in order to induce the right 'system modeling' process. In a second—but simultaneous—step, the system then adapts to the user by building a model of the user. Either way, explainability is a key feature to intelligent user adaptive systems.
However, this does not apply to all problems which deal with adaption. Low level adaption, such as adaption to motory skills (as in adaptive automatic gear boxes) or adaption in environments without any risks is not as sensitive to this commandment as high level or cognitive adaption where the system builds a complete model of the

user's interests, behavior, attitudes, opinions et cetera. Another way to circumvent the requirement of understandability is to adapt imperceptibly.
At this point, the risk of unwanted user manipulation emerges. Of course, users adapt to systems as well. Sometimes, user adaption is desirable—as long as it is evident and the user knows about it, but forcing the user against his own explicit will to adapt to a situation prescribed by a system is forbidden for ethical reasons.
Adaptive systems need to act in the background, but human–computer interaction and adaptive interfaces depend on feedback. Without feedback, the system has no chance to determine whether it made a mistake, what the user is interested in or whether a context change demands for another reaction. Obviously, user adaption without feedback is impossible. This seems to conflict with another crucial commandment:

> Don't bother the user.

Thus, we need to get going with as few feedback as possible, asking for additional information as seldomly as possible and presenting as few questionnaires as possible. In consequence, an adaptive system should be able to interpret the natural user behavior on all levels as user feedback—without the need for any further explicit feedback. The user model built upon this knowledge shall be inspectable and easy to understand for the user, the adaptive process itself should be scrutable, self–explaining and—if necessary—the user should be able to override system actions.

2.2 Machine learning

Knowledge and its acquisition plays a central role in intelligent behavior; thus learning can be regarded to as a key to intelligent behavior. On the other hand, acquisition of knowledge out of data sets is very important in today's computing machinery as well. The amount of data increases more and more—just imagine *data warehousing*—and with increasing knowledge we have to efficiently acquire and store this knowledge. In course of user modeling, we have huge datasets describing users — as, e.g. customer information in various business applications. A prototypical example for data mining is extracting knowledge about what credit may safely be granted from data about the customer's account (income), his address (social environment) and personal information (family, fix expenses, etc.). This is also an extreme example for user modeling, since rules extracted from data describing *all* customers can be used to derive information about a *special* customer. On the other hand, one might like to try to use information from server logs in order to induce rules that help to predict a user's next action. This can be used for pre–caching techniques, context sensitive help systems or recommender systems.
In this section we will first define the notion of a learning problem which is motivated by computational learning theory, see [Valiant, 1984]. After that we will go into detail and describe several methods of machine learning.

2.2.1 Machine learning problem

Valiant defined machine learning as a process, in which a machine acquires new data by any other means than explicit programming (c.f. [Valiant, 1984]). Of course, there are many other definitions all of which emphasize different aspects covered by the phenomenon of learning in general.
In general, a learning algorithm receives a sample with annotated classification and outputs a hypothesis which is a generalized description of the examples.[8⋆]

Definition 2.1 (Machine learning problem) *A learning algorithm* **A** *produces a hypothesis* $h \in \mathcal{L}_H$ *with respect to some background knowledge* Σ *(encoded in* $\mathcal{L}_\Sigma$*) on basis of examples (encoded in* $\mathcal{L}_E$*), see figure 1.2.*
A labeled sample of length m *is a sequence of examples together with a label which*

[8]It has been critizised, that definition 2.26 would be easier to understand if the following definition 2.1 would be motivated as an analogy to something that will be defined later on. However, it has been remarked by the same reviewer, that forward pointing cross–references should be strictly avoided. We therefore just kindly ask the reader to draw his attention to the following, really fundamental definition of a machine learning problem.

indicates whether the example is a member of the target concept h_t or not.[9]

$$\mathbf{s} = [\langle x_1, t_1\rangle, \langle x_2, t_2\rangle, ..., \langle x_m, t_m\rangle] \tag{2.1}$$

where $x_i \in \mathfrak{U}$ and $t_i = char(h_t)(x_i) \in \{0, 1\}$. That is, t_i is 1 iff the characteristic function for h_t is 1 on x_i. We abbreviate $char(h_t)$ by a target function $\mathbf{t}$.
In general, samples are generated by a function $S(m, \mathbf{t})$ which from $\mathfrak{U}$ chooses a sequence of m examples and labels with respect to $\mathbf{t}$. The choice of examples depends on a unknown probability distribution Δ on $\mathfrak{U}$. The set of all examples that agree with h_t is called the set of positive examples $\{x|\, \langle x, 1\rangle \in \mathbf{s}\} = E^+ \subseteq \mathfrak{U}$; those who disagree form the set of negative examples E^-.

Sometimes, labels t_i do *not* correspond to $\mathbf{t}$. This is called *noise* on the sample. In contrast to *supervised* learning tasks, samples can be unlabeled, too: In *unsupervised* learning scenarios, $\mathbf{t}$ is undefined on all examples: $\langle x_i, \uparrow\rangle$.
Learning algorithms can be described by two fundamental properties:

Definition 2.2 (Properties of A) **A** *is called a consistent learning algorithm, iff*

$$char(h)(x_i) = char(h_t)(x_i) = \mathbf{t}(x_i),\ \forall \langle x_i, \mathbf{t}(x_i)\rangle \in \mathbf{s} \tag{2.2}$$

i.e. the classification by h is equal to that of $\mathbf{t}$ on $\mathbf{s}$. When clear from context, we will omit the characteristic function and denote the above by $h(x_i) = \mathbf{t}(x_i)$.
A *is called a correct learning algorithm, iff*

$$char(h)(x_i) = char(h_t)(x_i) = \mathbf{t}(x_i),\ \forall x_i \in \mathfrak{U} \tag{2.3}$$

i.e. the classification by h is equal to that of $\mathbf{t}$ on the whole domain.

In other words, a correct learning algorithm is an algorithm which produces a function h that delivers a sufficiently precise approximation of $\mathbf{t}$ ($char(h) \approx \mathbf{t}$). Of course, the quality of a hypothesis depends on the sample: Samples usually are not deterministic, but they are not random either. In the PAC–learning setting it is assumed, that $S(m, \mathbf{t})$ draws examples from $\mathfrak{U}$ according to an *unknown* probability distribution Δ on $\mathfrak{U}$.[10] This assumption especially applies to the domain of learning user models: The sample will consist of user feedback which is unpredictable itself and based on a unknown user interest (see 2.3.2). In consequence, a Δ–weighted error sum tells us about the quality of the target approximation. Abstracting from Δ, one requires **A** to produce an error of at most ε, regardless to what Δ actually looks like.

[9] In general, we assume that $h_t \in \mathcal{L}_H$.
[10] The fact that Δ itself is unknown to **A** makes the PAC measure a quite pessimistic though realistic one.

2.2.2 Defining machine learning algorithms

Machine learning has several roots. As the key to artificial intelligence it is nearly as old as mankind. In our context, it is at least as old as modern computer science.
One of its founders, Alan M. Turing, already introduced concepts of machine learning along the Turing machine in his famous 1936 publication. During the second world war he devised a machine for the decryption of the Enigma (1939-1940), which was a mere ad-hoc solution from the viewpoint of his theoretical work in computer science: his ideas of a *learning* machine could not be realized in times, when decryption was a matter of survival. In 1947 and 1948 his work on artificial intelligence and neural nets already defined the width of current research efforts in machine learning: Symbolic and subsymbolic machine learning.
Given background knowledge Σ, a learning system (that is, an algorithm) $\mathbf{A}$ is a 'black box' containing meta knowledge, which on input examples produces hypotheses h according to its goal knowledge. For $h \in \mathcal{L}_H$ we especially want, that

$$\Sigma \cup \{h\} \mid\!\approx E^+ \qquad \text{and} \qquad \Sigma \cup \{h\} \not\mid\!\approx E^- \tag{2.4}$$

Once we determined the space of possible solutions we now have to seek for the best one. Often, we cannot find complete and correct hypotheses—either due to the instance space and our restricted languages or due to the task of keeping our description of the target concept as simple as possible. To 'navigate' in these cases, we shall use a ratio of covered negative and positive examples. Such ratios are defined according to the actual domain and are named *coverage* and *accuracy*:

Definition 2.3 (Coverage and accuracy) *The coverage of a hypothesis h is defined by the relative number of target evidence covered by h:*

$$\mathsf{cov}(h) = \frac{|\{e \in E^+ | \Sigma \cup \{h\} \mid\!\approx e\}|}{|E^+|} \tag{2.5}$$

The accuracy of a hypothesis h is defined by the number of target evidence covered by h in relation to all covered evidence:

$$\mathsf{acc}(h) = \frac{|\{e \in E^+ | \Sigma \cup \{h\} \mid\!\approx e\}|}{|\{e | \Sigma \cup \{h\} \mid\!\approx e\}|} \tag{2.6}$$

Note, that accuracy and coverage again can be computed with respect to (different) samples $\mathbf{s}$, to special test samples, and finally (if accessible) to whole $\mathfrak{U}$. In general, there are many different measures by which hypotheses and algorithms can be evaluated (see chapter 5 in [Mitchell, 1997]). The definition above, for example, does not take into account negative examples. Furthermore, it seems reasonable to measure the quality of

a hypothesis with respect to data which has not been used for training. In order to do so, one needs to define a test set T, $T \cap E = \emptyset$, of positive (T^+) and negative (T^-) examples. Taking into account both negative examples and a designated test sample, one can redefine accuracy by:

$$\mathsf{acc}(h) = \frac{|\{e \in T^+ | \Sigma \cup \{h\} \mid\!\approx e\}| + |\{e \in T^- | \Sigma \cup \{h\} \mid\!\not\approx e\}|}{|T|} \tag{2.7}$$

Taking into account negative evidence, the definition in equation 2.6 equals the definition of precision in the context of information retrieval (number of retrieved hits in relation to wanted hits).
Given such measures, we can search for optimal hypotheses. Carrying out such a search has several other requirements:

1. We must be able to identify our current position, i.e. we need kind of a metric on the space;
2. within guided search we need a measure of comparison, that is an ordering relation
3. and we finally need a search algorithm which most likely should have
4. several bounds as e.g. a depth search restriction, a language restriction or a breadth search restriction (i.e. a kind of 'focus')

We will examine the question of order relations in the section on generality and the question on boundaries in the section on bias later in detail.
In general, inductive reasoning as one central method of machine learning can be described as follows: If we have observed several instances of a rule

$$P(a_1) \Rightarrow Q(b_1) \quad P(a_2) \Rightarrow Q(b_2) \quad \ldots \quad P(a_n) \Rightarrow Q(b_n)$$

induction leads to a general law:

$$\forall x(P(x) \Rightarrow Q(y))$$

Restrictions imposed on the instantiations of the variables x and y then define the degree of inductive generalization: the less restrictions one applies, the more general is the inferred rule.
Given E^+ and E^- one seeks for a more general intensional description of a yet unknown new concept which satisfies formula 2.4. Due to the nature of induction and its application in order to yield a correct hypothesis in the sense of definition 2.3 one further requires that

$$|\{e|\Sigma \mid\!\approx e \text{ and } \mathbf{t}(e) = \mathtt{1}\}| < |\{e|\Sigma \cup \{h\} \mid\!\approx e \text{ and } \mathbf{t}(e) = \mathtt{1}\}| \tag{2.8}$$

$$|\{e|\Sigma \mid\!\not\approx e \text{ and } \mathbf{t}(e) = \mathtt{0}\}| > |\{e|\Sigma \cup \{h\} \mid\!\not\approx e \text{ and } \mathbf{t}(e) = \mathtt{0}\}| \tag{2.9}$$

Note, that we cannot argue by deductive closures on Σ since induction is not truth preserving. Therefore, induction is a kind of goal directed generalization which leads from a set of facts to a rule that describes the intensional concept.

2.2.2.1 Subsumption

If term or a proposition φ is *more general* than ψ, φ *subsumes* ψ (short $\varphi \precsim \psi$). There are different models of generality; which can roughly be discriminated by their *syntactic* or *semantic* definition. We shall apply the notion of *subsumption* to syntactic generality, while *generality* shall describe semantic generality.

Definition 2.4 (Generality) *Let φ, ψ be formulas. φ is called more general than ψ, if*

$$\varphi \precapprox \psi \text{ and } \psi \not\succapprox \varphi.$$

In other words, every model of φ is a model of ψ, and there is a model of ψ which is not a model of φ. If φ is more general than ψ we say that φ subsumes ψ, written $\varphi \precsim \psi$. This notion is used regardless to the formal system in which φ and ψ occur.

A special form of subsumption is θ–subsumption:

Definition 2.5 ($(\theta$–$)$ subsumption, $\preccurlyeq$) *A term t_1 is subsumes a term t_2 ($t_1 \preccurlyeq t_2$), iff there is a substitution θ such that*

$$t_1\theta = t_2.$$

For two literals of equal polarity L_1 and L_2, $L_1 \preccurlyeq L_2$ iff $L_1\theta = L_2$. A term or literal g is called a generalization over a set of terms or literals t_i, iff for every t_i, g subsumes t_i:

$$g \preccurlyeq \{t_1, ..., t_n\} \overset{\text{Def}}{\iff} \forall 1 \leq i \leq n : \; g \preccurlyeq t_i.$$

In this case we also say that g subsumes $\{t_1, ..., t_n\}$. The definition of θ–subsumption can easily be extended to clauses as well: Let there be two clauses C_1 and C_2. Then,

$$C_1 \preccurlyeq C_2 \overset{\text{Def}}{\iff} C_1\theta \subseteq C_2.$$

The substitution θ in the definition above is also known as the θ–difference of t_1 and t_2. Note, that $\varphi \preccurlyeq \psi$ implies that $\varphi \precsim \psi$, but not vice versa.

Definition 2.6 (Relative generality) *φ is more general than ψ relative to χ if:* [11]

$$\chi \wedge \varphi \precapprox \psi \text{ and } \chi \wedge \psi \not\succapprox \varphi.$$

In general, we shall abstract from the concrete type of generality and denote the generality relation by $\precsim$.

[11] The case that $\chi = \varphi$ leads to a contradiction ($\exists \mathfrak{M} : \mathfrak{M} \models (\varphi \wedge \psi) \rightarrow \mathfrak{M} \not\models \varphi$) and is therefore excluded.

2.2.2.2 Generalization

Give a (semi–) lattice induced by a relation $\precsim$, search along $\precsim$ yields successively *generalized* or *specialized* terms or propositions. In context of inductive machine learning, we seek for a hypothesis derived by generalizing on given data.[12]
The largest concept which is consistent with our background knowledge, and excludes all negative examples is called the *most general generalization* (mgg). The smallest such concept is called *least general generalization* (lgg).

Definition 2.7 (Least general generalization: lgg) *g is called a least general generalization of terms or literals t_1 and t_2, iff the following hold:*

1. *g subsumes both t_1 and t_2:*
   $g \precsim \{t_1, t_2\}$
2. *g is the most special generalization; i.e. any other generalization g' is more general than g:*
   $g \precsim \{t_1, t_2\}$ *and* $g' \precsim \{t_1, t_2\}$ *imply that* $g' \precsim g$

According to [Plotkin, 1970], $\precsim$ usually is interpreted as θ–subsumption, $\lesssim$.[13]

On the other hand, one can consider a subsumption relation of a completely different kind: namely based on logical derivability. So one can define a variant of the subsumption relation as

$$L \precsim L' \overset{\text{Def}}{\iff} \exists C, \theta : \Sigma \cup \{L\} \vdash_{\mathsf{SLD}} C \text{ and } L' = C\theta$$

or even

$$L \precsim L' \overset{\text{Def}}{\iff} \Sigma \cup \{L\} \approx L'.$$

It is clear, that such definitions are not applicable in sense of an efficient algorithmic definition.
To define the lgg of clauses relative to some background knowledge Σ we transform the 'context' into premises:

Definition 2.8 (Relative least generalization (of clauses): $r$lgg) *Let Σ be a conjunction of ground facts. Then,*

$$r\mathsf{lgg}(\{L_1, L_2\}) = \mathsf{lgg}(\{L_1 \leftarrow \Sigma, L_2 \leftarrow \Sigma\})$$

[12] Still, such a hypothesis can be found by both generalizing bottom up from the data as well as specializing top down from the maximum of the lattice. This is rather a question of implementation, not a question of principle.

[13] A set of well formed formulas Φ and the subsumption relation $\precsim$ defines a lattice as follows ($\varphi, \psi \in \Phi$): $\varphi \sqcup \psi = \mathsf{lgg}(\varphi, \psi)$ with $\varphi \precsim \psi$ iff $\mathsf{lgg}(\varphi, \psi) = \varphi$ and $\varphi \sqcap \psi = \mathsf{unif}(\varphi, \psi)$ with $\varphi \precsim \psi$ iff $\mathsf{unif}(\varphi, \psi) = \psi$. The result of meet and join operators are unique up to variable renaming. The top element of $\langle \Phi, \precsim \rangle$ is a free variable X, the bottom element is defined as a special error symbol $\bot = \mathsf{unif}(\varphi, \psi)$ for two not unifiable formulas φ and ψ.

One big disadvantage of using $r\mathsf{lgg}$ is the possibly huge amount of body literals—which gives rise to the need for a bias. This way, we have defined an ordering relation $\preccurlyeq$ on our search space the objects of which are clauses.

2.2.3 Concept learning

Decision trees. Given an information system $\mathcal{I}$ (a formal definition will be given in definition 2.21) the goal is to classify any entity $x \in \mathfrak{U}$ as $x \in E^+$ or $x \in E^-$ by successively asking for attribute values $f(x)$ with $f \in \mathfrak{F}$. This process is formalized by so–called decision trees. Decision trees can be constructed using a labeled sample by the following algorithm:

1. The root of the decision tree subsumes all entities $x \in \mathfrak{U}$.
   Choose one $f \in \mathfrak{F}$ and create successor nodes for each $f(x) \in \mathsf{cod}(f)$.
2. For all nodes:
   (a) If all entities subsumed by the current node either belong to E^+ or to E^-, label the node with p or n respectively.
   (b) Otherwise, choose another $f \in \mathfrak{F}$ which does not occur on the path from the current node back to the root and create successor nodes for each $f(x) \in \mathsf{cod}(f)$. If there is no attribute left, stop and report 'Unsuccessful attempt'.

This algorithm is the base for the learning algorithms which perform a top–down induction of induction trees (TDIDT, c.f. [Quinlan, 1986] and following; see [Quinlan, 1993] for an overview.). The problem of choice makes brute force approaches intractable. Furthermore, in most applications, the number of features does not suffice to allow a discrimination into all classes such that the tree cannot be correct at all. Moreover, it is often necessary to force inconsistency in the order to avoid overfitting (seemingly better consistency may imply loss of correctness).
A measure that guides the choice of alternative features is that of information gain. It is based on Shannon's entropy measure, [Shannon and Weaver, 1949]:

Definition 2.9 (Entropy, $\hbar$) *Let there be a sample $S \subseteq \mathfrak{U}$. Then, given n target classes c_i induced by a classification feature f, the entropy $\hbar$ on S is defined as*[14]

$$\hbar(S) = -\sum_{i=1}^{n} \mathsf{frq}(S, c_i) \cdot \log_2 \mathsf{frq}(S, c_i)$$

where $|\mathsf{cod}(f)| = n$ and

$$\mathsf{frq}(S, c_i) = \frac{|\{x \in S | x \text{ agrees with } c_i\}|}{|S|}$$

[14] We use $\hbar$ instead of H in order to avoid confusion with sets of hypotheses.

For a simple supervised learning task with a binary label function for positive and negative examples, we obtain:

$$\hbar(S) = -p \log_2 p - (1-p) \log_2 (1-p)$$

where p is the probability (or rather: frequency) of a positive piece of evidence. Given a set we now want to have a measure based on $\hbar$ for the information gain we expect when further partitioning the classes:

Definition 2.10 (Entropy relative to f)

$$\hbar_f(S) = \sum_{i=1}^{n} \frac{|\{x|f(x) = i\}|}{|S|} \cdot \hbar(\{x|f(x) = i\})$$

Thus, $\hbar_f$ is a f–frequency weighted sum of entropies of all blocks induced by f. Using $\hbar$ and $\hbar_f$, computing the gain comes down to the mere difference of given entropy and expected entropy:

Definition 2.11 (Gain, Gn) *The best feature f shall be the feature with best information gain:*

$$\mathsf{Gn}(f, S) = \hbar(S) - \hbar_f(S)$$

Further measures are introduced along the discussion of the role that bias plays in symbolic machine learning.

2.2.4 Rule induction

Instead of grouping objects into clusters of known or newly invented concepts, rule induction rather tries to find explanations for newly encountered entities by means of already acquired knowledge. In our case, we consider Horn clauses as $\mathcal{L}_E$,$\mathcal{L}_\Sigma$ and $\mathcal{L}_H$ (see definition 2.1). Thus, the goal is to induce a Prolog program which allows the proof of positive examples and fails for negative ones.

Basic terms. Each resolution step incorporates a unification of the involved complementary literals. This unification is carried out by applying an mgu. In parallel, the new hypotheses turn out to be clauses obtained by *inverse* substitutions.

Definition 2.12 (Inverse substitution) *Let t be a term and $t_1, ..., t_n$ subterms of t; $v_1, ..., v_n$ are variables. Let θ be a substitution acting on t. Then, θ^{-1} is called an inverse substitution, if:*

$$t\theta\theta^{-1} = t.$$

For $\theta = \{v_1/t_1, ..., v_n/t_n\}$ *we obtain*

$$\theta^{-1} = \left\{ \begin{array}{c} \left[t_1, \left\{p_{\langle 1,1\rangle}, ..., p_{\langle 1,m_1\rangle}\right\}\right]/v_1, \\ ..., \\ \left[t_n, \left\{p_{\langle n,1\rangle}, ..., p_{\langle n,m_n\rangle}\right\}\right]/v_n, \end{array} \right\}$$

with $p_{\langle i,j\rangle}$ *the positions of the subterms in* t*.*

Now, the problem is to define a calculus which provides a set of rules that by application of inverse substitutions and literal invention allows for the generation of new rules. The first idea of inverse resolution was published by [Muggleton and Buntine, 1988], who, among others, introduced the three operators of truncation, intra–construction and absorption.

Truncation. Any successful resolution proof ends in an empty clause which has been derived by a single literal resolution. Thus, given two unary clauses one might assume existence of a unary clause that subsumes both of them.

1. $\{\neg L_1\} \sqcap_{\mathsf{RES}} \{L\}\sigma_1 = \square$
2. $\{\neg L_2\} \sqcap_{\mathsf{RES}} \{L\}\sigma_2 = \square$

Adding the new hypothesis L' to the database with $L' \preccurlyeq L \preccurlyeq L_1 = \{C_1\}$ and $L' \preccurlyeq L \preccurlyeq L_2 = \{C_2\}$ we can remove C_1 and C_2 for they are now redundant clauses. All generalizations over L_1 and L_2 are possibly L' which means, that the original theory can be expanded by a reasonably large amount of new theorems—potentially too many, if we generalize too carelessly. A very strict constraint in choosing a generalization would be a restriction to least general generalizations; i.e. instead of choosing an arbitrary σ_i^{-1} we restrict ourselves to $L' = L = \mathsf{lgg}(\{L_1, L_2\})$. Another method is to carry out a best-first-search in the subsumption lattice. This top–down search then of course has to be guided by heuristics. We shall investigate the search guidance in the next chapter when speaking about inductive logic programming.

Intra–Construction. This operator can be illustrated nicely by the fanning and folding principles known from logic programming (see figure 2.2) plus a generalization component. The general rule of intra–construction can be described as follows:

$$\frac{C \leftarrow B, B_1 \quad C \leftarrow B, B_2}{H \leftarrow B_1 \quad C \leftarrow B, H \quad H \leftarrow B_2} \quad IC$$

That is, given a set of clauses with a set of common literals, the unique literals are used to define a new predicate. Note, that by way of generalization during defining H, the new rule set will be more general than the old one.

Theoretically, we consider two or more parallel resolution steps that are linked together by using one common but, as yet unknown, parent clause H:

$$R_1 = C_1 \sqcap_{\sqcap_{\mathsf{RES}}} H, R_2 = C_2 \sqcap_{\sqcap_{\mathsf{RES}}} H, \ldots, R_n = C_n \sqcap_{\sqcap_{\mathsf{RES}}} H \tag{2.10}$$

All resolvents R_i are elements of Σ. H will be the constructed clause and all the C_i will turn out to be specifications of the newly invented predicate we use in the premise (rule body) of the new clause H. Restricting to unary clauses $C_i = \{L_i\}$ we obtain:[15] $R_i = (H - \{\neg L\})\theta_{H_i}$. Again we compute a generalization $\widehat{R}$ over all R_i such that $\widehat{R} = (H - \{\neg L\})$ and by using the last equation we obtain $R_i = \widehat{R}\theta_{H_i}$; that means that $\theta_{H_i} = \widehat{R} -_\theta R_i$. Since R_i have the same term structure, we can find a generalization $\widehat{R} = H - \{\neg L\}$ over all R_i. By definition of the resolution principle, $\theta_{H_i}\theta_{C_i}$ is the mgu of L and L_i. Since any substitution can be decomposed, we find that $L\theta_{H_i} = L_i\theta_{C_i}$ wherein $\mathsf{vars}(\neg L) \subseteq \mathsf{dom}(\theta_{H_i})$. Therefore it seems to be quite suitable to generate a new n–ary predicate

$$L = p^n(v_1, ..., v_n) \tag{2.11}$$

with $\{v_1, ..., v_n\} = \bigcup \mathsf{dom}(\theta_{H_i})$. Of course, $L \not\leqslant L_i$ such that unifying L and L_i results in L_i again. But,—while $L\theta_{H_i} = L_i\theta_{C_i}$—we also get $\mathsf{unif}(L, L_i) = L_i\theta_{C_i}$. Since θ_{C_i} must be empty, this finally leads to the following equation: $C_i = \{L_i\} = \{L\}\theta_{H_i}$ which provides us with the definition of the newly invented predicate p.
Intra-construction without generalization, as described above, actually is—seen by itself without context of other operators—rather useless in machine learning. The result of this intuitive procedure is a clause which would be called an *overfit* hypotheses. It therefore is not a useful tool for theory enlargement. Furthermore, intra–construction poses the problem of choosing the right 'relevant' variables (as in equation (2.11)) which is computationally intractable without a proper bias.

Absorption. Theoretically, this operator enables one to compose and decompose recursive term structures which are very common in Prolog (every list is a recursive dotted–pair structure). In practice, decomposing complex recursive terms in Prolog is done by recursive rules. The absorption rule can be described by

$$\frac{C \leftarrow B_1, B_2 \quad H \leftarrow B_1}{C \leftarrow H, B_2 \quad H \leftarrow B_1} \quad Abs$$

using according inverse substitutions again. Absorption like operators are hardly used within ILP systems nowadays. This is also due to the huge computational effort that is

[15] This restriction is due to the theoretical foundations of intra–construction. Nevertheless, one can run a kind of enhanced intra–construction on non–unary clauses as well. This has been proposed in several publications, e.g. [Wirth, 1989] and [Müller, 1995].

Imagine a set of m clauses with the same rule head c:

$$\begin{aligned} c &\leftarrow p_{\langle 1,1\rangle} \wedge ... \wedge p_{\langle 1,n_1\rangle} \\ &\vdots \\ c &\leftarrow p_{\langle m,1\rangle} \wedge ... \wedge p_{\langle m,n_m\rangle} \end{aligned}$$

Furthermore, we assume a number of j literals $\tilde{p}$ to occur within each of this clauses; in other words: all clauses share a set of common body–literals.

$$\exists\, 1 \leq j \leq min\{n_1, ..., n_m\} : \tilde{p}_k\theta_{\langle k,i\rangle} = p_{\langle i,n_i\rangle} \text{ for } i = 1, ..., m \text{ and } k = 1, ..., j$$

Then, we 'cut out' all common literals and move the remaining literals $p'_{\langle i,k_i\rangle}$ into the definition of a new predicate q:

$$\begin{aligned} c &\leftarrow q \wedge \tilde{p}_1 \wedge ... \wedge \tilde{p}_j \\ q &\leftarrow p'_{\langle 1,1\rangle} \wedge ... \wedge p'_{\langle 1,k_1\rangle} \\ &\vdots \\ q &\leftarrow p'_{\langle m,1\rangle} \wedge ... \wedge p'_{\langle 1,k_m\rangle} \end{aligned}$$

Figure 2.2: Folding and unfolding

needed even when provided with a strong bias. Since the invention of recursive rules is not in focus of our work, we renounce a rather tedious sequence of formulas here and restrict ourselves to a short description of how absorption could work (instead of why it should work).
Muggleton and Buntine ([Muggleton and Buntine, 1988]) proposed the algorithm in figure 2.3 to 'search' for suitable substitutions which generate hypotheses for C. Of course, this algorithm is suitable as a guideline only. Analyzing the algorithm reveals, why one must not implement this algorithm in order to actually compute a new hypothesis in a considerable amount of time. Despite of this 'operational lack' absorption is a very powerful theoretical approach to cope with the invention of recursive predicates.

Example. To illucidate the power of the previously defined operators we now give an example which also shows, how those operators interact in order to induce a new hypothesis.
Given three facts

$$mammal(\texttt{mouse}). \quad mammal(\texttt{rat}). \quad mammal(\texttt{weasel}).$$

Imagine a resolution step $C = (C_1 - \{L_1\})\theta_1 \dot{\cup} (C_2 - \{L_2\})\theta_2$.
Transforming and replacing yields $((C - (C_1 - \{L_1\})\theta_1) \cup \{\neg L_1\}\theta_1)\theta_2^{-1}$.
Restricting C_1 to unary clauses results in $C_2 = (C \cup \{\neg L_1\}\theta_1)\theta_2^{-1}$.
The following algorithm computes C_2:

$\rightarrow$ C and $C_1 = \{L_1\}$ given

1. Collect all (sub–) terms t and store them together with their position p within the clauses in a set T_p
2. Now choose an arbitrary subset $T'_p \subseteq T_p$.
3. Construct a partition P on T'_p such that:
   (a) every block $B \in P$ is of the form:
   $B = \{(r, p_1), ..., (r, p_n)\} \cup \{(s, q_1), ..., (s, q_m)\}$
   (b) s subsumes r ($s \preccurlyeq r$)
   (c) All (r, p_i) are terms occurring in C
   (d) All (s, q_i) are terms occurring in $\{\neg L_1\}$
4. Compute θ_1 for all B in $\theta_1 = \bigcup(s -_\theta r)$
5. Then, $\theta_2^{-1} = \{(r, \{p_1, ..., p_n, q_1, ..., q_m\})/V|$ for all $B\}$, wherein all V are different variables which do not occur within $(C \cup \{\neg L_1\})$.

$\leftarrow$ $C_2 = (C \cup \{\neg L_1\}\theta_1)\theta_2^{-1}$

Figure 2.3: An intuitive algorithm for absorption

we would obtain *mammal*(X) by truncation. Of course, this generalization is not true, as

bird(`eagle`). *bird*(`falcon`). and *reptile*(`gecko`). *reptile*(`lizard`).

shows. By taking into account new observations

predator(`eagle`)`:-`*claws*(`eagle`). *predator*(`weasel`)`:-`*claws*(`weasel`).

application of intra–construction, leads to the invention of a new predicate — which we call *carnivore*:

predator(X)`:-`*claws*(X), *carnivore*(X)

The definition of the new predicate *carnivore* is the result of applying substitutions delivered by the generalization step (this corresponds to the fanning operator in logic programming).

carnivore(`eagle`). *carnivore*(`weasel`).

Finally, the system encounters a further new example which describes a new and unknown concept:

$$bird\_of\_prey(\texttt{eagle})$$

Since we also know the unary clause $\{predator(\texttt{eagle})\}$, absorption infers

$$\begin{array}{ccc} \{predator(\texttt{eagle})\} & & \{\neg predator(\texttt{eagle}), bird\_of\_prey(\texttt{eagle})\} \\ & \searrow \quad \nearrow & \\ & \{bird\_of\_prey(\texttt{eagle})\} & \end{array}$$

We repeat the process by choosing the newly acquired clause as resolvent of a new absorption step. The according parent clause is the fact $bird(\texttt{eagle})$. This time, the inverse substitution generalizes from `eagle` to a variable X (we omit positions, since the term occurs only once).

$$\begin{array}{ccc} \{bird(\texttt{eagle})\} & & \left\{ \begin{array}{c} \neg predator(X), \\ \neg bird(X), \\ bird\_of\_prey(X) \end{array} \right\} \\ & \searrow \quad \nearrow \sigma^{-1} = \{\texttt{eagle}/X\} & \\ & \left\{ \begin{array}{c} bird\_of\_prey(\texttt{eagle}), \\ \neg predator(\texttt{eagle}) \end{array} \right\} & \end{array}$$

Thus we have found, that birds of prey are predator birds:

```
bird_of_prey(X) :-
        predator(X),
        bird(X).
```

Inverse resolution is a suitable means for deriving meaningful hypotheses but the result crucially depends on

1. examples and their sequential ordering
2. choice of known clauses for applying operators
3. choice of inverse substitutions.

This gives rise to the question of how to define a suitable bias.

2.2.5 Bias

Following Michalski's paradigm of learning as search, one needs to restrict the search space. In terms of machine learning we need a *bias*.
In this section we will present several kinds of bias and illucidate the ideas of bias by concrete examples. Especially the refinement operators as used within Progol and the brief description of Foil are relevant for the theoretical background of this thesis.
When running through a lattice of hypotheses in general an exhaustive search is far too expensive in terms of computational complexity. Restricting the search space therefore means defining a bias which guides us through the hypothesis space or restricts it before carrying out a search.
In other words, to help $\mathbf{A}$, we need a bias β which helps to find a hypothesis more efficiently. There are different kinds of bias:

- *language* bias is used to restrict the search space extensionally, a priori. This is achieved by realizing β as a function which chooses an appropriate subset of $\mathcal{L}_H$. Here, of course, the definition of 'appropriate' is crucial: Proposing that $\mathbf{t} \in \mathcal{L}_H$, we also want that $\mathbf{t} \in \beta(\mathcal{L}_H)$. In order to define a sufficiently efficient search bias, this cannot be always guaranteed.

- *Search* bias seems to be of a different kind; since it does not give a prior boundary on where to look for a hypothesis but, rather, determines the behavior on *how* to search for it. Accordingly, β modifies the learning algorithm itself. For example, different search methods like depth–first or breadth first, greedy search or A* result in different behavior of $\mathbf{A}$. Here, the purpose is to yield an efficency gain with respect to the learning domain. It is clear, that different versions of $\mathbf{A}$ may deliver different solutions; sometimes they might be not able to find the best hypothesis at all, but the tradeoff between the gain of efficency and a loss of correctness up to a PAC bound may justify this.

- *Validation* bias is a kind of built–in tradeoff function which tells us to stop as soon as the ratio of time used and quality has reached a certain level. Speaking of validation bias or *stopping criterions* we cannot ensure correctness or consistency; i.e. sometimes it may be clear that $\mathbf{A}(\mathbf{s})$ can never become $\mathbf{t}$ (although $\mathbf{A}$ is still consistent for all $\mathbf{s}$).
  This is not as bad as one might expect—as long as in the PAC model $\mathsf{err}(\mathbf{A}(\mathbf{s})) < \varepsilon$ with the probability at least $1 - \delta$.[16]

[16]Therefore, it is not $\mathbf{t}$ which has to be preserved in $\beta(\mathcal{L}_H)$ but rather at least one 'sufficient close' hypothesis.

Language bias

h–easy ground models. The definition of *r*lgg gives rise to the question of how to achieve clauses as required input for an *r*lgg. The idea is simple, though expensive up to a certain degree determined by a parameter h:

Definition 2.13 (h–easy ground models, $\mathfrak{M}_{\Sigma|h}$) *Let there be a background theory Σ (a logic program Π) which contains intensional knowledge. We extract from Σ a set of ground clauses (i.e. its extension) by computing the closure under $\vdash_{SLD}$. The length i of the derivation—that is the depth of the SLD proof tree —is thereby restricted to h:*[17]

$$\mathfrak{M}_{\Sigma|h} = \{\varphi | \Sigma \vdash^{i}_{SLD} \varphi \text{ where } i \leq h\}$$

Using h–easy ground models, the notion of generality based on derivability $\vdash$ (as shown in definition 2.7) can be further restricted by limiting the maximum proof length.

Determinacy. ij–determinacy bias was introduced with the Golem system described in [Muggleton and Feng, 1990]. It deals with number of variables used within a literal and the 'linkage' of variables within a certain depth criterion i.
First of all, dealing wih h–easy ground models means dealing with extensional knowledge instead of intensional descriptions. A ground model only has a chance to be complete if there are no infinite paths and finite ground atoms. As one can see by these trivial considerations, the notion of h–easy ground models is already a severe restriction on $\mathcal{L}_H$.

Definition 2.14 (Determinacy) *An ordered Horn clause*

$$C \text{ :- } L_1, ..., L_n.$$

is called determinate (with respect to E and Σ) iff for all $i = 1, ..., n$ the following holds:

$$\forall\theta : (C\theta \in E^+\theta) \rightarrow (\exists\sigma_i : \{L_1, ..., L_{i-1}\}\theta\sigma_i \in \mathfrak{M}_{\Sigma|h})$$

A predicate is called determinate if its definition consists of determinate clauses only.

In other words: a literal is determinate if all of its variables that did not occur within previous literals are uniquely determined in their binding given the instantiations of all preceding literals. Then, a clause is determinate, if all of its literals are determinate. Note, that determinacy of clauses is always with respect to the underlying knowledge. So, for determinate terms (and literals) we have the following intuitive description: Whenever a new variable occurs in a literal, its binding is *completely determined* by all other bindings of variables occurring in the preceding body literals.

[17] This definition is a simplification of the more abstract idea of h–easiness: h is recursive, computable function, such that for each φ_i which is valid in our model we need at most $h(i)$ derivation steps to deduce φ_i in Σ. Here, we implicitly defined h to be constant.

Furthermore, we find a kind of 'linkage' degree which is expressed by the chain length of variable instantiations which link an arbitrary variable in the rule body to a variable in the rule head. This idea gives rise to the definition of *depth*:

Definition 2.15 ((Variable) depth, $\mathsf{dep}_{\mathsf{vars}}$) *Consider again an ordered Horn clause*

$$C\text{:-}L_1, ..., L_n.$$

where the head consists of an n–ary predicate $p \in \mathsf{Prd}^n$. We define

$$\mathsf{dep}_{\mathsf{vars}}(X) = \begin{cases} 0, \text{ for } X \in \mathsf{vars}(C). \\ \max(\{\mathsf{dep}(X') | X' \in \mathsf{vars}(C :- L_1, ..., L_{i-1})\}) + 1 \end{cases}$$

for a variable X occurring in L_i for the first time (i.e. not in any of the $L_1, ..., L_{i-1}$). Canonically we widen the definition of $\mathsf{dep}_{\mathsf{vars}}$ such that for any literal L, $\mathsf{dep}_{\mathsf{vars}}(L) = \max(\{\mathsf{dep}_{\mathsf{vars}}(X) | X \in \mathsf{vars}(L)\})$. By 'depth of a literal' we denote depth of its determinate variables.

This basic idea needs further specification with respect to multiple simultaneously occurring variables by the definition of the term *degree*: Instantiations of variables in the current literal under consideration can be seen as functions of earlier substitutions. Now, instantiations of L_i needs not to be unary; i.e. the value of multiple variables may depend on the earlier substitutions. Thus, the ratio of the number of determinate and yet unbound variables also allows a qualitative ordering on clauses. This can be achieved by means of the term *degree*:

Definition 2.16 (Degree, deg) *Let there be a Horn clause $C \leftarrow L_1, ..., L_n$ with an m–ary body literal*

$$L_i = p(t_1, ..., t_m) \text{ with } p \in \mathsf{Prd}^m$$

From those m terms, let there be k uninstantiated variables:[18]

$$L_i = p(X_1, ..., X_k, ..., t_m) \text{ with } p \in \mathsf{Prd}^m$$

The degree $\mathsf{deg}(L_i, X_j)$ of the variable X_j with $1 \leq j \leq k$ in L_i is defined by the number of those remaining variables, which also occur within the other literals (i.e. which do not occur only in L_i):

$$\mathsf{deg}(L_i, X_j) = |(\{X_1, ..., X_k\} \backslash \{X_j\}) \cap \mathsf{vars}(C, L_1, ..., L_n)|$$

The degree of a literal $\mathsf{deg}(L_i)$ is the maximum degree of all variables occurring in the literal and the degree of a clause is the maximum degree of all literals.

[18] Ordering of arguments is not significant here.

In other words, it is the number of variables that the following literals $L_{i+1}, ..., L_n$ have in common with those already considered.
Now we have defined two means for restricting $\mathcal{L}_H$: Degree and depth are two measures for which we can define dynamic biases for Horn clauses. Together with determinacy we obtain the following definition:

Definition 2.17 (ij–determinacy) *Let Π be a logic program representing Σ (i.e. by h–easy ground models). Examples E are represented by unary ground clauses. Then, C :- $L_1, ..., L_n$ is called ij–determinate, iff:*

1. *i is the maximum depth at which a determinate variable occurs.*
2. *j is the maximum degree of any variable occurring in $L_1, ..., L_n$.*

Predicate Schemata. Another idea to restrict the $\mathcal{L}_H$ is to introduce predicate schemata and modes. Schemata like

> The target is a 3–literal Horn clause which matches $p(\_, \_, X)$:- $Q(X, Y), p(\_, X, Y)$.

are used in Mobal. Modes like

> The target has a head literal, the arguments of which shall satisfy the I/O behavior determined by $p(+X, +Y, -Z)$

can be found in Progol as so–called mode declarations or in mFoil.
Mode declarations often also incorporate additional knowledge on possible underlying signatures; thus describing variable domains (sorts or *types*), predicate *declarations* or even further predicate characteristics such as symmetry or similar. A set of mode declarations **MD** defines a subset of a given language $\mathcal{L}_H$ which can be searched more efficiently.
Predicate schemata as described in [Kietz and Wrobel, 1992] are used within Mobal (so called *rule–models*). Rule models for n–ary clauses have the form $\Gamma = L \leftarrow L_1, ..., L_n$, where each L, L_i are literal schemata.

Decision tree pruning. Another example for language bias is decision tree pruning. In order to overcome the problem of overfitting, tree size is restricted. A simple pre–pruning method would be to demand a minimum information gain; i.e. $\mathsf{Gn}(f, S) \geq \vartheta$. This could be combined with tree depth, tree size or tree complexity measures in order to inhibit further node induction at a certain point. Nevertheless, pre–pruning bears the risk of myopic actions. Imagine a node S with four objects: a^+, A^-, b^-, B^+. It has an entropy of 1. Now, ordering with respect to letter or capitalization would deliver no gain at all—and thus would be inhibited by any threshold ϑ. Applying *both* features allows for an optimal solution which now has been cut off by the myopic pre–pruning process.

This argument of course also applies to nodes with more objects and more complex descriptions.
Much more reliable are post–pruning methods, where the decision trees are pruned after a completed tree induction process. Sadly, this is a more computationally expensive task: Finding the smallest correct decision tree is a $\mathcal{NP}$–hard problem (since exponential in $|\mathfrak{F}|$, as shown by Hyafil and Rivest, 1976; see [Garey and Johnson, 1979]).

Search bias

Different gain functions in decision tree induction. The gain function Gn (see definition 2.11, section 2.2.3) has a severe drawback of overestimating multi–valued attributes (see [Quinlan, 1993]). This leads to the following definition:

Definition 2.18 (Normalized gain, NGn) *Normalized gain is defined as the ratio of gain* Gn *and the number of possible values for the chosen feature f:*

$$\mathsf{NGn}(f, S) = \frac{\mathsf{Gn}(f, S)}{\log_2(\mathsf{cod}(f))}$$

This measure of course may now underestimate multivalued attributes: Imagine a key feature with n possible values, but only two of them actually given to objects in S: Then, Gn is penalized by $\log_2(n)$. In consequence, the feature would be rejected, although it might have been the optimal choice for obtaining a decision tree (imagine, that the two feature values induce the same pertition on the current node as $\mathbf{t}$). Thus, we need to choose f by its actual impact on S. In other words, we need to take into account the entropy with respect to f *and* $\mathbf{t}$, not only $\mathbf{t}$.

Definition 2.19 (Split information)

$$\mathsf{sI}(f, S) = -\sum_{\mathsf{cod}(f)} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

Learning by sI only is condemned to fail, since f does not necessarily express all relevant properties of $\mathbf{t}$. We therefore define a gain ratio GR of the gain Gn and split information:

Definition 2.20 (Gain ratio)

$$\mathsf{GR}(f, S) = \frac{\mathsf{Gn}(f, S)}{\mathsf{sI}(f, S)}$$

Entropy measure based gain functions are also used as a search bias in induction of rules. The Foil system uses a gain based heuristic in order to choose literals that shall be added to the antecedent of a rule.

The refinement operator in Progol. In the last paragraph we have described the top–down specialization method as implemented in Foil. The search was upper bounded by the initial single literal clause $C\theta_1$. The greedy information gain heuristic provides a strong but myopic bias. A lower bound for the search is not defined.
Another problem within the ILP approach is that of missing negative evidence.[19] The heuristic employed in Cigol (c.f. [Muggleton and Buntine, 1988]), which was a pretty straightforward implementation of the idea of ILP, was based on a compression measure. Compression was measured in terms of the number of function and predicate symbols used within a hypothetical logic program. Without any evidence this leads to the most general and most compressing hypothesis $p(X)$ for some unary target concept p. Of course, this hypothesis is not of any use.
The problem of choice of operators was solved by Golem (c.f. [Muggleton and Feng, 1990]) which used $r$lgg as single operator and ij–determinacy to guide the search through a subsumption lattice. But with application of $r$lgg another problem arises: Theoretically, the number of literals of $r\mathsf{lgg}(S(m,t))$ is upper bounded by $(|\Sigma|+1)^m$. Muggleton and Feng report clauses of tens of thousands literals while trying to learn arithmetic multiplication. Actually,

> [...] in the case of constructing a $r$lgg of 6 quicksort examples, with 15 ground elements of the model of `partition`, 49 instances of `append` and 16 instances of `quicksort`, the clause will have $15^6 + 49^6 + 16^6 + 1$ literals.
> [Muggleton and Feng, 1990]

which is approximately 14 billion literals.
Progol ([Muggleton, 1995]) tries to overcome the drawbacks that are based upon the paradigm of searching the subsumption lattice by the notion of implication and (inverse) entailment. It is obvious, that

$$\varphi \lesssim \psi \text{ implies } \varphi \rightarrow \psi \quad \textbf{but} \quad \varphi \rightarrow \psi \text{ does not imply } \varphi \lesssim \psi \tag{2.12}$$

The aforementioned ILP methods were based on the fact, that for a given ψ, a sequence of φ could be generated such that $\varphi \lesssim \psi$ which then implies $\varphi \rightarrow \psi$. In contrast to that, there is no constructive efficient method for a given ψ to generate a sequence of φ such that $\varphi \rightarrow \psi$.
Inverting entailment allows for a different view on the topic. Recalling equation (2.4), we can rewrite the requirements on h as follows:[20]

$$\Sigma \cup \{\bar{e}\} \gtrapprox \bar{h} \tag{2.13}$$

where the over-bar expresses negation of clauses. Now,

$$\Sigma \cup \{\bar{e}\} \gtrapprox \bar{s} \gtrapprox \bar{h} \tag{2.14}$$

[19] A situation that is very likely to occur in the context of user modeling tasks.
[20] Note, that $H = \{h\}$ and $E^+ = \{e\}$ consist of only one clause each.

where $\bar{s} = \bigwedge L_i$ for all (possibly infinitely many) ground literals L_i which satisfy $\Sigma \cup \{\bar{e}\} \not\approx L_i$. Reversing entailment again, this yields that $h \not\approx s$ for all h. A subset of H is the set of clauses h_i for which $h_i \not\preccurlyeq s$ holds.
The search space for h is lower bounded by s, and h shall be a clause which θ–subsumes s. But since s can be infinitely large, Progol uses both modes (see 2.2.5) and variable depth as a bias. An h must satisfy constraints on the usage of head and body literals (predicate names) as well as on variable instantiation and depth i (see definition 2.15). By considering only those s_i which have a depth of at most i and require a maximum number d of SLD resolution steps such that

$$\Sigma \cup \{s_i\} \cup \{\bar{e}\} \vdash^{d}_{\text{SLD}} \square, \tag{2.15}$$

s_i is the most specific hypothesis of the search lattice which is upper bounded by the empty clause. This lattice is searched using a refinement operator. Starting with a clause h from H, a sound refinement operator $\rho(h)$ delivers a subset $H' = \{h' \in H : h \not\preccurlyeq h'\}$. Progol's refinement operator ρ basically works as follows: Starting on an example e which is compatible with mode declarations for the head literal of the target clause, H' in the next step includes all those clauses which contain at most one more body literal (that is compatible with body mode declarations) and according sets of substitutions that allow to unify all h' with subsets of s_i. The number of added literals is upper bounded by the number of literals in s_i. Furthermore, within each step, the substitution θ' is required to be constructed from the last step θ by application of another substitution σ. Alternatively, a variable occurrence may be *splitted*. In this step, from one variable, two are made—while the restrictions on the variables are weakened or inherited to only one of the successor variables. For example, from

$$mammals([X|R]) \leftarrow mammal(X), mammals(R)$$

one could obtain

$$mammals([X|[Y|R]]) \leftarrow mammal(X), mammals(R)$$

by splitting R in the head into Y and R—where the restrictions on Y are lost.
The search is carried like an A* search with an Occam compression measure as a guiding heuristic.[21]
A detailed and self–contained description of ILP and inverse entailment can be found in [Muggleton, 1995].

[21] As a rough idea, the Occam compression of φ relative to ψ is $len(c(\psi)) - len(c(\varphi))$ for two clauses with $\varphi \not\approx \psi$ (and an encoding function c).

2.3 Machine learning for user modeling

The parallels of user modeling and machine learning have already been pointed out in figure 1.3. The two cycles of ML and UM can be unified in order to yield an abstract architecture of an ML4UM system. The result is shown in figure 2.4: The user query

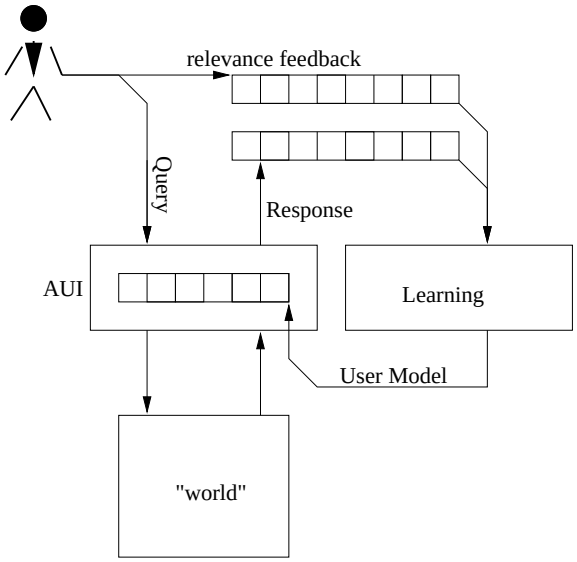


Figure 2.4: Machine learning and adaptive user interfaces

is sent to an adaptive user interface. By use of an internal user model, the query can be pre–processed and then submitted to the underlying information system (here, the 'world'). Responses are, again using the user model, post–processed and presented to the user. Then, feedback given in relation to the answers is used by an adaptive algorithm (here, a learning component) to adapt the user model in order to perform better next time.

This idea is not new, however. Iteratively refining an interface's capabilities by taking into account user feedback has been discussed in information retrieval pretty early already. A short overview on similar approaches to our problem from the information retrieval community is given in the next section.

2.3.1 Information retrieval*

When talking about 'information retrieval' (IR), many people think of database systems in the first place. Of course, there is much more to it as we shall see in this section. The underlying idea is to have an abstract system that holds all data and which is able to

answer certain questions about our data and as such, delivers information to the user. Such a system is called an *information system*.

Definition 2.21 (Information systems) *An information system* $\mathcal{I} = \langle \mathfrak{U}, \mathfrak{F} \rangle$ *consists of:*

1. *A set of objects* d*, also called Universe* $\mathfrak{U}$*,*[22⋆]
2. *A set of features or attributes* $\mathfrak{F}$*,*
3. *for each* $f \in \mathfrak{F}$ *a set* $\mathsf{cod}(f)$ *of possible values, and*
4. *an information function*

$$\mathsf{inf} : \mathfrak{U} \times \mathfrak{F} \to \bigcup_{f \in \mathfrak{F}} \mathsf{cod}(f)$$

which for each feature $f \in \mathfrak{F}$ *assigns a value* $v \in \mathsf{cod}(f)$ *to each object* $d \in \mathfrak{U}$*.*[23⋆]

In 1979, [van Rijsbergen, 1979], the information retrieval community already defined the term 'information retrieval' in opposition to 'data retrieval', as shown in table 2.1. Due

	Data Retrieval	Information Retrieval
Matching	Exact	Partial, Best
Inference	Deductive	Inductive
Model	Deterministic	Probabilistic
Query Language	Artificial	Natural
Query Specification	Complete	Incomplete
Items Wanted	Matching	Relevant

Table 2.1: Data Retrieval versus Information Retrieval

to the metaphor of dealing with information instead of data one can require to deliver *relevant* data instead of *matching* data. But what is the *best*? And how shall the system be able to improve? In [van Rijsbergen, 1979] it is stated that:

> [...] When the retrieval system is on-line, it is possible for the user to change his request during one search session in the light of a sample retrieval, thereby, it is hoped, improving the subsequent retrieval run. Such a procedure is referred to as *feedback*.

[22]Here, objects of the domain are web documents. Accordingly, we choose d as a variable name for elements of the domain: $d \in \mathfrak{U}$. In earlier versions, we used x in context of IR, o for abstract **o**bjects in ML4UM and d, when the domain of the ML4UM problem was restricted to a set of **d**ocuments. Context sensitive usage of different variables however decreased readability. We therefore chose d for all those variables and ask the attentative reader to abstract from the application domain whenever appropriate.

[23]Sometimes, we will refer to this function as a ternary relation by identifying $\mathsf{inf}(d, f) = v$ with $\langle d, f, v \rangle \in \mathsf{inf}$.

This leads to an architecture as depicted in figure 2.5 which is a slightly modified version of an abstract IR system architecture described in [van Rijsbergen, 1979].

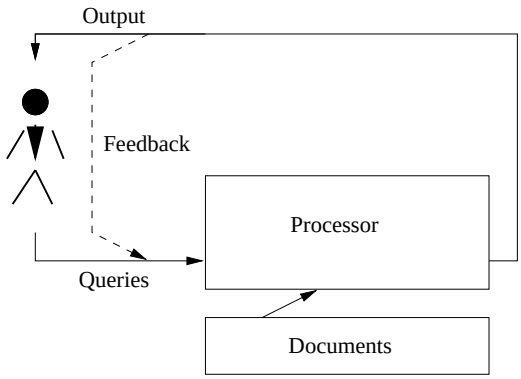


Figure 2.5: An (adaptive) Information Retrieval System

Quality measures: Recall and Precision. The core concepts in IR systems evaluation are *precision* and *recall* which relate to the quality measures of accuracy and coverage in machine learning. Let $\mathcal{L}_R$ be an information retrieval language. A query $q \in \mathcal{L}_R$ is an operation on a information function inf as, for example, in

$$\text{What is the set of solutions to: } f_i(d) = a \text{ or } (f_j(d) \neq a \text{ and } f_k(d) = b)?$$

The result is a set of objects which satisfy the underlying proposition:

$$\begin{aligned} r &= \{d \in \mathfrak{U} | \langle\langle d, f_i\rangle, a\rangle \in \mathsf{inf} \vee (\langle\langle d, f_j\rangle, a\rangle \notin \mathsf{inf} \wedge \langle\langle d, f_k\rangle, b\rangle \in \mathsf{inf})\} \\ &= \{d \in \mathfrak{U} | \mathsf{inf}(f_i) = a \vee (\mathsf{inf}(d, f_j) \neq a \wedge \mathsf{inf}(d, f_k) = b)\} \end{aligned}$$

Note, that for different q_1, q_2 the resulting operation may be equivalent and thus $r_1 = r_2$. It is clear, that the more powerful the operators are, the longer it takes for an IR system to deliver an answer. Operations on relational algebras, as e.g. products (i.e. `select ... from` x, y) and intersections (i.e. `select ... from` x, y `where` φ) soon turn out to be computationally expensive—and at least queries incorporating lattices in object oriented databases even turn out to be $\mathcal{NP}$ complete. Furthermore, modern IR systems provide the user with query languages that allow for string matching, near misses, and so on.
Now again let $q \in \mathcal{L}_R$ be a query; and let $r_{\mathbf{t}}$ to be the known target response. With queries incorporating weak or fuzzy quantifiers or operators or with noisy or incomplete databases (i.e. partial f_i), the actual outcome r might differ from $r_{\mathbf{t}}$. Recall is defined to be the ratio of actual responses to target responses whereas precision is the ratio of errors to target responses:

Definition 2.22 (Recall and Precision) *Let $q \in \mathcal{L}_{IR}$ be a query, and $r_{\mathbf{t}}$ a known target response (i.e. the set of relevant answers) for q.*
The recall is the number of relevant retrieved answers in relation to relevant answers:

$$\mathsf{rcl}_q(r) = \frac{|r \cap r_{\mathbf{t}}|}{|r_{\mathbf{t}}|}$$

The precision is the number of relevant ($r_{\mathbf{t}}$) retrieved answers (r) in relation to the number of retrieved answers:

$$\mathsf{prc}_q(r) = \frac{|r \cap r_{\mathbf{t}}|}{|r|}$$

Note, that for unknown $|r_{\mathbf{t}}|$ (uncountable or unknown $\mathfrak{U}$) neither recall nor precision can be evaluated.
A further measure is the fallout; it is defined as the number of irrelevant retrieved answers in relation to the number or irrelevant answers:

$$\mathsf{flt}_q(r) = \frac{|r \cap -r_{\mathbf{t}}|}{|-r_{\mathbf{t}}|}$$

Note that $|-r_{\mathbf{t}}|$ is unknown if $\mathfrak{U}$ or $r_{\mathbf{t}}$ is unknown.

Both recall and precision can be incorporated into an integrated quality measure called the *f–measure*, where a parameter β allows to emphasize either precision or recall:

Definition 2.23 (f–measure) *Let $q \in \mathcal{L}_{IR}$ be a query, and $r_{\mathbf{t}}$ a known target response for q.*
The f–measure with respect to q is defined to be

$$\mathsf{fms}(q) = \frac{(\beta^2 + 1)\,\mathsf{rcl}(q)\,\mathsf{prc}(q)}{\beta^2\,\mathsf{rcl}(q) + \mathsf{prc}(q)}$$

The smaller the value of β, the more recall is emphasized. Furthermore, one can relate precision, recall and fallout by the notion of *generality*.

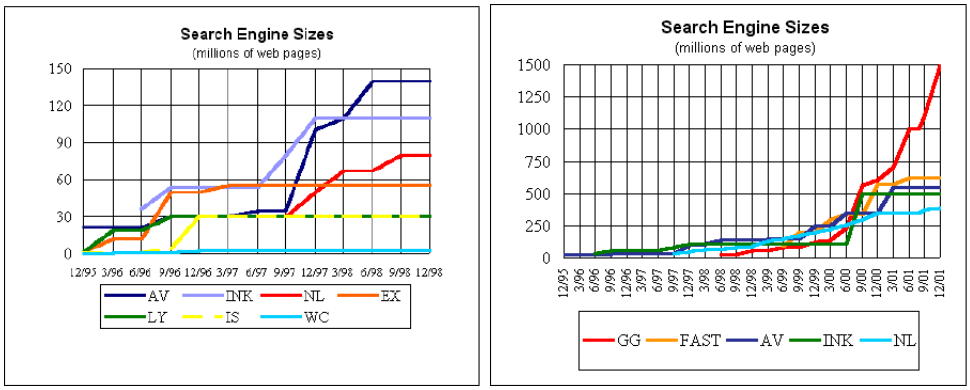
Definition 2.24 (Generality) *Let $q \in \mathcal{L}_{IR}$ be a query, and $r_{\mathbf{t}}$ a known target response for q. Furthermore, let $|\mathfrak{U}| = n$. Then, generality is the relative frequency of relative answers in $\mathfrak{U}$:*

$$\mathsf{gnr}_q(r_{\mathbf{t}}) = \frac{|r_{\mathbf{t}}|}{n}$$

Given $\mathsf{gnr}_q(r_{\mathbf{t}})$, $\mathsf{prc}_q(r)$, $\mathsf{rcl}_q(r)$ and $\mathsf{flt}_q(r)$, the following holds:

$$\mathsf{prc}_q(r) = \frac{\mathsf{rcl}_q(r) \cdot \mathsf{gnr}_q(r_{\mathbf{t}})}{\big(\mathsf{rcl}_q(r) \cdot \mathsf{gnr}_q(r_{\mathbf{t}})\big) + \mathsf{flt}_q(r)\,\big(1 - \mathsf{gnr}_q(r_{\mathbf{t}})\big)}$$

Information Retrieval and Adaptive User Interfaces. Obviously, the quality measurements from IR cannot be applied directly in the domain of user adaptive web search. First of all, no–one knows the entire web such that neither the number of documents nor the set of relevant documents actually can be determined. Nevertheless, any search engine will return a number of results $|r|$ which is much higher than an expected $|r_\mathbf{t}|$ (though most likely, $r_\mathbf{t} \not\subseteq r$). Though the biggest search engines still do not cover more than half the web, recall is not the problem in information retrieval from the web. In contrast to that, precision is much lower.
This is, where user satisfaction enters the game—neither precision or recall as a measure of theoretical system performance can be used to describe a system's quality. It is rather the relevance of a document with respect to a users interest that counts for a satisfied user. Thus, we do not need results of high precision or high recall with respect to a user query but rather a set of documents that are interesting to the user (and are not necessarily 'answers' to the query).
The problem of precise search with good coverage becomes evident, when looking at the size of the world wide web and the number of web pages indexed by search engines as shown in figure 2.6. When starting work on OySTER, the biggest search index was that



http://www.searchenginewatch.com

Figure 2.6: Search engine index sizes

of AltaVista, with approximately 150 million pages (left graph). Today (see right graph), Google's index is more than ten times bigger, covering 2 billion documents (including PDF documents and archived Usenet news which date back until 1981, Google claims to cover even more than 3 billion documents). The estimated number of web sites in the web has increased from 2,851,000 in 1998 to 8,745,000 today.
However, in the meantime it has become impossible to count the number of documents in the world wide web. For example, the proportion of dynamically generated documents increases, many documents are mirrored in several places and many documents can be retrieved by different addresses and site– or server aliases.

2.3.2 A brief formal view on the topic

In the previous section, we formally defined a learning problem. Thus it seems reasonable to extend the formal definition of a learning problem to a formal definition of a user modeling scenario.
Adaptive user interfaces can be interpreted as a kind of information systems as defined in the last section.[24] An information request is a set of partially instantiated triples $\langle d, f_i, v_i\rangle$ and the response is the set of all elements of inf which unify with them for all i. Usually a request asks for objects that satisfy certain properties, such that the request has the form $\{\langle \cdot, f_i, v_i\rangle\}_{i\in I}$ and the result performs an implicit projection on the first argument of inf.[25]

Definition 2.25 (Information interaction) *An information interaction scenario is a situation where a user u submits a query q to a user interface I_u with the intention of receiving a target response $r_{\mathbf{t}}$.*
The query q is translated into a set $I_u(q)$ of retrieval tasks $\mathsf{inf}(\cdot, f_i) = v_i$. The actual response r of I_u is the set of all relevant information objects $\bar{I}_u(\{\pi_1(\langle d_i, f_i, v_i\rangle)\}_{i\in J})$ for which $\mathsf{inf}(d_i, f_i) = v_i$ and $\langle d_i, f_i, v_i\rangle$ are elements of the information system's response to the query translation $\langle \cdot, f_i, v_i\rangle$.

Thus, the adaptive interface performs two user dependent actions. The actual response r is equal to the target response of I_u, only if $I_u(q)$ is a perfect translation of q and the relevance filter $\bar{I}_u$ is based upon a correct user model as described below. A 'perfect' translation and a 'correct' user model means, that there is no information loss in the interface, and thus, the delivered data corresponds to the target (in the sense of a target function in machine learning). We will illucidate the formal description of an information interaction again using our architecture of an abstract AUI, figure 2.7: The *Request* in figure 2.7 corresponds to the query q in definition 2.25. The interface I_u (here, **AUI**) translates q and yields *Query* corresponding to the retrival task. The systems *Response* is the set of solutions to the task and it is post–processed by the interface ($\bar{I}_u$) to deliver an *Answer* to the user.
The problem is to define the feedback loop in figure 2.7. It is obvious, that the unknown function from feedback labels to object of the domain defines a machine learning task. We will therefore define a user modeling problem in analogy to a machine learning problem. Together with definition 2.1, definition 2.25 can be used to formally describe the circle in figure 2.4.

[24] Adaptive help systems or systems which change their menu structure with respect to a user's preferences would not be regarded as information systems in the common sense, but actually those systems are information systems indeed: The help text needed is the information that is being searched for—and user interaction with the system is implicit feedback which reveals information need. A similar argument applies for adaptive menu structures and other adaptive user interface systems.

[25] Another form of request is to ask for property values for certain object; in other words, for a subset of $\mathfrak{U}$, (a subset of) $\mathfrak{F} \to \bigcup_{f\in\mathfrak{F}} \mathsf{cod}(f)$ is requested.

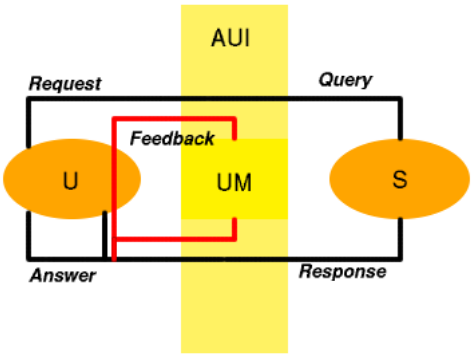


Figure 2.7: An abstract AUI

Definition 2.26 (User modeling problem) *An adaptive user interface I_u produces a user model $\mathbf{M}_u \in \mathcal{L}_\mathbf{M}$ for a user u with respect to some background knowledge Σ (encoded in $\mathcal{L}_\Sigma$) on the basis of user feedback.*
User feedback is a sequence of example response entities $d_i \in r$ together with a label l which indicates whether d is an object of the users interest $\mathfrak{I}_u$.[26]

$$\mathbf{f} = [\langle d_1, l_1 \rangle, \langle d_2, l_2 \rangle, ..., \langle d_m, l_m \rangle] \tag{2.16}$$

where $d_i \in r$ and $l_i = char(\mathfrak{I}_u)(d_i) \in \{0, 1\}$.[27] *We abbreviate $char(\mathfrak{I}_u)$ by a target function $\mathbf{i}_u$. In a theoretical framework, we assume that feedback given by a user is generated by a functional $F(m, \mathbf{i}_u)$ (where m is the sample size). The user model $\mathbf{M}_u$ models $\mathbf{i}_u$ such that $\mathbf{M}_u \models d$ if and only if $\mathbf{i}_u(d) = 1$.*

This definition allows us to define the special case of a learning problem for the user modeling component of an adaptive user interface:

Definition 2.27 (User model learning problem) *A user model learning algorithm $\mathbf{A}$ produces a hypothesis $\mathbf{M}_u \in \mathcal{L}_\mathbf{M}$ with respect to some background knowledge Σ (encoded in $\mathcal{L}_\Sigma$) on the basis of a sample $\boldsymbol{f}$ which consists of system responses and according user feedback.*

[26] In other words, $\mathfrak{I}_u$ is the real, unknown user interest which is a predicate on $\mathfrak{U}$.

[27] The attentative reader will have noticed, that the label was defined by the users interest ($l_i = char(\mathfrak{I}_u)(d_i)$). In footnote 26, however, we stated that this function unknown. Therefore, we assume that user feedback is realized by a function $F(m, \mathbf{i}_u)$, which according to the (unknown) target $\mathbf{i}_u$ delivers m examples whose labels 'agree' with $\mathbf{i}_u$. This (somehow cyclic) definition corresponds to the standard definition of learning problems (see definition 2.1) which will be incorporated in the following definition 2.27.

$\mathbf{M}_u$ is called correct, if it agrees with $\mathbf{i}_u$ on $\mathbf{f}$; it is called complete, if it agrees with $\mathbf{i}_u$ on $2^{\mathfrak{U}}$. Note, that $\mathbf{i}_u$ corresponds to the target function $\mathbf{t}$ in a achine learning problem and the sample $\mathbf{s}$ is realized by feedback $\mathbf{f}$ (see definition 2.1).

In other words: the optimal model $\mathbf{M}_u$ agrees with the actual user interest $\mathbf{i}_u$ by learning from examples $\mathbf{f}$ taken from feedback F in the same sense as a hypothesis h agrees with a target $\mathbf{t}$ learned from a sample $\mathbf{s}$ delivered by a sample function S. The final aim of adaptive user interfaces is to iteratively learn and refine user models $\mathbf{M}_u$ such that after repeated queries and responses r is equal to $r_{\mathbf{i}_u}$. In other words, $\mathbf{f} = r \times \{1\}$. At this point we have closed the circle of information flow as displayed in figure 2.4.

Goal of the thesis. The goal of this thesis is to show how one can learn $\mathbf{M}_u$ which approximates $\mathbf{i}_u$. The overall performance (measured by trying to satisfy $\mathbf{f} = r \times \{1\}$) of I_u, in our case the search engine OySTER, is far beyond the scope of this thesis.

The inherent problem of interpreting interactions.* There is, however, one inherent problem connected to the idea of interpreting observable interactions as feedback: It is the interpretation function itself. From the viewpoint of a designer of an AUI, the information flow in the system can be visualized as in figure 2.8. Observable interactions (as,

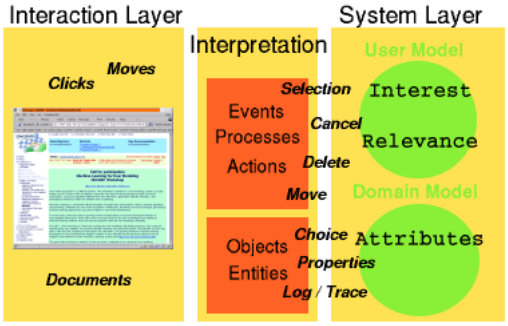


Figure 2.8: Interpreting interactions as feedback*

e.g., mouse clicks or requested documents) need to be interpreted as events or objects related to actions. All actions, events or objects are usually grouped into higher–level descriptions which correspond to partial plans in interaction sequences (as, e.g., selections, requests, deletions). Finally, after interpreting observed interactions, we evaluate all the data in a system layer, where information regarding the domain and the user model is processed. At first glance, this process seems to be rather easy to describe and

realize. But there is one simple example which shows the crucial problem in machine learning for user modeling: Observing a click on 'Cancel' in a web browser is interpreted as aborting the download of a document. But how shall one interpret this action with respect to the underlying user model? Is it disinterest in the downloading document or is a low bandwidth the real reason for cancelling the download?

2.3.3 Examples

As this work focuses on information retrieval from the world wide web we will put emphasis on recent research in this field only. Thus we give a more detailed overview of adaptive recommender and information systems and take only a brief look in more or less related but highly interesting application domains for adaptive user interfaces.

2.3.3.1 Recommender systems

Recommender systems are designed to recommend information to the user that might be helpful for him. Nevertheless, those systems do not necessarily have to be systems which are based on individual user models. As an extreme example one might consider Amazon which recommends books or other media to customers based on collaborative modeling: '*...people who have bought this item, also bought ...*'.
A popular example for a web page recommender system is Syskill&Webert, which was developed at UCI; see [Pazzani et al., 1996, S.Gaffney et al., 1996]. It consists of a meta search engine which offers the opportunity to give explicit feedback for each result. The feedback is used to build an individual user model that contains sets of boolean key word vectors which consist of n 'most informative words'. Using trained Bayesian classifiers, web documents (i.e. links on currently displayed pages) are recommended with respect to the user model.
The WebWatcher (developed at the CMU, [Armstrong et al., 1995, Joachims et al., 1996] and [Joachims et al., 1997]) is a more unobtrusive approach to the same domain. Without needing explicit feedback, links are recommended during a web browsing session. The browsing behavior is recorded in order to build a user model, again consisting of word vectors that have been derived using TFIDF. Collaborative user modeling is performed using reinforcement learning. As a successor, the Personal WebWatcher is a system for web page recommendation based upon individual user modeling techniques, [Mladenic, 1998].[28]
A recommender system for news is the News–Dude, developed at the UCI and now being marketed, [Billsus and Pazzani, 1999]. The NewsDude focuses especially on the difference between long–term and short–term interests: '*... a user's information need*

[28]For further literature consult `http://www.cs.cmu.edu/~webwatcher/` for the WebWatcher and `http://www.cs.cmu.edu/afs/cs/project/theo-4/text-learning/www/pww/` for the Personal WebWatcher.

changes as a direct result of interaction with information'. The system reads or displays news to the user and awaits explicit feedback, including two further types of positive feedback ('interesting, but I already know', formerly often covered by negative feedback, and 'tell me more!'). Documents are mapped onto TfIdf–based representations and are chosen by a nearest–neighbor algorithm for short–term interests and Bayesian classifiers for long–term interests (again, based on boolean word vectors).
Taking into account different presentation methods such as pagers, mobile phones, office PCs and home PCs, one needs to decide which news is interesting with respect to the current location of the user. This question is the topic of the InformationValet project, Rutgers, described in [Macskassy et al., 2000].
A very nice and comprehensive overview of web based recommendation systems is given in [Pretschner and Gauch, 1999].

Search or information systems. As already pointed out in the last section, Syskill & Webert is a user adaptive search engine. Nowadays, nearly any search engine allows for a certain amount of 'personalization', which is mostly restricted to source selection or presentation methods. Nevertheless, one can imagine the outcome of a mix between AltaVista and Amazon. On the other hand, it must be stated, that individual user modeling for a meta search engine for the whole world wide web is beyond all computational means. The number of a billion web pages and over 200 million web users simply outstrips the computational power of the whole planet.
Thus, adaptive search services currently are solution for parts of the web only. One approach for information retrieval of web documents with respect to an classification ontology is the Ontobroker, [Fensel et al., 1998]. The idea of the Ontobroker is that web documents are attached additional information which describes type and content of the document. This information has to be added manually but allows for a very precise, Kl-One like query[29] with precise results.
Actually, the fusion of OntoBroker and the PersonalWebWatcher together with the technique also used in the WebKB (see [Craven et al., 1998a, Craven et al., 1998b]) project defines the idea behind OySTER pretty well: Let there be a conceptual description of web pages obtained by suitable classifiers. Then, a document is interesting for the user, if the document's description fits into his user model. The user model is refined using explicit feedback from the user.

2.3.3.2 Route and place advisors

Another type of information recommender systems suggest to users is information in the sense of information about entities or objects.

[29] Actually, L_{Lilog}.

Navigation systems for cars are becoming more and more ubiquitous. For a given goal the route is computed independently of the driver. Daimler–Chrysler and the CSLI Stanford developed an adaptive RouteAdvisor, [Langley, 1999], which recommends routes and alternatives with respect to the driver's driving preferences. The trick is to offer several routes which are described by attributes like estimated time driven on a segment, intersections per segment, left–, right– and u–turns and street type (such as road, highway, freeway). The driver's choice is interpreted as relation of preference between the vectors describing those routes. The learning algorithm is a simplified variant of support vector machines (c.f. [Fiechter and Rogers, 2000])[30]: A so–called subjective function $f(r) = w \cdot r$ models the preference relation by mapping 'better' routes r described by the feature vectors to smaller values than those describing 'bad' routes. In other words, w is a user model describing the user's driving behavior by means of the features described above.
Once a route has been planned it seems desirable to be able to recommend restaurants as well. Again, choice of restaurants depends on user's preferences and the current context consisting of location, time of day, day of week, history etc. The interactive PlaceAdvisor tries to recommend suitable restaurants using spoken natural language dialogs (see the publications [Thompson and Göker, 2000, Göker and Thompson, 2000b]).

2.3.3.3 Personal assistants

Starting with the Advisor recommender systems, there are many other personal assistants which are enhanced by user modeling techniques.
First of all, one of the first systems in this context were that of Patti Maes; NewT, a user adaptive news filtering and presentation system, [Maes and Sheth, 1993] which already learned from examples, might be the best known of the early systems. Today, the community of all personal agents has grown to an immense number that cannot be surveyed any more.
Personal agents do not have to be bound to the desktop: Within the HIPS and Hippie (c.f. [Specht and Kobsa, 1999, Oppermann and Specht, 1999]) projects, handhelds are used to guide a user through museum exhibitions. The guide can be adaptive but is not a fully functional adaptive guide yet. More emphasis is put on the adaptive behavior in the READY project, which simulates an adaptive guide on an airport. This scenario, as described in [Berthold and Jameson, 1999], deals with all trapdoors and tripwires that were discussed in section 5.

[30] Previous versions used a perceptron for learning the target function, [Rogers et al., 1999].

II

INDUCING CONCEPTUAL USER MODELS

In this part, we describe our approach of conceptual user modeling.

In a first chapter, we give a 'virtual tour' through the OySTER system which illucidates the idea behind conceptual user models through a user–centered view on our approach.

In the following chapter, we define the core concepts behind conceptual user modeling which is based on our formalization of user modeling problems from the first part. We then focus on the user model induction task.

The last chapter of part two presents results from our evaluation and gives a short description of possible improvements and how the obtained user models can be used for document filtering.

Chapter 3

THE META SEARCH ENGINE OYSTER

From the user's viewpoint, OySTER is a meta search engine. One can submit a search query which is forwarded to other search services and results are ranked and integrated into one single result page. In the first part of this chapter we give a brief overview about search engines on the web. After that, we describe the very high idea of how OySTER, as a user adaptive meta search engine, helps to overcome the drawbacks of a standard search engine. The following section provides a short tour through the current meta search system from a user's viewpoint and the fourth section discusses enhancements that have to be made so that the current OySTER system actually meets the requirements as described in the second section. In the last section of this chapter, we describe the internals of the system.

3.1 Searching the web

In 1990, Tim Berners–Lee invented the world wide web during his work at CERN. One year later the first web browser, Mosaic, was released. In 1994, the web traffic on the first web server ever, was already a thousand times higher than at the beginning. In the same year, David Filo and Jerry Yang started creating a manually edited web index for interesting web sites from their own bookmarks. Filo and Yang called themselves 'yahoos' since they dared to start such a hopeless project. At this time, the number of sites (that means hosts) in the WWW was about 2,700 with 13.5% being commercial sites.
Only two years later, in 1996, the number of sites had reached the magic 100,000 with the proportion of fifty percent being commercial sites.
In November 2000, a company named Yahoo! employs 150 editors which maintain over one million links to web pages. The number of sites has reached a million with the majority of providers being commercial sites. At the same time, the largest search engine

for the world wide web, Google, has indexed 1,247,000,000 web pages.[1] Without search services or catalogs, it is impossible to find relevant information on the world wide web.

3.1.1 Search engines

Search engines do not search the web online. The web is scanned and documents are indexed and stored in a local database. On these databases the search is carried out. This suggests a generic architecture as shown in figure 3.1.

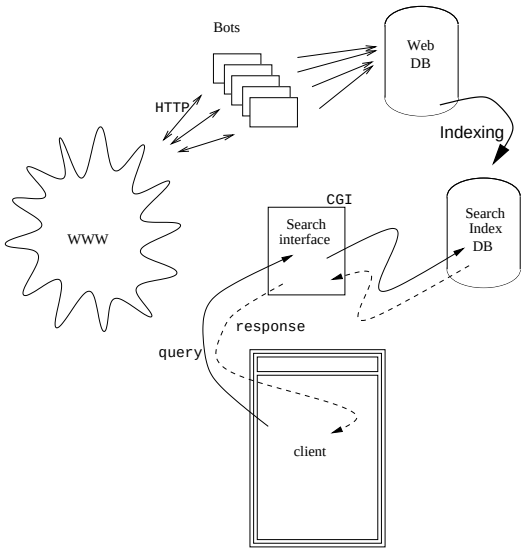


Figure 3.1: A generic search engine

Indexing the Www. Given all the data a robot has collected, the Www database needs to be indexed in order to allow for an efficient search. For each document found, several data can be used for indexing:

1. Word occurrence, word frequency, n-grams, phrases
2. Domain and language
3. Text Category

The third property seems to be used infrequently (an exception is NorthernLight), whereas the other two are regarded as almost obligatory. Sadly, none of the search engines reveal their secret of indexing—since it is their key to precision.

[1] Statistical data taken from `http://www.searchenginewatch.com/` and `http://www.mit.edu/people/mkgray/net/`.

Word occurrence and frequency are mostly determined using derivatives of the TfIdf measure on stemmed snippets of the document. Html tags are sometimes used to give a better 'significance score' for keywords. The same method applies to phrases or n–grams. Another add-on are `META` tags. They provide a list of keywords, an abstract and authoring information for some documents.[2⋆] The so-derived keywords are used as keys in the index database and the current document Url is added or updated.
This mechanism needs not be automated; one of the most popular search engines which also provides a fully browsable theme index does all the indexing manually: Yahoo.
It is worth mentioning that one of the most recent and most popular search engines, Google, uses a completely different method for indexing and computing a document's relevance: Instead of viewing a document as an entity of its own, cross references from other documents are taken into account. All matching pages are collected in a set r. To each document $d_i \in r$ a value is attached which describes the number of links from other documents $d_j \in r$ to d_i. Thus, pages which are referred to often by other documents that match the search query receive a higher score.

Searching the index.⋆ When a client u seeks information, a query q submitted by u to a search engine involves key words or key phrases which the user thinks precisely describe his interests. Query formulation is a crucial quality argument; since most common search engines are not user adaptive, the user has to have a certain idea about the expected result and how the desired result can be achieved by chosing the words he thinks will yield the best match based on the search engine's internal indexing methods. Only few users spend much cognitive effort on this 'guess'; a search query usually consists of two or three words only (see below). An alternative is the search service provided by AskJeeves which accepts complete natural language queries.[3⋆] Nevertheless, such queries are neither parsed nor refined by synonyms or user dependent information. AskJeeves rather performs a more elaborate stop word filtering which also includes query phrases like *Where can I find....* Google uses the same technique to eliminate common words and furthermore tries to recognize phrases or other structural properties of queries. As an example, search strings that resemble addresses are forwarded to the Yahoo map service.
Search queries on the intranet of the University of California, Berkely, collected over a period of four months in 1997 revealed an average query length of 1.8 words. Considering search queries for the whole world wide web, the top ten search queries posted to Google have an average length of 1.4 words only (data as of October 2001). Counting fix phrases such as names or products like *Microsoft Works* as single words, the average drops down

[2]Using data from `META` tags is prone to noise, since many web authors try to cheat web indexes in order to gain more web traffic on their web pages. As a consequence, many web pages contain faked content descriptions. This phenomenon wich is only likely to be observed in rather dubious communities is referred to as *pollution*.

[3]as of December, 2000.

to 1.1 words per query.[4⋆]
A query q is processed by the interface (like removal of stop words) and the result $I_u(q)$ is handed over to a database retrieval system. The corresponding result $r = \bar{I}_u(\pi_1(\langle d, f, v\rangle_i))$ of the database query is then transformed back into the search query result page.
Some search engines allow for a minimal configuration by defining a profile that determines presentation or search scope. In general, modern search engines like AltaVista, NorthernLight or Google deliver sufficiently good results provided there is a good query. Subjective precision of the results can be dramatically increased using meta search engines like the MetaCrawler since multiple occurrences of search results are ranked as better (more precise) results (the MetaCrawler is described in [Selberg and Etzioni, 1995]).

A short note on the revised version.⋆ The last paragraph reflects the intention behind building a user adaptive search engine that increases search results as it was in 1998 when the work on the OySTER prototype was started. Also in 1998, Google, a former research project at Stanford university, became a public search service. It soon stripped out the then–market leaders AltaVista and Inktomi (see figure 2.6). Meanwhile, with recent advances in web crawling, indexing techniques and a massive increase of computational power[5⋆], 'traditional', index based search services like Google offer an incredible coverage of web documents, a stunning precision and even more features like, e.g., document text classification with respect to the largest ontology (the dmoz–project) at an impressive speed.

3.1.2 The very high idea behind OySTER

First of all, we tried to increase subjective precision by the simple method of meta search. A meta search engine does not create its own index of the Www but forwards the search request to other search engines and then aggregates results. The idea is as follows: Utilizing several search services s_i, we receive responses r_{s_i} which build the union $r = \bigcup_i r_{s_i}$. Now, since $r \supset r_{s_i}$ and $|r| \gg |r_{\mathbf{i}_u} \cap r|$, we gain a better recall but we need to break down r in order to increase precision. Increasing precision by means of user adaption is the main goal of OySTER.

[4]It is a sobering fact, that observations like these are not always taken into account when evaluating new methods for user adaptive interfaces. In one article for which we deliberately do not want to give a reference, it is stated that: *[...] The average length of queries was 2.30 words. Users averaged 1.31 distinct informational goals per day, and performed 3.27 queries per goal.* In this article, a method is presented that predicts interactive steps (refinement strategies) on an ontology of 15 categories using Bayesian networks. An evaluation—no matter what kind of evaluation model or assumptions—is not carried out at all.

[5]Google uses the world's largest Linux cluster consisting of 10,000 servers for indexing web documents, archived Usenet news and PDF documents.

The second idea was to make the search process user adaptive; i.e. I has to be parameterized by the user u. Furthermore, the idea was to overcome the lack of opaque user models, where a description of $\mathbf{i}_u$ consists of a large word vector. We have already stated, that in the domain of information retrieval from the WWW several quality measures from all contributing disciplines are often mixed: user satisfaction is measured by acc, precision of $\mathbf{A}$ is approximated by some rcl, and so on. We wanted to draw a distinctive borderline between the document, its components (i.e. words) and its content (that is, its meaning). Of course, when modeling a user's interest or a user's information need, this shall not be modeled by words but rather by concepts. Thus, the borderline is realized by document classification with respect to an underlying ontology of document content categories. Additionally, we crafted a concept hierarchy of document types which are interpreted as sorts over categories: When searching for an individual homepage of a researcher who works on machine learning and user modeling, a query *q=homepage user modeling machine learning* is not adequate: the word *homepage* describes the type of desired document, but not its content.
Now, given a set of appropriate classifiers, we do not take words into account any more but instead work on conceptual descriptions of categories. In other words, each document is described by a vector of conceptual descriptions, where each component is attached a confidence value. It is a straightforward idea, to represent a user model in terms of such categories as well: The user model $\mathbf{M}_u$ which shall approximate $\mathbf{i}_u$, is a conceptual expression that is lucid and understandable. Basically, one might think of the conceptual hierarchies as trees, where the user model consists of tagged subtrees. As soon as a new document is classified into such subtrees, it is assumed that the document is interesting. This approach also enabled us to clearly define a learning problem for conceptual descriptions with a given feedback sample $\mathbf{f}$ derived from feedback $F(m, \mathbf{i}_u)$, where:

- the characteristic function of the user's interest, $\mathbf{i}_u$ corresponds to the learning target $\mathbf{t}$,
- the feedback sample $\mathbf{f}$ to a machine learning sample $\mathbf{s}$ and
- the feedback function together with a sample enlargement function Γ corresponds to the sampling function $S(m, \mathbf{t})$.

3.2 Searching the web with OySTER

In this section we describe the realization of the idea presented in the last paragraph. We first give an illustrative tour from the user's viewpoint and then describe the internals of the system.

3.2.1 Preliminary remarks

Ordinary search engines present results to the user which are ordered by 'statistical relevance'. This relevance measure is not visible to the user. It is not visible to the public either, since its quality pays off in terms of user satisfaction and thus, turnover. One can state though, that those measures are based on phrase occurences.
The OySTER approach is different. We will illucidate the idea behind our approach by a simple metaphor in the next section.

3.2.1.1 Looking for a book in a library

Imagine, you were looking for a book dealing with a certain topic in a library, but you do not know whether there exists such a book (or a whole set of them) or not. Accordingly, you do not have knowledge about the title, the author or other bibliographic information. The usual procedure in such a case is to ask an expert. In other words, one requests a book about a certain topic from a librarian.
In this metaphor, librarians who work like current index based search engines would have read every book in the library. They have collected all statistically relevant phrases they have encountered while reading every single book, and for each book they know, which phrases occur at which frequency. In order to satisfy your search request, they quickly write a list of book titles ordered by the frequency of occurence of the words that were contained in your query. This list will be pretty long, and the results will be pretty bad for 'vague' queries (e.g. if they contain ambigous words).
Librarians who work like OySTER work completely different. For each book, they ask an expert to classify the book by its content with respect to a taxonomy. The taxonomy is realized by a huge archive where books are stored in shelves ordered by their content. Books with multiple topics or books that cannot be classified into one single place are copied—and each copy is printed in different shades of grey representing the relevance of the shelf for the book. When you request a book by the same question you asked the other librarian, interesting things happen:
First, the librarian asks ten other librarians (Al Tavista, E. Xi-Te, G. O'Ogle, and so on) in different libraries who work the way we have decribed above. The librarian then collects all the lists, compares them and strikes out all multiple occurences and reorders the list by aggregating the preorders. He immediatly gives a copy of the list to you, and asks you to wait until he gets more detailed information.

Then, for each book on the list, the librarian asks the classification expert, what an appropriate location for the book in this library would be. The classifying expert checks whether he already knows the book and, if so, informs the librarian. If the expert has never seen the book before, he quickly reads it, makes copies and stores them in the shelves and reports the location back to the librarian.
The librarian now knows the location for each book, although he does not know the book itself, nor what the book actually is about. You, as a customer, have an understanding of the taxonomy of books. Therefore, the librarian's information about a book location means to you information about the content of that book.
Nevertheless, it might be the case, that you find same books in some shelves that are not interesting with respect to your initial request. Since you are a friendly customer, you go back to the librarian after you have flipped through a few books from some shelves and tell the librarian which books you liked and which not.
Now, the librarian becomes active (so far he has delegated all work to others: classifying experts and other librarians): He tries to find out, what is interesting for you. He does so, by taking a map of the library and marking all shelves with a green dot for each book you liked from this helf. Books you did not like appear as red dots on his map.
Next time you visit the library, the librarian has a certain impression of your interest: he knows your preference of shelves—which to you means your interest. Again, you request a book from the librarian. The librarian proceeds exactly as the last time—until he receives the classification results from his colleague. Then, he presents those books to you, which belong to shelves which have the most green dots and the least red dots. The order of books is determined by the librarian's idea of whether the book classifies into a shelf (or near to a shelf) which best fits his impression of the best 'green shelf'. To you, it means that the librarian has tried to learn a description of your interest in terms of the taxonomy.
Provided the taxonomy is broad enough to cover all books and fine enough to distinguish between different, but similar books and provided that you gave enough feedback, the classifiers work reliable and the librarian is pretty good in guessing what the 'ideal green shelf' would look like, you will be presented the most interesting book first.

3.2.1.2 Looking for a document in the web

OySTER is divided into several parts which correspond to the staff and tasks from the example in the last section.
First of all, it has to be said that documents are classified into so–called categories by their content. Additionally, OySTER tries to classify documents by their type as well (homepages, publications, ...). Both types and categories are arranged in independent taxonomies. These categories and types and the subsumption relation for the taxonomies establish the vocabulary by which the user interest is described and by which documents are represented.

Taking into account user feedback which describes relevance of documents to the user's interest, OySTER tries to induce a user model which describes the interest by conceptual expressions. Conceptual expressions consist of document categories (describing the content of the document) and document types. The user model can be used to filter search results by proving relevance of documents with respect to those conceptual expressions.

3.2.2 A brief overview of OySTER

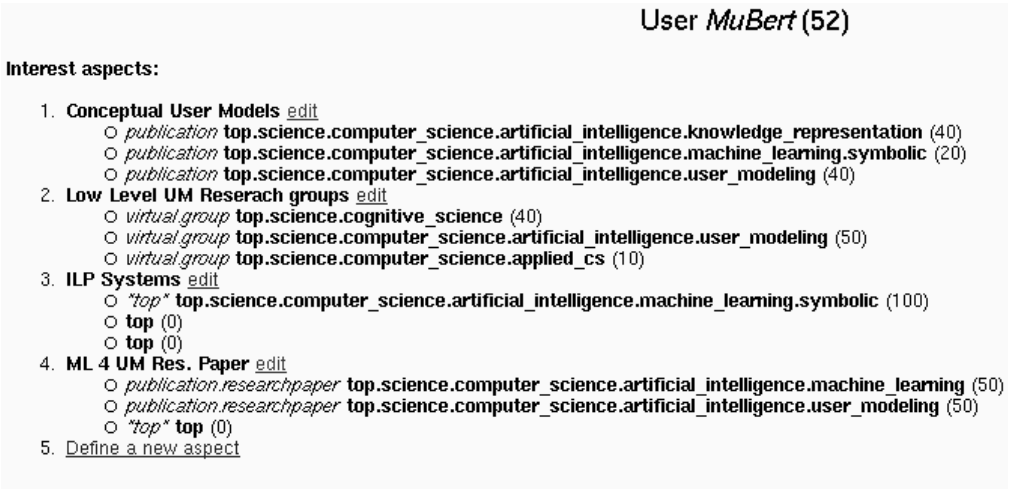
Initially, OySTER needs to collect data about the user in order to induce a first user model. This can be accomplished by scanning the user's homepage which is treated as a simple web document that has to be classified with respect to our document category hierarchy (c.f. [Clodo et al., 2000]) but currently is not included in the prototype. Thus we need another method for gaining a first impression of the user's interest.[6]
Users are rarely interested in only a single topic but rather have a broader field of interests that can be described by a set of *interest aspects*. Aspects represent distinct topics of interest; together they form the whole user interest. In contrast to similar approaches, aspects are not used to discriminate between short–term and long–term interest (c.f. [Billsus and Pazzani, 1999]) but to represent the whole interest as sets of *rather disjoint* sub topics of interest.[7] Since our user models are conceptual descriptions with respect to a category hierarchy, those models can be visualized very easily and allow for a understandable presentation and a manual editing process. As an example, see the user model for the user `MuBert` in the upper part of figure 3.2. It shows four different interest aspects (which are interpreted as a disjunction). Each aspect is defined by up to three types and categories. The first aspect and its definition is displayed in figure in the lower part of figure 3.2. The natural interpretation of an aspect is a 'meet' over the product lattice of document types and categories where each conjunct has a weight attached.[8] In this metaphor, the whole user model corresponds to a 'join' operation on all aspects. Actually, the interpretation of the data that can be used for user modeling depends on the user modeling techniques one wants to realize within OySTER.
Now, this information can be used in order to carry out a search with more precise results. After submitting a search request, the user receives a handle by which he may

[6]Note, that this assumption is not made within the user model learning problem. There, we will try to learn user models from the scratch; that is by feedback only and without any prior knowldege.

[7]Of course, interest aspects may overlap. A paper on 'machine learning for user modeling' is both relevant for machine learning and user modeling. However, the idea behind our approach is to make the user create (or even to suggest to the user to create) a new interest aspect, which only includes such documents. This way, the disjunction becomes an exclusive disjunction, or, the 'rather disjoint' sets become strictly disjoint.

[8]The weights that can be added to categories here are not used for the user modeling process as described later on, since evaluation is based on learning user models without prior knowldege. The weights displayed here are intended to serve as a relative relevance measure for the filtering process.— For a description of the lattices of document types and categories, see appendix.

A user model consists of different aspects:



Each aspect is defined in terms of the taxonomy:

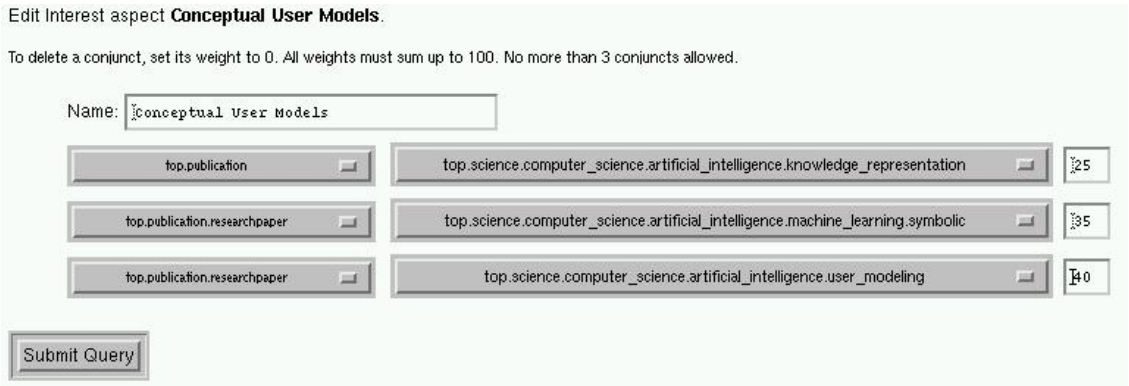
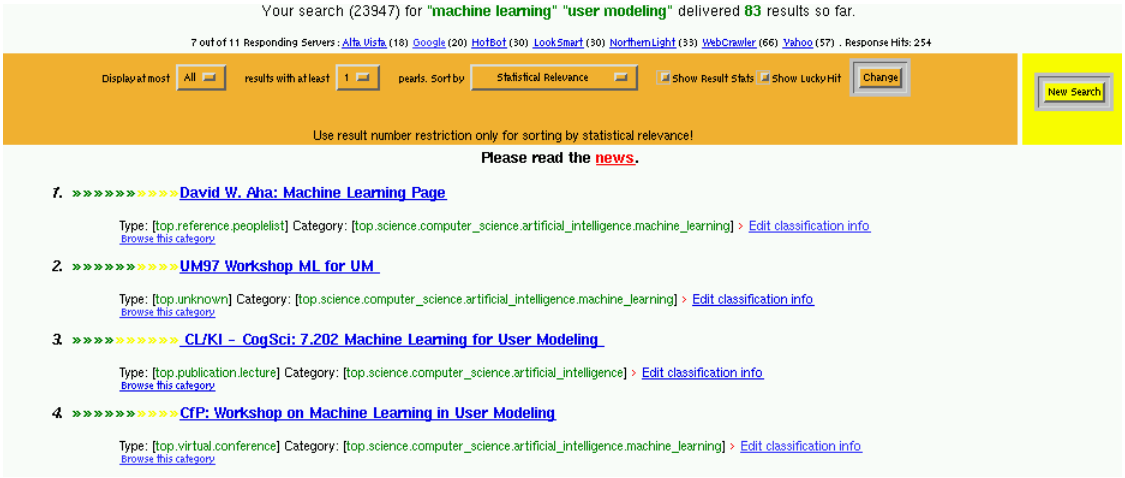


Figure 3.2: A user model for user `MuBert`

request search results. The reason for this method is to allow for an any–time behavior of the search engine: The user receives the 'best' result at any time, independent of the states of the responding search engines or the classifiers of the search results.



Note, that only one category with the highest confidence is displayed in the search result.

Figure 3.3: Search result for *"machine learning" "user modeling"*

After a while more results have been gathered and information from the local database has been added to some of the already known URLs. To receive a search result with nearly completely classified references as in figure 3.3 it may take up to an hour or more depending on whether the collected results were known to the local database or not. Thus, it seems more reasonable to mail the search results to the client instead of forcing him to call back frequently. After all the results have been classified it is reasonable to sort results by their document categories.

The effect of clustering the results by document categories reveals one big advantage of using OySTER: Instead of using query key phrases only, the documents are classified and thus allow for disambiguation of homonyms. This property becomes especially important in the context of user adaptive information filtering, where results that belong to uninteresting classes can be filtered. As an example, consider a single word query for *decompression*. The upper part of figure 3.4 shows a few results that were classified as documents relating to *file decompression* while the same query also delivered results for *decompression sickness* as displayed below. Finally, OySTER asks the user for feedback regarding a selected search result with respect to all of his interest aspects (figure 3.5). The simple click action can also be interpreted as a 'luke warm' feedback for the aspect which fits the category best.

decompression as in *file decompression*:

decompression as in *decompression sickness*:

Figure 3.4: Results for ambiguous queries: *decompression*

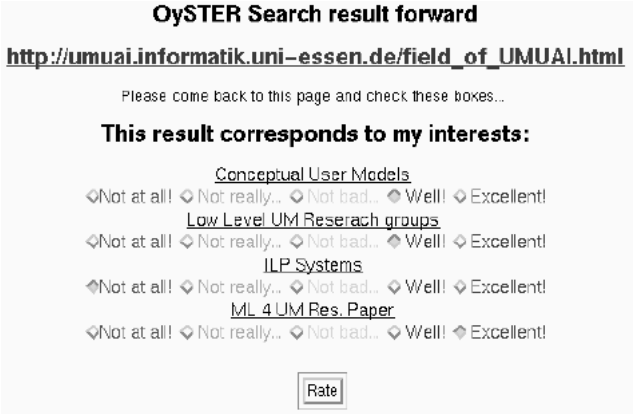


Figure 3.5: Asking for feedback

3.2.3 Enhancements

Query refinement. Having a certain model of the user in mind, consisting of user interest aspects, this knowledge can easily be used to refine a user's query. Suppose, a user submits an ambiguous query like the one shown in figure 3.4. Since all of our categories are attached a set of key phrases which are used by the classifiers, we can use those phrases to enlarge q. Given, for example, interest aspects a_i which are defined over categories c_{j,a_i} the query q can be enlarged to $\bigcup \mathsf{keys}(c_{j,a_i}) \cup q$ which is interpreted as a large (weak) `AND` query by all other search engines. For many interest aspects this method would generate a huge overhead of results that have to be filtered after retrieval, such that a pre-filtered two stage query process seems more reasonable: First, q is processed as usual. In a second step, r is clustered with respect to the ontology. Grouper (c.f. [Zamir and Etzioni, 1998, Etzioni and Zamir, 1999]) performs web document clustering by applying suffix tree clustering on the documents.[9]
Instead of using the whole powerset we now only use key phrases of those aspects which can be mapped onto the first n most reported categories.

Gathering feedback. The current version asks for relevance feedback concerning documents with respect to all interest aspects using a special form as shown in figure 3.5. This form pops up on result selection and as such is a rather bothersome method for acquiring explicit feedback. Instead of forcing the user to complete this questionnaire, we plan to include another method for receiving user feedback which was first proposed by the Slider interface, [Balabanovic, 1998]. The idea is to present the user interest as-

[9]The aim of this clustering method is to dynamically generate a search result, where similar (in terms of words) documents are grouped together. Thus, clusters delivered by this algorithm are not related to any underlying document category hierarchy.

pects as folders which contain interesting links. This bookmark metaphor allows for re-organizing uninteresting or wrongly categorized messages by simple drag-and-drop operations. Deletion, re-categorization or re-ordering can thus be interpreted as explicit feedback. This method has been realized within the Bikini project[10], a student's project for user adaptive news classification which was carried out in the context of OySTER (see [Braun et al., 2001]).

Wrapping. The task of implementing wrappers which extract relevant information from a web page is a Sysiphus' like work in our context: The web pages under consideration are dynamically generated pages of search results from other search engines. Search engines often change their HTML layout such that a static wrapping procedure has an average lifetime of about three months only.
This has given rise to the development of automatic wrapper induction (see the PhD thesis [Kushmerick, 1997]) for special domains (i.e. [Perkowitz et al., 1997]). Presently, all wrappers used in the OySTER prototype are manually developed and perform a simple pattern matching search in order to extract URLs, titles and ranks. A conservative enhancement would be to include snippet extraction for fast keyword detection and the extraction of score information for a better arithmetic over–all score computation. With more utilized search engines being involved, wrapper design and maintenance cannot be carried out manually. Search engine result pages offer a simple structure, from which patterns can easily be learned by only a few examples. With a set of known results (i.e. search requests, result URLs and URL titles) we also have an oracle which should enable supervised learning.
In the course of the Bikini project a wrapper learning tool for newspages on the web has been developed, [Braun et al., 2001]. The wrapper component consists of a generator which generates wrapper descriptions for URLs and an interpreter which executes the wrapper description files and actually extracts information from those URLs. URLs (submitted by a user as a resource for news pages) are cached until two different versions of the same URL are known. Then, the source code of both document versions is parsed and represented as HTML trees. In a next step, differing nodes or subtrees are analyzed. The problem of extracting URLs for single news from a page that offers a list of links to several recent news documents is relatively simple: Most of the news offered by the news server will make use of common URL prefixes. Different methods and a strong bias realized by several well–motivated heuristics are used to search for the most probable prefix.
Generating wrappers for the actual news text is a bit more sophisticated, but since there are only two (unique, distinct and continuous) units of information to be extracted, the problem is feasible: Starting with an HTML parse of a pair of documents from the list of URLs, text differences in similar layout environments are searched. If such tag

[10] C.f. `http://www.cl-ki.uni-osnabrueck.de/~bikini`.

environments have been found and proved on a set of URLs, they are stored as wrapper descriptions that are executed by the wrapper interpreter.
As a fallback solution two robust default wrappers have been implemented. The first one simply returns the HTML `title` tag as news title and the heuristically best rated part of text from the `body` as the news text. The second default wrapper is a bit more elaborate and takes into account considerations such as non–tabular formatting of the news body, precedence relations of title and text, font size and so on. Clearly, this rather ad–hoc defined method is condemned to fail in several cases as for example in the presence of cascading style sheets—but it offers a neat basis for further development of wrapper induction. Another alternative is that presented in the approach HYQL, a hypertext query language developed at the DFKI, [Bauer et al., 1999].
With growing demands by incorporating more sophisticated services, an ILP based approach for learning grammars looks interesting (see [Huck et al., 1998] for a discussion of pattern and grammar based wrapping, [Hammer et al., 1997] for a pattern extraction from web resources and [Cohen, 1995] for an ILP approach to text categorization).

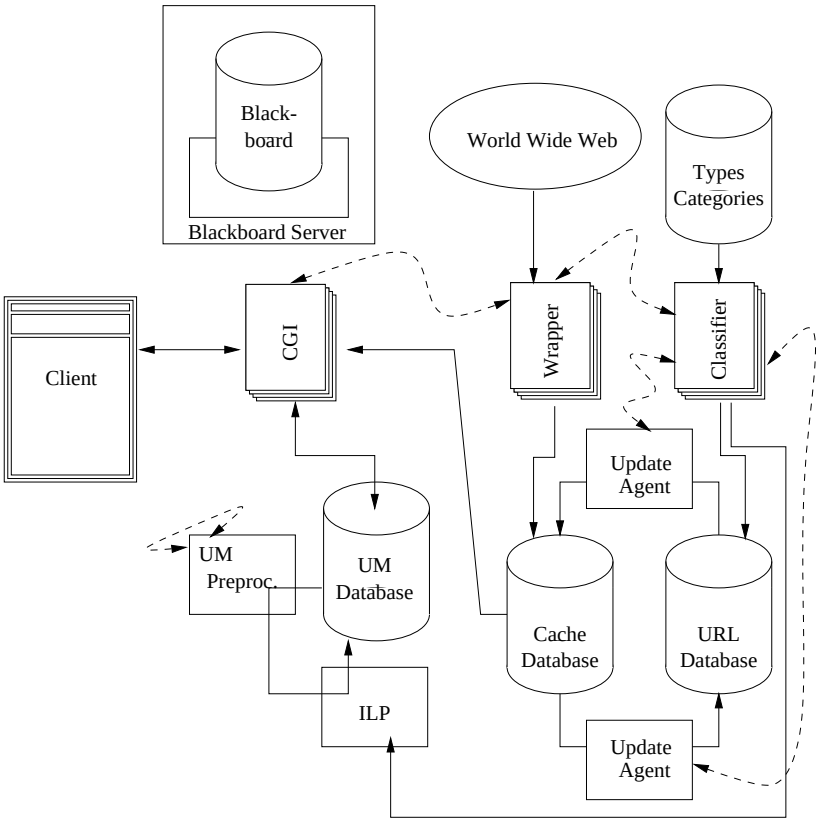
3.2.4 OySTER backstage

After the short virtual tour through the surface level of OySTER we now describe the internal structure of the system. The search engine provides an ideal testbed for user modeling techniques due to its implementation as a multi-agent system.
OySTER consists of three main components: A user model (database) server which provides history and feedback information, a URL database server which provides classifications of URLs and a blackboard server. All components communicate using a special protocol language by way of the blackboard.

Meta search functionality. A search query is added on the blackboard by a CGI agent which returns a handle to the client. The query is decomposed into a list of queries that are submitted to a set of search engines by another agent. Results are gathered and written into a cache database by wrapper agents. Then, the cache database and URL are updated against each other and additional data is added by classifier agents. Results can be requested using the handle and are presented by another CGI agent.
Any of these agents can be substituted by variants while the system is running; furthermore multiple instances of those agents can be started on different machines. This way, we can handle *families* of agents where different agents of the same functionality co–operate or compete for the benefit of optimal results. A brief sketch of the OySTER system architecture is depicted in figure 3.6—a detailed description can be found in the appendices.

User modeling. OySTER is a useful tool to generate and collect interesting data about users. Currently, we can collect *user histories* including search queries and result queries, *traces* with a proxy add-on, *reliable explicit feedback* given by categorization of results with respect to interest aspects and *vague explicit feedback* by detecting simple URL selection. The domain for user interests is described by two concept hierarchies: one for document types and one for document categories (see appendix). Both hierarchies are stored in a database and are accessible through web interfaces which allow for an easy adaption of the hierarchies. Classifiers for those hierarchies were partially developed in the course of the Bikini project (for a description of the document type classifiers see [Heißing, 1999] and [Heißing, 2000]).

User data and domain knowledge can then be used to both induce and explicitly construct user models using appropriate agents.



Solid lines represent data flow which actually is routed through the blackboard server. Dashed lines show agent dependencies which are resolved by messaging through the blackboard. For reasons of readability, blackboard server traffic is not shown.

Figure 3.6: Inside OySTER.

OySTER is a workbench which can easily be used for the empirical evaluation of user modeling techniques rather than a fully functional meta search web engine. Usage of OySTER is guaranteed by the implementation of the system as a set of individual agents and agent families which both co-operate and compete. Any of the involved agents can be easily replaced, modified or relocated during runtime; similarly the system functionality can be expanded by simply adding new agents.
In its current version, the workbench is used for inducing conceptual user models which is in the focus of our current research work. Implementational issues of the underlying multi agent system are described in the appendix (see also [Müller, 1999]).
The underlying techniques used for user modeling and machine learning are described in the next chapter.

3.3 Related work

Search engines, information retrieval as well as user adaptive information filtering and recommendation have become research areas of growing interest. This is exemplified by the multitude of different services, projects and systems that are under construction, already available on the web or currently being planned or developed. Nevertheless, the roots can be easily identified: The emergence of the world wide web has emphasized the need for more sophisticated retrieval methods since the early 90's. From the 70's onwards two distinct research communities worked on user adaptive interfaces in the domain of information systems: Information retrieval and user modeling. A recently published concise book which covers exactly the intersection of these two aspects with the world wide web is [Belew, 2000].
Nevertheless, there are too many systems that deal with machine learning based user modeling in the WWW domain for us to list them all. Thus, we only give a very brief overview which is by no means complete.

3.3.1 Search

Manually edited catalogs were soon outstripped by search engines with crawlers that helped to build web indices. AltaVista, which started in 1995, was among the top three search engines (measured by number of indexed pages) until summer 2000. Competitors were Inktomi and FastSearch (as well as Excite in 1997). In summer 1998, NorthernLight entered the scene and was the search engine with the biggest index for a short period in 1999. Google, starting in 1998 as a research project, overtook AltaVista in late summer 2000 and has more than doubled its size since then.
Currently, as of June 2001, it seems that Google is the unchallenged search engine with respect to index size, rating system and performance.[11] One competitor is NorthernLight which includes a sophisticated document classification system. [12⋆]

3.3.1.1 Meta search

Since search engine business is a hard business, search accuracy has to be put in relation to time and computing power needed in order to determine a trade–off. Commercial

[11] Although its actual index size is estimated to be something around 600 million (and thus twice as big as AltaVista's but not significantly bigger than FastSearch), the method of taking into account links from indexed pages to unseen pages as well, doubles the index size.

[12] The trend observed in summer 2001 has continued. Google now (Spring 2002) claims to have indexed around 1.3 billion documents, more than three times more than NorthernLight. It has also dramatically improved its functionality including address search, image search and a directory. The speed is simply stunning; a search for 'user modeling' yielded approximatley 949,000 results in 0.29 seconds. Precision is very high (first hit being the UM Inc homepage) and many of the results are also known from the web directory.

search engines do not try to deliver the best results in terms of accuracy and coverage but rather deliver the best result given a certain budget in order to guarantee a minimum profit which is determined by expense spent on hardware (computing and bandwidth) and income by commercial advertising.
Recall and precision of search engines can be enhanced by meta search engines. Furthermore, the development of meta search engines allows for an easy specialization with respect to the topic of documents. Currently, `www.searchenginewatch.com` lists 45 different generic (not specialized on certain topics) meta search engines.[13⋆] The first meta search engines were the MetaCrawler, now Go2Net, and SavvySearch. Another very well known meta crawler is ProFusion. All abovementioned search services originally started as research projects.
SavvySearch, [Howe and Dreilinger, 1997], offers an interesting option in querying search facilities: While usual meta search engines forward the search query to a static set of search services, SavvySearch builds a meta–index in order to rate search engines with respect to queries. Then, only selected search engines will be asked and the search engine quality for the current query is taken into account while ranking the aggregated results.
The MetaCrawler technology was further enhanced by a clustering mechanism (Grouper) to yield the HuskySearch engine which performs web document clustering on meta search results by significant words using suffix trees, see [Zamir and Etzioni, 1998] and the follow-up [Etzioni and Zamir, 1999]. Since it does not use any conceptual descriptions or user models, but performs clustering on document phrases (with respect to stemming and stop words), the delivered clusters often result in a 'weird' but still very helpful organization of results.

3.3.1.2 Wrapper specialists

Finding Homepages. Ahoy! (c.f. [Shakes and Langheinrich, 1997]) is a homepage specialist. Upon input of a name (first name and last name) and optionally specifying institution, region and e-mail, Ahoy! returns a list of links to homepages of the sought person. Ahoy! utilizes heterogenous information sources as input: On the one hand, it carries out a search on MetaCrawler using the `NEAR` search facility in order to ensure a full name search that is invariant to first and last name ordering. Additionally, e–mail services (WhoWhere) provide user names and a database of institutions (Yahoo) provides server names. Results are aggregated and grouped by cross filtering, where the ranking of each service is interpreted as one dimension in the search space. Incremental aggregation thus means cutting a cube down to a plane, the plane down to a ray, and the ray—in the

[13] As of June, 2001. Current figures (February 2002) include 15 major generic search services, 11 further global search engines, and 6 collaboratively maintained directory based search engines (such as `dmoz`). There are more than 30 news specialized search engines, 21 meta search engines, and over 180 topic specific search engines, not including more than 20 multimedia search services.

ideal case—to one point. All references returned are grouped by their over-all quality into "buckets", where the quality is determined by the position of the document in the search space. Herein, the dimension that is associated to the MetaCrawler is weighted more significant as, e.g., the server name database, since the name of the person the user looks for is assumed to be more important than the person's location.[14] High quality buckets then include a list of promising candidates. On a test sample Ahoy! has a recall of approximately 85% and a precision of approximately 75%.[15] The recall is 9% percent higher than that of MetaCrawler which is due to the fact, that Ahoy! tries to guess URLs where there are none actually found. Several near misses including information about institutions (as found by MetaCrawler through staff member lists, online publications etc) deliver hints of where to find a homepage; for example `www.isiv.uni-osnabrueck.de` and `mir.cl-ki.uni-osnabrueck.de`. Together with a query for some person named *Martin Müller*, Ahoy! successively tries to find homepages by guessing URLs by extending the server info by paths: Such candidates are: `/~martin`, `/~mmueller`, `/~mmueller`, `/~mm` or `/staff/M.Mueller.html` and so on. The URL pattern generator derived 23,000 patterns for 6,000 institutions kept in a local DB by an offline–learning algorithm. Limitations of Ahoy! are multiple (often "fan") homepages (with a large set of responses from MetaCrawler) or homepages of non–existent persons (a search request for *Alan Turing* delivered 116 results).

Finding cheap bargains. The Shopbot seeks for WWW catalogs. The Shopbot project (c.f. [Perkowitz et al., 1997]) started as a research project at the university of Washington and became a commercial service at Excite (`http://www.jango.excite.com/`).
It is specialized on several topics and according web resources. ShopBot aggregates information from various resources and integrates them into one unifying layout. In other words, ShopBot is a wrapper par excellence. Using a knowledge base concerning query interfaces for different online shops, the ShopBot forwards the search query to the resource services. From the responses, the information is extracted and then put into the meta–search response document. Again, it is clear, that for a small change in WWW presentation the wrappers have to be redefined. The technical problem of wrapping in information integration is discussed in detail in [Perkowitz et al., 1997]; the conclusion was to make the process of wrapper design more comfortable. This resulted in automatic wrapper induction, see [Kushmerick, 1997] and [Kushmerick and Doorenbos, 1997].

Both the Shopbot and Ahoy! are based on the MetaCrawler: The MetaCrawler is the

[14]It is clear, that here again we have to compare incomparable qualities; a dilemma that cannot be resolved. But for this domain, the chosen preferences seem to be well founded (for the problem of evaluation from heterogeneous information sources, see [Müller, 1996]).

[15]The test sample was derived from David Aha's page of machine learning researchers at `http://www.aic.nrl.navy.mil/~aha/people.html`. Precision here means: If the target was listed as first link—if found at all—delivers a precision of 1. The value of 75% is the average precision over the sample.

information aggregation component which is prior to further specialization or personalization. From the scientific point of view, the ShopBot has a further ancestor, namely the internet learning agent, ILA. This agent focused on learning interfaces (and wrappers) instead of hard–wiring them. An overview of the whole family of softbots created at the University of Washington is given in [Etzioni, 1997].

3.3.2 User adaptive filtering and recommendation

Some of the most popular systems are the previously mentioned WebWatcher and Syskill& Webert systems. In section 2.3.3 we also gave a brief overview of the NewsDude system and the InformationValet prototype.

Letitzia. One of the first web browsing assistants was Letitzia, [Lieberman, 1995]. The system is a recommender system which tries to guess relevance of web pages based upon observations. The user model is entirely made up of relevant key words. Therefore, no additional effort needs to be put into modeling the user's interest based upon a background knowledge base. Recommendation is performed by making use of a set of rather weak heuristics which together contribute to a picture of the user's interest. The heuristics take into account time spent on a page in relation to the document's length, following further links or navigating back. Since documents are usually read top down and left to right, skipped links are also interpreted as uninteresting. In contrast to many similar systems, Letitzia does not classify documents as interesting or irrelevant but rather ranks the set of links available in the currently viewed document. From such a preference ordering, relevant documents can be determined by a high ranking. If the preference ordering is 'dense'; i.e. all links are ranked similarly, there is no significant 'most interesting' link which should be recommended and the system remains silent in the background. Due to its usage as an online browsing assistant, the system performs a breadth first search in order to analyze the referenced documents.

Fab. In [Balabanovic and Shoham, 1997], the authors describe a system which by simultaneously applying content based and collaborative methods tries to overcome the drawbacks of each approach by using the advantage of the other. Fab is a web page recommender, which builds individual user models by means of content based user modeling and compares those models to other user's models thus performing collaborative user modeling, too. On the one hand, this allows for derivation of group models and can be used to prevent overfitting (thus eliminating the drawbacks of content based user modeling)—on the other hand Fab is able to decide whether to recommend new items, which have not been rated by other users, on the basis of its content (and thus overcomes the weakness of collaborative approaches). Content based user modeling requires content based indexing of the web pages on which recommendation will be performed. Once clas-

sified and indexed, the pages can be recommended to users with similar, matching user models (collaborative). Content based recommendation is based on relevance feedback which the user gives explicitely rating the above recommended pages. There is already an interesting variant of the idea of user interest aspects within Fab : Instead of a clear distinction between several aspects, Fab adapts to different kinds of filtering components: On the one hand, a set of so–called collection agents are adapted to both specialize in topics as well as to cover the set of all web pages. This process is user independent. On the other hand, user dependent filtering agents (which implement the individual, content based user model) choose documents by choosing appropriate collection agents. The underlying model again is based on the vector space model of word occurrences.

Slider. The Slider interface, [Balabanovic, 1998], is very similar to the OySTER approach. User models also consist of different aspects (here: topics) and actions performed on incoming news are interpreted as implicit feedback (explicit feedback is not asked for). In contrast to OySTER, documents and user models are represented using word vectors. Elements of the vector are chosen canonically by stemming and TfIdf. The user model is represented by a set of vectors; each vector representing a topic. A topic vector $\vec{t}$ is updated by feedback–weighted document vectors $\vec{d}$: $\vec{t} := \vec{t} + \lambda\vec{d}$. The user can perform several actions on the set of incoming news, these include: Creation of new topics, moving news from one topic to another, reading a news item, deleting it, explicitly rating it or deleting an entire topic. All these actions are mapped onto distinct topic update functions with different, but fixed, λ–values. Moving d from t_1 to t_2, for example, results in $\vec{t_1} := \vec{t_1} - \lambda\vec{d}$ and $\vec{t_2} := \vec{t_2} + \lambda\vec{d}$. The actual weights are chosen in relation to the reliability of performed actions: A NIL–action sets $\lambda = 0.25$, reading sets $\lambda = 0.5$ and all other actions are described by an update value of $\lambda = 3$.

Amalthaea. Amalthaea, [Moukas, 1996], is a very close relative of OySTER from the implementational point of view. Although Amalthaea is keyword based, it consists of two classes of agents, which—with respect to a user model—filter and discover information. Discovery agents act as information retrieval components and are being optimized with respect to their results according to the user model. Information filtering agents act on user models and refine the models by using explicit feedback from the client.

Profile. Profile, [Simons, 1997], a project at Nijmegen University, deals with information filtering on a dynamic archive. Queries are refined and extended with respect to the user model and domain structure is represented in a hand crafted ontology, which corresponds to $\mathcal{C}$. Profile lies somewhere between Amalthaea (modulo missing ontology) and the OntoBroker (modulo missing user modeling).

Chapter 4

CONCEPTUAL USER MODELS

Conceptual user models contain conceptual knowledge about the user. In our case, we want to describe a user's interest. In contrast to word occurrences, (sequences of) interaction primitives or other direct evidence for user interaction, conceptual user models contain a description of meaningful symbols representing concepts which can be used to intensionally describe a complete set of observations. As an example, a conceptual user model would contain knowledge about 'Undo-Actions' instead of a sequence of keystrokes consisting of deletions, backspacing or mouse clicks on 'Undo' buttons.
In this section, we describe the concept lattice we use for describing a user's interest in the subdomain of web documents which deal with research publications and related document types in the field of computer science. We also show, how to represent such 'ontologies' in the form of logic programs as a base for the following inductive processes.

4.1 Concepts as descriptions of a user's interest

In ML4UM user models are often represented by n–ary vectors. Matches are then determined by applying an appropriate measure in vector space. In the course of content–based document recommendation, the vectors represent significance of "key"–phrases for the user's interest which is defined as the frequency of those phrases in the documents the user has rated as interesting in the past. Vector length is reduced by extracting the most relevant phrases using TFIDF measures and the labeling of examples is achieved by relevance feedback which assigns target function values to vectors, see [Billsus and Pazzani, 1999, Balabanovic, 1998, Billsus and Pazzani, 1997, Joachims et al., 1997, Pazzani et al., 1996, Lieberman, 1995]. In order to decide whether to recommend a document or not, the current document's classification (i.e. the corresponding vector) is compared to the user model vector.
Though the vectors *represent* a user's interest, they do not explicitely *describe* a user's interest. This fact can be illustrated by a simple Gedanken experiment: Given a user model consisting of such a word vector—what would be the system's answer to the

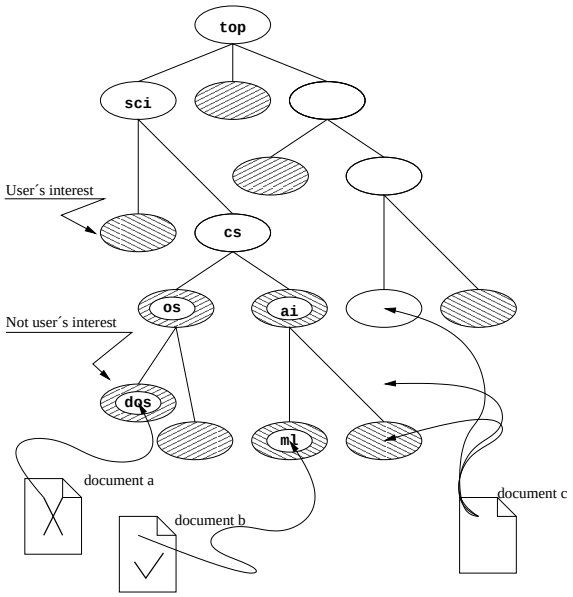
user's questions '*What is your model of my interests?*'. Showing the vector to the user is like answering '*This is my representation of your interest*' but it is by no means a description the user could easily understand. Thus, our motivation was to find a transparent formalism which is accessible to every user and which allows for an easy translation into the user's 'language'. Such a language could be the language of concept hierarchies.
The idea behind using such category hierarchies is, that they will be used both for representing documents as well as for representing user models. A pictorial view on the question of whether a certain document is interesting for a user u with respect to the underlying user model $\mathbf{M}_u$, is shown in figure 4.1.
Our approach allows for a more lucid explanation in an entirely different problem as well. In user adaptive systems, where user interest is defined by means of words, the borderline between classification and recommendation is veiled. Thus, when measuring accuracy (or rather precision in such cases) of the whole system it is not clear whether significance of key words or the actual user modeling contributes to increasing performance. When using conceptual user models, the accuracy of user models is determined by accuracy measures on the machine learning problem exclusively; word frequencies are invisible to the conceptual user model.

4.2 Concept hierarchies

Concept hierarchies are graphs, where nodes represent (atomic) concepts with a designated meaning and edges represent relations between those nodes. Here, we will only discuss a simple subset of semantic networks: the graph needs to be directed and acyclic; the direction is defined by the inverse of the only reflexive relation 'superconcept'. In other words, the concept hierarchies we are about to discuss are trees. Within OySTER we use two such concept hierarchies: One for document types ($\mathcal{T}$) and one for document content (categories, $\mathcal{C}$).

4.2.1 Concept hierarchies as lattices

More formally, $\langle \mathcal{T}, \sqcup_{\mathcal{T}} \rangle$ is a semi-lattice of document types $t \in \mathcal{T}$ with an ordering relation $\sqsupseteq_{\mathcal{T}}$ where $t \sqsupseteq_{\mathcal{T}} t'$ if and only if $t \sqcup_{\mathcal{T}} t' = t$. Similarly, $\mathcal{C}$ is a semi-lattice of document categories c with an ordering relation $\sqsupseteq_{\mathcal{C}}$ where $c \sqsupseteq_{\mathcal{C}} c'$ if and only if $c \sqcup_{\mathcal{C}} c' = c$. We extend $\mathcal{C}$ by an artificial bottom element $\perp_{\mathcal{C}}$ and define $c \sqcap_{\mathcal{C}} c' = \perp_{\mathcal{C}}$ for any c, c' with $c \sqcap c' \notin \{c, c'\}$ to yield a lattice $\langle \mathcal{C}, \sqcup_{\mathcal{C}}, \sqcap_{\mathcal{C}} \rangle$. A lattice $\langle \mathcal{T}, \sqcup_{\mathcal{T}}, \sqcap_{\mathcal{T}} \rangle$ can be defined in the same way.
Given such concept hierarchies, a set of appropriate classifier algorithms $\mathbf{A}_{\mathcal{C}}(d)$ and $\mathbf{A}_{\mathcal{T}}(d)$ which, for a document d, delivers a category c and a type t, we can use to describe documents by those classifications.



Documents a and b are assigned a unique class using the classification algorithm $\mathbf{A}_{\mathcal{C}}$. Document c belongs to at least two known classes.
$\mathcal{C}$ is also used to define a user model $\mathbf{M}_u$ which consists of sets of classes the user definitely is interested in and definitely is not interested in.
Thus, the user is interested in document b and not in document a. It cannot be proven, that document c belongs to those documents, the user is not interested in.

\–shaded nodes represent the user's interest, while /–shaded nodes represent the user's disinterest. The arrows stand for document classifications. The checkmark represents positive user feedback; the cross–checked (X) document was assigned negative feedback.

Figure 4.1: Using $\mathcal{C}$ for both document classification and user models

Document types are independent from document content.[1] Thus, we interpret $\mathcal{T}$ as sorts over expressions in terms of $\mathcal{C}$. In other words, the sorted term $t : c$ with sort (i.e. type) $t \in \mathcal{T}$ and category $c \in \mathcal{C}$ denotes a concept that 'contains' any document for which $\mathbf{A}_{\mathcal{T}}(d) = t'$ with $t \sqsubseteq t'$ and $\mathbf{A}_{\mathcal{C}}(d) = c'$ with $c \sqsubseteq c'$.
The concept hierarchies used in the system are described in the appendix.

4.2.2 Describing documents by concepts

Working on document contents instead of word vectors means that we need classifiers. Definition or implementation of efficient and accurate classifiers is not the main focus of this thesis. Nevertheless, we needed to implement classifiers for our special domain since available concept hierarchies with appropriate classifiers were not suitable for our project.

Choosing a concept hierarchy of document categories. The domain we had to choose had to be a well defined and mostly self–contained set of documents. Thus, our decision was

[1] Here, 'independent' does *not* mean independent in the sense of probability theory.

Query result for “Maschinelles Lernen” (Machine Learning):

Titel	seminar "lernende agenten im www"
URL	http://www-ai.cs.uni-dortmund.de/LEHRE/AGENTEN97/agenten97.html
Überschrift	seminarank ndigung wichtiger hinweis einleitung in das themengebiet agenten grundlagen und verfahren lernen von strukturen und grammatiken agenten zur unterst tzung beim browsing auf dem www agenten zum filtern von e-mail agenten zum filtern von usenet news faq finder informationsfilterung und klassifikation von texten benutzermodellierung und benutzerschnittstellen pers nliche assistenten java und html (hypertext markup language)
Zugeordnete Verzeichniseinträge	• INTELLIGENZ, LERNEN, LERNFAEHIGKEIT • INTELLIGENZ, LERNEN UND LERNFAEHIGKEIT • INFORMATIK + COMPUTERWISSENSCHAFTEN • LINDENMAYER-SYSTEM • QUANTENZUSTAND EINES MIKROPHYSIK. SYSTEMS • LERNEN

Result:
“Learning Agents in the WWW”
Result Classification:

- intelligence, learning, ability to learn
- intelligence, learning and ability to learn
- computer science
- Lindemayer system
- quantum state of a microphysical system
- learning

Figure 4.2: Gerhard: Using the UDC for document classification

to build our own small concept hierarchy which describes the field of artificial intelligence and neighboring communities in the discipline of computer science. The OySTER category hierarchy $\mathcal{C}$ is a handishly coded hybrid that emerged out of the universal decimal classification (UDC), the Google web directory, the dmoz project[2] and—last but not least—personal experience with classifying web documents. $\mathcal{C}$ could be adapted to nearly any application domain, as long as all classes are initially described by a set of phrases which are indispensable for reliable document classification. The initial idea of incorporating already used ontologies failed due to unavailability, coarseness, incompleteness or complexity reasons. For example, the UDC which was considered due to the fact that we had existing classifiers at our disposal, offers the advantage of already including syntactical means in order to describe new classes. The UDC has been used within the German search index for scientific publications that was developed within the course of the Gerhard system. The Gerhard project (c.f [Möller et al., 1999]) uses the UDC for the automated classification of documents.[3] Choosing the UDC as the category ontology seems to be a more promising approach, since it offers three major advantages: (1) it is much finer, (2) it includes a language for defining new concepts by way of existing ones using modifiers (where ‘c_1 modifies c_2’ differs from ‘c_2 modifies c_1’) and (3) its concept names are generic (i.e. numeric) such that newly invented concepts can be named in a proper way. As an example of using the UDC, see figure 4.2 for a screenshot from a Gerhard search result. On the other hand, the UDC inherited many properties of the DDC[4], which means that some included knowledge structures dating back to the

[2]See `http://www.dmoz.org/about/`.
[3]For documentation, see `http://www.gerhard.de/info/index_en.html`.
[4]Dewey Decimal Classification; see, e.g. `http://www.oclc.org/fp/`.

last century. Furthermore, the UDC is simply too big: More than 60,000 classes (as in the ETHZ version) organized in almost arbitrarily deeply nested paths describe nearly any topic in great detail. But since concepts can be defined by asymmetric modification relations and lattice operations, it is almost impossible to find a self contained subset of the UDC. The long tradition behind the UDC also induces several problems: computer science and its sub-discipline artificial intelligence has the superconcept electrical engineering. This means, in consequence, that in our naïve reading of such a hierarchy either computational learning theory is a part of electrical engineering or that contributions to learning theory force implications on electrical engineering—which is at least a questionable proposition.
On the other hand, the Google directory turned out to be too coarse in the required substructure while the dmoz project yielded an ontology which almost reaches UDC's complexity. Another candidate, the classification system of the University of Osnabrück's library, which also provided annotated keywords and classifiers as implemented in the course of the Osiris project (c.f. [Ronthaler, 1998]) was also too coarse and does not allow for the invention of new classes.
Thus we decided to design a small category hierarchy that best met our needs and which united the benefits of the aforementioned classification systems by pruning their drawbacks. The resulting ontology consists of 69 classes with a maximum depth of 5. It is described in the appendix.

Defining suitable classifiers. Since we crafted our own hierarchy of concepts we also needed to implement our own classifier. Here, substantial work was carried out in the Bikini project. The classifiers used are based on a vector space model. Each concept is represented by a word vector which is compared to the document's word vector. The phrases contained within the category vectors were determined by a bootstrapping method: For each concept we defined a small number of relevant keywords which were used to generate a set of queries to several search engines (including OySTER itself). The documents received for each query were concatenated to yield single documents on which relevant phrases were extracted using a TFIDF method. The resulting phrases were handishly postprocessed. Now, given category description vectors $\vec{c}$ and word vectors $\vec{d}$ of documents d, the document is classified by a simple similarity measure:

$$\|\vec{c} - \vec{d}\| := \frac{\sum c_i \cdot d_i}{|\vec{d}|} - \vartheta_c \tag{4.1}$$

Herein, ϑ_c is a c–specific bias determined by checking pre–classified documents against $\vec{c}$. The classifiers are described in detail in [Braun et al., 2001, Eilert et al., 2001].

Document types. Similarly we developed a hierarchy of document types which roughly can be divided into *homepages*, *research articles* and *lecture material*. Document or

text types are discussed within computational linguistics but are rather unknown in the domain of text classification for user–centered document retrieval.[5]
Currently, the classification of documents with respect to their types is carried out using a simple regular expression match on the URL of the document only. The motivation behind this approach was to provide the search engine with a fast classifier that returns preliminary classification results until a deeper investigation of the document delivers a more accurate result. This has been discussed in [Heißing, 2000]. Document type classifiers of sufficient accuracy need to take document structure and word or phrase analysis into account.
The hierarchy of document types contains 35 classes with a maximum tree depth of 4. The type hierarchy is described in the appendix.

Representing documents. Using our classifiers $\mathbf{A}_{\mathcal{C}}$ and $\mathbf{A}_{\mathcal{T}}$ we represent a document d by a pair of document classifications:

$$\mathbf{A}_{\mathcal{T}\times\mathcal{C}}(d) = \langle \mathbf{A}_{\mathcal{T}}(d), \mathbf{A}_{\mathcal{C}}(d) \rangle = t : \vec{c} = \langle t :: p, \langle c_1 :: p_1, c_2 :: p_2, \ldots, c_n :: p_n \rangle \rangle \tag{4.2}$$

where t is the document's type and p the confidence of the classification and c_i are categories with decreasing values of confidence p_i. Currently, n is limited to 3 in order to achieve a stronger bias on the learning task.

4.2.3 Describing user models by concepts

As already mentioned above and illustrated in figure 4.1, the same category hierarchies $\mathcal{T}$ and $\mathcal{C}$ are used to represent the model of a user's interest. Instead of representing a user's interest by a set of word vectors which are used to determine relevance of documents by use of vector space operations such as cosine measures, the user interest is expressed in terms of document contents that are described by categories and types. Furthermore, a user model consists of several aspects as pointed out in section 40. This means, that a user model is represented as a set of conceptual descriptions which are interpreted as weighted meets over atomic concepts. One such aspect is depicted in the lower part of figure 3.2. This aspect is interpreted as:

> A document d is considered relevant with respect to the aspect **Conceptual User Models** if d is classified as:

[5] Although text types like 'newspaper article', 'obituary' or 'poem' are widely accepted, the 'new' document types that emerged with the WWW have not been discussed to a satisfying extent which would allow for a reference here. Researchers agree that there are text types like 'homepage' but it is still argued whether those types are already known types which have been adapted to new media (in this case: 'application' or 'curriculum vitae') or whether they are genuine types. Either way, to the author's knowledge there is no published work which focuses precisely on this topic and which tries to give a list or hierarchy of known document types.

1. a *publication* about *knowledge representation* with a confidence of at least 40 **and**
2. a *publication* about *symbolic machine learning* with a confidence of at least 20 **and**
3. a *publication* about *user modeling* with a confidence of at least 40,

Thus, a user model is a conceptual description of the form

$$\bigsqcup_i \sqcap_j (t :: c :: p)_{i_j}. \tag{4.3}$$

In other words, it is a set of Horn clauses C_i, whose premises are conjunctions of literals $L_{i_j} = (t :: c :: p)_{i_j}$. Relevance with respect to an aspect i thus means that all literals L_{i_j} are satisfiable.[6]

4.2.4 The meaning of inheritance

Dealing with concept hierarchies always poses an important question: what does inheritance mean? At the beginning of this section we stated that $c \sqsupseteq_{\mathcal{C}} c'$ means that c is a superconcept of c'. In the context of real ontologies, the inverse $\sqsubseteq_{\mathcal{C}}$ is usually read as *is_a*. Here, the ordering relation actually has two readings:
For representing documents, a naïve interpretation would imply that a document about c' is also about c. This only holds to a certain degree, of course. In our case, *also_is_about* is realized by taking into account costs that are attached to the edges defined by $\sqcap$. Roughly, content similarity decreases the closer we get to the top element of the lattice. Vice versa, it is not the case, that a document about c also covers the topic c'. In consequence, one would have to represent the concept hierarchies as a flat hierarchy, thus losing any inheritance information which is not desirable either.
Regarding the user interest, inheritance or rather implication cannot be defined clearly as well. Interest in c' does not necessarily imply interest in c though it is a kind of supporting evidence. The dual case does not hold either.
In consequence, we define edge costs and a distance measure on $\mathcal{C}$ which reflects the most important properties of a concept hierarchy.

1. The higher an edge is located in the tree, the higher the cost. The reason for this is, that the class similarity increases the closer one gets to the leaves.

2. Long paths consisting of more edges are more expensive than short ones.

[6] Actually, each aspect again may consist of a set of clauses. This will be motivated and discussed later on; for clarifying the general idea behind our approach the simplified description given here is sufficient and more suitable. See section 4.3.

3. Costs of paths of equal length are correlated to the cost of their highest edge.

To achieve this we first need to extend the definition of $\sqsupseteq$:

Definition 4.1 (Path subsumption, $n \sqsupseteq_{\mathcal{C}}^{p} m$) *We extend the notion of subsumption in a lattice by computing the minimum path length p from a node n to m. We write*

1. $n \sqsupseteq_{\mathcal{C}}^{0} n$ *for any node* n.
2. $n \sqsupseteq_{\mathcal{C}}^{1} m$ *iff:*
   (a) $n \neq m$
   (b) $n \sqsupseteq_{\mathcal{C}} m$
   (c) $n \sqsupseteq_{\mathcal{C}} m$ *and* $n \sqsupseteq_{\mathcal{C}} l \sqsupseteq_{\mathcal{C}} m$ *implies* $n = l \vee l = m$.
3. $n \sqsupseteq_{\mathcal{C}}^{p+1} m$ *iff* $n \sqsupseteq_{\mathcal{C}}^{p} l$ *and* $l \sqsupseteq_{\mathcal{C}}^{1} m$.

This way, $\sqsupseteq_{\mathcal{C}}^{p}$ *calculates path length.*

Since our approach for computing distances on the hierarchies will be a cost–based traversing algorithm, we define costs of a link as follows:

Definition 4.2 (Cost annotation in $\mathcal{C}$, cost) *For two nodes* n, m *with* $n \sqsupseteq_{\mathcal{C}}^{1} m$ *the cost attached to the edge is*

$$\mathsf{cost}(n, m) = \mathsf{cost}(m, n) = \alpha^{\mathsf{dep}(\mathcal{C}) - p(m,n)} \tag{4.4}$$

where $p(m, n) = \min\{p | \top \sqsupseteq_{\mathcal{C}}^{p} m, \top \sqsupseteq_{\mathcal{C}}^{p} n\}$ *and* $\mathsf{dep}(\mathcal{C}) = \max\{l | \top \sqsupseteq_{\mathcal{C}}^{l} c \text{ for all } c \in \mathcal{C}\}$.

For our work a value of $\alpha = 1.5$ has been proved to deliver satisfying results. Using cost, we can now penalize edges according to our requirements mentioned previously. Next, we need to define path costs. Since generalization shall be more expensive than specialization we need to define two different measures. The cost $\mathsf{down}(m, n)$ for traversing the tree from m strictly downwards to n is defined as the sum of costs of all edges that establish the path from m to n:

Definition 4.3 (Downward path cost, down) *Let* $x_1 \sqsupseteq_{\mathcal{C}}^{1} \cdots \sqsupseteq_{\mathcal{C}}^{1} x_n$. *Then,*

$$\mathsf{down}(x_1, x_n) = \sum_{i=1}^{n-1} \mathsf{cost}(x_i, x_{i+1}) \tag{4.5}$$

Note, that specialization becomes cheaper as we get nearer to the leaves. Though interest in c' implies interest in some c with $c \sqsupseteq_{\mathcal{C}} c'$, c is a less precise description of interest. Thus, generalizing paths are additionally penalized by their length:

Definition 4.4 (Upward path cost, up) *Again, let* $x_1 \sqsupseteq_{\mathcal{C}}^{p} x_n$. *Then,*

$$\mathsf{up}(x_n, x_1) = \log_2(p+1)\mathsf{down}(x_1, x_n) \tag{4.6}$$

Now, we can define an asymmetric distance measure δ on our document category hierarchy $\mathcal{C}$ as follows:

Definition 4.5 (Distance measure δ on $\mathcal{C}$) *For two nodes* $n, m \in \mathcal{C}$, *the distance from* n *to* m *is defined as*

$$\delta(n, m) = \mathsf{up}(n, l) + \mathsf{down}(l, m) \tag{4.7}$$

where $l = \mathsf{lub}(\{n, m\})$ *and the distance is the sum of costs from* n *up to* l *and from there down to* m.

The asymmetric nature of δ may sound a bit unfamiliar: Let $c \sqsupseteq_{\mathcal{C}}^{1} c'$ and $c \sqsupseteq_{\mathcal{C}}^{2} c''$ but not $c' \sqsupseteq_{\mathcal{C}} c''$. Then, c'' is less related to c' than c' to c''. Nevertheless, it is well founded as explained by the following example: Consider, for example, c'' =‘symbolic machine learning and c' =‘logic programming’. The least upper bound is ‘artificial intelligence’. Now, ‘logic programming’ is closer to ‘symbolic machine learning’ than vice versa—since ‘logic programming’ is directly subsumed by ‘artificial intelligence’.
How does this relate to our statement that c'' is less related to c' than c' to c''? In this concrete example it means, that ‘symbolic machine learning’ is less related to ‘logic programming’ than ‘logic programming’ is related to ‘symbolic machine learning’. In fact, this can be explained by a simple argument: ‘Symbolic machine learning’ is more specific than ‘logic programming’, because it is a subtopic of ‘machine learning’ (which is on the same level in the taxonomy as ‘logic programming’). Therefore, if something deals with ‘symbolic machine learning’, it may play a certain, but smaller, role in the general concept of ‘machine learning’. The same holds for ‘artificial intelligence’. In other words, the relevance of a document about a special category becomes weaker, the more general the categories are. In order to relate ‘symbolic machine learning’ to ‘logic programming’, we need two generalization steps and one specialization (which means, that the target is more general as the source, too). In consequence, ‘symbolic machine learning’ is related to ‘logic programming’, but not very tightly.
In the inverse case, we come to the conclusion, that ‘logic programming’ is related more strongly to ‘symbolic machine learning’: Thinking of a relatively more general idea of logic programming, one can easier relate the general method of ‘logic programming’ to a more special field such as ‘symbolic machine learning’. The impact of ‘logic programming’ is weakened only once by generalising to ‘artificial intelligence’.
To put it into one simple phrase, an article about "*Efficient SLD resolution*" is more relevant to symbolic machine learning (say, ILP), than an article about "*Bias in symbolic learning*" is relevant to logic programming (say, constraint logic programming).

Any distance measure or semantics of inheritance are subject to criticism. In our case, we do not at all postulate to have closed the argument on this issue. During the work on our formalisation we rather concluded, that the philosopher's stone providing us with a true and everlasting formalism of how to craft ontologies slips through our fingers faster the harder we try to get a grip on it.
Nevertheless, our assumption of decreasing edge cost seems to be well founded. As a calming fact, cost actually will be the only function we will have to use during the user model induction process. The distance measure δ which might be more questionable will be used for generating feedback and thus simulating user interaction only; it is not involved in the learning process. Now one might argue whether such a measure can simulate a real user's behavior since its design is not all cognitively founded. Sadly, we cannot do anything but accept this criticism, although the results we achieved using this measure actually looked pretty much like feedback of real users. The feedback generated is slightly worse than expected—which in our case is a big advantage: Modest results become even more promising, the noisier the training samples and the harder the learning problems become.
Thus, we could even reverse the argument: The worse the modeling character of δ, the harder is our simulation and the more our approach will work in practice.

4.3 Concept hierarchies and user models as logic progams

As already mentioned in the last section, we will represent both the concept hierarchies and the user models as logic programs. In order to be able to induce user models, we also need a suitable representation according to the learning algorithm. Accordingly, background knowledge like inheritance relations in the document type and category hierarchies are represented by Horn clauses. We will first discuss the hierarchies.

4.3.1 Representations of lattices

A concept hierarchy $\mathcal{C}$ can be represented as a logic program using many different methods. A simple idea would be to declare any element of a concept hierachy as a tree node by asserting $\texttt{node}(c_i)$. Subsumption can then be represented extensionally by defining $\texttt{subsumes}(c, c')$ for any $c \sqsupseteq^1_{\mathcal{C}} c'$. Thus, transitivity of inheritance would be modeled by a depth limited or depth penalized recursive definition on subsumes. Another idea is to incorporate subsumption information into the node names. Then, a node c with $\top_{\mathcal{C}} \sqsupseteq^n_{\mathcal{C}} c_1 \sqsupseteq^1_{\mathcal{C}} \cdots \sqsupseteq^1_{\mathcal{C}} c_n \sqsupseteq^1_{\mathcal{C}} c$ would be represented as a list $[\top_{\mathcal{C}}, c_1, \ldots, c_n, c]$. Subsumption of $c \sqsupset_{\mathcal{C}} c'$ would be checked by proving that $[\top_{\mathcal{C}}, \ldots, c'] = [\top_{\mathcal{C}}, \ldots, c] \circ [\_]$ which again is a recursive operation on recursive term structures. Finally, subsumption can be defined by means of implication. The advantage is that inheritance is implicitely carried out by a theorem prover. Nevertheless, the notions of *is_a_superconcept_of* and

is_also_about suggest two different readings, namely to represent $c \sqsupset_{\mathcal{C}} c'$ either as c`:-`c' or c'`:-`c. Since the used learning algorithm is based on the principle of inverse entailment we chose the former method; that is subsumption is represented as c`:-`c' (*is_also_about* is realized by reading '`:-`' as implication '$\leftarrow$'; the procedural interpretation realizes *is_a_superconcept_of*).
The category hierarchies $\mathcal{C}$ and $\mathcal{T}$ are represented as a set of Horn clauses which model inheritance through entailment. This means that the intuitive subsumption relation *is_a* is realized as logical implication. Other representations by lists with node names carrying the whole path and thus allowing for checking subsumption by comparing difference lists or explicit modeling of inheritance are not suitable for our learning approach.
A relation $c \sqsupseteq^1 c'$ is transformed into a Horn clause $c(X)$`:-`$c'(X)$. Thus, a document d concerning c *also_is_about* c'. From the procedural viewpoint, `?-`$c(d)$ succeeds if we have direct evidence for d dealing with c or if the proof `?-`$c'(d)$ succeeds.
This initial representation scheme has to be refined in order to meet two important requirements: First, by rule of resolution, this means that for any d, $\top_{\mathcal{C}}(d)$ holds, which is not desireable. Second, user models will demand a minimum confidence of classification confidences as shown in equation 4.2 and 4.3. Thus, generalization along entailment has to be penalized by a special predicate `genpenalty` which implements a threshold that is realized by u as defined in definition 4.6. Accordingly, the above represenation would yield a rule

$$c(X, C) \texttt{ :- } c'(X, C'), \texttt{genpenalty}(C', C). \tag{4.8}$$

and `genpenalty` is defined such that $C < C'$. One method for defining this penalty is to choose

$$\texttt{genpenalty}(C', C) \text{ iff } C = \frac{C'}{\mathsf{cost}(C', C)} \tag{4.9}$$

a stronger threshold would be defined by $\mathsf{up}(C', C)$ instead of $\mathsf{cost}(C', C)$. We will discuss the issue of penalties in detail in the section 7.2 about using the induced user models for document filtering and in section 7.2.1 about proof models for relevance levels. According to our concept hierarchy, the tree is represented as a set of Horn clauses as shown in figure 4.3. Representing $\mathcal{T}$ is carried out analogously.
background knowledge remains to be represented; that is the set of known document classification data represented as facts. This is shown in figure 4.4.

4.3.2 Representations of conceptual descriptions

Now, given a canonical method for representing concept hierarchies as Horn clauses, we need to express conceptual descriptions using Horn clauses as well. Consider again equation 4.3:

$$\bigsqcup_i \sqcap_j (t :: c :: p)_{i_j}.$$

```
cat_..._cs(X,D) :-
  cat_..._cs_programming(X,C), genpenalty(C,D).
cat_..._cs_programming(X,D) :-
  cat_..._cs_programming_languages(X), genpenalty(C,D).
cat_..._cs_programming_languages(X,D) :-
  cat_..._cs_programming_languages_procedural(X), genpenalty(C,D).
```

Figure 4.3: Representing concept hierarchies as Horn clauses

```
type_..._publication_researchpaper(urlid_5121,68).
cat_..._intelligence_machine_learning_symbolic(urlid_5121,92).
cat_..._intelligence_nat_lang_proc_generation(urlid_5121,78).
cat_..._intelligence_machine_learning_subsymbolic(urlid_5121,20).
```

Figure 4.4: Representing classification data

where t is the type interpreted as a sort, c is the category and p is a threshold that is interpreted as a lower bound for classification confidences. Types t and categories c are interpreted as binary predicates which are defined as shown in figure 4.4. Thus, an aspect i containing $t_1 :: c_1 :: p_1 \sqcap t_2 :: c_2 :: p_2$ for a user u means, that u is interested in documents that deal with c_1 with a confidence of at least p_1 **and** c_2 with a confidence of at least p_2. In principle, the same holds for the document type, but we do not allow for multiple assignments of text sorts to documents.[7] Compiling the sorts of $\mathcal{T}$ into predicates that are used within the Horn clause representation of $\mathcal{C}$ results in [8]:

$$t_1(D, \_) \wedge c_1(D, p_1) \wedge t_2(D, \_) \wedge c_2(D, p_2)$$

Relevance of d will be shown by a successful proof (i.e. derivation of an empty clause) against the background knowledge as shown in figure 4.4.

Aspects.* So far, we have motivated, that the user model consists of a set of Horn clauses. Taking into account our idea of dividing the user's interest into several more specific aspects of interest, the set of clauses should be divided into aspects, too. This will be explained in detail in section 5.1 which also introduces the explicit modeling of

[7]This is due to implementational issues. Currently, the text type classifier is not capable of delivering a vector of the n most probable document types for the document d under consideration. Sadly, this actually makes our formalization more complicated. If we applied the same methods we use for categories to types as well the formalization would in fact become easier though not less complex in terms of computational effort. See also footnote 4.

[8]Note, that the confidence of document type classification is omitted here, though it is actually available (again, see figure 4.4).

disinterest. Accordingly, the user's interest will be represented as a program consisting of rules as follows:

$$\texttt{interest}_u(\_, D) \texttt{ :- } t_1(D, \_), c_1(D, P_1), t_2(D, \_), c_2(D, P_2), P_1 \geq p_1, P_2 \geq p_2. \tag{4.10}$$

Discrimination between different aspects is obtained by simply introducing aspect identifiers a which determine unification:

$$\texttt{interest}_u(a, D) \texttt{ :- } t_1(D, \_), c_1(D, P_1), t_2(D, \_), c_2(D, P_2), P_1 \geq p_1, P_2 \geq p_2. \tag{4.11}$$

Talking about a single user u, the interest as modelled by the predicate `interest` has the signature $\texttt{interest}_u$ which is defined as follows:

$$\texttt{interest}_u \subseteq A \times \mathfrak{U} \tag{4.12}$$

where A is the set of possible aspects a of the user's u interest and $\mathfrak{U}$ is the set of all documents (represented by their unique id's).
In other words, $\texttt{interest}_u(a, D)$ succeeds for an instantiation d of D where d is of type t_j and belongs to c_j with a confidence of at least p_j. This would be interpreted as '*d is interesting for u with respect to u's interest aspect a*'. A set of clauses as in equation (4.11) establishes the definition of an aspect a; sets of those aspect definitions form the user model of u.
Now, it is obvious that the restriction to one document type implies a severe drawback. The clause displayed above can only be satisfied if either $t_1 \sqsupseteq t_2$ or $t_2 \sqsupseteq t_1$.
But as already pointed out in footnote 7, our formalization can easily be extended to multiple document types once we are provided with a suitable classifier. On the other hand, a clause as shown above is not actually intended as a manual user modeling formalism but rather as a scheme which is used as a bias for inducing more general rules describing the user's interest. This is the topic of the next chapter.

Chapter 5

INDUCING CONCEPTUAL USER MODELS

In the last section we have shown how conceptual user models can be represented as Horn clauses. We have demonstrated the underlying concept hierarchies in which inheritance is modeled by cost based inverse entailment and we have already pointed out the way in which user models as Prolog programs can be used for document filtering.
In this section, we will discuss how such user models can be induced given appropriate evidence in the form of feedback **f**.
Note, that this approach and the following evaluation is based on several assumptions and focuses on a special aspect within the whole approach of user adaptive web search:

[A-1]: Knowledge about documents and knowledge about users is strictly discriminated. Although both documents and user interests are described by categories, classification knowledge is not available to the user modeling procedure and vice versa. User modeling approaches which integrate knowledge about documents and users are mainly based on phrase occurrence. A user model for u's interest is a representation of a prototypical[1] 'most interesting' document d_u. Accordingly, relevance of a new document is shown by similarity: *Document d is interesting for u if d is similar to d_u.*
Of course, 'similarity' is a user dependent as well (see the discussion of 'relevance' and 'interestingness' below): Two documents, whose content is similar to user u_1 need not be similar to user u_2. In contrast to this statement, similarity measures used in the above-mentioned approaches are **static** in the document space and are **not** user dependent.
Therefore, we argue, that knowledge about such a similarity between documents is not an adequate means to formalize user models. User models shall contain information about the users interest but not the way in which knowledge about the domain is represented in a user independent part of the system.

[1]The prototype d_u needs not actually to be a real, existing document, but may have been derived as an average of documents previously classified as interesting by the user.

We deliberately chose this approach in order to avoid quality statements about user models which depend on the notion of similarity between documents, although such additional knowledge makes induction of user models an easier task.
Similarly, knowledge like *If user u is interested in d, d most likely belongs to category c* is not used within the classifiers.

[A-2]: 'Relevance' and 'Interestingness' are synonyms. Following the usual (common sense) understanding, 'relevance' is somehow orthogonal to 'interestingness'; especially in the context of information retrieval.
According to the considerations in `A-1`, different features of the object of our domain are of different *relevance* to the user. For example, some users may focus on searching for a certain document type instead of document content; while for other users content is more relevant to their interest than type.
In this case, both relevance and interestingness are user dependent; though on different scales: Examining a user model d_u as in `A-1`, a representation of relevance corresponds to a user specific weighting of the components of the vector d_u: $d_u = \langle r_1 \cdot f_1, r_2 \cdot f_2, \ldots, r_n \cdot f_n \rangle$. Accordingly, one says that *features are of different relevance to a user's interest.*
Interestingness of d is—as already described in A-1—measured by d in relation to d_u. A simple measure would be a Hamming distance. A more precise picture of interestingness can be obtained by taking into account personal relevance weights: When comparing components of the vectors d and d_u, matching components are weighted by their relevance r_i, thus yielding a higher score in the Hamming distance measure.

In our approach, however, 'relevance' is not visible to the user model, since the classification of documents is encapsulated and opaque to the user model. In contrast to the information retrieval community, 'relevance' in the context of user modeling is used as in the term *relevance feedback*. Relevance feedback given by a user u for a document d means, that d is relevant with respect to u's interest. In other words, d is an example for the users interest, which is interpreted as: *u is interested in d.*[2]

[A-3]: Feedback is strictly discriminated from labels. Feedback ('*relevance feedback*'; see A-2) is given with respect to documents: "*This document is interesting with respect to my interest aspect a.*" As a consequence of discriminating knowledge about documents and knowledge about users, labels are generated by interpreting this feedback as relevance feedback with respect to categories. A document d which is positive evidence for the user's interest and which belongs to category c, will result in an example which states that c supports the model of the user's interest.

[2]This assumption does not apply to the relevance values that can be manually added to the user aspects and which can be used for filtering. See footnote 8.

At this point it becomes clear, that the learning task we are going to analyze is much harder than learning a user's interest by comparing the documents themselves: A single document is classified into several categories simultaneously. For example, a document d about 'machine learning for user modeling' might be classified as a document about c_1 ='machine learning' with a confidence of 65%, while it is also classified as a document about c_2 ='user modeling' with a confidence of, say, 75%. If the user u gives relevance feedback which states, that d is interesting to u, then *both* c_1 and c_2 will be labeled as positive evidence for the user's interest. It is clear, that for documents which are 'heterogenous' according to the underlying category hierarchy, the sample generated out of the feedback is inherently 'noisy' in terms of machine learning.

[A-4]: Learning user models from scratch. The description of our application domain, web search, and its realization within the web search engine OySTER, were motivated by the idea that users should be able to investigate the user model the system has induced. This idea was motivated by the commandment that user models should be scrutable to the user (see section 2.1.4) and has been realized in the user modeling editing interface (see figure 3.2) in section 3.2.2).
This methodology might suggest, that the user is *required* to define an initial user model which is *refined* during the learning process. Actually, this method would pose a much easier learning problem. On the other hand, we try to meet the requirement of not bothering the user with tedious work. As a consequence, our approach is based on the assumption, that *no* prior knowledge is available. This means, that our user modeling problem can be described by the aim to learn user models from only a few examples with no further knowledge about the user.
A compromise between both approaches would be to require the user to submit a list of interesting links (a bookmark file). Such a list can be interpreted as a set of positive feedback which most likely contains a considerable amount of data (instead of only five examples).

[A-5]: Evaluation. Learning user models is incorporated into an (user adaptive) information retrieval scenario, but is encapsulated and evaluated as an isolated part.
In A-1 to A-3 we have explained, that the problem of learning a user model can be properly dissected from its surroundings. This allows for the evaluation of the user modeling process as a machine learning problem. Based on a sample (instead of feedback), we induce a user model which is independent of the internal representation of the domain and the way that documents are mapped onto the underlying categories. Even more, the learning process does not require any additional information nor does our approach presuppose certain properties which restrict the application domain. As a consequence, the results obtained can easily be outstripped by approaches which either take into account more information (for example, word occurrences) or impose severe restrictions on pos-

sible application domains due to prerequisites (for example, independence of features).

This section first defines user modeling samples which are used to learn user models that represent a user's interest (see A-1 and A-2). After that, the generation of samples from feedback (see A-3) is described. This allows the definition of a new learning problem (A-4) which is evaluated according to A-5 in the next section.

5.1 A more detailed investigation of representing user models

The aim of the presentation of conceptual user models in the last chapter was to demonstrate the feasibility of representing user models as Horn clauses. However, thinking towards a working user model induction process and thus having in mind a feasible learning problem, one needs to refine the rough idea of modeling a user's interest a bit further.

5.1.1 Interest, non-interest and disinterest

One of the most prominent problems within user model induction is sparse negative feedback. This leads to several problems:

1. one would need to employ a learning algorithm which works for little data
2. one would need to employ a learning algorithm which works for positive data only (as for example shown in [Schwab et al., 2000a, Schwab et al., 2000b])
3. the algorithm will tend to overfitting hypotheses
4. the user model will be very pessimistic since the proof that documents are not interesting by negation as failure will succeed very often for very accurate hypotheses

Thus, we introduced the novel approach of explicitly modeling a user's 'disinterest'[3]: By interest, we denote what the user is interested in. If something is not of interest or if it is of non-interest, it means that we have no evidence that the user is interested in this entity. By disinterest, we refer to concepts the user explicitly is not interested in. And in parallel, if something is not of disinterest we have no evidence that the user dislikes the concept.
At a first glance, this notion seems to complicate rather than to help in the solution of the above problems: All we have done so far is to prepare the ground for defining a new, kind of dual learning problem. The nice thing about dual problems however, is that evidence can be mutually interchanged under certain circumstances. We will explain this method in detail in section 5.2 about generation of feedback samples from evidence. As

[3]We chose this word in analogy to 'dis–like'.

an idea, one might consider, that (positive) evidence for disinterest is negative evidence for interest and vice versa.

User models for (dis–) interest. Again, consider figure 4.1. Here differently shaded nodes or subtrees were already used to visualize concepts that were of interest or which were known not to be of interest. One also could interpret non–interestingness in this figure as explicit disinterest. This leads to the following definition:

Definition 5.1 (User model with respect to (dis–) interest, $\mathbf{M}_u$) *A user model $\mathbf{M}_u$ is a tuple $\langle M_u^+, M_u^- \rangle$, where M_u^+ models the user's interest and M_u^- models the user's disinterest. M_u^+ is formalized using a binary predicate $\mathtt{p\_interest}_u$; M_u^- is formalized using a binary predicate $\mathtt{n\_interest}_u$.*
Formally, aspects a are subsets of $\mathbf{M}_u$: $\mathbf{M}_u(a) = \langle M_u^+(a), M_u^-(a) \rangle$ where each aspect is meant to represent a special topic of interest (see section 3.2.2). Interest of u with respect to aspect a is represented by the binary predicate $\mathtt{p\_interest}_u(a, \_)$; disinterest by the according binary predicate $\mathtt{n\_interest}_u(a, \_)$.

The following clause is an element of $M_u^+(a) \subseteq M_u^+$:

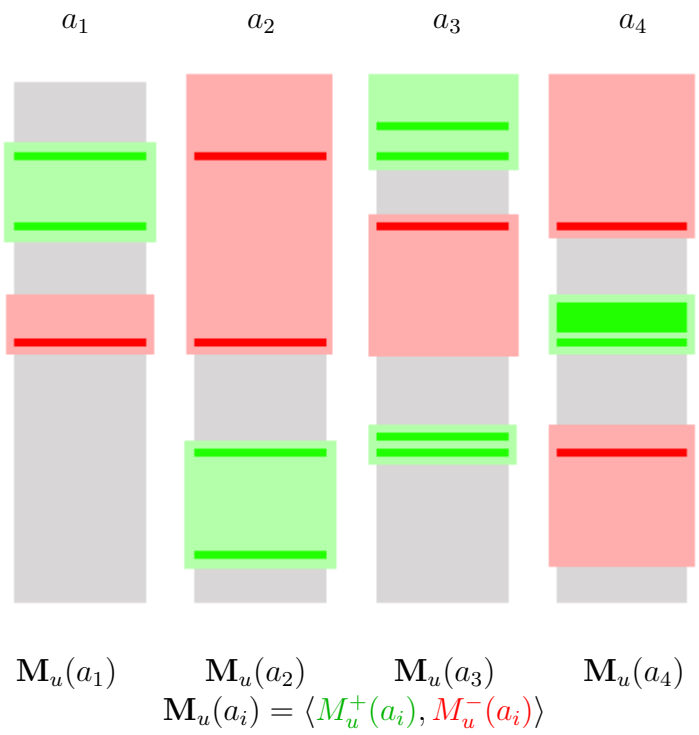
$$
\begin{array}{l}
\mathtt{p\_interest}\_u(a, D) :- \\
\quad \mathtt{type}\_\_t_1(D, T_1), ..., \mathtt{type}\_\_t_{n_t}(D, T_{n_t}), \\
\quad \mathtt{cat}\_\_c_1(D, C_1), ..., \mathtt{cat}\_\_c_{n_c}(D, C_{n_c}), \\
\quad \mathtt{thresh}(T_1, \vartheta_1), ..., \mathtt{thresh}(T_n, \vartheta_n), \mathtt{thresh}(C_1, \vartheta_{n+1}), ..., \mathtt{thresh}(C_n, \vartheta_{2n}).
\end{array} \tag{5.1}
$$

where u is the user id, a the aspect id and D is instantiated with the id d of the document currently under consideration. Document types (t_i) and categories (c_j) are assigned confidence values tc_i and cc_j, respectively. Finally, `thresh`olds can be defined in order to require a certain value V_k (one of $\{tc_1, ..., tc_{n_t}, cc_1, ..., cc_{n_c}\}$) to be greater than a certain boundary ϑ_k.[4] Currently, `thresh` is realized by the two relations $<$ and $>$ as already motivated in equation 4.11.
Note, that explicit modeling of the user's disinterest is a rather novel approach in user modeling ([Widyantoro et al., 1999] use three different kinds of feature vectors to describe long– and short term interest where the short term is described by both explicit interest and disinterest). Most of the time, $\mathbf{M}_u$ is identified with M_u^+ and M_u^- is regarded as negative evidence with respect to M_u^+.
For the sake of brevity, $\mathbf{M}_u(a)$ shall denote the set of all clauses unifying either

$$\mathtt{p\_interest}\_u(a, D) \text{ or } \mathtt{n\_interest}\_u(a, D)$$

for an aspect a; i.e. $\mathbf{M}_u(a) = M_u^+(a) \cup M_u^-(a)$. Figure 5.1 shows a user model with its

[4] Thresholds on type classifications (i.e. $tc_1, ..., tc_{n_t}$ which correspond to $\vartheta_1, \ldots \vartheta_n$) have not proven to be a very useful tool for appropriate bias because the type classifiers used in the prototype did not deliver weighted classifications but only a 'winner–takes–all' classification. For our evaluation, however, data was simulated (see the next chapter) thereby bybassing the flaws of the document type classifier; c.f. footnote 7.



For this visualization, we assume a content based ordering of documents of the domain along the vertical dimension of our picture.

The user's u interest is modeled by four aspects $a_1, \ldots, a_4$ (columns from left to right). Relevance feedback is given in relation to each aspect; brightly colored bars represent positive (green) and negative (red) feedback. As one can see, feedback is not given with respect to all documents or aspects.

Each aspect $\mathbf{M}_u(a_i)$ of the user model $\mathbf{M}_u$ consists of a model of interest and disinterest: $\mathbf{M}_u(a_i) = \langle M_u^+(a_i), M_u^-(a_i) \rangle$. They are represented by faded color regions around the relevance feedback.

Interest $\mathbf{M}_u^+(a_i)$ is modeled by the predicate `p_interest_`u`(`a_i`,D)`, while disinterest $\mathbf{M}_u^-(a_i)$ is modeled by a predicate `n_interest_`u`(`a_i`,D)`.

Note, that the models are much bigger than the region supported by rated documents. This is due to the generalization and sample enlargement procedure (see figure 5.2). Note also, that in our implementation feedback and models cover only a very small fraction of the whole domain.

Figure 5.1: A user model with aspects

parts describing interest and disinterest and different aspects.
Relevance of documents with respect to aspect is not neccessarily 'disjoint': Of course, the same document can be interesting to several aspects, if the aspects are 'similar'. For our formalization, we need assume that feedback is 'disjoint'. This restriction is required for using feedback given with respect to aspects a and a' as a basis for larger samples which include relevance information from a as examples for a' and vice versa. For our evalution we used feedback data which violates this assumption (relevance as indicated by feedback may overlap for different aspects). In consequence, the problem we will tackle is harder than the problem our methods are designed for.

5.1.2 Subsumption, implication and entailment

In the previous paragraphs we have informally described both the input data and bias information which can be used for inducing user models. The background knowledge consists of $\mathcal{T}$ and $\mathcal{C}$ (see figure 4.3) and document classification data (as shown in figure 4.4). An abstract definition of the sample is a subset of $\texttt{interest}_u$ as defined in equation (4.12); the target scheme is shown in equation 4.11.
In detail, any hypothesis for the target $\mathbf{i}_u$ consists of a set of clauses with head literals which unify with $\texttt{p\_interest\_}u(a, D)$ or $\texttt{n\_interest\_}u(a, D)$. Admissible body literals are specializations of $\texttt{type\_}\ldots$ and $\texttt{cat\_}\ldots$ literals (as included in the background knowledge). Furthermore, $<$ and $>$ are admissible body literals, if their first arguments are bound to variables which occur in type or category declarations. This yields a target clause as shown in equation 5.2.
Consider now evidence of interest for some document d: $\texttt{p\_interest}_u(a, d) \in \mathbf{f}_u$. Assuming that

$$\mathbf{A}_{\mathcal{T}\times\mathcal{C}}(d) = \langle t :: p, \langle c_1 :: p_1, c_2 :: p_2, c_3 :: p_3\rangle\rangle$$

this yields a most specific hypothesis clause:

$$\begin{aligned}&\texttt{p\_interest\_}u(a, D) :- \\ &\quad \texttt{type\_\_}t(D, P),\quad \texttt{cat\_\_}c_1(D, P_1),\quad \texttt{cat\_\_}c_2(D, P_2),\quad \texttt{cat\_\_}c_3(D, P_3), \\ &\quad P \geq p, P_1 \geq p_1, P_2 \geq p_2, P_3 \geq p_3.\end{aligned} \tag{5.2}$$

This clause can be generalized using three different methods:

1. decreasing the constants $p, p_1, \ldots, p_3$ means that the confidence required on the concept classification is decreased

2. dropping pairs of literals determining the variables $P, P_1, \ldots, P_3$ means to abolish requirements concerning the type t or categories c_1, c_2, c_3, accordingly

3. replace some $\texttt{cat\_\_}c_i(D, P_i)$ by $\texttt{cat\_\_}c_j(D, P_j)$ where $\texttt{cat\_\_}c_j(D, P)\texttt{:-cat\_\_}c_i(D, P)$ can be unified with a subset of a clause from Σ.[5]

[5]Type restrictions $\texttt{type\_\_}c_i(D, P_i)$ are treated analogously.

All three methods, with increasing importance for rule induction are performed during rule induction by inverse entailment. Thus, a hypothesis derived by inverse entailment subsumes this most specific clause and also entails examples as intended.
It remains to be explained, how feedback (in the sense of F) has to be interpreted in order to obtain a sample $\mathbf{f}$ which allows for inducing a sufficiently accurate subset of $\mathbf{M}$ which is to approximate $\mathbf{i}$.

5.1.3 Negation*

It is clear, that (explicit) modeling of both interest and disinterest gives rise to several questions concerning negation. On the one hand, there is no doubt that proving interest of a certain document by resolving a horn clause means, that the document needs to meet *all* requirements as represented by the body literals of the clause (see section 4.2.3). The reverse case, however means that a document is *not interesting* as expressed by the horn clause under consideration. As soon as one literal cannot be resolved, the proof for the clause fails. This means, that "not being interesting" can be shown by satisfying the disjunction of the negated literals. To make things even more complicated, negation in Prolog is simulated by negation as failure.
However, a failing proof for interestingness must by no means be confused with a succeeding proof for being *disinteresting*. In other words, for some documents we are able to prove that it "is non–interesting" (as opposed to "not being interesting"). In some cases this helps to overcome the drawback of a weak negation as realized by negation as failure for interestingness.
Now, one might argue, that conjunctive representations of user models are inappropriate for representing disinterest:

> "*I am interested in research articles on machine learning but not in homepages of researchers or students dealing with machine learning*"

In such a case we again stress the fact, that each aspect is represented by *sets* of clauses:

(5.3)

$$\texttt{p\_interest\_}u(a,D) :- \\ \quad \texttt{type\_\_...researchpaper}(D,T_1), \\ \quad \texttt{cat\_\_...machine\_learning}(D,C_1), \\ \quad \texttt{thresh}(T_1,\vartheta_1), \texttt{thresh}(C_1,\vartheta_2).$$

$$\texttt{n\_interest\_}u(a,D) :- \\ \quad \texttt{type\_\_...homepage...researcher}(D,T_1), \\ \quad \texttt{cat\_\_...machine\_learning}(D,C_1), \\ \quad \texttt{thresh}(T_1,\vartheta_1), \texttt{thresh}(C_1,\vartheta_2). \\ \texttt{n\_interest\_}u(a,D) :- \\ \quad \texttt{type\_\_...homepage...student}(D,T_1), \\ \quad \texttt{cat\_\_...machine\_learning}(D,C_1), \\ \quad \texttt{thresh}(T_1,\vartheta_1), \texttt{thresh}(C_1,\vartheta_2).$$

Although a Prolog program is a DNF representation, disjunction is representable as shown above. The apparant argument against our formalization is, that modeling dis-

interest this way becomes a tedious work, for one might have to list all possible combinations for the different document types (like in this example). At a closer look, this argument is a strong argument for our approach—and not against: Collecting two feedback events where one expresses disinterest in a researcher's homepage on machine learning and the second one disinterest in a student's homepage on machine learning enables the machine learning component to generalize by making use of the underlying type hierarchy: In this case *homepages* in general seem to be non–interesting, such that the final representation of the disinterest part of the aspect becomes:

$$\begin{aligned} &\texttt{n\_interest\_}u(a, D) : - \\ &\quad \texttt{type\_\_...homepage}(D, T_1), \\ &\quad \texttt{cat\_\_...machine\_learning}(D, C_1), \\ &\quad \texttt{thresh}(T_1, \vartheta_1), \texttt{thresh}(C_1, \vartheta_2). \end{aligned} \tag{5.4}$$

This general formulation of disinterest in homepages on machine learning can even be weakened again by adding the following clause (given there is evidence):

$$\begin{aligned} &\texttt{p\_interest\_}u(a, D) : - \\ &\quad \texttt{type\_\_...homepage...employee}(D, T_1), \\ &\quad \texttt{cat\_\_...machine\_learning}(D, C_1), \\ &\quad \texttt{thresh}(T_1, \vartheta_1), \texttt{thresh}(C_1, \vartheta_2). \end{aligned} \tag{5.5}$$

Of course, if we now encounter a new document which is classified as an employee homepage on machine learning, both interest (by the clause in equation (5.5)) as well as disinterest (by clause (5.4)) can be shown. Inheritance on $\mathcal{T}$, however, is penalized, such that the cost for proving disinterest becomes higher than the cost for proving interest (this is described in section 7.2).

5.2 Generating samples

In our framework, we assume that feedback F_u given by a user u will be used to construct a labeled sample $\mathbf{f}_u$ according to the users interest $\mathfrak{I}_u$ using a function Γ. We want to approximate the target function $\mathbf{i}_u$ which is induced by $\mathfrak{I}_u$ on our domain $\mathfrak{U}$ by a user model $\mathbf{M}_u$.

In contrast to user feedback $F_{u,a}$, which contains feedback with respect to aspects, feedback F_u does *not* contain any information regarding interest aspects[6]. Therefore, user feedback F_u actually only provides information that can be used to build a sample $\mathbf{f}$ as introduced in definition 2.26.

It is clear, that a sample for multiple aspects as it will be defined in definition 5.2 cannot be generated from feedback F_u as specified at the beginning of the next paragraph.

[6]In case one restricts user models $\mathbf{M}_u$ to consist of one aspect only, then F_u and $F_{u,a}$ contain the same information.

Therefore, we will consider two cases: In the first case, interest is interpreted as consisting of only one single aspect. In this case, F_u is sufficient[7]. For multiple aspect interests, we need feedback $F_{u,a} \subseteq \mathfrak{U} \times A \times V$, where A is the set of aspects.[8]
This section introduces Γ–functions, which allow for generating samples $\mathbf{f}_u$ (see definition 5.2 below) which carry much more information.
The feedback given by user u is a relation F_u between documents d and feedback values $V = \{-2, -1, 0, 1, 2\}$ representing strong and weak dis–/interest and indifferent interest as shown in the feedback form in figure 3.5. Note, that F is not a function of u for two reasons: u will not provide us with feedback concerning every document d. Thus, F is not total. Second, a single document d can be assigned different feedback values at the same time.
Furthermore, the sample $\mathbf{f}_u$ has to provide some more information. Recall, that relevance feedback is always given with respect to an aspect $\mathbf{M}_u(a)$. In other words, one single document d can occur several times within $\mathbf{f}_u$: It might be interesting with respect to a but is not interesting with respect to a'. Furthermore, $\mathbf{M}_u$ is divided into two separate models each of which defines an own learning target: M_u^+ ($\texttt{p\_interest}_u$) and M_u^- ($\texttt{n\_interest}_u$). Thus, samples $\mathbf{f}$ in our framework actually carry more information as described in equation 2.16:

Definition 5.2 (User modeling sample, $\mathbf{f}_u$) *A sample that is used for a learning problem in user modeling consists of a sequence of labeled pieces of evidence*

$$\mathbf{f}_u = \left[\langle d_1, a_1, v_1 \rangle^{\{+,-\}}, \langle d_2, a_2, v_2 \rangle^{\{+,-\}}, \ldots, \langle d_n, a_n, v_n \rangle^{\{+,-\}} \right] \tag{5.6}$$

where $d_i \in \mathfrak{U}$ and $v_i \in \{0, 1\}$, a_i are valid aspects in $\mathbf{M}_u$ and the superscript denotes the target M_u^+ or M_u^-.

Note, that according to definition 5.1, $\mathbf{f}_u$ actually includes label data that will be used for different learning targets, namely $M_u^+(a_i)$ and $M_u^-(a_i)$. The sample provides labels $v_i \in \{1, 0\}$ for documents d_j with respect to a whole set of, say m, aspects: $\{a_1, \ldots, a_m\} = \{a : \langle \cdot, a, \cdot \rangle \in \mathbf{f}_u\}$.

5.2.1 An example

To clarify the contents and information provided by feedback and different samples we have depicted an example in figure 5.2[9].[10⋆] The domain contains three documents

[7] Although we will have to introduce a single, new aspect in order to construct triples as in $\mathbf{f}_u$ instead of tuples as in $\mathbf{f}$.

[8] One can interpret F_u as a set $F_{u,a}$ where the aspect dimension in figure 5.2 contains only one aspect.

[9] Feedback for document d_1 shows that aspects are disjoint (see end of paragraph 5.1).

[10] Please note, that this example is not to be mistaken as a definition. It is just to give the reader a rough idea of the following more formal sections on single and multiple aspect user models.

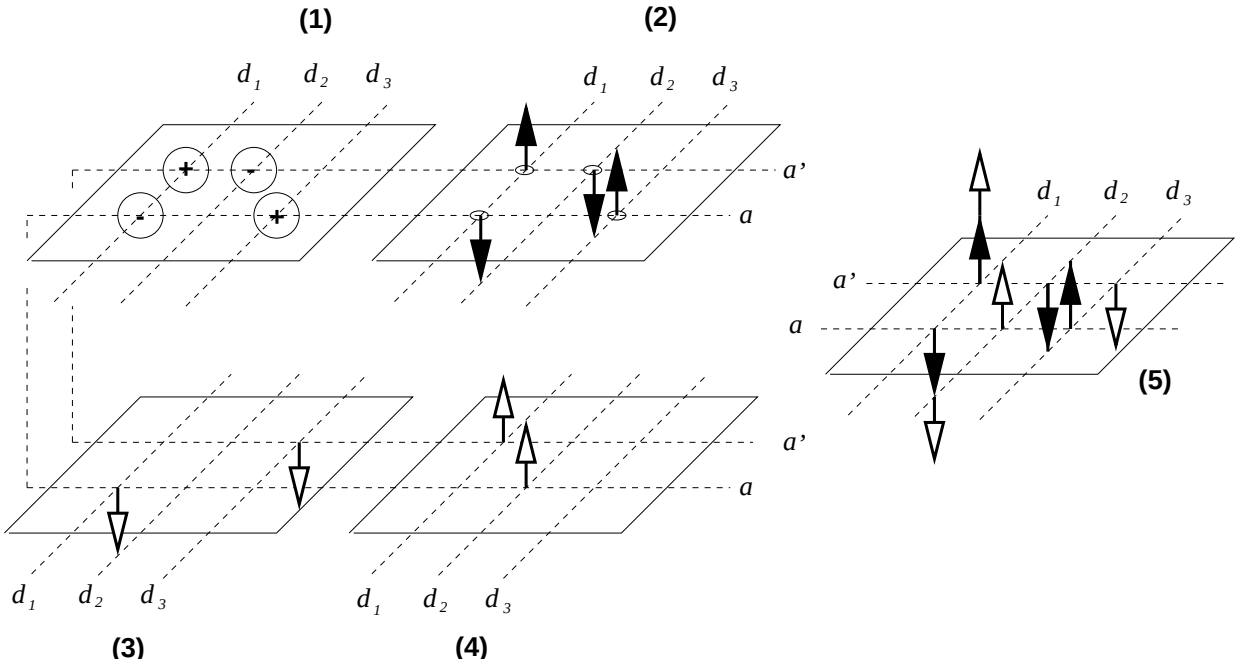


Figure 5.2: Feedback $F_{u,a}$ and a sample $\mathbf{f}_u$

d_1, d_2, d_3. The learning target is split into two aspects a, a'.

Part (1) shows feedback as given by the user: A plus sign (+) represent a positive label while a minus sign (-) represents negative feedback. Obviously, document d_1 is interesting with respect to a', but not with respect to aspect a. Document d_2 on the other hand, is not interesting with respect to a'; its relevance to a is unknown. The third document is interesting for a, but relevance with respect to a' is unknown. For some documents there is only positive (d_3) or only negative evidence (d_2). As one can see, $F_{u,a}$ does not describe the whole instance space: There are many 'gaps' in the sample especially if one considers that we have explicit models of interest and dis–interest. Note, that for m aspects and n documents under consideration, the feedback relation F_u most likely will *not* contain $m \cdot n$ tuples. It is much more likely, that the user provides us with partial feedback only such that—especially in the domain of sparse user feedback—we need to grasp for every straw in order to collect more evidence about the user's interest. From this feedback data we want to derive a sample as defined in definition 5.2.

In a first step, (2), the given feedback is mapped one–to–one onto labeled examples. Upward pointing bars indicate feedback with respect to M_u^+, while downward pointing arrows indicate relevance for disinterest. As an example, consider document d_3: Since it was rated interesting with respect to a, we have positive evidence for $M_u^+(a)$. But this also means, that d_3 represents negative evidence for $M_u^-(a')$. One might claim, that

from the feedback gathered we might conclude that d_3 is a negative example for $M_u^+(a')$ as well: The same holds for d_1 and $M_u^-(a)$. This is shown in part (3).
Thinking a step further, one might even guess, that d_2 might be of interest with respect to a—since it is not of interest with respect to a'. The same argument applies to d_1 and a', see part (4). Of course, this is a very vague guess which may rather cause noisy samples instead of large, reliable samples.
Putting it all together we yield a sample as shown in part (5). Note, that in this figure, we have not depicted whether feedback for some d with respect to a is interpreted as evidence for M_u^+ or M_u^-. This will be formally defined in the following sections.
Now having defined the sample we need for learning user models the question arises how to generate those samples from feedback. First, we consider single aspect user models only.

5.2.2 Single aspect user models

The single aspect user model is a user model which contains only one single aspect in M_u^+ or M_u^-. For reasons of brevity, we only consider M_u^+; the dual case can be constructed canonically.
It is a straightforward idea to use extrema from F_u only since they provide the most reliable information about the user's interest. Thus, we generate a sample $\mathbf{f}_u$ from F_u using a simple function Γ which is defined as follows:

Definition 5.3 (Sample Generation, Γ) *We generate a sample from given feedback using a function $\Gamma : F_u \to \mathbf{f}_u$ and a new aspect id a (see footnote 7):*

$$\Gamma^+(a)(\langle d, v\rangle) = \begin{cases} \langle d, a, \mathtt{0}\rangle^+ & \text{for } v = \min(V) \\ \langle d, a, \mathtt{1}\rangle^+ & \text{for } v = \max(V) \end{cases} \tag{5.7}$$

In our case, the extrema in V are -2 and 2. Note, that $\mathbf{f}_u$ now contains only extreme feedback, since Γ is undefined in all other cases.

Since $\mathbf{f}_u$ is most likely too small to define a feasible learning task, we can weaken the condition such that all negative feedback is interpreted as negative examples and all positive feedback as positive examples:

$$\Gamma^+(a)(\langle d, v\rangle) = \begin{cases} \langle d, a, \mathtt{0}\rangle^+ & \text{for } v < 0 \\ \langle d, a, \mathtt{1}\rangle^+ & \text{for } v > 0 \end{cases} \tag{5.8}$$

Now, the sample generated by Γ from equation (5.8) contains the sample as generated using definition (5.7), but the information is less reliable.
We still have much more information available to generate further, larger samples. Recall, that the user's interest in $\mathbf{M}_u$ is modeled by two sets of Horn clauses M_u^+ and M_u^-, which

both represent a separate learning problem on the same set of examples. Clearly, positive examples in $\mathbf{f}_u$ are positive evidence for M_u^+, that is for the target predicate $\texttt{p\_interest}_u$. Similarly, negative examples from $\mathbf{f}_u$ are negative evidence for $\texttt{p\_interest}_u$. This is the standard interpretation for samples; though sample interpretation is independent of sample generation, we will denote this case by Γ^+ (meaning that Γ produces a sample for the target concept M_u^+).
Since we learn both interest and disinterest simultaneously, we can use same example from $\mathbf{f}_u$ twice. In addition to the considerations above, it is quite reasonable, that the reverse case holds for the target $\texttt{n\_interest}_u$. Thus, an example $\langle d, a, \texttt{1}\rangle$ is a positive instance for $M_u^+(a)$ *and* a negative instance for M_u^-. The dual case holds for examples $\langle d, a, \texttt{0}\rangle$. Interpreting the sample as input for the learning target M_u^- will be denoted Γ^-, respectively. Γ^+ and Γ^- together form $\Gamma^\pm$.
Since it is likely, that there is only a little negative feedback in $F(\mathbf{i}_u)$, we only have a few examples for $\texttt{n\_interest}_u$ and only a few examples against $\texttt{p\_interest}_u$. Thus one might conclude that the hypothesis for $\texttt{p\_interest}_u$ has a bad accuracy, i.e. is too general, while it will be hard to find a hypothesis for $\texttt{n\_interest}_u$ at all (and if there is any, it will be a set of overfitted clauses).
Thus, we need to take into account feedback that was given with respect to different aspects.

5.2.3 Multiple aspect user models

A document d which is interesting for u with respect to an interest aspect a is most likely non–interesting for another aspect a'[11]. Since we want to prove relevance of d with respect to aspects, a hypothesis for $M_u^+(a')$ should exclude d. Relevance of d shall only be provable by means of $M_u^+(a)$. This leads to the idea of using feedback $\langle d, a, v\rangle \in F_{u,a}$ that was given with respect to a certain aspect a as feedback for a' as well. Figure 5.3 shows different ways to interpret relevance feedback as feedback that will be provided by $\mathbf{f}_u$. Note, that $\oplus$ and $\ominus$ are interpreted differently according to whether Γ was defined as in equation 5.7 or 5.8. The upper part of the table describes $\Gamma^\pm$ as described in the previous paragraph. Taking into account relevance feedback with respect to other aspects a', $\mathbf{f}_u$ can be enlarged by decreasingly reliable information as provided by $\Gamma_1, \ldots, \Gamma_4$.
Γ_1 simply states the fact, that aspects are exclusive. In other words, positive evidence for relevance of d with respect to a' is negative evidence for interest in a. Γ_2 is a kind of Γ_1^-: positive evidence for interest in a' is interpreted as positive evidence for disinterest in a. This inference is not as reliable as Γ_1. Therefore, Γ_2 should only be considered in cases, where there is very little labeled data available for M_u^-.

[11] This assumption is based on the general idea of aspects: they describe rather disjoint fields of interest which together form a whole picture of the user's interest. On the other hand, this assumption has not been empirically validated.

	$\langle d, A, v\rangle \in F_{u,a}$		$\langle d, a, \mathbf{i}_u(d)\rangle \in \mathbf{f}_u$		
	A	v	$\mathbf{i}_u(d)$	target	
Γ^+	a	$\oplus$	1	M_u^+	$\texttt{p\_interest}_u(a, d)$
Γ^-	a	$\oplus$	0	M_u^-	$\texttt{:-n\_interest}_u(a, d)$
Γ^+	a	$\ominus$	0	M_u^+	$\texttt{:-p\_interest}_u(a, d)$
Γ^-	a	$\ominus$	1	M_u^-	$\texttt{n\_interest}_u(a, d)$
Γ_1	a'	$\oplus$	0	M_u^+	$\texttt{:-p\_interest}_u(a, d)$
Γ_2	a'	$\oplus$	1	M_u^-	$\texttt{n\_interest}_u(a, d)$
Γ_3	a'	$\ominus$	0	M_u^-	$\texttt{:-n\_interest}_u(a, d)$
Γ_4	a'	$\ominus$	1	M_u^+	$\texttt{p\_interest}_u(a, d)$

The `:-` –sign is used to identify negative examples by the Progol system (section 2.2.5).

Figure 5.3: Generating samples from feedback

Γ_3 states that an uninteresting (with respect to a') document d is not necessarily uninteresting for aspect a. Thus, it is (weak) negative evidence for $M_u^-(a)$, too. However, such documents d *can* be interesting for $M_u^+(a)$. This is stated by Γ_4, but is not reliable at all and prone to generate noise (thus, it is only defined for reasons of symmetry). Accordingly, we extend our definition of Γ:

Definition 5.4 (Sample Generation with respect to aspects, $\Gamma(a)$) *Given a target aspect a from $\mathbf{M}_u$ and another aspect $a' \neq a$ from $\mathbf{M}_u$ as well, we define $\Gamma(a)$ as follows:*

$$\Gamma_0^{\pm}(a)(\langle d, a, v\rangle) = \Gamma^{\pm}(\langle d, v\rangle) \tag{5.9}$$

$$\Gamma_1(a)(\langle d, a', \oplus\rangle) = \langle d, a, 0\rangle^+ \quad \textit{forall } a' \neq a \tag{5.10}$$

$$\Gamma_2(a)(\langle d, a', \oplus\rangle) = \langle d, a, 1\rangle^- \quad \textit{forall } a' \neq a \tag{5.11}$$

$$\Gamma_3(a)(\langle d, a', \ominus\rangle) = \langle d, a, 0\rangle^- \quad \textit{forall } a' \neq a \tag{5.12}$$

$$\Gamma_4(a)(\langle d, a', \ominus\rangle) = \langle d, a, 1\rangle^+ \quad \textit{forall } a' \neq a \tag{5.13}$$

5.3 A new user model learning problem

5.3.1 Motivation

Given a sufficiently large sample $\mathbf{f}$ of relevance feedback, user model induction on $\mathcal{T} \times \mathcal{C}$ is a (sloppily) supervised learning task. Since we neither need to cope with numbers, complex terms or recursive predicates which, e.g., makes application of Absorption–like operators much easier and since we have a strong bias on the structure of hypotheses we will be able to define a feasible learning problem.

As an intuitive motivation that is oriented on the more theoretical nature of inverse resolutionas described in section 2.2.4, we give a small example of the idea of ILP in this context. Note, that the formalization significantly differs from our considerations in sections 5.1 and 5.1.2 which was driven by the method of inverse entailment already. Thus, imagine the following initial user model:

```
interest(u, D) :-                    interest(u, D) :-
   class(D, machine_learning).          class(D, user_modeling).
```

Using classical truncation (c.f. 2.2.4), the induced user model would be `interest(u, D) :- class(D, X).` Since X is much too general we can specialize X to `machine_learning` $\sqcup_{\mathcal{C}}$ `user_modeling`. The same applies to sorted expressions, such that the body literals of

```
interest(u, D) :-                       interest(u, D) :-
   class(D, pers_hp::machine_learning).    class(D, group_hp::user_modeling).
```

would be generalized to

$$\begin{aligned} & (\texttt{pers\_hp} :: \texttt{machine\_learning}) \sqcup_{\mathcal{T}\times\mathcal{C}} (\texttt{group\_hp} :: \texttt{user\_modeling}) \\ = \; & \texttt{hp} :: (\texttt{machine\_learning} \sqcup_{\mathcal{C}} \texttt{user\_modeling}). \end{aligned}$$

Using Intra–Construction (see section 2.2.4), new concepts could be invented for describing user models: Given

```
interest(u, D) :-                       interest(u, D) :-
   class(D, pers_hp::machine_learning),    class(D, pers_hp::user_modeling),
   class(D, group_hp::user_modeling),      class(D, res_art::intelligent_agents),
   class(D, res_abs::web_search).          class(D, group_hp::machine_learning).
```

we can induce a user model with a body as follows:

$$\begin{aligned} & \texttt{class}(D, \texttt{hp} :: \texttt{machine\_learning} \sqcap_{\mathcal{C}} \texttt{user\_modeling}), \\ & \texttt{class}(D, \texttt{res} :: \texttt{new\_class}). \end{aligned}$$

which triggers two actions: Compute or ask for the value of

$$\texttt{machine\_learning} \sqcap_{\mathcal{C}} \texttt{user\_modeling}$$

and ask for a name for the newly invented class

$$\texttt{new\_class} = \texttt{intelligent\_agents} \sqcup_{\mathcal{C}} \texttt{web\_search}.$$

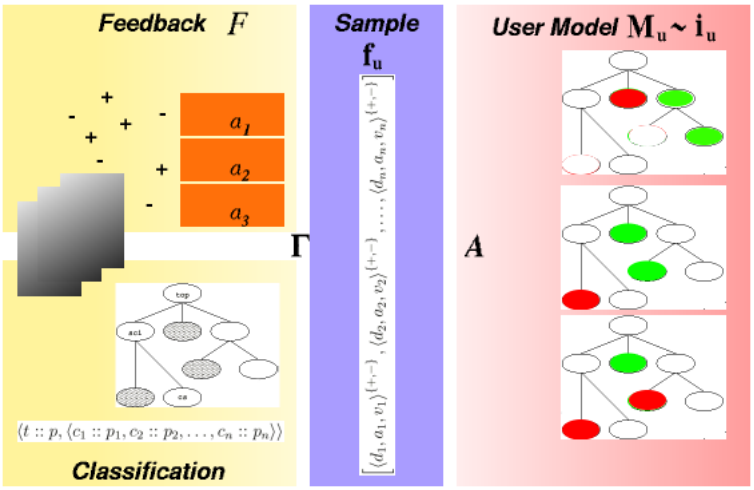


Figure 5.4: A more intuitive description of a user model learning task

5.3.2 A more abstract and formal description of the learning task

By now, we are able to define a clear learning problem. The learning problem is situated in a well defined setting of adaptive user interfaces. To illucidate the notion used in this thesis, we have put the most important concepts into figure 5.4. The user's u interest $\mathfrak{I}_u$ induces a target function $\mathbf{i}_u$ on our domain. The hypothesis to be learned by $\mathbf{A}$ is the user Model $\mathbf{M}_u$ which consist of the two target concepts M_u^+ and M_u^- which are further divided into aspects $M_u^{\pm}(a)$. In other words, $\mathbf{M}_u$ shall approximate $\mathbf{i}_u$. The input data is a sample $\mathbf{f}_u$ which is generated by Γ out of user feedback F_u. Examples from $\mathbf{f}_u$ are represented as positive and negative literals $\texttt{p\_interest}_u(a,d)$ and $\texttt{n\_interest}_u(a,d)$, respectively. Background knowledge provided to $\mathbf{A}$ are $\mathcal{C}$ and $\mathcal{T}$ as well as classification data for documents.

From a procedural point of view, figure 5.4 shows, that documents are represented by conceptual descriptions (lower left corner). They are represented as factual background knowledge. A user gives feedback (that is, by his interaction he realizes a function F) and assigns relevance values to documents with respect to his interest aspects (upper left corner). Γ functions are used to generate a feedback sample $\mathbf{f}_u$ (see center of the picture) from which the the learning algorithm $\mathbf{A}$ induces a user model $\mathbf{M}_u$ which shall approximate the target (that is, the real user's interest function $\mathbf{i}_u$; shown in the right hand part of figure 5.4).

As a learning algorithm $\mathbf{A}$ we chose ILP. Bias is provided using modes: Hypotheses for M_u^+ and M_u^- must be formulated using a clause head `p_interest` and `n_interest`,

respectively. Admissible body literals are any literals which describe concepts from $\mathcal{C}$ or $\mathcal{T}$ and threshold predicates. An example was already given in section 5.1.2.

Chapter 6

RESULTS

The aim of OySTER was to create a workbench which provided us with real–world data for evaluation. The data that was expected based on the fact that a search engine offers an attractive service in these days of information flood. The drawback of this idea however is that we underestimated the actual need for such a search engine. This resulted in a multitude of users who were searching for documents that were not located within our domain as covered by $\mathcal{C}$. Actually, OySTER got 'slashdotted' several times, when the service became public and was discussed on a German news web site in July 2000 (`http://www.heise.de/`).[1] Soon we were concerned with search queries for cheap car bargains or even requests for porn sites. As a result, the benefit of using OySTER as a source for user feedback turned into a drawback: Every–day search requests and feedback rather polluted the samples.

The evaluation of our approach was carried out in three steps: First, we performed a simple spot check on how the ILP methods perform in our domain. Section 6.2 summarizes results that were derived for two users (88 and 90) with rather precise interest and two users (92 and 93) with an increasingly vague description of interest.[2⋆] Our main task in this first evaluation was to achieve a rough impression of the number, accuracy and coverage of rules that were induced. Accordingly, factual knowledge (i.e. non–compressing rules) were not taken into account. The results are discussed by inspecting the user feedback. Having a rough impression in mind from the first spot check, we carried out a more detailed, statistical evaluation. In order to obtain significant results, we generated a set of fifty aspects, for each of which we carried out single and multiple aspect learning tasks by mutually combining the single aspects.
The second evaluation is an optimistic evaluation, where the feedback was noise–free and interest aspects were very specific. It shows both that accuracy increases with specificity

[1] The verb originates from the popular Unix–news site `http://www.slashdot.org` which often announces news and programs hosted by third parties. Due to the number of users who follow those links, the server sites often crash.

[2] See figures in appendix A.1.

of interest and with sample size and that the use of Γ functions helps to increase accuracy on small samples rather than on large samples. This evaluation is described in section 6.3.
Finally, a third evaluation was carried out in a worst-case setting: The feedback data was implicitly (multiple category assignment of documents) and explicitly (simulated classification errors) noisified. This test was carried out in order to determine a baseline for the performance of our approach and is described in section 6.4.

6.1 Simulating data for evaluation*

As already pointed out during the introducing remarks of this chapter, we needed to simulate users with approriate interests who give according feedback.[3]* The following sections describe, how data was simulated for the different evaluation tasks.

6.1.1 Domain data: Generating document classifications

In order to be able to simulate feedback, one needs to have documents that have to be rated. For this reason, we generated 10,000 URLs. All URLs are named `http://www.testurl.org/test/file-`n`.html`, where n is an integer from $[0, ..., 9999]$. The important part is to simulate classifications for each document which resembles classification data from real web documents.

6.1.1.1 Random data for a first impression of learning results

For our first test run, we *randomly* assigned document categories and types. Recall, that category classifiers are supposed to deliver a vector of classes and confidence values $\langle c_1 : p_1, c_2 : p_2, c_3 : p_3, ..., c_n : p_n \rangle$, where $p_j \leq p_i$ for $i < j$. In our approach, we restrict ourselves to a maximum of three classifications ($n = 3$). The document type instead is described by a single type identifier and a confidence value: $t : p_t$. Our test classification routine randomly defines c_1, c_2, c_3 and assigns confidence values as follows:

$$p_1 \in [100, 98, ..., 62],\ p_2 \in [60, 58, ..., 40],\ p_3 \in [20, 18, ..., 0].$$

p_t is randomly chosen from the interval $[62, 64, ..., 100]$.
In order to obtain a rough impression of the average URL specificity, we measured categroy 'entropy' by the average sum of δ–distances between c_i. The average distance sum for our test URLs is at about $\delta^t_{avg} = 27.11$. This corresponds, for example, to the distance between the categories *science* (second level) and *sports* (third level) but is still less than

[3] 'Appropriate' means, that we needed to restrict the domain of possible interests to the topics covered by our taxonomy.

the distances between *machine learning* and *science* (28) or *clustering* and *linguistics* (33).
After two months of collecting and classifying real web pages, the average sum of distances between the c_i is $\delta^r_{avg} = 17.27$ which corresponds, e.g. to the distance between *genetic algorithms* and *programming languages*. The longest distance measured within the *science* subtree turned out to be about 64, as for example for *clustering methods in machine learning* and *pragmatics in linguistics*. Larger values can only be obtained by completely crossing the tree at depth 2 already—*arterial gas embolism* (as a decompression sicknes in recreational sports diving) and *C* (as a procedural programming language in computer science) for example have a distance of 79.
Thus, our test URL classifications are of less quality (that is, 'noisier') than those delivered by our classifiers on real web documents after the two month harvesting period. This ensures that the results delivered by our machine learning algorithm perform better on real data than on simulated data.
Domain data simulated by this method was used in the evaluation described in section 6.2.

6.1.1.2 δ–specific simulation of real world documents

In addition to data that was generated as described in the last section, we generated a second set of URLs and feedback data sets in order to allow for a more detailed investigation.
Simulating URL classifications by random assignment of categories as described in the last section provides a setting for a pessimistic evaluation. On the other hand, random assignments contradict the idea of underlying category hierarchies such that we may assume a document classification diversity value of at most 17.27. As a consequence, a more realistic simulation requires documents with a more elaborate category classification.
For each category $c \in \mathcal{C}$ we defined a set of URLs d with

$$\mathbf{A}_{\mathcal{T}\times\mathcal{C}}(d) = \langle t :: p, \langle c_1 :: p_1, c_2 :: p_2, c_3 :: p_3\rangle\rangle$$

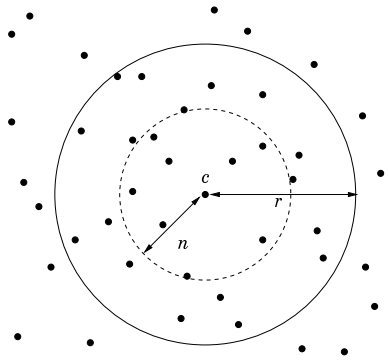
where $c_1 = c$ and $p_1 \in [85, 100]$. As in the first generation process, $t :: p$ is chosen randomly where $p \in [75, 100]$ (since we may assume independence of types and categories; see also footnote 1). $c_2 :: p_2$ and $c_3 :: p_3$ however are chosen in relation to c_1. Given c_1, we compute a 'circle' around c_1 with radius r by taking into account δ (see figure 6.1). Let δ_r denote the ordered sequence of categories $c_i \in \mathcal{C}$ for which $\delta(c, c_i) < r$. From δ_r, c_2 and c_3 are randomly drawn by a function *rand*:

$$c_{2,3} = \delta_{r_{2,3}}[rand(2, \min\{n_{2,3}, |\delta_{r_{2,3}}|\})]$$

The function *rand* delivers the index j of the j–th category in $\delta_{r_{2,3}}$, where $2 \leq j \leq \min\{n_{2,3}, |\delta_{r_{2,3}}|\}$.[4] Thus, n determines the number of n closest categories under consid-

[4]Lower bounding j to 2 ensures that $c_1 \neq c_{2,3}$.

eration which can be chosen by *rand*. Due to the distance measure on $\mathcal{C}$, the category



The 'radius' r determines the maximum δ distance around c. The number of closest categories which may be considered as candidates for c_2 or c_3 is defined by n (here: 8).

Figure 6.1: δ_r region around c

density is not uniformly distributed: The more special categories become, the more neighbors can be found within a radius r. In order to take this behavior into account, we only consider the n nearest categories
All in all we ran three generation procedures for three classes of URLs whose classification specificity in the average approximates real data:

1. Narrow class URLs: We generated ten URLs per c with $r_2 = 8$, $n_2 = 6$ and $r_3 = 18, n_3 = 12$. The average sum of $\delta(c_1, c_2)$ and $\delta(c_1, c_3)$ is approximately 12.40.

2. Average class URLs: For each c, we generated five URLs with $r_2 = 12$, $n_2 = 6$ and $r_3 = 24, n_3 > |\delta_{r_3}|$.[5] The average of $\delta(c_1, c_2) + \delta(c_1, c_3)$ is approximately 16.84.

3. Random class URLs: Here, the radius $r_{2,3}$ was defined as the maximum distance over $\mathcal{C}$ and $n_{2,3}$ was defined as the number of all known categories. Accordingly, this yielded a random distribution (the average sum of all $\delta(c_1, c_2)$ and $\delta(c_1, c_3)$ is 55.01 which pretty accuratly approximates the average class distance: $2\delta^t_{avg} = 2 \cdot 27.11 = 54.22$).

Confidence values $p_1 \geq p_2 \geq p_3$ are chosen as follows: $p_1 \in [85, 100]$, $p_2 \in [70, p_1]$ and $p_3 = [50, p_2]$.
The choice of r and n are—of course—subject to criticism. We chose both parameters as described above with respect to data from the real web: As already mentionend, the average distances between URL categories after two months of collecting documents

[5] Setting $n_3 > |\delta_{r_3}|$ means, that c_3 is chosen from all the candidates defined by the circle with radius r_3 (the number of all categories is $|\delta_{r_3}|$).

was at about 17.27. We tried to approximate this value by our 'average class URLs' by manually tuning r and n; yielding a pretty close value of 16.84. Additionally, we generated URLs with more specific and more vague classifications (see above); such that the overall average over all URLS accumulates to $(12.40 + 16.84 + 27.11)/3 \approx 18,78$ which again is slightly worse than real data and, therefore, a reliable foundation for our evaluation.

6.1.2 User Feedback: Simulating a user's interest

We assume, that the user's interest can be described in terms of concepts of our document category hierarchy.
Accordingly, user interests are simulated by (sets of) 'centroids', with each such centroid representing an individual center of interest (which is not to be confused with aspects). Single centroids correspond to single nodes of interest as shown in figure 5.4 (last aspect representation in bottom right corner). Interest aspects of different specificity are simulated by the number of centroids used per aspect as well as weighted δ–radius regions around the randomly picked categories. Aspects covering several categories are simulated by the union of several single centroids (therefore corresponding to sets of nodes as in the other aspects shown in figure 5.4).[6*] The actual parameters of simulation, that is specificity (radius), diversity (number of entroids per aspect) and additional noise are described along the line with each evaluation in sections 6.2–6.4.
It must be stressed that our simulation of feedback is pessimistic such that the applicability of our approach for real users can be assured within a realistic estimate.

6.1.2.1 A pessimistic ad–hoc simulation

The domain data for the first evaluation was generated randomly (see section 6.1.1.1). User feedback F (neither F_u nor $F_{u,a}$) cannot be chosen randomly. It cannot be defined using rules either—since rules are what we want to learn as $\mathbf{M}_u$. Thus, we generate user feedback with a δ-weighted random distribution over the document category hierarchy. This way, we obtain artificial noise and rough boundaries of feedback clusters around randomly chosen centroids.
For each user, one or up to three 'centroids' $c_k \in \mathcal{C}$ are randomly defined which can be interpreted as the 'center of interest'. Then, for each centroid c_k, 200 randomly chosen URLs (that is, 'documents') d_i are assigned a feedback value ranging from -2 to $+2$. The feedback value is determined by the distances $\delta(c_d(j), c_k) \cdot p(j)$, where $c_d(j)$ is the j–th category d has been assigned to by $\mathbf{A}_\mathcal{C}$. A small bonus is added

[6] If, for example, one wants to simulate an interest aspect "machine learning for user modeling" one would have to define *two* centroids: one is located near "machine learning", the other near "user modeling".

for document type agreement. The feedback value is then mapped onto the interval $\{-2,-1,0,1,2\} = \mathsf{cod}(F(\mathbf{i}_u))$.

An example is displayed in figure 6.2. Note, that the visualization veils some important information: Feedback (indicated by square brackets) is given with respect to documents. The picture in figure 6.2 shows feedback that is mapped onto categories. Since every document is represented by several categories and according classification confidence values, the figure was generated taking into account only the most reliable classification. If, for example, user 88 has rated a document d as very interesting, and $\mathbf{A}_{\mathcal{T}\times\mathcal{C}}(d) = \langle t :: p, \langle c_1 :: p_1, c_2 :: p_2, c_3 :: p_3\rangle\rangle$, then only c_1 is marked [++] in figure 6.2. As a consequence, this representation veils the fact, that d also belongs to c_2 and c_3 and is also a source for inherent noise (except for the case, that only c_1 will be taken into account; see section 6.1.2.3 and footnote 8).

[o][o][+]	3	1	1	top.science.computer_science.operating_systems
[−]	1	−1	−1	top.science.computer_science.operating_systems.dos
[−][+]	2	0	0	top.science.computer_science.operating_systems.unix
[−][o][o][o][o]	5	−1	−1	top.science.computer_science.operating_systems.unix.linux
[++][++][++][++]	4	8	16	top.science.computer_science.programming
[++]	1	2	4	top.science.computer_science.programming.languages
[++][++][++]	3	6	12	top.science.computer_science.programming.languages.functional
[+][++][++][++][++]	5	9	17	top.science.computer_science.programming.languages.functional.lisp
[++][++][++]	3	6	12	top.science.computer_science.programming.languages.functional.ml
[++][++]	2	4	8	top.science.computer_science.programming.languages.oo
[++][++][++]	3	6	12	top.science.computer_science.programming.languages.oo.smalltalk
[++][++][++]	3	6	12	top.science.computer_science.programming.languages.predicative
[+][+][++][++][++][++][++][++]	8	14	26	top.science.computer_science.programming.languages.procedural
[+][+][++]	3	4	6	top.science.computer_science.programming.languages.procedural.c
	0	0	0	top.science.computer_science.programming.languages.procedural.cpp
[+][++][++]	3	5	9	top.science.computer_science.programming.languages.procedural.perl
[+][+][+][++]	4	5	7	top.science.computer_science.programming.languages.procedural.python
[−][o]	2	−1	−1	top.science.computer_science.theoretical_cs
[−−][−][+]	3	−2	−4	top.science.linguistics
[−−][−][o]	3	−3	−5	top.science.linguistics.computational_linguistics
[−−][−]	2	−3	−5	top.science.linguistics.computational_linguistics.parsing
[−−][−−]	2	−4	−8	top.science.linguistics.computational_linguistics.pragmatics
[−−][−][−][−]	4	−5	−7	top.science.linguistics.computational_linguistics.semantics
[−−][−−][o][+]	4	−3	−7	top.science.linguistics.computational_linguistics.syntax
[−−][−−][−][−][−]	5	−7	−11	top.science.linguistics.morphology
[−−]	1	−2	−4	top.science.linguistics.phonology
[−−][−−][−−][−−][−−][−][o]	7	−11	−21	top.science.linguistics.psycholinguistics

(Truncated example)

This table is generated by the feeback browser script `fbb`. The leftmost column contains all feedback events collected with respect to the categories listed in the rightmost column. The second column simply counts the number of feeback events per line (that is, the number of square bracketed entries in the leftmost column). The next column contains the score as defined by the sum of all '+'–signs minus the sum of all '−'–signs. Column four contains a weighted score which doubles the number for the [++] and [−−] feedback events.

Figure 6.2: Relevance feedback as simulated for user 88.

This algorithm deliberately performs suboptimally for generating clear boundaries in interest evidence on $\mathcal{C}$:

1. Single centroids deliver 200 feedbacks, which on the other hand are relatively clear.
2. The higher the centroid is defined, the more probable 'off–topic' categories are labeled as interesting due to the definition of δ. This corresponds to 'unspecific' feedback of a new user.
3. The more centroids we have, the more likely it is that the feedback data concerning one category is 'mixed'. This corresponds to a user with different interest aspects that have not been discriminated so far.

The feedback generated by this procedure was taken as input for the first rough evaluation as presented in section 6.2.

6.1.2.2 A worst–case simulation

The user feedback that was simulated as described in the last section is very heterogenous in terms of frequency of relevance feedback and in terms of the number of different interest aspects per user. The data was used to obtain a first impression of the performance of our approach as explained in section 6.2. In table 6.2, the values of accuracy for `p_interest` show the impact of different qualities of user feedback. Furthermore it was impossible to derive decent results for `n_interest`, since we had no positive evidence for negative interest (i.e. negative feedback was simulated by growing distances from positive interest centroids).
In order to overcome these drawbacks and to allow for a more informative evaluation, we generated feedback that consisted of single aspects only. From those feedback sets we were able to construct more complicated cases of multiple aspects; where the relation between the involved aspects could be explicitly triggered.
In general, for each aspect a a centroid c_a was randomly chosen. The feedback value for a document d with $\mathbf{A}_{\mathcal{T}\times\mathcal{C}}(d) = \langle t :: p, \langle c_1 :: p_1, \ldots, c_n :: p_n\rangle\rangle$ whose most reliable category is c, is determined using a sigmoid

$$fb_a(d) = \beta \left(1 + \mathrm{e}^{-\alpha \left(\mu - \sum_{i=1}^{n} p_i \delta(c_a, c_i) \right)} \right)^{-1} + \chi$$

again, taking into account all classifications per document. Herein, $\beta = 4$ and $\chi = -2$ are used to normalize fb to the codomain values $[-2, 2] \in \mathbb{R}$. To control gradient and null of the function we use μ and α. The codomain of fb is divided into five equidistant intervals $[-2, 2] \in \mathbb{Z}$.

The outcome of our feedback function can be artificially noisified. A noise value of p_{noise} means that the outcome of the feedback simulation function is randomly changed with a probability of p_{noise}.[7]

The feedback generated by this procedure was taken as input for the detailed pessimistic evaluation as presented in section 6.4.

6.1.2.3 An average case simulation

Since our aim in the detailed evaluation was to provide results that come close to results that are to expected from real world data, we needed to reduce noise from 'cross–categorization' as it is inherently generated by the simulations described above. Therefore, user feedback is here defined only with respect to the most reliable classification c_1 of a document[8]. Accordingly, we defined four different 'users' with 50 aspects each; each aspect is defined by a center category c that was chosen randomly from the entire ontology. Then, aspects are centroids defined by δ–regions around c. The choice of δ is subject to discussion and is based on a crucial assumption: The implicit assumption is, that 'primitives' of interest, that is, (parts of) interest aspects, can be represented by concepts of our taxonomy. Then, a user model is a collection of complex descriptions based on such primitives. This is exactly the only fundamental assumption in our approach. We simulate interest by a function that delivers feedback with respect to a conceptual primitive that represents an aspect of a user's interest. The learning task is to induce a model which describes the interest in terms of the concepts without knowing what the user really is interested in; that is, without taking into account knowledge about δ. Simulation of feedback $F_{u,a}$, which is a function of the unknown interest $\mathfrak{I}_u$, is completely separated from the task to approximate the (unknown) learning target $\mathbf{i}_u$ by $\mathbf{M}_u$ based on a sample $\mathbf{f}$ generated by Γ functions from $F_{u,a}$.

The four users we simulate to give feedback differ in the interval of δ distances which are defined as positive or negative evidence for the user. Relevance feedback for documents d with respect to an aspect with a centroid c is defined as simulated for the different interest specificity as shown in table 6.1. During the evaluation, the goal was to learn both $M_u^+(a)$ and $M_u^-(a)$ using E^+ and E^- for each task.

The feedback generated by this procedure was taken as input for the detailed average case evaluation as presented in section 6.3.

[7] The generation of URLs and simulated feedback data was carried out using reserved key features on the actual OySTER databases using the functions in `um/inducum_2/mk_testdata`. The following user model induction, however, was carried out offline on a dumped subset of database contents.

[8] In this special case, the visualization of feedback as in figure 6.2 actually corresponds to the sample.

User	E^+	E^-
1 (444)	$\delta < 3$	$\delta > 8$
2 (111)	$\delta < 5$	$\delta > 5$
3 (222)	$\delta < 7$	$\delta > 15$
4 (333)	$\delta < 10$	$\delta > 10$

Table 6.1: User interests of different specificity

6.2 A first evaluation of the approach

The motivation for this first evaluation was to get a rough impression of the capability of our approach without sample enlargement by Γ–functions. We also needed to get a rough estimate of the runtime behavior of the system such that the further evaluation could be planned in detail. We therefore started with a rather simple representation and a stronger biased learning task (excluding thresholds for classification confidence). Furthermore, we did not evaluate the learning task for several aspects per user but assumed a single aspect interest setting.[9] Data used for this evaluation was domain data as decribed in section 6.1.1.1 and feedback as described in section 6.1.2.1.
Figure 6.2 gave a rough impression about user 88's interest based upon document feedback that was mapped onto the categories. The sample, however, contains much more information (since documents are described by multiple categories). The rules induced on the sample derived from all feedback data available are:

```
p_interest_88(_a,D) :-
  cat__top_science_computer_science_programming_languages(D,C1).

n_interest_88(_a,D) :-
  cat__top_science_linguistics(D,C1).
n_interest_88(_a,D) :-
  cat__top_science(D,C1),
  cat__top_science_computer_science_artificial_intelligence(D,C2).
```

Those three rules were found to cover ten pieces of evidence in all. Nevertheless, from other samples more complex rule sets, as displayed in figure 6.3, have been derived. User 90 shows the nice attribute of being interested in any 'publication'–like document about 'science'. Furthermore, he seems to be interested in 'computer science' and 'machine learning', if the document belongs to the category 'machine learning' with a confidence of at least 57. User 93 is interested in 'programming' if it coincides with 'artificial

[9] Accordingly, the argument of the head literal specifying the aspect id only carries a dummy variable. The rules presented here were postprocessed by renaming variables for better readability as well as to match definition (5.1). The files contained in the directory `/src/oyster/um/inducum_1/first_run/` of the CD–ROM have a reversed argument structure in the head literal (the rest remains unchanged).

```
p_interest_90(_a,D) :-
  cat__top_science_computer_science(D,C1),
  cat__top_science_computer_science_[ai]_machine_learning(D,C2),
  C2>56.
p_interest_90(_a,D) :-
  type__top_publication(D,C1),
  cat__top_science(D, C2).
p_interest_93(_a,D) :-
  cat__top_science_computer_science_programming(D,C1),
  cat__top_science_computer_science_artificial_intelligence(D,C2).

n_interest_93(_a,D) :-
  cat__top_science_computer_science_programming(D,C1),
  cat__top_rec_sports_water_scuba_diving(D,C2).
n_interest_93(_a,D) :-
  cat__top_science(D,C1),
  cat__top_science_computer_science_operating_systems_dos(D,C2).
```

Figure 6.3: Rules describing a user's interest

intelligence'—but definitely is not interested in documents about 'diving computers' or 'DOS'.

We have generated different feedback sets for ten simulated users, where Progol did not deliver any compressing rule at all for two feedback sets; which in one case is due to the almost equally distributed feedback over all categories.[10] Results are shown in table 6.2. The underlying interest consisted of a single centroid (located somewhere in the 'procedural programming' tree). Due to the nature of δ, most negative feedback fell into the category `linguistics` which formed a very clear image. Accordingly, only three rules were induced which deliver a relatively high coverage and accuracy for `p_interest`. Since `n_interest` is modeled by low δ values instead of special centroids, the training data is unspecific and rather noisy.

A growing number of centroids chosen within the feedback simulation function corresponds to multiple centers of interest. Usually, one would like to represent such a diversity of interest by different aspects, but our aim was here to simulate a blurred image of the user's single aspect interest. Since multiple aspects were not covered in the first test series, according results are rather bad: User 92's single aspect interest was simulated using two centroids[11] that were both located in the upper levels of the ontology's 'sci-

[10] In the second case, the depth limit on the search space was exceeded.

[11] Again, the use of two centroids does not correspond to multiple aspects! Centroids are simply a means to simulate user feedback. Therefore, simulating feedback for one aspect using two centroids means to 'blur' the feedback behavior.

	p_interest			n_interest			
user	*cov*	*acc*	$r_{\mathtt{p}}(c_{\mathtt{p}})$	*cov*	*acc*	$r_{\mathtt{n}}(c_{\mathtt{n}})$	time
88	93.8%	72.2%	1(2)	65.6%	26.1%	2(8)	2'07"
92	51.7%	52.4%	1(2)	57.1%	29.5%	4(15)	4'07"
93	41.6%	57.8%	3(12)	32.2%	26.8%	4(12)	7'54"

r_i is the number of rules induced for the target i; c_i is the number of facts that are covered by the rules.

Note, accuracy and coverage are computed only with respect to rules which actually compressed the sample; remaining rules covering only single pieces of evidence are not taken into account.

Table 6.2: Coverage & accuracy of induced rules

ence' part thus yielding a rather uniform distribution of positive feedback with average noise of interfering negative feedback. Least positive evidence in this branch was given in the 'computer science' / 'operating systems' classes; most positive evidence was located in the linguistics branch. The large number of rules for `n_interest` can be explained by the noisy negative feedback of the large positive field which might also explain the slightly better accuracy result of `n_interest`. Most important is the dramatic decrease in coverage and accuracy of `p_interest`, though the latter can be easily explained by inducing only one rule which subsumed the 'linguistics' branch and left out the whole branch of 'computer science' (containing approximately 70% of all positive feedback).

Finally, user 93, whose interest was defined by three centroids, showed the worst results. Two of the centroids were located in 'artificial intelligence', while the third was identical to the node 'science', which of course yielded (due to its high position in the hierarchy) lots of noise. Seven rules were induced, three for `p_interest`, four for `n_interest`. Nevertheless, the induced rules showed interesting results (three rules are shown in figure 6.3; a non–compressing rule that was found for user `93` is shown in figure 7.1).

A first conclusion shows that for increasing number of interest topics (as simulated by growing number of centroids for the feedback function), coverage decreases since compressing rules need to be more precise—thus generalizing too carefully (see table 6.2). The bad values for `n_interest` are due to our simulation of negative feedback.

A more pictorial view on the user interest for users `88`, `90`, `92` and `93` is given in appendix A.1.

6.3 A detailed evaluation

The second evaluation was carried out in order to obtain a more precise estimate of the quality of the sample generator functions Γ. Taking into account several document categories for each document, the sample is inherently noisy: Positive feedback with respect to a document yields positive examples for each of the categories the document was classified. For documents with wide–spread categorizations this means that positive evidence is distributed among the whole ontology.
For reasons of brevity, we only include a discussion of learning `p_interest` here; the impact of applying Γ for learning `n_interest` is discussed in detail during the pessimistic evaluation.
For this evaluation data generated by the procedure as decribed in section 6.1.1.2 was used; feedback was simulated as described in section 6.1.2.3.

6.3.1 Learning single aspects without sample enlargement

It is clear, that specific interests provide a better base for inducing user models than shattered or vague feedback. Accordingly, user `444`, who has the most specific interest, draws a rather clear picture of his interest. The left graph in figure 6.4 shows that accuracy increases both in specificity and sample size. As data from table 6.1 suggests,

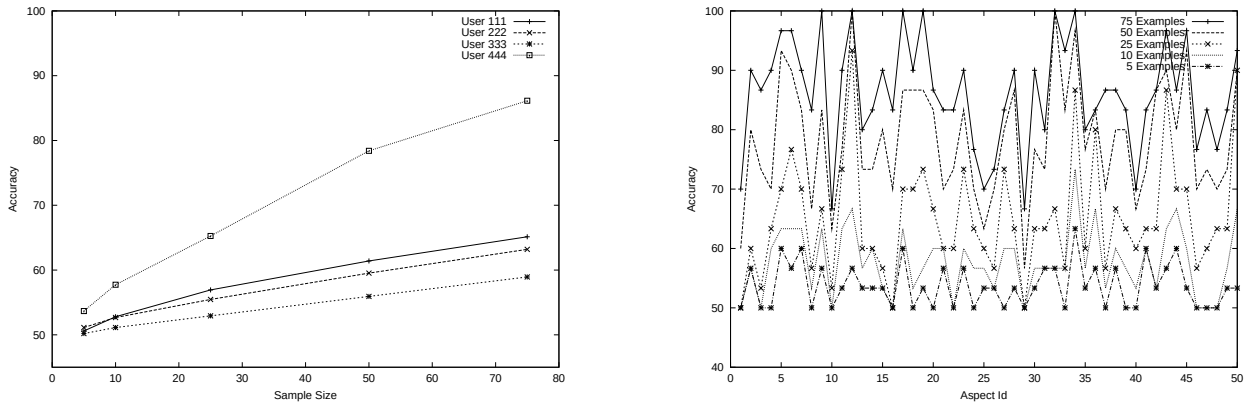


Figure 6.4: Learning `p_interest` for specific interests (user `444`)

accuracy for user `444` is the best derived for all users and sample sizes. Similarly, accuracy increases with growing sample size (with decreasing gradient). A more detailed evaluation of the accuracy for learning `p_interest` for user `444` only is shown in the

right graph in figure 6.4 (the graphs on the left hand side are average values derived from such data files).

6.3.2 Learning multiple aspects with sample enlargement

As described in the last section, accuracy for small samples is rather poor. This motivates using mutual feedback from other aspects in order to enlarge the sample. In the following evaluation, interest aspect `8` was chosen as aspect a' whose feedback data is used by Γ in order to gain more examples for learning aspects $a = [1, ..., 50]$.[12] The evaluation was carried out for users `111`–`444` with 50 aspects each, thus modeling 200 different aspects of different specificity. For each user, initial samples of length 5, 10, 25, 50 and 75 were generated (with an uniform distribution of positive and negative examples). Γ_1 and Γ_4 enlarged all samples by 15 examples that were generated from feedback given with respect to aspect `8`. Results are shown in figure 6.5. The average accuracy gain for the test shown in figure 6.5 is described in table 6.3.

	Sample size		
User	5	10	50
`111`	5.8 %	5.6 %	0%
`222`	5.5 %	6.4 %	0%
`333`	4.5 %	5.0 %	0%
`444`	4.4 %	6.6 %	0%

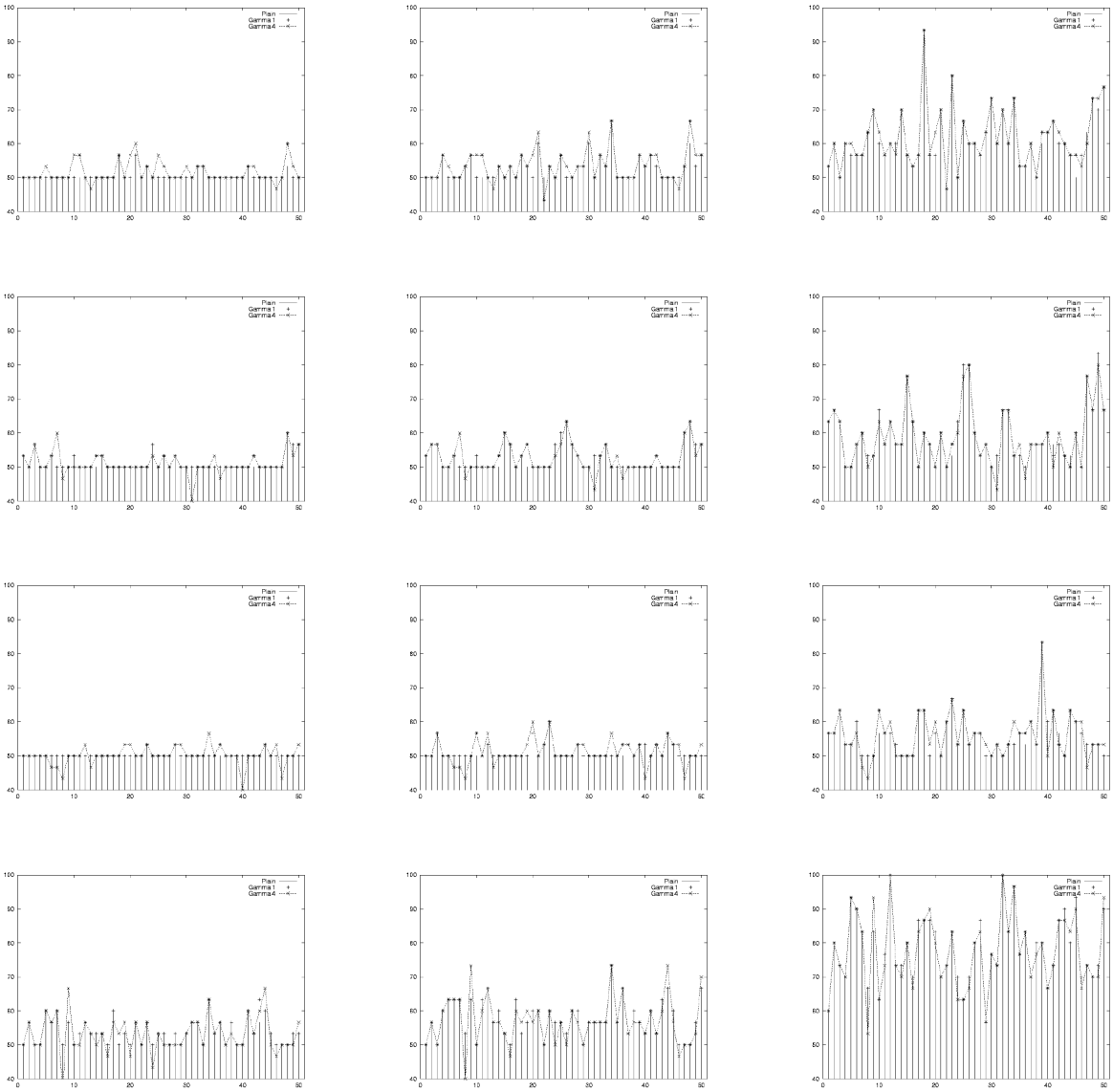
Table 6.3: Average positive accuracy gain

6.3.3 Discussion

From the results, we draw two major conclusions:

Only small samples should be enlarged. With growing sample size, the impact of Γ functions decreases.
Due to the fact that the accuracy of plainly derived hypotheses increases for larger samples, application of Γ functions does not contribute to the result. In such cases, enlarged samples noisify the input. As results for sample length 50 show, accuracy on Γ samples actually drops significantly below plain learning accuracy.

[12] This explains bad results for $a = 8$: The sample is enlarged by labeled data obtained from $a' = a = 8$ and therefore becomes very noisy.



From top to bottom: Users `111`, `222`, `333` and `444`; from left to right: Sample size 5, 10, 50

Each graph shows accuracy of user models as learned for each of the fifty text cases. The solid bars labelled 'Plain' show the accuracy gained when trying to learn the user's single aspect interest based on the feedback only. This value increases with sample size and specificity of interest. The '+'–sign shows accuracy when trying to learn the aspect under consideration taking into account Γ_1 applied to feedback from aspect 8. The '×'–signs (connected through a dotted line) show accuracy as obtained by using Γ_4.

The interesting part of the figures is where the accuracy can be increased by use of Γ–functions. This effect becomes visible in small samples only. It vanishes for growing sample size or extreme specific or unspecific interest. As a result, the Γ–functions behave as expected and help to improve results for small samples on average interest specificity.

Figure 6.5: Accuracy for users `111`–`444` for different samples

Specificity of interest. Over–specific interest as simulated by user 444 leads to highly accurate hypotheses (see figure 6.4). This explains a rather poor accuracy gain using Γ functions for user 444. The base line of plain learning accuracy can be exceeded in only very few (five) cases. For five initial examples, average accuracy gain is maximal for user 111. The accuracy gain values can be related to the number of aspect hypothesis in which enlarged samples delivered worse results than the initial sample (2, 3, 6 and 5 cases for users 111–444)[13]. Optimal results are delivered by specific interests which do not allow for overfitting.

For the shortest sample and user 111, the best accuracy reached is only 60%. On the other hand, the average accuracy gain obtained by using enlarged samples is 5.8%, and more than 20% of all aspects could be learned with a higher accuracy. For a sample size of 10 examples, similar results can be derived. It is noteworthy, that Γ helps to increase accuracy in 9 cases for user 111 while it helps for only 3 aspects for user 444.

The results obtained by our evaluation can be improved without violating our assumptions (A-5): All aspect learning tasks were evaluated using a fixed aspect $a' = 8$ for enlarging the sample. During the evaluation we did not take into account 'disjointness' of aspects (the extreme case is the result for learning aspect 8). This means, that for aspects similar to aspect 8, the results are distorted. A better evaluation method would have been to choose a set of aspects $a', a'', ...$ for each learning task $M_u(a)$, where $a', a'', ...$ significantly differ from a (this can be modeled by means of the δ–function). This evaluation method meets the assumption about modeling a user's interest $\mathbf{i}_u$ by an approximation $\mathbf{M}_u$ which consists of several distinct aspects $M_u(a)$ and will most likely deliver much better results.

Conclusions. It has been shown, that sample enlargement pays off for very small samples only. Of course, sample enhancement can by no means reach an accuracy gain that can be obtained by more examples.

Plain learning average accuracy results of between 50 to 52% do not sound very promising. Even when this level is increased to 58% one cannot feel overly impressed by it.

One must bear in mind though, that samples of length 5 pose a very hard learning problem. When taking this into account one can conclude that an accuracy gain of 10% (in relation to accuracy levels by plain learning) can be yielded in up to 20% of all cases where 5 examples are considered.

Especially in cases where there is very little feedback available (namely 5 or 10 examples) an accuracy gain of 5% should be viewed as a major improvement.

[13]These are the cases where the dotted line for Γ_4 drops below the baselines.

6.3.4 Further evaluation

So far, we have only evaluated cases, where the sample based on feedback for one aspect a was enhanced by Γ on one different aspect a'. It is clear, that for more aspects a'', a''' etc., application of Γ yields even larger samples. This hypothesis was not evaluated in detail, but we tested the method for some chosen sample sets. The results show, that for an initial sample of length 10, the baseline accuracy (i.e. without sample enlargement by Γ–functions) was 50 %. The sample under consideration was aspect 20 as defined by user 111; see the top left graph in figure 6.5. As already shown during this evaluation, application of Γ to $a' = 8$ yielded an accuracy gain of approximately 7 %. The same result could be derived by applying Γ to $a'' = 23$.
Incorporating both additional sample data, the accuracy could be further increased to nearly 60%. These results cannot be generalized, but suggest that a further investigation at this point may yield even better results.

6.4 A detailed worst–case evaluation

In this section, we describe an evaluation which was performed on deliberately noisy data. For each document, all categories are taken into account—the probability distribution by which categories were assigned to documents was a δ–weighted random distribution. Therefore, the outcome of this evaluation can be regarded as a baseline of minimal performance which is guaranteed by our approach.
During the pessimistic evaluation, the same domain data was used as in the last evaluation (see section 6.1.1.2). In contrast to the evaluation described in the last section, we take into account all categories of a document. As a result, generating user feedback is a more sophisticated process. Therefore, the user simulated user feedback is that defined in section 6.1.2.2.

6.4.1 Learning single aspects without sample enlargement

Again, for a first evaluation, we generated fifty different aspects for an artificial user. All aspects were single aspects with specific interest functions and five per cent noise.

Size of feedback sets. From each feedback set samples of size 25, 50, 75 and 100 were generated. A batch run of the rule induction system was then used to verify that larger samples generally imply a better means for learning compressing rules. Table 6.4 shows, that for growing samples more rules are derived (upper part). For a sample size of 25 feedback events only 4 out of 50 aspects were partially described by one single rule. With 100 training examples, 54% of all aspects were partially described by rules. In this case,

Sample type/ No. of Rules	Sample size 25			50			75			100		
$\mathtt{n}^{\{+,-\}}, \mathtt{p}^{\{+,-\}}$ / 1	**4**	100%	8%	**11**	73%	22%	**15**	54%	30%	**15**	38%	30%
$\mathtt{n}^{\{+,-\}}, \mathtt{p}^{\{+,-\}}$ / 2				**2**	27%	4%	**5**	36%	10%	**11**	55%	22%
$\mathtt{n}^{\{+,-\}}, \mathtt{p}^{\{+,-\}}$ / 3							**1**	10%	2%	**1**	7%	2%
sum	**4**		8%	**15**		26%	**28**		42%	**40**		54%
$\mathtt{p}^{\{+,-\}}$ / 1	**4**	100%	8%	**9**	69%	18%	**8**	67%	16%	**8**	80%	16%
$\mathtt{p}^{\{+,-\}}$ / 2				**2**	31%	4%	**2**	33%	4%	**1**	20%	2 %
$\mathtt{p}^{\{+,-\}}$ / 3												
sum	**4**		8%	**13**		23%	**12**		21%	**10**		19%

The bold face number in each cell describes the number of aspects that were described by a rule set of according cardinality (for example, for a samplesize of 50, **2** aspects were described by two rules each).
The following percentage shows the distribution of induced rules over the cardinality of rule sets. As an example, consider the samples of length 100: 55% of all rules belong to two–rule aspect descriptions. Only seven per cent belong to complex three–rule hypotheses while 38% belong to single rule aspects.
The last value shows, how many aspects were covered the rules. It corresponds to the term of the first number divided by 50.

Table 6.4: Learning aspects in relation to sample size*

55% of all rules belonged to a two–rule aspect description—which means that 22% of all aspects were described by two rules.
In a second batch run, the goal was to learn rules for only `p_interest` using both positive and negative feedback. The results from table 6.4 (lower part) suggest, that the algorithm performs worse for this learning problem, although the produced rules are exactly the same as the $\mathtt{p}^{\{+,-\}}$ subset in the former test run (detailed information about aspects for which hypotheses were generated are provided in the appendix; see table A.2).
It remains to be explained why most aspects $M_u^+(a)$ can be described by rules with relatively small sample sizes (25 to maximum 50), while the number of rules for $M_u^-(a)$ still increases. This phenomenon can be easily explained by analyzing the samples: Due to our simulation the relative amount of negative feedback increases with sample size. This property was intended to reflect the user's behavior while working with such a system: First, any user, in general, tends to give positive rather than negative feedback. Assuming that the first links followed by the user are the best matches delivered by the search engine upon the user's request, the first feedback given with respect to a search query will most likely be positive. In consequence we may assume, that they are not completely off the topic but rather fit M_u^+ instead of M_u^-. This explains, that for larger samples the hypothesis space is more strongly biased due to the fact that there are more negative examples and also because of the decreasingly scattered positive examples when trying to learn $M_u^+(a)$. The reverse case holds for learning $M_u^-(a)$.[14]

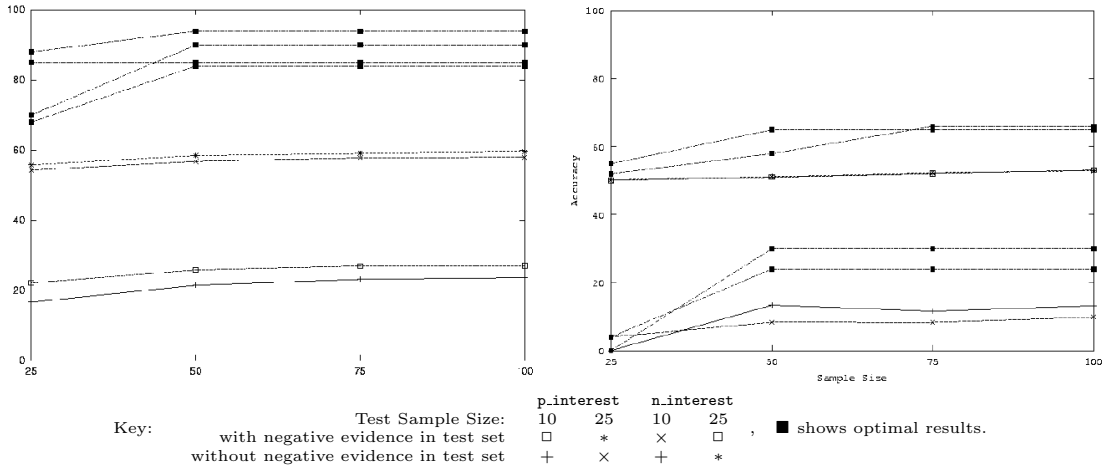


Figure 6.6: Accuracy for `p_interest` and `n_interest`

Accuracy is described by the two graphs depicted in figure 6.6 (`p_interest` left hand

[14]In both cases, constant noise can be neglected.

side `n_interest` right side). Both figures display the average accuracy over all aspects for which at least one rule has been learned (80%; ten aspects could not be compressed). Accuracy is measured with respect to a test sample and is computed only by taking into account the inferred rules. Two surprising results are derived from the statistics: First, accuracy differs significantly for different test sets. If the test set includes negative evidence for `p_interest`; i.e. URLs which shall not be provable as elements of $M_u^+(a)$, then accuracy is much higher. In other words: The rules learned for `p_interest` rather help filtering documents which are not of interest instead of describing the interest. The second surprising fact is that accuracy cannot be significantly increased by sample sizes larger than 50 feedback events.
Similar observations can be derived from the results obtained while learning `n_interest`, see the right hand side of figure 6.6:[15] A sample sizes of 25 is not sufficient at all. Furthermore, it seems that accuracy increases slower and never reaches values as reached by `p_interest`. Again, this can be explained by the fact that positive and negative feedback is not distributed equally (and thus, examples in the sample are not, either). In contrast to the learning problem `p_interest`, accuracy of `n_interest` is much more sensitive to the presence of negative evidence (in this case, *positive* feedback). Again, the hypotheses are more suitable to filter negative examples than describing positive instances.

6.4.2 Learning multiple aspects with sample enlargement

Results from the last section have explained that taking into account relevance feedback concerning aspects, other than the current aspect under consideration, should easily help to improve accuracy of the results. However, this task is not as easy as one might think in the first place. Accordingly, we start with rather sobering results.
In this section we briefly describe the results obtained while learning the positive interest `p_interest` (M_u^+) of a user. The samples provided contained both positive only and positive and negative feedback. The samples were truncated to different sizes in order to evaluate the performace of the Γ function in relation to the sample length (5, 10, 25, 50). The example generating functions Γ need to take into account feedback that was given with respect to another aspect $a' = \mathtt{42}$. This reference aspect was fixed for all test runs.

Medium size samples. The first evaluation was carried out on samples which contained 25 relevance feedback pieces of evidence each. The target was to learn $M_u^+(a)$. In one case, the sample contained positive feedback for `p_interest` only; the other cases contained

[15] Please note, that the "constant" values for $*$ and $\Box$ actually accumulate to an accuracy of 50 % by pure incident. This can be verified by comparison to the same lines in the left graph and the non–constant maximum graphs in the right graph.

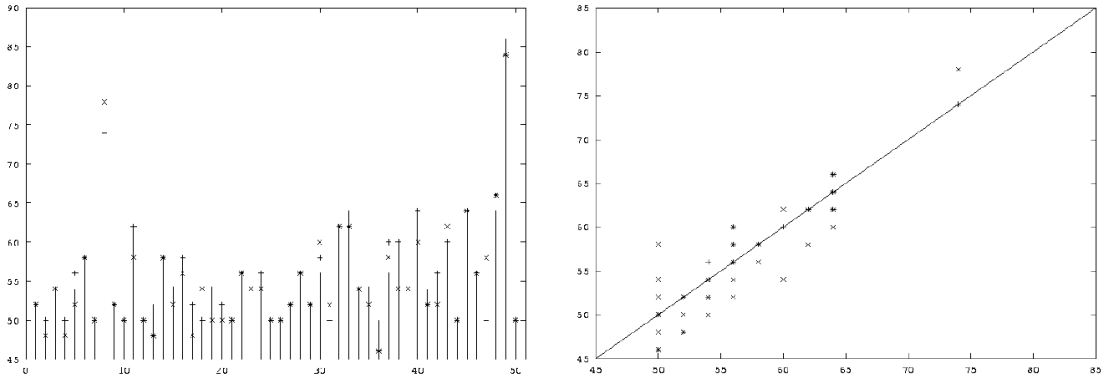
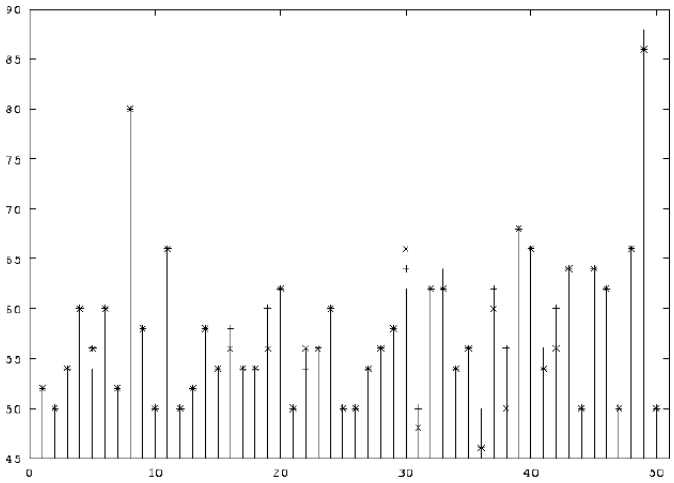


Figure 6.7: Accuracy of Γ–samples on $M_u^+(a)$ feedback with initial length 25

very little negative evidence. The last sample also contained examples for `n_interest`. Those samples were enlarged using Γ_1 and Γ_4 (see figure 5.3). However, the results showed that, on average, hypotheses were no more accurate than the initial hypotheses. Applying the Γ–functions sometimes resulted in an improvement, sometimes in worse results—with an average that does not significantly differ from the initial data. In other words, the use of Γ resulted in more hypotheses, that showed a slightly less accuracy than the initial sample, and few hypotheses, which were of significantly better accuracy. For reasons of brevity, we only show a graph for the behavior of Γ with respect to the first sample provided, namely positive feedback only for $M_u^+(a)$ in figure 6.7. The remaining graphs and statistics can be found in the appendix. Figure 6.7 shows the distribution of accuracy values of Γ–hypotheses around initial hypotheses. The accuracy of the initial hypotheses is drawn as a diagonal line. Applying Γ_1 (see '+'–marks), accurracy values are distributed rather closely around the baseline accuracy. Distribution increases even more for Γ_4 (see '×'–marks) which explains the different number of points drawn in figure 6.7.

Large samples. As shown in section 6.4.1, samples of length 50 already displayed nearly optimal results. In consequence, it is rather unlikely that additional information as provided by Γ helps to boost the accuracy of the derived hypotheses. Indeed, results show less significant results. Hypotheses generated using Γ are still more or less accurate than initital hypotheses (see figure 6.8) but differ less from the reference hypotheses than hypotheses do for samples of size 25 (see figure 6.7). The fact that better initial hypotheses are harder to improve further, can be derived from a direct comparison of figures 6.7 and 6.8: While in figure 6.7 the hypotheses derived from the sample of



Bars indicate accuracy of the initial sample; $\begin{smallmatrix}+\\ \times\end{smallmatrix}$ indicates $\begin{smallmatrix}\Gamma_1\\ \Gamma_4\end{smallmatrix}$ values.

Figure 6.8: Accuracy of Γ–samples on $M_u^+(a)$ feedback with initial length 50

length 25 could be enhanced in some points, figure 6.8 does not show any significant improvement at all (see aspects 5, 6, 8, 18, 20, 21, 30, 37, 43, 47 and 48 against 5 and 30 only).

Small samples. The quality of initial hypotheses and Γ–hypotheses is shown in figure 6.9. As one can see, the impact of Γ for very small samples is bigger than for large samples.

Summary. The results presented in this section are based on a very pessimistic basic assumption. Feedback was generated taking into account all document classifications c_1, c_2 and c_3. As a consequence, documents with "multiple topics" (i.e. c_i with large average δ distances), generate noisy learning and evaluation samples. Accordingly, results are not very promising but still demonstrate the potential behind Γ–functions. Accuracy gain is still obtained in several cases and, neglecting cases where Γ actually delivers worse results[16], follows the general rules of improving learning results. The following table summarizes results from figures 6.7, 6.8 and 6.9: It shows, that Γ performes best if used on samples of length 25. On shorter samples (length 5 and 10), Γ performs slightly worse

[16]Such cases can be prevented by testing sample accuracy of hypothesis for plain samples and Γ–enhanced samples. If sample accuracy for enhanced samples drops below accuracy on the initial sample, hypotheses are discarded.

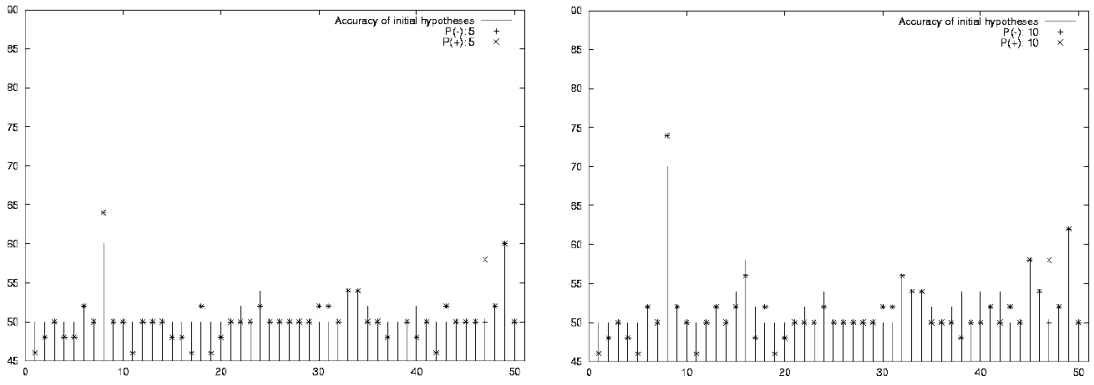


Figure 6.9: Accuracy of Γ–samples on $M_u^+(a)$ feedback with length 5 (left) and 10 (right)

Sample length	Γ_1	Γ_4	both	avg. accuracy gain
5	5	6	6	3.82
10	5	6	6	3.82
25	4	8	8	4.08
50	2	2	2	2.50

Table 6.5: Results for accuracy gain with Γ samples

(in number and average accuracy gain) than in the reference case. For longer samples, both number of and average accuracy gain are significantly smaller than in the reference case.

6.5 A brief discussion of the evaluation methods

At first glance, results presented in the preceding sections are not overly impressive. Partially, this was due to our motivation of giving a base line evaluation instead of presenting impressive results. In other words: Using our approach on real data, with better classifiers or more background knowledge, results can only be improved.
On the other hand, the evaluation carried out was even more pessimistic than actually required by our assumptions. Furthermore, several more effects need to be evaluated in order to gain a more precise image of how our approch will perform on real data.
The following ideas should be considered for further evaluation:

Bias on the learning algorithm: The language bias allows for building hypotheses which give both lower and upper bounds on document category confidences. In other words, both $>$ and $<$ are elements of the hypothesis language. This leads to overfit clauses which cover only single examples (for an example, where the category confidence was 93, two literals are added to the rule body: $\mathtt{C} < 94$ and $\mathtt{C} > 92$).
It has to be evaluated, how a further language bias, which still satisfies our formalization (that is, prohibiting the use of $<$), increases accuracy of our results.

Evaluation of Γ functions: During evaluation we chose one fixed reference aspect a'. We already have motivated, that taking into account further reference aspects, results can be optimized. Furthermore, our evaluation did **not** take into account, that aspects from our artificial users have overlapping regions. At this point, the basis for our evaluation is much weaker than our basic assumption, that aspects are disjoint.

Aspects: It has to be evaluated, how different characteristics of aspects can be learned. This includes specificity (as defined by δ–thresholds for feedback) as well as diversity (defined by the number of categories chosen as center of aspect centroids).

A more detailed summary of the results and a discussion of further work is given in the concluding chapter of the thesis, 8.1.

Chapter 7

IMPOVEMENTS

7.1 Improvements of the learning algorithm

The hypothesis generated during the induction process is a rule set S. The set of clauses generated by Progol can be roughly divided into two sets of compressing rules C and redundant rules R.[1] Most of the rules that were not taken into account in sections 6.2 – 6.4 are rules which yield no compression, but nevertheless carry valuable information. For example, user 93's (see table 6.2) interest in a certain URL yielded a most specific clause as an element of R which is shown in figure 7.1. Such clauses are redundant (because they describe exactly one example) and are not included in the hypothesis, since this would violate the minimum description length principle.
With respect to the knowledge described by the whole sample, such clauses are overspecific; one could say that such clauses are the most extreme form of overfitting. In order to prevent overfitting, one would try to generalize hypotheses.
In general, there are two ways to generate rule based hypotheses: One method is called '*general–to–specific*' which in this case means adding literals to the antecedent of an intially empty clause. This method is performed by the (m)–Foil systems.
The second version is a *specific–to–general* method which operates by dropping literals from the antecedent of the rule. This method is used in the context of post pruning decision trees.

7.1.1 Learning rules by adding literals

Learning rules by adding literals has been implemented in the Foil systems (as described in [Quinlan, 1990]). A straightforward idea to learn rules using redundant clauses is to

[1] Rules in R are called redundant since they have the same expressive power as the fact they were generated by. Since the encoding length of the rule is much greater than the length of the example, they are discarded. Thus, R is replaced by the examples E and the output hypothesis $H = C \cup E$ is less complex than **s**.

```
p_interest_93(_a,D) :-
   type__top_publication_publishedbook(D,C1),
   cat__top_science(D,C2),
   cat__top_science_[cs](D,C3),
   cat__top_science_[cs]_[ai]_machine_learning_learning_theory(D,C4),
   cat__top_science_[cs]_[ai]_machine_learning(D,C5),
   cat__top_science_[cs]_programming(D,C6),
   cat__top_science_[cs]_programming_languages_functional_lisp(D,C7),
   cat__top_science_[cs]_programming_languages_functional(D,C8),
   cat__top_science_[cs]_programming_languages_procedural_perl(D,C9),
   cat__top_science_[cs]_programming_languages_procedural(D,C10),
   cat__top_science_[cs]_programming_languages(D,C11),
   cat__top_science_[cs]_[ai](D,C12),
   C1>67, C4>71, C5>46, C7>7, C9>75, C10>50, C12>21.
```

Brackets, i.e. _[cs]_[ai]_, are used to abbreviate long category names.

See also footnote 9 on page 127.

Figure 7.1: A non–compressing rule

collect all literal pairs which share common variables and then run Foil on this set. This idea is formalized in the algorithm described in figure 7.2.

As a descendant of the attribute–value learning systems, Foil brings a top–down searching technique into the family of rule learning systems. The system works on large sets of examples where its search is guided by an information based heuristic. Again, the heuristic seeks to guarantee maximal compression; i.e. a complex description is preferred if its bit length does not exceed the number of bits required to encode all examples covered by the description. Information gain based heuristics can easily be implemented using a greedy search algorithm which always prefers locally maximal information gain. Such myopic algorithms may lead to garden paths.

The learning target shall be defined by a logic program; i.e. a set of function free Horn clauses. Foil runs on ground models just as Golem does, since it also uses ij–determinacy as a means for search bias. Examples are tuples of objects for which the corresponding relations hold. Due to a closed world assumption, negative examples are defined through positive examples; but they can also be defined explicitely.

While searching for a hypothesis which covers most of E^+ and as few E^- as possible, Foil successively specializes the hypothesis clause by literal adding. Once such a literal is added it cannot be removed again due to the greedy search mechanism. Stopping criterion is the previously determined description length of concepts.

The top–down algorithm proceeds a follows: Starting with the rule head C whose predicate name is that of the target concept the system successively adds literals. Suppose,

- $\mathbf{A}(\mathbf{s}) = H = C \cup R$ with $acc_{\mathbf{s}}(H) = cov_{\mathbf{s}}(H) = 1$
- Literal adding with FOIL:

```
foreach r_i ∈ R
   foreach (L(_, c_L), c_L ≷ ϑ_L) ∈ r_i
      L := L ∪ {(L(_, c_L), c_L ≷ ϑ_L)}
   done
done
foil(L, s)
```

- $H' = C \cup bestof(\bar{R})$.

Figure 7.2: Learning rules by adding literals

we already had a hypothesis

$$h_{i-1} = C \leftarrow L_1, \ldots, L_n$$

with $\mathsf{vars}(h) = \{X_1, \ldots, X_m\}$. There, each variable is bound to a ground term such that $C\theta \in E^+$. Attached to this intermediate hypothesis is the (ordered) set of used variables, which is enlarged by introducing a new body literal:

$$h_i = C \leftarrow L_1, \ldots, L_n, L_{n+1}$$

where

$$L_{n+1} = p(X_{i_1}, \ldots, X_{i_{n_i}})$$

The ordered sets of $\mathsf{vars}(h_i) \supseteq \mathsf{vars}(h_{i-1})$ differ by the variables X_{i_j} introduced by L_{n+1}. For the bindings of the X_{i_j} one demands unique (due to determinacy) ground terms $t_{i_1}, \ldots, t_{i_{n_i}}$ such that $p(t_{i_1}, \ldots, t_{i_{n_i}}) \in \Sigma$. In other words, we choose a literal for which we have a substitution $\theta_{i+1} = \{X_{i_1}/t_{i_1}, \ldots X_{i_j}/t_{i_{n_i}}\}$ such that $L_{n+1}\theta_{i+1} \in \Sigma$. This gives a sequence like

$$\begin{aligned} C\theta_1 &\in E^+ \\ (C \leftarrow L_1)\theta_1\theta_2 &\in \mathfrak{M}_{\Sigma|h} \\ (C \leftarrow L_1, L_2)\theta_1\theta_2\theta_3 &\in \mathfrak{M}_{\Sigma|h} \\ \vdots \quad \vdots &\quad \vdots \\ (C \leftarrow L_1, \ldots, L_n)\theta_1\theta_2\theta_3 \cdots \theta_{n+1} &\in \mathfrak{M}_{\Sigma|h} \end{aligned}$$

It remains to be explained which predicate p the algorithm shall choose. This is done by a modified information gain heuristic: The information gain measure is based on the

ratio of instantiations of the predicate schemata to positive ground facts to the number of overall known constants. Accordingly,

$$I_{\mathsf{Foil}}(S) = -\log(\frac{|S \cap E^+|}{|S|})$$

At each step the next literal is chosen with respect to maximal information gain. As a consequence, one might conclude that the literal which has, relatively, the best support by positive examples is chosen.
Quinlan writes in [Quinlan, 1991]:

> The form of the gain allows significant pruning of the literal space, so that Foil can usually rule out large subspace without having to examine any literals in them. Foil thus tames the hypothesis space problem by a stepwise greedy search for clauses.

Of course, this is not always case: The introduction of new variables is consequently underestimated. This again is due to the fact that, relations are unique in their arguments. Imagine such a binary literal $f(X, Y)$. Now that X in $\mathsf{vars}(L_i)$ and Y is a function f of X, the enlarged set of variables $\mathsf{vars}(L_i) \cup \{Y\}$ carries exactly the same information! In other words, if all values for all variables $\mathsf{vars}(L_i)$ are determinate to ground terms, Y automatically is determinate to a unique value, too. Consequently, the information gain must be zero. In such cases, new variables could not be introduced.

7.1.2 Learning rules by dropping literals

The second version is a *specific–to–general* method which means to drop literals from the antecedent of the rule. With a growing set of examples induction of decision trees tends to deliver overfitted hypotheses. As a consequence, accuracy on the training sample increases, while it drops on an evaluation sample. One method to avoid overfitting is to stop the tree induction process. This can be achieved by requiring a minimum information gain in each step. Nevertheless, so called 'pre–pruning' (i.e. pruning during the tree growing procedure) is rather insecure (as it is myopic).
Post–pruning, on the other hand is much more expensive in terms of computing complexity since it includes both growing and iteratively shrinking. In general, such methods try to prune subtrees which exceed a certain error–complexity measure which is determined either on the training set or on an exclusive validation set. Pruning of whole subtrees however poses the problem of re–structuring the tree in cases where intermediate nodes have been deleted. Intermediate nodes may subsume subtrees which shall remain in the final tree as well.
Rule based post pruning tries to overcome these drawbacks by first growing a tree to its maximum depth deliberately allowing for overfitting. The problem of reorganizing

trees is circumvented by transforming the tree into a set of rules. Hereby, every path is translated into a rule the antecedent of which consists of the whole path from the root to the leaf node. Then, the 'tree' is reduced by deleting whole rules (i.e. complete paths) or by literals (that means, intermediate nodes).
In our case this could be achieved by an information gain guided literal dropping method. Thus, for each rule $r \in R$ we recursively drop least informative pairs of literals

$$\langle l(\_, c_l), \texttt{thresh}(c_l, \vartheta_l) \rangle$$

yielding more general rules $r' \in R'$.[2] Since coverage increases with each step, the process is stopped if the information content of the whole rule r' drops below a predefined value (accuracy of r initially is 1; but since r is only a part of the complete hypothesis, using acc as a bias here would be rather myopic). In a second step, we delete rules from R' until $acc(R' \cup C)$ reaches a lower bound and output $H = R' \cup C$ as a final hypothesis. This algorithm is shown in figure 7.3. Here, inf corresponds to $\hbar_f$ as defined in definition

- $\mathbf{A}(\mathbf{s}) = H = C \cup R$ with $acc_{\mathbf{s}}(H) = cov_{\mathbf{s}}(H) = 1$
- Drop literals:

  foreach $r_i \in R$
  $k = 0$; $\bar{r}_i^k := r_i$;
  repeat
  $\bar{r}_i^k = \bar{r}_i^{k-1} \setminus \{L(\_, c_L), c_L \gtrless \vartheta_L\}$
  with $inf(L(\_, c_L), c_L \gtrless \vartheta_L)) = \min(body(r_i^{k-1}))$
  until $gain(\bar{r}_i^k, \bar{r}_i^{k-1}) < \vartheta_{gain}$ or $acc(\bar{r}_i^k) < \vartheta_{acc}$
  $\bar{R} := \bar{R} \cup \{\bar{r}_i^k\}$
  done

- $H' = C \cup bestof(\bar{R})$.

Figure 7.3: Learning rules by dropping literals

2.10 while *gain* can be realized by any of the functions Gn (see definition 2.11), NGn or GR (see section 2.2.5).

7.1.3 Asking for feedback

Results cannot only be improved by using mutual information for generating larger samples but also by explicitly asking the user for more feedback. In contrast to interpreting

[2] A similar technique will help in identifying aspects: sudden leaps in decreasing information gain while literal dropping suggest a border crossing.

already existing feedback, this method does not really count as an improvement of the learning algorithm.

At a certain point, the generation of samples from sparse feedback becomes very vague and unreliable. Though one central aim is not to bother the user, we can define a concrete situation in which very little user feedback can help a lot. Thus it seems reasonable to explicitly ask the user for feedback under certain circumstances.

Imagine some clauses $m_u^i = m_v^j$; i.e. equivalent clauses in different user model aspects for different users u and v.

By use of collaborative filtering we can use v's positive feedback for u and vice versa. In other words, the set of positive examples can be enriched by examples that were labeled by other users with 'similar' interests. Here, similarity can be defined in a completely discrete manner using our notion of interestingness. Since semantic entailment of different user models like $M_u \approx M_v$ is hard to show, we make use of our relations $\Vdash$ and $\vdash$. The choice of a set of documents d for which the following holds determines the notion of similarity:

1. If $\mathbf{M}_u \Vdash \mathbf{A}_{\mathcal{T}\times\mathcal{C}}(d)$ implies $\mathbf{M}_v \Vdash \mathbf{A}_{\mathcal{T}\times\mathcal{C}}(d)$, u's interest is similar to v's interest: $u \approx v$. Note, that $\approx$ is not necessarily symmetric.

2. Similarity is weaker, if we replace $\Vdash$ by $\vdash$. Accordingly, we denote this case by $u \sim v$.

More labeling information can be extracted from other users where E_v^- coincides with E_u^+. We now consider the case where we will ask u for further feedback (the other case can be formalized canonically).

We now choose a set of documents $D_{F_v}^+(u) = \{d | \langle d, p\rangle \in F_v \wedge \langle d, \cdot\rangle \notin F_u\}$ with positive feedback evidence p for which $\mathbf{M}_u \Vdash \mathbf{A}_{\mathcal{T}\times\mathcal{C}}(d)$ such that for $D_{F_v}^+(u)$ it holds that $v \approx u$. In other words, $D_{F_v}^+(u)$ contains documents v has rated as interesting and u seems to be interested in according to $\mathbf{M}_u$ though u has not given any feedback. Now, $D_{F_v}^+(u)$ can be ordered by the proof costs R we obtain when trying to prove $\mathbf{M}_u \vdash \texttt{p\_interest}_u(a, D, R)$ (see section 89). Then, the 'cheapest' d is the one which is most likely interesting for u, while the most expensive one is a candidate for disinterest.

Similar to $D_{F_v}^+(u)$ we also choose a set $D_{F_v}^-(u) = \{d | \langle d, n\rangle \in F_v \wedge \langle d, \cdot\rangle \notin F_u\}$ which contains documents v did not like. After ordering with respect to $\mathbf{M}_u \vdash \texttt{n\_interest}_u(a, D, R)$, the cheapest is most likely un–interesting while the most expensive might be of interest.

We choose exactly those four documents to ask u for explicit feedback. This way, we obtain new evidence for F_u which defines the most discriminant documents for v and u.

The same method is applied for the dual case $u \approx v$ to obtain more examples in F_v.

If however $\mathbf{M}_u$ and $\mathbf{M}v$ cannot be made disjoint in terms of clauses, the problem seems to be of a different quality: Suppose, as an extreme case, that $\mathbf{M}_u = \mathbf{M}_v$ but for F_u and

F_v it holds that:

$$\langle d, p\rangle \in F_u \quad \text{implies} \quad \langle d, x\rangle \in F_v \rightarrow x < 0$$
$$\langle d, p\rangle \in F_v \quad \text{implies} \quad \langle d, x\rangle \in F_u \rightarrow x < 0$$

for some $p > 0$. Then, it is obvious, that u and v have different interests which cannot be expressed by means of $\mathcal{C}$. In this case we would have to introduce new concepts in $\mathcal{C}$, which poses the problem of interfering with static ontologies.

7.2 Conceptual user models for filtering

Having induced a user model $\mathbf{M}_u$, the generated Prolog program can be used to deduce relevance of a web document. Thus our user models allow for a highly individual content based filtering method. The filtering methods have not been implemented in the current work of OySTER. Our focus lies on inducing such user models. Since user models as Prolog clauses also have a pretty clear procedural semantic, the idea of using them for a filtering process is quite straightforward.

7.2.1 Proving relevance

Given a user model $\mathbf{M}_u$, relevance actually can be proven: If $M_u^+ \vdash_{\mathsf{SLD}} \texttt{p\_interest}_u(a, d)$, u is interested in d (with respect to aspect a) according to the user model. The same holds for disinterest and a program $P_u^- \subseteq M_u^-$.
For example, consider the following rule which is part of $M_{111}^+(\texttt{13})$:[3]

```
p_interest_111(13,Doc) :-
   type__top_reference_peoplelist(Doc,C), C>99,
   cat__top_rec(Doc,E), E>16,
   cat__top_rec_sports(Doc,F), F>42,
   cat__top_rec_sports_water(Doc,D), D>67,
   cat__top_rec_sports_water_scuba_diving_medical_age(Doc,G), G>92.
```

Obviously, $M_{111}^+(\texttt{13})$ describes an interest about lists of people who are associated to scuba diving and related medical issues. Now imagine we had to decide whether an URL with `urlid_0001` is of any interest to user `111`. Looking up `urlid_0001` in the database yields the following classification information:

```
type__top_reference_peoplelist(urlid_0001,100).
cat__top_rec_sports_water_scuba_diving_medical_age(urlid_0001,97).
cat__top_rec_sports_water(urlid_0001,73).
cat__top_science_computer_science_os_dos(urlid_0001,63).
```

[3]Literals have been reordered for the sake of readability.

Obviously the document contains a list of references to people who are somehow associated to scuba diving and DOS. For example, the document might contain a list of people who work on computer simulations of decompression processes in diving. Then, document `urlid_0001` could be proven to be interesting for user `111` with respect to his interest aspect `13`:

1. The first pair of literals concerning the document type can be directly inferred from the first fact (`C=100` and `100 > 99`).

2. Similarly, the fourth and fifth literals can be proved using facts three and four, respectively (`D=73, G=97`)

The second and third literals however cannot be derived directly, but, taking into account background knowledge about the concept hierarchy, the remaining literals can be resolved, too:

```
cat__top_rec(X,D) :-
   cat__top_rec_sports(X,C), D is C - 25.
cat__top_rec_sports(X,D) :-
   cat__top_rec_sports_water(X,C), D is C - 25.
```

Since we know, that `cat__top_rec_sports_water(urlid_0001,73)`, we can prove

```
cat__top_rec(urlid_0001, 23).
cat__top_rec_sports(urlid_0001, 48).
```

Now, `E=23` and `F=48`, and since `23 > 16` and `48 > 42`, we have shown that

```
p_interest_111(13,urlid_0001).
```

Therefore it has been shown that `urlid_0001` obviously is relevant with respect to user `111`'s interests. Proving explicit disinterest is carried out analogously; the goal then is a literal $\texttt{n\_interest\_111}(a, D) \in M^-_{111}$. Negations, i.e. non–interest (or non–disinterest) can be proven by negation as failure: document `urlid_0001` is non–interesting, if there is no proof of $\texttt{p\_interest\_111}(A, \texttt{urlid\_0001})$. This is explained in the next section.

7.2.2 Different levels of relevance

Taking aspects into account again, documents can be of different levels of interestingness, too: Given a document d, for which $M^+_u(a) \vdash_{\mathsf{SLD}} \texttt{p\_interest}(a, d)$, we have shown, that d is relevant to u with respect to a. If, however, the proof fails, and there is a different aspect a', for which $\texttt{p\_interest}(a', d)$, d is still of some interest[4]. Finally, if there is no

[4]Taking into account the search query q, one can quantify the notion of 'some' by trying to classify q and computing $\delta(\mathcal{C}(q), \mathcal{C}(d))$.

a, such that relevance of d can be proven, it is likely to say that d is not interesting. However, d is definitely not interesting if there is some aspect $\overline{a}$ for which $M_u^-(\overline{a}) \models \mathcal{C}(d)$. This way, we can define a hierarchy of different levels of interestingness:

Definition 7.1 (Interestingness) *Let $\mathbf{M}_u = \langle M_u^+, M_u^- \rangle$ be a user model. We define the level of interestingness for relevance of a document d as follows:*

1. *u is interested in a document d, if $\exists a : M_u^+ \vdash$ **p_interest**(u, a, d).*

   We denote this case by $\mathbf{M}_u \Vdash_a \mathbf{A}_{\mathcal{T}\times\mathcal{C}}(d)$.

2. *d could be interesting for u, if $\forall a : M_u^- \nvdash$ **n_interest**(u, a, d).*

   We denote this case by $\mathbf{M}_u \Vdash \mathbf{A}_{\mathcal{T}\times\mathcal{C}}(d)$.

3. *u is indifferent about d, if $\forall a : M_u^+ \cup M_u^- \nvdash$ **p_interest**$(u, a, d) \vee$ **n_interest**(u, a, d).*

   We denote this case by $\mathbf{M}_u \nVdash \mathbf{A}_{\mathcal{T}\times\mathcal{C}}(d)$.

4. *d could be non-interesting for u, if $\forall a : M_u^+ \nvdash$ **p_interest**(u, a, d).*

   We denote this case by $\mathbf{M}_u \nVdash \mathbf{A}_{\mathcal{T}\times\mathcal{C}}(d)$.

5. *u is not-interested in a document d, if $\exists a : M_u^- \vdash$ **n_interest**(u, a, d).*

   We denote this case by $\mathbf{M}_u \Vdash_a \mathbf{A}_{\mathcal{T}\times\mathcal{C}}(d)$.

Furthermore, any successful proof by clauses of $\mathbf{M}_u^+$ or $\mathbf{M}_u(a)$ (of which there might be several) has a certain length (since satisfaction of body literals again may have to be checked against the concept hierarchy). The minimum number of resolution steps used for a proof thus can be interpreted as a quality measurement as well. The simple sum of resolution steps can further be weighted by the δ distance measure in literal proofs. This method allows for a more detailed notion of relevance as demonstrated by the example in section 7.2.1.

Using confidences. In section 7.2.1 we have shown how relevance of a document can be proven. Irrelevant documents are documents for which disinterest can be proven. Failing proofs can be used to derive different levels of interestingness (see last paragraph).
Since in most aspect definitions, the membership of documents to classes is combined with a minimum confidence threshold, one might conclude that for decreasing differences between actual confidence and required threshold, relevance of the whole document also decreases. In order to take such information into account, one would need to extend a hypothesis clause as displayed in equation 5.2 by a return value for confidence which is

computed by a function ρ. The abstract scheme for such rules is shown below:[5]

$$\begin{array}{l} \texttt{p\_interest\_}u(a, D, R) : - \\ \quad \texttt{type\_\_}t_1(D, T_1), ..., \texttt{type\_\_}t_{n_t}(D, T_{n_t}), \\ \quad \texttt{cat\_\_}c_1(D, C_1), ..., \texttt{cat\_\_}c_{n_c}(D, C_{n_c}), \\ \quad \texttt{thresh}(C_1, \vartheta_1), ..., \texttt{thresh}(C_{n_c}, \vartheta_{n_c}), \\ \quad \texttt{rho}([(C_1, \vartheta_1), ..., (C_{n_c}, \vartheta_{n_c})], R). \end{array} \tag{7.1}$$

Note, that during SLD–resolution, all C_i are instantiated and the elements of the result list can be arithmetically evaluated. Here, R is a newly introduced variable which is used to describe an overall threshold. Continuing our example from the last section, one would obtain:

```
p_interest_111(13,Doc,R) :-
   type__top_reference_peoplelist(Doc,C), C>99,
   cat__top_rec(Doc,E), E>16,
   cat__top_rec_sports(Doc,F), F>42,
   cat__top_rec_sports_water(Doc,D), D>67,
   cat__top_rec_sports_water_scuba_diving_medical_age(Doc,G), G>92.
   rho([ (D,67), (E,16), (F,42), (G,92)], R).
```

In this clause, variables in the last literal were instantiated while proving the former classification literals. Accordingly, the following substitution has been applied: {`D/17, E/28, F/48, G/97`}. Then, `rho` could return the sum of differences between thresholds and actual values and return them using the Variable `R`; in our example, `R = 29`. Using a system–wide (predefined) threshold value for `R`, the predicate `rho` might fail—although all other literals have been satisfied. In this example the minimum value for `R` which still allows for the other literals to be provable would be `4`. If, on the other hand, one wants to require a higher precision, the overall threshold for `R` could be set to `15`; thus forcing `rho` to fail. Dual considerations apply to `n_interest`: A higher threshold makes disinterest more precise which results in a more generous filter.

Computing proof cost. A further idea is to use penalties that were collected during a proof in order to determine the quality of a proof. In the example above, two literals were not provable by single resolution steps: The system had to consider clauses from the background knowledge which describe the category hierarchy. Since it was known that `cat__top_rec_sports_water(urlid_0001,73)`, the clauses

```
cat__top_rec(X,D) :-
   cat__top_rec_sports(X,C), D is C - 25.
```

[5] In this example we focus only on document categories. One could easily extend this approach document types as well.

```
cat__top_rec_sports(X,D) :-
   cat__top_rec_sports_water(X,C), D is C - 25.
```

were used to derive two lemmas:

```
cat__top_rec(urlid_0001, 23).
cat__top_rec_sports(urlid_0001, 48).
```

Note, that classification confidence was decreased. In our implementation, category movement (here: generalization) is penalized by a constant factor (25). One could also use the δ measure which allows for a more sophisticated penalty system[6]. Whether one takes δ into account or simply wants to take proof length (which would be equivalent to a generalization penalty of 1 instead of 25)—both require a slightly different representation of both the user model clauses and the concept hierarchies. In such a case, each subgoal needs to report its 'computing cost' which can then be aggregated by a modified version of rho.

$$\begin{array}{l}\texttt{p\_interest\_}u(a,D,R):-\\ \quad \texttt{type\_\_}t_1(t_1,D,T_1,\rho_{t_1}),...,\texttt{type\_\_}t_{n_t}(t_{n_t},D,T_{n_t},\rho_{t_{n_t}}),\\ \quad \texttt{cat\_\_}c_1(c_1,D,C_1,\rho_{c_1}),...,\texttt{cat\_\_}c_{n_c}(c_{n_c},D,C_{n_c},\rho_{c_{n_t}}),\\ \quad \texttt{thresh}(C_1,\vartheta_1),...,\texttt{thresh}(C_{n_c},\vartheta_n),\\ \quad \texttt{rho}([(C_1,\vartheta_1,\rho_{c_1}),\ldots(C_{n_c},\vartheta_n,\rho_{c_{n_t}})],R).\end{array} \tag{7.2}$$

The background knowledge then needs to be reformulated as follows: For each rule

$$\begin{array}{l}\texttt{cat\_\_}c_i(D,R):-\\ \quad \texttt{cat\_\_}c_j(D,S),\\ \quad \texttt{R is S} - 25.\end{array}$$

one needs to add two new rules. In the case where we are trying to prove the membership of a document to a category directly, this becomes

$$\begin{array}{l}\texttt{cat\_\_}c_i(c_i,D,0,R):-\\ \quad \texttt{cat\_\_}c_i(D,R).\end{array} \tag{7.3}$$

If on the other hand, we need to prove membership by taking into account subsumption, things get a bit more complicated:[7]

$$\begin{array}{l}\texttt{cat\_\_}c_s(c_i,D,Cost,R):-\\ \quad \texttt{cat\_\_}c_j(c_s,D,OldCost,OldR),\\ \quad Cost\ \texttt{is}\ OldCost+\delta(c_j,c_s),\\ \quad R\ \texttt{is}\ OldR - \texttt{GENPENALTY}.\end{array} \tag{7.4}$$

[6]In order to avoid the result of learning being predetermined by the input, this was not realized within this work: the δ-measure was used to simulate user interests which were used to generate feedback. Taking into account the δ–measure in the learning process would mean to provide much more information about the user to the system. This argument does not hold for real world applications where the feedback is real feedback from real users. In such cases one could include the δ–measure; most likely with a significant accuracy gain.

[7]Categories c_i and c_j are as chosen by the rule displayed in equation 90.

Imagine, one would have to prove a `cat_c`$_i$ literal in a goal as displayed in 7.2. If the document D under consideration actually belongs to c_i, the proof will succeed using the rule as shown in equation 7.3, as only the body literal can be satisfied by a fact from the background (classification) knowledge. If there is no such evidence, category subsumption has to be taken into account. In such a case, the rule shown in equation 7.4 is triggered:
It has to be shown, that D somehow belongs to category c_i. Since `cat_c`$_i(D, R)$ failed, one considers subconcepts c_j which are subsumed by categories c_s. For subsumption paths of length 1 it holds that $c_j = c_s$, such that the first body literal becomes

$$\texttt{cat\_c}_s(c_s, D, OldCost, OldR)$$

This literal can only be satisfied by rules of the former type (equation 90). If the proof fails again, the same rule scheme as in equation 7.4 is applied again. In this recursive step, the variables are instantiated as follows: $c_s := c_j$, $c_i := c_s$ and the new c_j are subconcepts of c_s (formerly c_j).
Either way, if the proof succeeds (with a subsumption path length of at least 1), we obtain values $OldCost$ describing the proof annotated cost and $OldR$ for subsumption penalties (`GENPENALTY`, a constant `25` in the preceding examples). To those values, δ–distances and generalization penalties are added to yield $Cost$ and R, respectively.
Such an approach would allow for the setting of the `GENPENALTY` to a minimum value of `1` thus calculating path length (that is, proof length). Together with the δ cost, the proof length could be further weighted: $Cost$ `is` $OldCost + \delta(c_j, c_s) \cdot (OldR + 1)$.

III

CONCLUSION AND PROSPECTS

Chapter 8

SUMMARY AND CONCLUSION

8.1 Summary

This thesis presented both a formal and practical approach to user modeling by inducing conceptual user models.
After a brief survey of the contributing fields of user modeling and machine learning, we described the parallels of machine learning and user modeling and introduced a formal learning problem in section 2.3.2.
In the second chapter, we described the OySTER system as a testbed for evaluating user modeling techniques. This was followed by a description of related work.
The core chapter of the thesis introduced the notion of conceptual user models. The idea of representing models of the user's interest by conceptual user models in terms of Prolog clauses is a novel approach in user modeling research. The big advantage is that such models can be visualized as trees (as shown in figure 4.1), explained to and edited by the user (as, e.g., shown in figure 3.2) and represented and learned by Prolog clauses as shown in formula 4.11 and in figure 7.1.
A further new and promising idea is that of explicit modeling of disinterest which allows for more distinct information filtering and sample enhancement.
Based upon the representational properties of user models, we developed a framework for induction of such user models in chapter 5. Representing conceptual user models as Prolog clauses defines a clear machine learning task for inductive logic programming which is described in section 5.3.

8.2 Conclusion

It has been shown, that ILP delivers accurate results for sufficiently large samples and that accuracy of results for very small samples can be increased by taking into account mutual information from other interest aspects as well as from explicit disinterest. In

sections 6.2– 6.4, we presented results that were derived based on pessimistic assumptions.
For an initial sample which contained 10 examples only, accuracy of the compressing rules reached 57% (depending on the specificity of the user's interest, see figure 6.4). The relative average accuracy gain when taking into account 10 more examples from different aspects reached approximately 11 % (6.6 % for user 444 and an initial accuracy of 57%, see table 6.3). Using a pessimistic simulation for user feedback, accuracy gain dropped to 3.82 % (see table 6.5).

This thesis presented at least three novel approaches to the current research in machine learning for user modeling:

1. The use of concept hierarchies for describing document contents overcomes the drawbacks of representing content by word vectors. This allows for better inspectability of user models which is one key feature required for a increasing user acceptance.

   Furthermore, this approach allows for discriminating knowledge about users and knowledge about the domain (see assumption A-1 in section 5), although the lack of information about the domain poses a harder learning problem.

2. Explicit representation of interest and disinterest allows for the generation of larger samples than actually provided by sparse feedback.

   Furthermore, the label data derived out of feedback data helps to define a better bias for the user model induction process. Additionally, the dual concepts together with Prolog's negation as failure already allows for a multi truth–valued like proof method for recommendation which can be further enhanced using weighted feedback values.

   This approach was motivated by the idea of discriminating feedback from samples (see A-3, section 5) and by the need to generate larger samples from little feedback data, since we induce user models from scratch instead of refining predefined initial models (A-4).

3. In order to induce conceptual user models based upon conceptual input data, we employ methods of inductive logic programming. The logic programs can be used to prove relevance or irrelevance of retrieved documents. It is noteworthy again, that the logic programs derived do not describe the user's interest by means of word occurrences but by content based conceptual descriptions only.

Of course, our approach also has at least three major disadvantages:

1. The overall performance of the system cannot be measured in terms of the quality of the derived user models only. It crucially depends on the quality of text classifiers which produce the conceptual description. The distinction between document and category is invisible to the user—and thus a faulty classification may lead to an erroneous recommendation, although the user model is correct.

2. In this approach we restricted ourselves to a rather static ontology of document types and contents. Static, handcrafted ontologies have at least two disadvantages: Firstly the domain that is to be covered cannot be entirely overseen at the time of building the ontology. This means that for growing document sets, and an increasing number of documents, the need for more and more special categories will arise. Secondly, one ontology for all users presupposes a unique understanding of all categories for all users—which most certainly is not the case.

3. The system as described in this thesis depends on explicit feedback. Although samples can be enlarged in order to increase precision up to a certain point, it has been shown, that larger samples (i.e. more feedback) are the only reliable method for gaining more precise user models. At this point, the tradeoff between bothering the user and precision gain has to be evaluated very carefully.

Taking into account the 'ten commandments', of user adaptive systems ([Miller, 2000], as briefly described in section 2.1.4; see table 8.1) an evaluation of our approach yields the following result:

1	Make many correct conversational moves for every error made	?
2	Make it very easy to override and correct your errors	✓
3	Know when you are wrong	?
4	Don't make the same mistake twice	✓
5	Don't show off	✓
6	Be able to talk explicitly about what you are doing and why	✓
7	Understand the implications of interaction on all levels	–
8	Adapt to individual, cultural, social, contextual differences	✓ / –
9	Be aware of what the user knows (don't repeat yourself)	✓
10	Be cute only to the extent that it helps	✓

Table 8.1: The ten commandments of Human–Computer Interaction

The system presented in this thesis is an approach to more transparent individual user modeling. The subject of the user models are aspects of the user's interest—thus also covering a certain type of contextual differences. Still, the system is based on pure individual, content based user models which fail to explain cultural, social, or—in general—collaborative effects (commandment 8).

Conversational moves in context of a search engine are query/response tuples. The problem is to define what an 'error' means in this context. If a result which contains the desired information is a correct answer then, due to recall, the overall search procedure certainly meets the requirement of the first commandment. If, on the other hand, an error is defined as a greatest lower bound on accuracy, our approach qualifies for a low error probability provided there is sufficient feedback. Taking into account the categorical imperative of HCI, "Don't bother the user", one only has little feedback available and error probability increases, thus violating the first commandment. Similarly, the need for explicit feedback as the only source for samples shows, that commandment seven certainly is not fulfilled.

Commandments 2 and 6 are certainly met since they are a direct implication of our central idea—conceptual user models. The user models can be easily visualized, the filtering process is lucid to the user and can be verbalized such that the whole system becomes scrutable. Furthermore, the user interface (see figure 3.2) already allows for correcting the underlying user model as well as the actual search result (see the 'Edit classification info' facility in figure 3.3).

The system cannot be sure, whether it is wrong or not (though a measurable sample accuracy may give a hint about the reliability of hypotheses). Nevertheless, once the system knows it was wrong, this information is incorporated into the user model and helps to refine filters.

Chapter 9

FUTURE WORK AND OPEN PROBLEMS

Induction of conceptual user models as introduced in this thesis offers a very promising base for further research. Accordingly, there are many prospects for both further theoretical and practical work.

9.1 Inducing concpetual user models

For each problem identified, tackled and solved during our work, a whole set of even more interesting problems arose. Some of those problems which are worth a deeper investigation are described in the following paragraphs.

Generating samples from feedback. The Γ functions described and used within our evaluation yielded a gain of 10%, in relation to initial accuracy, in 20 per cent of all cases with five examples only.
For an accuracy of 50% after plain learning this means 55% accuracy which is still far to low. Therefore, different methods for sample generation have to be further investigated. Two approaches seem to be very promising:

1. Using the δ–measure, reliability of Γ samples could be increased:
   Instead of applying Γ_4 to the whole domain (i.e. transforming every negative feedback with respect to a into a positive example for $M_u^+(a')$), inherent noise could be reduced by taking into account only those negative examples for a, which are within a δ–region around the current center of a. Similar considerations apply for Γ_3 and $M_u^-(a')$. For the reverse case, only positive examples for a outside a δ–region around the center of a should be used to generate negative examples for $M_u^+(a)$ by Γ_1 and positive examples for $M_u^-(a)$ by Γ_2.

   It remains to be evaluated if the tradeoff between noise reduction and sample shrinking pays off in terms of accuracy.

2. During our evaluation we only considered pairwise Γ enhancement. It is obvious, that for more than two aspects with a sufficient average δ distance the sample size can be increased even further—even if noise reduction restrictions as described above are applied. This was shown in section 6.3.4. It remains to be explained, how to choose a proper δ distance and how many different aspects should be taken into account in order to determine the ideal trade–off between sample enlargement and additional noise.

Learning from real and little data. The initial idea behind implementing OySTER was to provide a testbed for different user modeling approaches. One main motivation was to allow for an evaluation of methods with real world data. As described in the introduction to chapter 6, we encountered severe problems in trying to obtain real world data: Users who worked with OySTER gave too little feedback (less than five examples) and in general had widespread interests which were not covered by the ontology and which were not divided into aspects. In consequence, our work was evaluated against simulated user feedback.
Nevertheless, OySTER offers the opportunity of collecting real world data. Given an ontology, text classifiers can be trained very efficiently for almost any domain. Using OySTER as a web portal search engine, one could observe users in a rather static context which focuses user interests onto a certain domain under consideration. Again, it remains to be explained how feedback can be obtained in such environments.
The idea of asking for feedback where needed has already been discussed in section 7.1.3. Asking for feedback in order to yield a better precision in each document category gave rise to newly invented concepts. This leads to the following open question.

Learning ontologies and user models simultaneously. If there is a model $\mathbf{M}$ that applies for several users u_i, $\mathcal{C}$ obviously is too coarse. Then, by introducing new categories (i. e. splitting clusters in a clustering approach or predicate invention in an ILP approach), the F_{u_i} now induce new $\mathbf{M}_{u_i}$.
The pitfall which is likely to be overlooked is that dynamic category hierarchies also demand a constant re-training of classifiers. The problem to re–classify all documents is even harder. Once all documents are reclassified, the samples $\mathbf{f}$ are changed inherently: Recall, that all documents d in $\mathbf{f}$ are interpreted as conceptual descriptions which then naturally must change.
Similar problems occur when examining the phenomenon of concept drift and concept shift. Concept drift means a slow movement of the user interest; in our case it forces re-learning of interest aspects as accuracy decreases over time. Concept shift means sudden leaps in the user's interest. In such cases, current interest aspects are not involved (though some might be less important after a shift) but rather force rapid induction of new aspects (for example during Olympic games with a temporary interest in certain

disciplines which were uninteresting before and will be after the games). Both phenomena also lead to situations where the current ontology needs to be refined in order to be able to describe the new interest aspects.
Learning ontologies and user models simultaneously poses an interesting learning problem which deserves further research. One must bear in mind, that the result of such a learning process might pose even larger problems. The tradeoff thus has to be estimated very carefully. Within our work, however, it was impossible to also include re–training classifiers and re–learning user models.
Nevertheless, the task of collaboratively learning an ontology or rather one ontology with user dependent views on it defines another highly interesting research question. Results will be of great importance as they provide us with methods that free us from the need of handishly defining (static) ontologies which will always be the subject of criticism.

Order sorted ILP. One main initial motivation for this work was also to incorporate order sorted ILP as, e.g. described in [Müller, 1995]. Actually the distinction between document types and document categories was motivated by the idea of representing types as sorts and categories as predicates. This approach was described in an example for order sorted ILP operators in section 5.3.
Since the learning problem for this domain has been specified in detail in section 5.3, it seems adequate to implement a more problem specific learning system. One successor system of Progol, the Aleph system[1], is completely written in Prolog making use of the Yap Prolog interpreter[2]. Thus, an adaption of Aleph and its underlying methods to the domain of order sorted user models poses a further interesting research question.

Evaluation: Accuracy vs. user satisfaction. The evaluation presented in this thesis is an evaluation of the learning algorithms only—with respect to accuracy gained measured by a test sample. Our approach of discriminating classification and user modeling was mainly driven by the demand of independently evaluating the performance of the user modeling component. In HCI however, the scale on which adaptive systems should be measured is defined by user satisfaction.
In order to evaluate OySTER as an adaptive web search engine we would have to automate and incorporate the user modeling process into the current prototype and then evaluate the whole system by investigating the change in user satisfaction. The problem in such an empirical test is that several components of the whole system contribute to the overall performance but the outcome cannot be traced back to each module. In such an evaluation many more aspects of HCI would have to be taken into account: Interface design contributes to interaction and perception of results (including feedback) and thus

[1] http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph_toc.html
[2] http://www.ncc.up.pt/~vsc/Yap/

affects user satisfaction in several ways. Ontology design poses the problem of different understandings of categories by different users which leads to misunderstanding of classification results; classifiers themselves are also a constant source of noise.
This list could be added to further—but luckily, the system developed during the course of the thesis offers an ideal workbench for testing different interfaces, classifiers and user modeling techniques and its open architecture also allows for gathering data that can be used for carrying out an evaluation of user satisfaction.

9.2 Implementational issues

Most issues discussed in the thesis have been realized with a first prototype of an adaptive search engine. However, there still remains a lot of work to be done.

System integration. First of all, the user modeling component has not been integrated into the online accessible system. This is due to the fact, that evaluation was carried out on simulated users. Since the definition of initial user models is already supported, we will first include filtering mechanisms. In a second step, the user modeling process shall be integrated.

Multi agent system. All parts of the whole system are realized as independent agents. Since the implementation of a meta search engine was not the focus of the thesis, wrapper agents and database update agents are minimalistic ad–hoc solutions. These need to be re-implemented and enhanced; for example the database is not updated by revisiting crawlers which check for persistence of URLs. However, the system is currently distributed over three machines and is (with few exceptions) very stable and robust against failure of single components. For higher system load an intelligent agent scheduling with respect to computing resources is indispensable.

Upscaling. Currently, the ontology covers only a tiny part of the web. With a growing ontology, computational resources increase. Instead of upscaling, it seems more reasonable to generate several different instantiations of OySTER for different, enclosed domains. One promising idea is to use OySTER as a search engine for portals or intranets.

User interface. User feedback was assumed to exist. The current prototype relies on explicit feedback from pop–up windows asking for user ratings for search results. Such a system behavior is by no means ‘user friendly’ behavior since it severely violates the rule of not bothering the user.

Classifiers. Feedback is given in relation to documents, but example labels are based upon target concepts which are made up from document categories. This means, that the overall performance of OySTER—and thus user satisfaction—heavily depends on the accuracy of classifiers. The development of more accurate classifiers is the subject of ongoing student projects.

One important feature of classifying documents with respect to several categories is to deliver relative confidences instead of absolute values. Currently, the classifier delivers a vector $\mathbf{A}_C(d) = \langle c_1 :: p_1, c_2 :: p_2, \ldots, c_n :: p_n \rangle$, where each $p_i \in [0, 100]$. The resulting noise could be decreased, if the sum of all p_i is normalized to 100 and the importance of p_i is weighted (decreasing with growing i).

Thinking a step further

The experience gathered while working on *Inducing Conceptual User models* and with OySTER leads to further research questions. The most intriguing question is how conceptual user models and ontologies could be developed and learned simultaneously. From the viewpoint of OySTER as an application, one would conclude that a generic meta search engine is only a suboptimal application domain for our theory. As already mentioned above, web portals seem to establish a more promising application domain for different reasons. One could imagine a user adaptive forum for a loosely organized special interest group of researchers. By contributing to the forum and interactively building a glossary of key words, one could induce a concept graph (rather than a hierarchy). Then, ontologies are user dependent as well: they are user centered views on the concept graph. At this point, we could incorporate 'relevance' in the sense of importance of features into the user model (see assumption A-2 in section 5). This allows for customizing classifiers as well: according to the different models of relevance, the same document could be classified into different categories for different users.

User adaption would result in user tailored representations of content and recommendations for further retrieval. Data collected in such an environment by far exceeds the data available in the approach described here: One could not only collect explicit or implicit feedback but also relationships between documents by means of collaboratively analyzing transitions between documents and sub–graphs.

Furthermore, the personalization of both the representation of categories and the classifiers need to be described in the user model. This provides the learning algorithm with a whole set of useful background knowledge (thus weakening the assumptions A-1 and A-4) which was not available in the approach presented in this thesis.

Conclusion

This thesis presented the novel approach of conceptual user models. Easy inspectability and understandability of conceptual user models was demonstrated and scrutability of the system's actions was shown.
It has been shown, that ILP can be applied for the task of inducing user models from feedback, and that feedback concerning different interest aspects can be used for sample enlargement.

Results were evaluated independently of domain knowledge within a clear machine learning problem definition.
The whole approach is based on several assumptions (A-1 to A-5), which:

- do not presuppose certain requirements on the domain (like, for example, independence of features)
- do not require additional background knowledge about the domain that is incorporated into the user model (like word occurrences)
- do not assume prior knowledge about the user (as, for example, initial user models provided by the user himself or a minimum amount of feedback needed)

The results obtained constitute a baseline of performance using ILP as the machine learning method in user modeling. Taking into account more knowledge, the results can only be improved.

During the work on the thesis, most parts of what has been described was realized in a meta web search engine prototype, OySTER. All code, documentation and data is published and accessible through the world wide web from the project's homepage: `http://www.aye-aye.de/oyster/`.

IV

LITERATURE AND APPENDICES

References

[André and Rist, 2000] André, E. and Rist, T. (2000). Presenting through performing: On the use of multiple lifelike characters in knowledge-based presentation systems. In *Proc. of the Second International Conference on Intelligent User Interfaces (IUI 2000)*.

[Armstrong et al., 1995] Armstrong, R., Freitag, D., Joachims, T., and Mitchell, T. (1995). Webwatcher: A learning apprentice for the world wide web. In *Working Notes of the AAAI Spring Symposium: Information Gathering from Heterogeneous, Distributed Environments*, pages 6–12, Stanford University. AAAI Press.

[Balabanovic, 1998] Balabanovic, M. (1998). An interface for learning multi-topic user profiles from implicit feedback. In *AAAI-98 Workshop on Recommender Systems*, Madison, Wisconsin.

[Balabanovic and Shoham, 1997] Balabanovic, M. and Shoham, Y. (1997). Fab: Content-based, collaborative recommendation. *Communications of the ACM*, 40(3).

[Bauer et al., 1999] Bauer, M., Dengler, D., Meier, M., and Paul, G. (1999). Trias: Trainable information assistants for cooperative problem solving. In Rautenstrauch, C., Wrobel, S., Perner, P., Paass, G., Jörding, T., and Schröder, O., editors, *Lernen, Wissen, Adaptivität — Proc. 7th ABIS–99*, Otto–von–Guericke Universität Magdeburg. 39016 Magdeburg. ISBN 3–929757–26–5.

[Belew, 2000] Belew, R. (2000). *Finding Out About*. Cambridge University Press.

[Berthold and Jameson, 1999] Berthold, A. and Jameson, A. (1999). Interpreting symptoms of cognitive load in speech input. In [Kay, 1999].

[Billsus and Pazzani, 1997] Billsus, D. and Pazzani, M. (1997). Learning probabilistic user models. In Jameson, A., Paris, C., and Tasso, C., editors, *User Modeling: Sixth International Conference, UM97*. Springer Wien New York, Vienna, New York. Workshop Paper.

[Billsus and Pazzani, 1999] Billsus, D. and Pazzani, M. (1999). A hybrid user model for news story classification. In [Kay, 1999], pages 99–108.

[Braun et al., 2001] Braun, T., Eilert, S., Dörfler, A., Haase, R., Kleber, A., Lahrkamp, I., Reiter, C., Rolf, R., Trenkamp, F., and Sievertsen, F. (2001). Bikini—Benutzerbasierte Intelligente Klassifikation voninformation aus dem Internet . Technical report, Institute for Semantic Information Processing, University of Osnabrück.

[Clodo et al., 2000] Clodo, C., Schwarzkopf, E., and Bauer, M. (2000). Einsatz adaptiver Techniken für die UM2001. In *Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen*.

[Cohen, 1995] Cohen, W. W. (1995). Text categorization and relational learning. In Prieditis, A. and Russell, S., editors, *Machine Learning*, pages 124–132. Morgan Kaufmann.

[Craven et al., 1998a] Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., and Slattery, S. (1998a). Learning to extract symbolic knowledge from the world wide web. Technical Report CMU-CS-98-122, School of Computer Science, Carnegie Mellonc University. shorter version published as [Craven et al., 1998b].

[Craven et al., 1998b] Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., and Slattery, S. (1998b). Learning to extract symbolic knowledge from the world wide web. In *Proc. of 15th Nat. Conf. on Artificial Intelligence (AAAI-98)*. AAAI Press.

[Eilert et al., 2001] Eilert, S., Mentrup, A., Müller, M. E., Rolf, R., Rollinger, C.-R., Sievertsen, F., and Trenkamp, F. (2001). Bikini: User adaptive news classification in the world wide web. In *Workshop "Machine Learning for User Modeling", 8th Intl. Conf. on User Modeling*.

[Etzioni, 1997] Etzioni, O. (1997). Moving up the information food chain: Deploying softbaots on the world wide web. *AI Magazine*.

[Etzioni and Zamir, 1999] Etzioni, O. and Zamir, O. (1999). Grouper: a dynamic clustering interface to web search results. In *Proc. 8th World Wide Web Conference*, Toronto.

[Fensel et al., 1998] Fensel, D., Erdmann, M., and Studer, R. (1998). Ontobroker: How to make the www intelligent,. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based Systems, Workshop (KAW98)*, Banff, Canada.

[Fiechter and Rogers, 2000] Fiechter, C.-N. and Rogers, S. (2000). Learning subjective functions with large margins. In [Rogers and Iba, 2000], pages 40–47.

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability*. Freeman and Company.

[Göker and Thompson, 2000a] Göker, M. and Thompson, C. (2000a). The adaptive place advisor: A conversational recommendation system. In *Proceedings of the 8th German Workshop on Case Based Reasoning*, Lammerbuckel, Germany.

[Göker and Thompson, 2000b] Göker, M. and Thompson, C. (2000b). Personalized conversational case-based recommendation. In *Proceedings of the 5th European Workshop on Case Based Reasoning*, Trento, Italy.

[Hammer et al., 1997] Hammer, J., Garcia-Molina, H., Cho, J., Crespo, A., and Aranha, R. (1997). Extracting semistructured information from the web. In *Proceedings of the Workshop on Management of Semistructured Data, ACM SIGMOD International Conference on Management of Data*.

[Heißing, 2000] Heißing, C. (2000). Klassifizieren von URLs durch Generalisierung von Pfadnamen. Master's thesis, Institute for Semantic Information Processing, University of Osnabrück.

[Heißing, 1999] Heißing, C. (1999). Clustering URLs by Pathname Generalisation. In *LWA 99: Lernen, Wissen und Adaptivität*, pages 301–306, Magdeburg. Universität Magdeburg.

[Howe and Dreilinger, 1997] Howe, A. E. and Dreilinger, D. (1997). Savvysearch: A metaserach engine that learns which search engines to query. *AI Magazine*.

[Huck et al., 1998] Huck, G., Fankhauser, P., Aberer, K., and Neuhold, E. J. (1998). Jedi: Extracting and synthesizing information from the web. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems*, pages 32–43, New York. IEEE-CS Press.

[Jameson et al., 1997] Jameson, A., Paris, C., and Tasso, C., editors (1997). *User Modeling: Proceedings of the Sixth International Conference, UM97*. Springer.

[Joachims et al., 1996] Joachims, T., Freitag, D., and Mitchell, T. (1996). Webwatcher: A tour guide for the world wide web. Technical report, CMU.

[Joachims et al., 1997] Joachims, T., Freitag, D., and Mitchell, T. (1997). Webwatcher: A tour guide for the world wide web. In *Proceedings of IJCAI 97*.

[Jörding, 1999] Jörding, T. (1999). Adaptive shopping in the web: Individual product presentations for every customer. In *Proc. HCI International 1999*.

[Kay, 1999] Kay, J., editor (1999). *User Modeling: Proceedings of the Seventh International Conference, UM99*. Springer.

[Kietz and Wrobel, 1992] Kietz, J.-U. and Wrobel, S. (1992). Controlling the complexity of learning in logic through syntactic and task-oriented models. In Muggleton, S., editor, *ilp*.

[Kobsa, 1986] Kobsa, A. (1986). Generating a user model from wh-questions in the vielang system. In Hellwig, P. and Lehmann, H., editors, *Trends in der Linguistischen Datenverarbeitung*. Olms.

[Kobsa and Trost, 1984] Kobsa, A. and Trost, H. (1984). Representing belief models in semantic networks. In Trappl, R., editor, *Cybernetics and Systems Research II*. North–Holland.

[Kushmerick, 1997] Kushmerick, N. (1997). *Wrapper Induction for Information Extraction*. PhD thesis, University of Washington.

[Kushmerick and Doorenbos, 1997] Kushmerick, N. and Doorenbos, R. B. (1997). Wrapper induction for information extraction. In *IJCAI 97*.

[Langley, 1999] Langley, P. (1999). User modeling in adaptive interfaces. In [Kay, 1999], pages 357–370.

[Lieberman, 1995] Lieberman, H. (1995). Letizia: An agent that assists web browsing. In *Proc. of the IJCAI 95*, pages 924–929.

[Macskassy et al., 2000] Macskassy, S., Dayanik, A. A., and Hirsh, H. (2000). Information valets for intelligent information access. In [Rogers and Iba, 2000], pages 68–73.

[Maes and Sheth, 1993] Maes, P. and Sheth, B. (1993). Evolving agents for personalized information filtering. In *Proceedings of 9th IEEE Conference on Artificial Intelligence for Applications*, Los Alamitos, CA, USA. IEEE Comput. Soc. Press.

[Miller, 2000] Miller, C. A. (2000). Rules of Etiquette — or How a Mannerly AUI should Comport Itself to Gain Social Acceptance and be Perceived as Gracious and Well-Behaved in Polite Society. In *AAAI Spring Symposium on Adaptive User Interfaces*, pages 80–81, Menlo Park. AAAI Press.

[Mitchell, 1997] Mitchell, T. (1997). *Machine Learning*. McGraw Hill.

[Mladenic, 1998] Mladenic, D. (1998). *Machine Learning on non-homogeneous, distributed text data*. PhD thesis, University of Ljubljana, Slovenia.

[Möller et al., 1999] Möller, G., Carstensen, K.-U., Diekmann, B., and Wätjen, H. (1999). Automatic classification of the world-wide web using the universal decimal classification. In McKenna, B. and Graham, C., editors, *Proceedings of the 23rd International Online Information Meeting (Online99)*, pages 231–237.

[Moukas, 1996] Moukas, A. (1996). Amalthea: Information discovery and filtering using a multiagent evolving ecosystem. In *Proc. 1st Int. Conf. on The Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM)*, London.

[Muggleton, 1995] Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, (13):245–286.

[Muggleton and Buntine, 1988] Muggleton, S. and Buntine, W. (1988). Machine Invention of First–Order Predicates by Inverting Resolution. In *Proceedings of the International Workshop on Machine Learning 1988*, Ann Arbor. Morgan Kaufmann.

[Muggleton and Feng, 1990] Muggleton, S. and Feng, C. (1990). Efficient induction of logic programs. Tokyo. Ohmsha Publishers.

[Müller, 1995] Müller, M. (1995). Eine Erweiterung der inversen Resolution auf sortierte Hornklauselmengen. Master's thesis, Universität Osnabrück, Institut für semantische Informationsverarbeitung.

[Müller, 1996] Müller, M. E. (1996). Zur Problematik der Bewertung heterogener Informationen am Beispiel der Umweltbewertung alternativer Produktionspläne. In Grützner, R., editor, *Werkzeuge für Simulation und Modellbildung in Umweltanwendungen*. Gesellschaft für Informatik, FA 4.6, AK 5.

[Müller, 1999] Müller, M. E. (1999). An Intelligent Multi-Agent Architecture for Information Retrieval from the Internet. Technical report, Institute for Semantic Information processing, Univ. Osnabrück.

[Oppermann and Specht, 1999] Oppermann, R. and Specht, M. (1999). A nomadic information system for guidance and learning on demand. In *Proc. of the Eighth International Conf. on Human Computer Interaction, HCI–99*.

[Pazzani et al., 1996] Pazzani, M., Muramatsu, J., and Billsus, D. (1996). Syskill & Webert: Identifying interesting Web Sites. In *Proc. of the National Conference on Artificial Intelligence*, Portland, OR.

[Perkowitz et al., 1997] Perkowitz, M., Doorenbos, R., Etzioni, O., and Weld, D. (1997). Learning to understand information on the internet: an example-based approach. *Journal of Intelligent Information Systems*, 8(2).

[Petrelli et al., 1999] Petrelli, D., Angeli, A. D., and Convertino, G. (1999). A user–centered approach to user–modeling. In [Kay, 1999].

[Plotkin, 1970] Plotkin, G. D. (1970). A note on inductive generalization. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 5*, chapter 8, pages 153 – 163. Edinburgh Univ. Press, Edinburgh.

[Preece, 1994] Preece, J. (1994). *Human–Computer Interaction*. Addison Wesley.

[Pretschner and Gauch, 1999] Pretschner, A. and Gauch, S. (1999). Personalization on the web. Technical Report ITTC–FY2000–TR–13591–01, Dep. Electrical Engineering and Computer Science, Univ. Kansas.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, (1):81–106.

[Quinlan, 1990] Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3):239 – 266.

[Quinlan, 1991] Quinlan, J. R. (1991). Determinate literals in logic programming. In *IJCAI 91*. IJCAI.

[Quinlan, 1993] Quinlan, J. R. (1993). *C 4.5 — Programs for Machine Learning*. Morgan Kaufmann.

[Rogers et al., 1999] Rogers, S., Fiechter, C.-N., and Langley, P. (1999). An adaptive interactive agent for route advice. In *Proc. 3rd Int. Conf. on Autonomous Agents*. ACM Press.

[Rogers and Iba, 2000] Rogers, S. and Iba, W., editors (2000). *Adaptive User Interfaces*, number SS–00–01 in Technical Report, Menlo Park, California. AAAI Press.

[Ronthaler, 1998] Ronthaler, M. (1998). Osiris: Qualitative Fortschritte bei der Literaturrecherche. In Dassow, J. and Kruse, R., editors, *Informatik '98 : Informatik zwischen Bild und Sprache*. Springer.

[Ronthaler, 2000] Ronthaler, M. (2000). *Dialogschnittstellen an Online–Informationssystemen: Notwendigkeit, Leistungsfähigkeit und Entwicklungsmöglichkeiten am Beispiel des OSIRIS–Systems*. PhD thesis, Fachbereich Sprach– und Literaturwissenschaft, Universität Osnabrück.

[R.Penner and Steinmetz, 2000] R.Penner, R. and Steinmetz, E. S. (2000). Digbe: Adaptive user interface automation. In *AAAI Spring Symposium on Adaptive User Interfaces*, pages 98–101, Menlo Park. AAAI Press.

[Schwab et al., 2000a] Schwab, I., Koychev, I., and Kobsa, A. (2000a). Learning about users from observation. In [Rogers and Iba, 2000], pages 102–106.

[Schwab et al., 2000b] Schwab, I., Koychev, I., and Pohl, W. (2000b). Learning to recommend from positive evidence. In Lieberman, H., editor, *Proc. Int. Conf. on Intelligent User Interfaces, (IUI-2000)*.

[Selberg and Etzioni, 1995] Selberg, E. and Etzioni, O. (1995). Multi–service search and comparison using the metacrawler. *Proc. of the WWW Conference*.

[S.Gaffney et al., 1996] S.Gaffney, Hettich, S., Khoo, G., Kim, D., Klefstad, R., Ludeman, A., Omori, K., Pazzani, M., Semler, D., and Yap, P. (1996). Syskill & Webert. Technical report, UCI.

[Shakes and Langheinrich, 1997] Shakes, J. and Langheinrich, M. an Etzioni, O. (1997). Dynamic reference sifting: A case study in the homepage domain. In *Proc. 6th Int. WWW Conf.*, pages 189–200.

[Shannon and Weaver, 1949] Shannon, C. and Weaver, W. (1949). The mathematical theory of communication. Technical report, University of Illinois, Urbana.

[Simons, 1997] Simons, J. (1997). Using a semantic user model to filter the World Wide Web proactively. In [Jameson et al., 1997], pages 455–456.

[Specht and Kobsa, 1999] Specht, M. and Kobsa, A. (1999). Interaction of domain expertise and interface design in adaptive educational hypermedia. In *Proc. of the Eighth International World Wide Web Conference, Workshop paper*.

[Spooner and Edwards, 1997] Spooner, R. I. W. and Edwards, A. D. N. (1997). User modelling for error recovery: A spelling checker for dyslexic users. In [Jameson et al., 1997], pages 147–157.

[Thompson and Göker, 2000] Thompson, C. A. and Göker, M. H. (2000). Learning to suggest: The adaptive place advisor. In *AAAI Spring Symposium on Adaptive User Interfaces*, pages 130–135, Menlo Park. AAAI Press.

[Trewin and Pain, 1997] Trewin, S. and Pain, H. (1997). Dynamic modelling of keyboard skills: Supporting users with. In Jameson, A., Paris, C., and Tasso, C., editors, *User Modeling: Proceedings of the Sixth International Conference, UM97*, pages 135–146. Springer Wien New York, Vienna, New York.

[Valiant, 1984] Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27:1134–1142.

[van Rijsbergen, 1979] van Rijsbergen, C. J. (1979). *Information Retrieval*. Butterworths.

[Widyantoro et al., 1999] Widyantoro, D. H., Yin, J., Nasr, M. S. E., Yang, L., Zacchi, A., and Yen, J. (1999). Alipes: A swift messenger in cyberspace. In *Proc. AAAI Spring Symposium on Intelligent Agents in Cyberspace*, Palo Alto.

[Wirth, 1989] Wirth, R. (1989). Completing logic programs by inverse resolution. pages 239 – 250, London/San Mateo, CA. Pitman/Morgan Kaufmann.

[Zamir and Etzioni, 1998] Zamir, O. and Etzioni, O. (1998). Web document clustering: A feasibility demonstration. In Croft, W. B., Moffat, A., van Rijsbergen, C., Wilkinson, R., and Zobel, J., editors, *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 46–54, Melbourne, Australia. ACM Press, New York.

Appendix A

EVALUATION DETAILS

A.1 Sample results for a single aspect learning task

The following four figures illustrate the user interests as used in the evaluation of section 6.2. They were generated using the feedback browser interface (`fbb`) and show the relevant part of the feedback distribution.

[o][o][o][o][o][+][+]	7	2	2	84: top.science.computer_science.artificial_intelligence.reasoning.deduction
[o][o][o][o][o]	4	0	0	81: top.science.computer_science.artificial_intelligence.reasoning.nonmonotonic
[−][o]	2	−1	−1	58: top.science.computer_science.artificial_intelligence.robotics
[−][o][o][o]	4	−1	−1	85: top.science.computer_science.artificial_intelligence.search
[o][+]	2	1	1	119: top.science.computer_science.artificial_intelligence.user_modeling
[−][o][o][o][o]	5	−1	−1	73: top.science.computer_science.computer_aided
[o][o]	2	0	0	74: top.science.computer_science.computer_aided.design
[−][−][−][+]	4	−2	−2	76: top.science.computer_science.computer_aided.learning
[−][−][−][o][o]	5	−3	−3	77: top.science.computer_science.computer_aided.learning.language
[o]	1	0	0	75: top.science.computer_science.computer_aided.manufacturing
[o][o][+]	3	1	1	88: top.science.computer_science.database_systems
[o][o][o][+][+]	5	2	2	118: top.science.computer_science.hci
[−][−][o]	3	−2	−2	66: top.science.computer_science.information_retrieval
[o][o][+]	3	1	1	102: top.science.computer_science.operating_systems
[−]	1	−1	−1	105: top.science.computer_science.operating_systems.dos
[−][+]	2	0	0	103: top.science.computer_science.operating_systems.unix
[−][o][o][o][o]	5	−1	−1	104: top.science.computer_science.operating_systems.unix.linux
[++][++][++][++]	4	8	16	89: top.science.computer_science.programming
[++]	1	2	4	90: top.science.computer_science.programming.languages
[++][++][++]	3	6	12	92: top.science.computer_science.programming.languages.functional
[+][++][++][++][++]	5	9	17	94: top.science.computer_science.programming.languages.functional.lisp
[++][++][++]	3	6	12	95: top.science.computer_science.programming.languages.functional.ml
[++][++]	2	4	8	100: top.science.computer_science.programming.languages.oo
[++][++][++]	3	6	12	101: top.science.computer_science.programming.languages.oo.smalltalk
[++][++][++]	3	6	12	93: top.science.computer_science.programming.languages.predicative
[+][+][++][++][++][++][++][++]	8	14	26	91: top.science.computer_science.programming.languages.procedural
[+][+][++]	3	4	6	96: top.science.computer_science.programming.languages.procedural.c
	0	0	0	97: top.science.computer_science.programming.languages.procedural.cpp
[+][++][++]	3	5	9	98: top.science.computer_science.programming.languages.procedural.perl
[+][+][+][++]	4	5	7	99: top.science.computer_science.programming.languages.procedural.python
[−][o]	2	−1	−1	55: top.science.computer_science.theoretical_cs
[−−][−][+]	3	−2	−4	67: top.science.linguistics
[−−][−][o]	3	−3	−5	68: top.science.linguistics.computational_linguistics
[−−][−]	2	−3	−5	69: top.science.linguistics.computational_linguistics.parsing
[−−][−−]	2	−4	−8	72: top.science.linguistics.computational_linguistics.pragmatics
[−−][−][−][−]	4	−5	−7	71: top.science.linguistics.computational_linguistics.semantics
[−−][−−][o][+]	4	−3	−7	70: top.science.linguistics.computational_linguistics.syntax
[−−][−−][−][−][−]	5	−7	−11	116: top.science.linguistics.morphology
[−−]	1	−2	−4	117: top.science.linguistics.phonology
[−−][−−][−−][−−][−−][−][o]	7	−11	−21	115: top.science.linguistics.psycholinguistics

Figure A.1: Interest of user 88

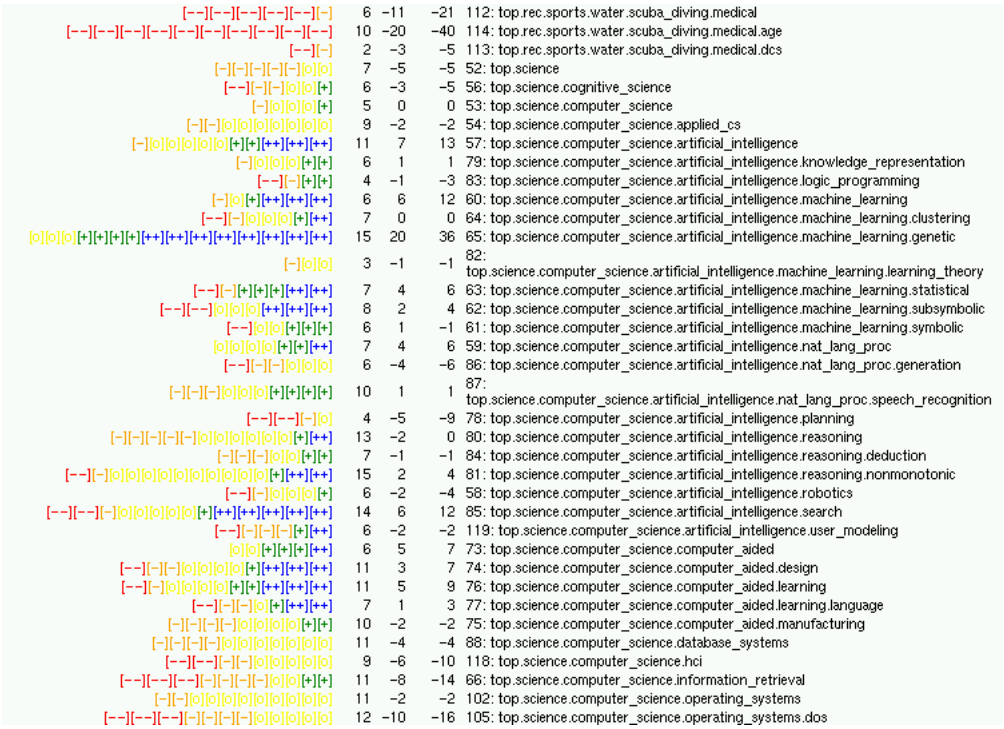
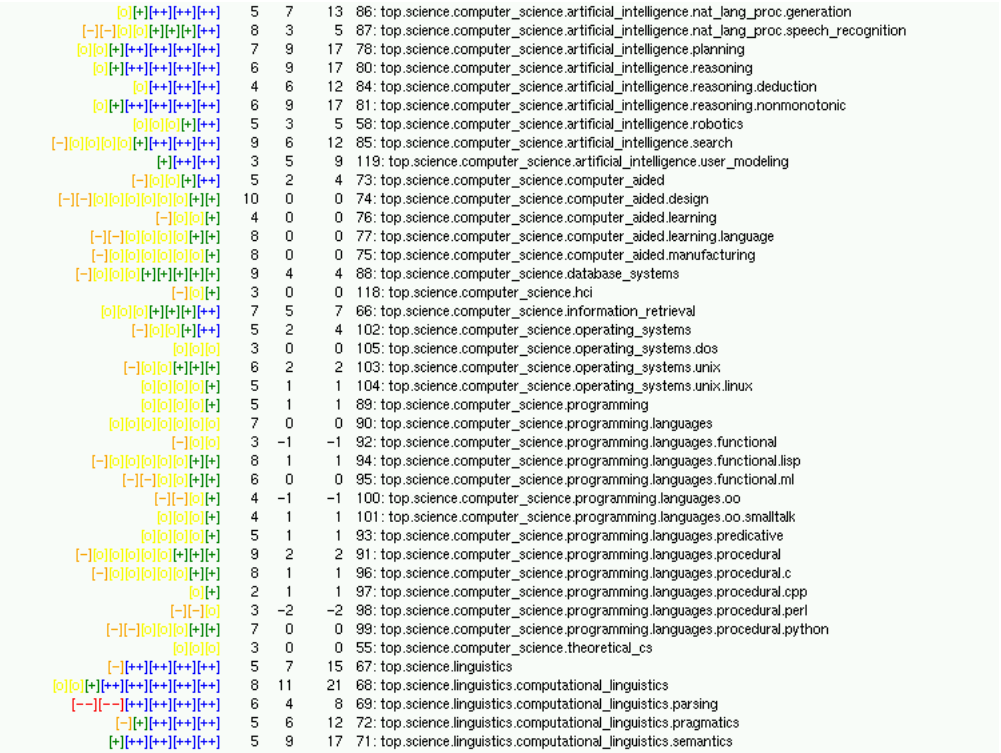


Figure A.2: Interest of user 90



[o][+][++][++][++]	5	7	13	86: top.science.computer_science.artificial_intelligence.nat_lang_proc.generation
[−][−][o][o][+][+][+][++]	8	3	5	87: top.science.computer_science.artificial_intelligence.nat_lang_proc.speech_recognition
[o][o][+][++][++][++][++]	7	9	17	78: top.science.computer_science.artificial_intelligence.planning
[o][+][++][++][++][++]	6	9	17	80: top.science.computer_science.artificial_intelligence.reasoning
[o][++][++][++]	4	6	12	84: top.science.computer_science.artificial_intelligence.reasoning.deduction
[o][+][++][++][++][++]	6	9	17	81: top.science.computer_science.artificial_intelligence.reasoning.nonmonotonic
[o][o][o][+][++]	5	3	5	58: top.science.computer_science.artificial_intelligence.robotics
[−][o][o][o][o][+][++][++][++]	9	6	12	85: top.science.computer_science.artificial_intelligence.search
[+][++][++]	3	5	9	119: top.science.computer_science.artificial_intelligence.user_modeling
[−][o][o][+][++]	5	2	4	73: top.science.computer_science.computer_aided
[−][−][o][o][o][o][o][o][+][+]	10	0	0	74: top.science.computer_science.computer_aided.design
[−][o][o][+]	4	0	0	76: top.science.computer_science.computer_aided.learning
[−][−][o][o][o][o][+][+]	8	0	0	77: top.science.computer_science.computer_aided.learning.language
[−][o][o][o][o][o][o][+]	8	0	0	75: top.science.computer_science.computer_aided.manufacturing
[−][o][o][o][+][+][+][+][+]	9	4	4	88: top.science.computer_science.database_systems
[−][o][+]	3	0	0	118: top.science.computer_science.hci
[o][o][o][+][+][+][++]	7	5	7	66: top.science.computer_science.information_retrieval
[−][o][o][+][++]	5	2	4	102: top.science.computer_science.operating_systems
[o][o][o]	3	0	0	105: top.science.computer_science.operating_systems.dos
[−][o][o][+][+][+]	6	2	2	103: top.science.computer_science.operating_systems.unix
[o][o][o][o][+]	5	1	1	104: top.science.computer_science.operating_systems.unix.linux
[o][o][o][o][+]	5	1	1	89: top.science.computer_science.programming
[o][o][o][o][o][o][o]	7	0	0	90: top.science.computer_science.programming.languages
[−][o][o]	3	−1	−1	92: top.science.computer_science.programming.languages.functional
[−][o][o][o][o][o][+][+]	8	1	1	94: top.science.computer_science.programming.languages.functional.lisp
[−][−][o][o][+][+]	6	0	0	95: top.science.computer_science.programming.languages.functional.ml
[−][−][o][+]	4	−1	−1	100: top.science.computer_science.programming.languages.oo
[o][o][o][+]	4	1	1	101: top.science.computer_science.programming.languages.oo.smalltalk
[o][o][o][o][+]	5	1	1	93: top.science.computer_science.programming.languages.predicative
[−][o][o][o][o][o][+][+][+]	9	2	2	91: top.science.computer_science.programming.languages.procedural
[−][o][o][o][o][o][o][+][+]	8	1	1	96: top.science.computer_science.programming.languages.procedural.c
[o][+]	2	1	1	97: top.science.computer_science.programming.languages.procedural.cpp
[−][−][o]	3	−2	−2	98: top.science.computer_science.programming.languages.procedural.perl
[−][−][o][o][o][+][+]	7	0	0	99: top.science.computer_science.programming.languages.procedural.python
[o][o][o]	3	0	0	55: top.science.computer_science.theoretical_cs
[−][++][++][++][++]	5	7	15	67: top.science.linguistics
[o][o][+][++][++][++][++][++]	8	11	21	68: top.science.linguistics.computational_linguistics
[−−][−−][++][++][++][++]	6	4	8	69: top.science.linguistics.computational_linguistics.parsing
[−][+][++][++][++]	5	6	12	72: top.science.linguistics.computational_linguistics.pragmatics
[+][++][++][++][++]	5	9	17	71: top.science.linguistics.computational_linguistics.semantics

Figure A.3: Interest of user 92

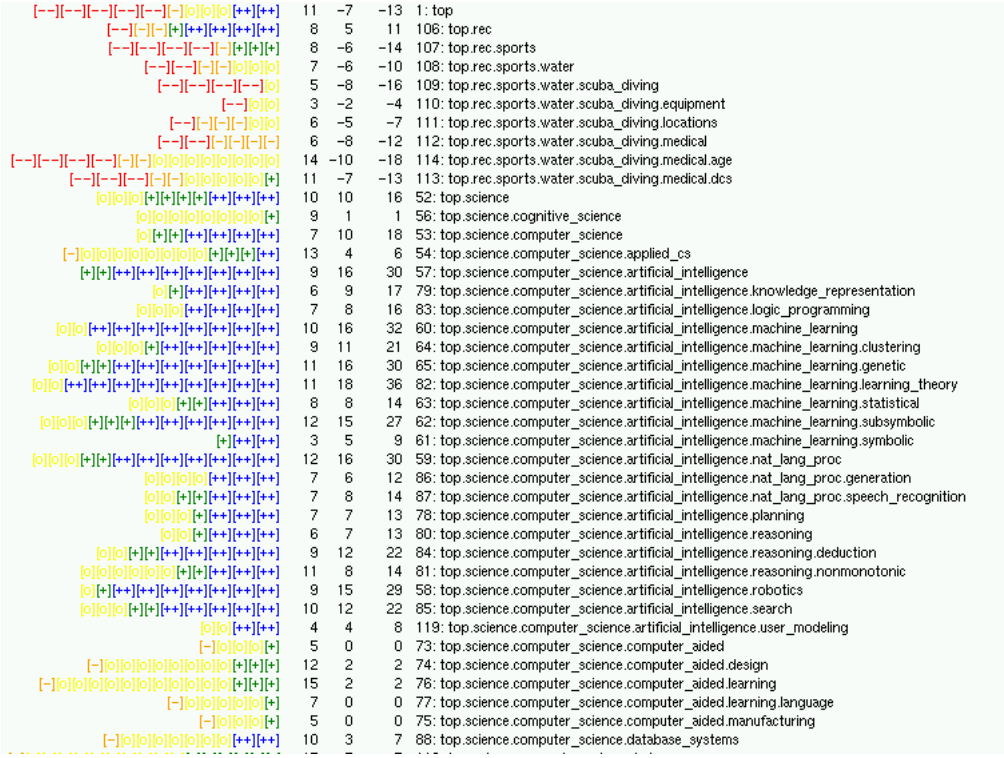


Figure A.4: Interest of user 93

A.2 Detailed evaluation

A more detailed evaluation was carried out in two steps: For a set of pre-classified URLs, we simulated user feedback. The definitions of user interests were defined by single aspect user models; complex interests were simulated by merging several aspects. The optimistic evaluation is carried out on noise–free data while the pessimistic evaluation is performed on inherently noisy classification data plus a mis-classification noise with probability of 10%.
The data for the optimistic evaluation is available in the directories `111`–`444`, the pessimistic evaluation was carried out on data as provided in the directories `666`–`888`.

A.2.1 Additional data for pessimistic evaluation

The following tables lists the aspect id's for which compressing rules have been generated during the course of our pessimistic evaluation. The data shown in table A.1 was used for the statistics in table 6.4 on page 135 (upper part). The data shown in table A.2 was

	Sample size							
No. of Rules	25		50		75		100	
per aspect	p	n	p	n	p	n	p	n
1	14, **22**, 37, 38		08, 14, 20, **22**, 24, 38, 41, 44, 48	19, 40	08, 24, 38, 41, 43, 44, 45, 48	05, 13, 15, 19, 28, 29, 34, 40, 44, 48, 49, 50	08, 09, **22**, 38, 41, 43, 44, 48	03, 05, 08, 11, 15, 19, 28, 33, 44, 46, 48, 49, 50
Num	4	0	9	2	8	12	8	13
Sum		4		11		20		21
2			37, 43		**22**, 37	24, 35	37	13, 17, 24, 29, 34, 39, 40
Num	0	0	2	0	2	2	1	7
Sum		0		2		4		8
3								35
Num	0	0	0	0	0	0	0	1
Sum		0		0		0		1

Table A.1: Evaluation (I): Induced rules

used for the statistics in table 6.4 on page 135 (lower part).

	Sample size			
No. of Rules per aspect	25	50	75	100
1	14, **22**, 37, 38	08, 14, 20, **22**, 24, 38, 41, 44, 48	08, 24, 38, 41, 43, 44, 45, 48	08, 09, **22**, 38, 41, 43, 44, 48
Num	4	9	8	8
2		37, 43	**22**, 37	37
Num		2	2	1

Table A.2: Evaluation(II): Induced rules

A.2.2 Accuracy of induced interest aspect descriptions

All figures containing an evaluation based on accuracy of induced models are based on data derived from running several batch jobs.

Generating data. The program `bin/mk_testdata` is used to generate data as described in section 6.1.1.2. This includes generation of preclassified URLs, user data, interest aspect id's and feedback. It is invoked with one single argument which is one out of `U`, `R`, `A` or `F` in order to generate **u**sers, U**r**Ls, **a**spects or **f**eedback, respectively. The range of URL, user and aspect id's is defined within the code; furthermore, the Perl program also includes several variants of feedback generation functions. The feedback variants determined the feedback behavior of the user thus specifying more specific or rather blurred pictures of the individual interests.
Note, that every user model consists of one single aspect only; complex models are simulated by composing several single aspect user models. Furthermore, the number of feedback values is constant; different feedback behavior is simulated by the sample generation process (see below). All data is written directly to the respective databases. With all data generated, `bin/mk_sample` generates sample files which can be used as input files for the Progol ILP system. `bin/mk_sample` takes a number of arguments, which are:

1. The user id of the user for whom a sample shall be generated
2. The aspect id
3. The sample length (to simulate different numbers of feedback values submitted by the user)

4. A mode description:

   (a) At least one out of {P, N}. Defines whether p_interest (P) and/or n_interest (N) shall be learned.

   (b) One out of {w, s} defines **w**eak or **s**trong (i.e. steep) interest functions. For w, a feedback value > 0 is interpreted as positive feedback; for s, the feedback value has to be > 1 (and the dual case for negative feedback).

   (c) Optionally a flag n which determines, whether negative examples (i.e. :-p_interest or :-n_interest) shall be included in the sample.

5. A posonly flag. Has to be 1 if the learner shall induce models from positive data only; 0 otherwise.

In order to test hypotheses that were generated using samples generated by mk_sample, we need test files which include validation data. Those files are generated by bin/mk_valdata. This program is very similar to mk_sample but delivers randomly chosen validation data.
Since we would have to run mk_sample and mk_test for every aspect and sample size (50×4 plus special cases;i.e. posonly or different targets) by hand, this procedure was automated by utilities which generate batch shell scripts. bin/util/mk_mksamplebatch generates a batch bin/util/samplebatch and mk_mkvaldata generates a batch for generating test files and Progol command files. All sample data are stored in the directory evaluation/samples.
Samples that were used for the first evaluation (see section 6.2) are located in the subdirectory 666.

Available datasets: Optimistic Evaluation. The optimistic evaluation has been carried out on four simulated users with fifty interest aspects each. The difference between users 111—444 was their specifity of interest (see section 6.1.1.2). Each directory contains test samples (v*XXX*_*A*) for determining accuracy of induced user models (and which were disjoint from the learning data). The user id (as already determined by the directory name) is *XXX*, *A* is the aspect identifier. The data is then further discriminated with respect to the learning target which could either be both M^+_{XXX} and M^-_{XXX} (these are the NP-s0 directories) or M^+_{XXX} only with (P-s0) or without (P-s0) negative feedback available. The next directory level distinguishes between the input sample size (5, 10, 25, 50 and 75). The files are named using the following naming convention:

- c*M*_*A*.pl
  Command batch files which are executed by PROGOL. *M* determines the use of Γ samples (g*–files) that are consulted.

- `uXXX_A.pl`
  Initial user feedback files. These files contain all background knowledge, the target and bias declarations and the initial input sample

- `gM_XXX_A.pl`
  Γ–samples include additional information for learning. The values for *M* correspond to the definition of Γ_i as given in figure 5.3. For *M*=5, both Γ_1 and Γ_4 have been applied.

Results of learning are stored in the files `c*evlg.pl`. The statistical data was extracted by `grep`ping accuracy values from the files and visualizing them using `gnuplot`.

Available datasets: Pessimistic Evaluation. Input files used for the evaluation described in section 6.4 are stored in the subdirectory `777`. The naming convention for those files is as follows:

`u777_a_`			user 777, aspect `a`
	`25_`		sample length: 25
	`50_`		sample length: 50
	`75_`		sample length: 70
	`100_`		sample length: 100
		`NPns0.pl` [`.o`]	steep interest, negative examples, two targets
		`Psn0.pl` [`.o`]	steep interest, negative examples, `p_interest` target only
		`Ps1.pl`	steep interest, no negative examples, `p_interest` target only, `posonly` flag
		`Psn1.pl` `.o`	output files for `Ps1.pl`

Evaluation data (test files as generated by `bin/mk_valdata` in the `bin/util/mktestbatch` batch) are named as follows:

`u777_a_`		user 777, aspect `a`
	`10_`	sample length: 10
	`25_`	sample length: 25
		`P+` positive examples for `p_interest`
		`N+` positive examples for `n_interest`
		`P-` positive/negative examples for `p_interest`
		`N-` positive/negative examples for `p_interest`

Finally, the evaluation is triggered by a batch file `/evaluation/samples/777/evalbatch` which is generated by the `bin/util/mk_eval` script. It requires two arguments specifying the target (`p` or `n`) to be evaluated and the sample mode to be used for learning

(see naming conventions). Running the `evalbatch` triggers a multitude of Progol runs which are all logged in files with a postfix `evlg`. The naming convention is as follows: `u777_a_(`*`ss`*`-`*`ts`*`)`*`Flg`*`_c.pl.evlg` where *`ss`* is the sample size of the input file used, *`ts`* is the size of the test file and *`Flg`* is a flag (`+` or `-`) indicating whether negative examples for the target concept were contained in the test file (`a` is an aspect id). Single aspect learning log files for the target `p_interest` were stored in the subdirectory `evlg-1`; `evlg-2` contains the files for the target `n_interest.` Both subdirectories contain:

1. files named `u777_a_(`*`ss`*`-`*`ts`*`)`*`Flg`*`_c.pl.evlg` (naming convention see above). There are 800 such files in each directory:
   (a) 50 aspects `a` ($[1, 50]$)
   (b) 4 sample sizes *`ss`* ($\{25, 50, 75, 100\}$)
   (c) 2 test file sizes ($\{10, 25\}$) and
   (d) 2 negative evidence flags ($\{+, -\}$)
2. the file `SUMMARY`, which only contains contingency tables from all 800 logs
3. the file `SSUMMARY`, which only contains the accuracy data
4. 16 files `Summary.`*`ss`*`.`*`ts`*`.`*`flg`*`.s` with
   (a) 4 sample sizes *`ss`* ($\{25, 50, 75, 100\}$)
   (b) 2 test file sizes ($\{10, 25\}$) and
   (c) 2 negative evidence flags ($\{+, -\}$)

   which contain only appropriate lines from `SSUMMARY`
5. the file `SUMSUMSummary` which contains average values computed by the script `ev.pl`. Data is divided into blocks that are `index`ed as input files for `gnuplot` (see figure 6.6 on page 136).

Appendix B

IMPLEMENTATIONAL ISSUES

B.1 Concept hierarchies

Conceptual user models are based upon concept hierarchies. The concepts defined in such hierarchies are the syntactic elements by which user models are represented. Within OySTER we used two different concept hierarchies. $\mathcal{T}$ describes the set of document types, while $\mathcal{C}$ is used to represent a document's content (also called category). Both hierarchies are stored in the local OySTER URL database.

B.1.1 Document types

The document type hierarchy contains 35 classes:

top
top.publication
top.publication.dissertation
top.publication.lecture
top.publication.lecture.notes
top.publication.lecture.resource
top.publication.lecture.syllabu
top.publication.manual
top.publication.manual.document
top.publication.manual.online
top.publication.publishedbook
top.publication.publishedbook.abstract
top.publication.researchpaper
top.publication.researchpaper.abstract
top.publication.researchpaper.full
top.publication.researchpaper.full.published
top.publication.researchpaper.full.unpublished
top.publication.selfdescription
top.reference
top.reference.linklist
top.reference.otherlist
top.reference.peoplelist
top.reference.publicationlist
top.unknown
top.virtual
top.virtual.conference
top.virtual.group
top.virtual.group.university
top.virtual.group.university.facdep
top.virtual.group.university.group
top.virtual.group.university.project
top.virtual.individual
top.virtual.individual.employee
top.virtual.individual.researcher
top.virtual.individual.student

The document types are stored in the `url_dtype` table in the `oyster_url` database on the `URLDB`–server. For more interactive browsing an editing tool is provided by the `typed` Cgi script which is located in the `cgi` subdirectory of OySTER. It can be invoked by requesting the Url `http://localhost/ocgi/typed`.[1]

B.1.2 Document categories

The document content class hierarchy contains 69 classes. On the top level it is divided into two sub-hierarchies: `science` and `rec`. The former contains two further general topics, namely `computer_science` and `linguistics` while the latter specializes to `scuba_diving`.

```
top.rec
top.rec.sports
top.rec.sports.water
top.rec.sports.water.scuba_diving
top.rec.sports.water.scuba_diving.equipment
top.rec.sports.water.scuba_diving.locations
top.rec.sports.water.scuba_diving.medical
top.rec.sports.water.scuba_diving.medical.age
top.rec.sports.water.scuba_diving.medical.dcs
```

This part of the ontology was created to examine the example of resolving the ambiguous search query for `decompression`. Actually, classifier accuracy was much higher than expected (though not evaluated against a manually pre-classified set). This is due to the set of relevant phrases which was generated using the bootstrapping method described in section 53. The initial queries used were (in order of the above listed categories):

```
recreational
recreational, sports
recreational, sports, water
recreational, sports, scuba-diving
scuba-diving, equipment, jacket, bcd, regulator, fin
divespot
scuba-diving, medical
"arterial gas embolism", AGE
"decompression sickness", caisson, DCS
```

The `science` branch of the ontology is divided into three sub-categories (in addition to the two further elaborated categories mentioned above, there is terminal node `cognitive_science`). The `linguistics` branch is rather shallow:

```
top.science.linguistics
top.science.linguistics.computational_linguistics
top.science.linguistics.computational_linguistics.parsing
top.science.linguistics.computational_linguistics.pragmatics
top.science.linguistics.computational_linguistics.semantics
top.science.linguistics.computational_linguistics.syntax
top.science.linguistics.morphology
top.science.linguistics.phonology
top.science.linguistics.psycholinguistics
```

Within `computer_science`, the hierarchy mainly distinguishes between `artificial_intelligence`, `programming` and `computer_aided`:

[1] In case of a distributed installation of the OySTER system, the `localhost` should be replaced by the proper name of the Cgi server.

top.science
top.science.cognitive_science
top.science.computer_science
top.science.computer_science.applied_cs
top.science.computer_science.artificial_intelligence
top.science.computer_science.artificial_intelligence.knowledge_representation
top.science.computer_science.artificial_intelligence.logic_programming
top.science.computer_science.artificial_intelligence.machine_learning
top.science.computer_science.artificial_intelligence.machine_learning.clustering
top.science.computer_science.artificial_intelligence.machine_learning.genetic
top.science.computer_science.artificial_intelligence.machine_learning.learning_theory
top.science.computer_science.artificial_intelligence.machine_learning.statistical
top.science.computer_science.artificial_intelligence.machine_learning.subsymbolic
top.science.computer_science.artificial_intelligence.machine_learning.symbolic
top.science.computer_science.artificial_intelligence.nat_lang_proc
top.science.computer_science.artificial_intelligence.nat_lang_proc.generation
top.science.computer_science.artificial_intelligence.nat_lang_proc.speech_recognition
top.science.computer_science.artificial_intelligence.planning
top.science.computer_science.artificial_intelligence.reasoning
top.science.computer_science.artificial_intelligence.reasoning.deduction
top.science.computer_science.artificial_intelligence.reasoning.nonmonotonic
top.science.computer_science.artificial_intelligence.robotics
top.science.computer_science.artificial_intelligence.search
top.science.computer_science.artificial_intelligence.user_modeling
top.science.computer_science.computer_aided
top.science.computer_science.computer_aided.design
top.science.computer_science.computer_aided.learning
top.science.computer_science.computer_aided.learning.language
top.science.computer_science.computer_aided.manufacturing
top.science.computer_science.database_systems
top.science.computer_science.hci
top.science.computer_science.information_retrieval
top.science.computer_science.operating_systems
top.science.computer_science.operating_systems.dos
top.science.computer_science.operating_systems.unix
top.science.computer_science.operating_systems.unix.linux
top.science.computer_science.programming
top.science.computer_science.programming.languages
top.science.computer_science.programming.languages.functional
top.science.computer_science.programming.languages.functional.lisp
top.science.computer_science.programming.languages.functional.ml
top.science.computer_science.programming.languages.oo
top.science.computer_science.programming.languages.oo.smalltalk
top.science.computer_science.programming.languages.predicative
top.science.computer_science.programming.languages.procedural
top.science.computer_science.programming.languages.procedural.c
top.science.computer_science.programming.languages.procedural.cpp
top.science.computer_science.programming.languages.procedural.perl
top.science.computer_science.programming.languages.procedural.python
top.science.computer_science.theoretical_cs

The document categories are stored in the `url_dcat` table in the `oyster_url` database on the `URLDB`–server. A more interactive browsing editing tool is provided by the `onted` Cgi script which is located in the `cgi` subdirectory of OySTER. It can be invoked by requesting the Url `http://localhost/ocgi/onted`.[2]

B.1.3 Browsing and Editing

As already mentioned, OySTER includes a Cgi script which allows for browsing and editing of the concept hierarchies more comfortably. A sample screenshot of the document type category is shown in figure B.1 on the following page. New categories can be added by a simple point and click action on the superconcept of the newly introduced concept. Subsumption relations are computed by the Cgi script such that the representation of the categories in the `oyster_url` database always remain consistent. The data structures used to store the hierarchies in the database are explained in the database documentation on the compact disk.

[2]See footnote 1 on the preceding page.

Figure B.1: The OySTER ontology browser/editor

B.2 The OySTER multi agent system

OySTER is realized as a multi agent system for different reasons. First, computing power required is too great to be carried out on a single machine in a tolerable amount of time. When receiving a search request, the system load quickly reaches a peak value of 8 on a dual Pentium III machine. Though the average processor load is rather low it is necessary to provide high peak performance resources. The second reason is that a multi agent system (once designed well enough) is robust: On the one hand it is fault–tolerant in the sense, that missing components do not hinder the other agents. On the other hand, this allows for easy enhancement of the system since single agents, or sets of agents, can be changed during runtime. Finally, such an agent system with families of agents which are specialized on different tasks is scalable in a rather primitive way: If more performance in a certain subtask is needed, we are able to invoke several independent instances of the same agent.

For these reasons, the architecture of a multi agent system was chosen during the design of OySTER. The agent families are roughly divided into interface agents and classifier agents; the user modeling process is currently carried out offline but would simply establish a new agent class. Communication between agents is carried out using a blackboard, which is maintained by a special blackboard server agent. Thus, any communication between agents is piped through the blackboard using a special protocol. Messages written on the blackboard may be encoded arbitrarily which enables us to incorporate arbitrary agents as well.

The architecture of the underlying blackboard centered multi agent system is sketched in figure B.2 on the next page. Read access could be made much faster if we allow agents to read the blackboard file themselves. But this would also imply several severe disadvantages:

1. Security issues

2. Distributed agents would need an NFS accessible blackboard file (thus again security)

3. The blackboard syntax should be encapsulated. With a new server agent, a new blackboard file syntax would force the re-implementation of all agent's read access procedures.

Thus, through the price of a large protocol overhead, we avoid the above mentioned disadvantages and obtain a highly open and flexible multi agent environment.

The blackboard protocol is described in the next section.

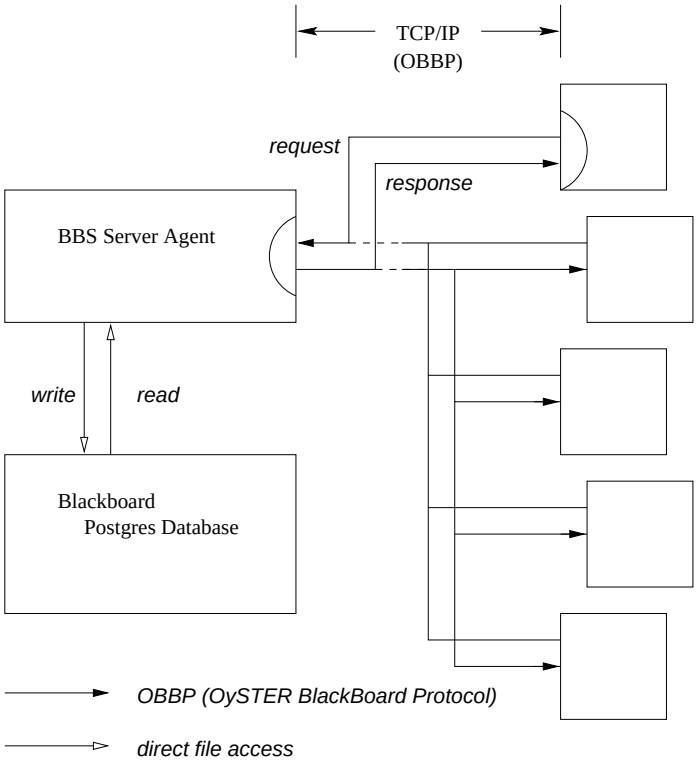


Figure B.2: A brief sketch of the OySTER BBS

B.2.1 The OySTER blackboard protocol

Any request, response or daemon activity is co-ordinated through the blackboard. For example, instead of a linear process for a CGI triggered meta search the process of search is decomposed and encapsulated into several independent tasks in the multi agent environment:

1. a CGI agent receives a search request from a client.
2. the search request is sent to the BBS and a response `id` is returned.
3. the search request is processed based upon the blackboard entries by appropriate agents
4. Upon a further request by some other agent referring to the response `id`, an answer is sent back to the agent which is then read by a client.

The OBBP generally knows three distinct modes of communication:

1. A client agent sends a *request* to the BBS server;

2. The BBS server *acknowledges* a client request

3. The BBS server *responds* to a client agent.

The OBBP is realized by using TCP/IP sockets; server and port are specified in the configuration file. Currently, the protocol data is not encrypted; for future releases this should be considered for privacy reasons in the user modeling process. The plain text protocol is described in the following sections. In general, the string beginning with the first non-space character after a key until the next EOL followed by the `::keyword` is interpreted as key value (i.e. multiple arguments can be passed by using control characters, whitespaces or new line characters).

B.2.1.1 A Grammar for the OBBP

Simple juxtaposition including spaces means concatenation. The symbol '_ ' means a sequence of at least one whitespace character.

⟨*OBBP Item*⟩ ::= ⟨*OBBP Start*⟩ ⟨*OBBP Cont*⟩ ⟨*OBBP End* ⟩
⟨*OBBP Start*⟩ ::= `::oyster-bbs-p.01` _ ⟨*MessageType*⟩
⟨*OBBP Cont*⟩ ::= ⟨*::Agent*⟩ ⟨*::ReqType*⟩ ⟨*::ReqCmd*⟩ ⟨*::ReqArg*⟩ ⟨*::ReqPar*⟩ ⟨*::Att*⟩ ⟨*::Addinfo*⟩
⟨*OBBP End* ⟩ ::= `\n ::oyster-bbs-p.0.1 _ end`

⟨*MessageType*⟩ ::= (`request` ‖ `response` ‖ `ack`)

⟨*::Agent*⟩ ::= `\n ::agent` _ ⟨*AgentClass*⟩ `.` ⟨*AgentName*⟩
⟨*::ReqType*⟩ ::= `\n ::reqtype` _ (`r` ‖ `w` ‖ `a` ‖ `c` ‖ `d` ‖ `x` ‖ `m`)
⟨*::ReqCmd*⟩ ::= `\n ::reqcmd` _ ⟨*BBReqType*⟩ `:` ⟨*AgentClass*⟩ `.` ⟨*AgentCommand*⟩
⟨*::ReqArg*⟩ ::= `\n ::reqarg` _ ⟨*$text$*⟩ (see Agent descriptions)
⟨*::ReqPar*⟩ ::= `\n ::reqpar` _ ⟨*$text$*⟩ (see Agent descriptions)
⟨*::Att*⟩ ::= `\n ::att` _ (`asap` ‖ `now` ‖ `-` ‖ (`+` ⟨*int*⟩(`M` ‖ `H`)))
⟨*::Addinfo*⟩ ::= `\n ::addinfo_` ⟨*$text$*⟩ (see Agent descriptions)

⟨*AgentClass*⟩ ::= ⟨*$string$*⟩ (see Agent descriptions)
⟨*AgentName*⟩ ::= ⟨*$string$*⟩ (see Agent descriptions)
⟨*AgentCommand*⟩ ::= ⟨*$string$*⟩ (see Agent descriptions)

⟨*BBReqType*⟩ ::= (`r` ‖ `a` ‖ `d`)

⟨*int*⟩ ::= $\{0,1,2,3,4,5,6,7,8,9\}^\star$
⟨*$string$*⟩ ::= any sequence of characters except for `\n`
⟨*$text$*⟩ ::= any sequence of characters except for `\n::`

Remark: The expression

```
\n ::oyster-bbs-p.0.1 _ end
```

matches the following character sequences:

- `\n::oyster-bbs-p.0.1␣end`
- `\n::oyster-bbs-p.0.1␣␣␣␣end`
- `\n::oyster-bbs-p.0.1␣\t␣␣end`

but *not* any of the following:

- `\n␣::oyster-bbs-p.0.1␣end`
- `\n::oyster-bbs-p.0.1␣\n␣␣end`

B.2.1.2 Requests and Responses

Any request sent to the BBS Server is treated as a command or goal that has to be executed. The request is parsed for correctness and (if security allows) added to the blackboard. There it waits for an agent to come and execute the task. Upon termination, the agent sends a signal which causes the server agent to delete the request from the blackboard.
The OBBP allows for two major classes of messages:

1. *Requests* and *Responses* are for information transfer and use mainly the same message format
2. *Acknowledgments* from the server are short messages that are used for receipt notification and transmission error checks.

We first describe the main common protocol for both queries (i.e. `request`s) and `response`s.

Example: A Query for database information about an URL. The request displayed in figure B.3 on the facing page is a query for database information about an URL. In general this request asks for the `title` entry in the `url_url` table, where the URL is `http://mir.cl-ki.uni-osnabrueck.de/~martin/oyster.html`.

Example: A Response with database information about an URL. In figure B.4 on the next page you find a query/response pair for an answer read request and the delivered answer. It corresponds to the very last two interactions in figure B.6 on page 216. An agent which sends a request and waits for an acknowledgment can be implemented using Perl and `OySTER.pl` very easily. An example is shown in figure B.5 on page 196.
In figure B.6 on page 216 we have illustrated the communication flow for a CGI triggered database lookup for an URL title. As one can easily extract from the picture, it is a massive communication overkill for passing the single information that the ***title*** for the URL ***XYZ*** is ***foo-bar***. But, on the other hand, the diagram also shows the advantage of the multi agent architecture.

```
1   ::oyster-bbs-p.0.1   request              # this is a request
2   ::agent              cgi.url-info         # from a cgi agent named "url-info"
3   ::reqtype            w                    # write to the blackboard the following ...
4   ::reqcmd             r:dbi.url-url.read   # "a read request for dbi agents
5   ::reqarg             full_url=http://mir.cl-ki.uni-osnabrueck.de/~martin/oyster.html
6   ::reqpar             title                # get the title for that url
7   ::att                -                    # not time critical
8   ::oyster-bbs-p.0.1   end                  # Good bye.
```

(Line counters and comments are *not* part of the OBBP!)

Figure B.3: OBBP Request

```
1   ::oyster-bbs-p.0.1   request              # this is a request
2   ::agent              cgi.url-info         # from a cgi agent named "url-info"
3   ::reqtype            r                    # read from the blackboard ...
4   ::reqcmd             a:dbi.url-url.read   # an answer about
5   ::reqarg             full_url=http://mir.cl-ki.uni-osnabrueck.de/~martin/oyster.html
6   ::reqpar             a.123                # with that id
7   ::att                asap                 # now
8   ::oyster-bbs-p.0.1   end                  # Good bye.
```

The according response is:

```
1   ::oyster-bbs-p.0.1   response             # this is a response ...
2   ::agent              bbs                  # from the bb server agent ...
3   ::reqtype            r                    # which read information follows:
4   ::reqcmd             a:dbi.url-url.read   # "an answer about ...
5   ::reqarg             full_url=http://mir.cl-ki.uni-osnabrueck.de/~martin/oyster.html
6   ::reqpar             a.123::title=foo-bar # ... with that id" delivered "title=foo-bar"
7   ::att                -                    #
8   ::oyster-bbs-p.0.1   end                  # Good bye.
```

(Line counters and comments are *not* part of the OBBP!)

Figure B.4: AOBBP Request/Response pair

```
#!/usr/bin/perl
# ============================================================================
# Blackboard test client
# ----------------------------------------------------------------------------
# Author  : Martin E. Mueller
# ============================================================================

require 5.002;
require "flush.pl";
require "OySTER.pl";
use Socket;

&__oy_initvars() or
&__oy_decease(1, "Variable intialisation failed") ;

&__oy_bbsconnect("SOCK") or
&__oy_decease(2, "BB Conn couldn´t be established") ;

%msg  = (agent   => "cgi",
         reqtype => "a",
         reqcmd  => "r:dbi.url_url.read",
         reqarg  => "url=http://mir.cl-ki.uni-osnabrueck.de",
         reqpar  => "title",
         att     => "now" );

&__oy_bbssend( "request",                  # send a request
               "SOCK",                     # to stream SOCK
               $msg                        # with this content
             ) or
         &__oy_decease(3, "BBS send failed") ;

&__oy_bbsdisconnect("SOCK");

exit;
```

Figure B.5: A OBBP Agent

Actually, any pair of boxes representing an agent that sends and receives can be instances of *different* agents. Or, they can be different processes or threads. Thus, many agents from nearly anywhere can access the blackboard at "any" time. For example, after the answer has been written to the blackboard, *many* agents may read it—until it gets deleted either by a explicit delete command or by a garbage collector that clears the blackboard from already processed commands.
In detail, the communication runs as follows:

1. A user asks for a title for an URL; either explicitly or implicitly during the course of a search request. The request is caught by a CGI agent which then starts the whole communication process.

2. The CGI agent sends a `request` to the blackboard server in order to force a `Write` on the blackboard which represents a *Req*uest for **title** information about url=XYZ. The addressed agent class is dbi; the information source is the url_url table, and the action to perform is a read.

3. The BBSERVer agent writes the *Req* to the blackboard and implicitly `Mark`s it as **todo**. This request is assigned a unique blackboard id r:123.

4. The BBSERVer agent `ack`nowledges receipt of a `Write` *Req*uest with arguments as above and returns the id r:123.

5. Now, a DBI agent sends a `request`; namely to `Check` for *Req*uests that are waiting to be processed by an agent from the dbi class.

6. The BBSERVer finds an according entry and `ack`nowledges the request by additionally returning the id of the matching blackboard entry (r:123).

7. Using this id, a DBI agent `request`s to `Read` the *Req* with id r:123.

8. The BBSERVer's `response` includes all information that was initially sent by the first `Write` request.

9. A DBI agent uses this id to `Mark` the `Req` as being in **in_process** in order to prevent others doing the same work.

10. This is canonically `ack`nowledged by the server.

11. Now, the actual lookup takes place, where a DBI agent queries the URL database using **title** as selection criterion and **url=XVY** as restriction.

12. The outcome (**title=foo-bar**) is then sent to the blackboard as a `request` for an *Ans*wer `Write`. Simultaneously, an implicit `Mark` is carried out, which switches the state of r:123 to **ready**.

13. During the `Write`, the BbServer assigns a unique id `a:123` to this entry.
14. This id is sent back along with the `ack`nowledgment of the `Write request`.
15. Finally, a Cgi agent again `Check`s for *`Ans`*wers matching the initial *`Req`*uest
16. The server sends an `ack` including the id `a:123`,
17. this is used as a last `request` for an *`Ans`*wer `Read`.
18. Finally, the `response` includes the information, that the **title** is **foo-bar**.
19. This is used by a Cgi agent to forward the desired information to the client.

B.2.2 The blackboard database

The blackboard itself is realized as the table `bb` in the database `oyster_bb` on the blackboard server. The only program with actual read/write access on this table is the blackboard server agent `bb_server` (in the directory `/oyster/bbs/bb_server_agents`). The server agent `bb_server` connects via Pg sockets on port `5432` to the `postmaster` and listens for client requests on port `4711`.
The database `oyster_bb` contains one static table `bb` which is described in the documentation on the compact disk. Specific blackboard maintenance tasks are performed by so–called daemon agents. All daemons reside in the `bbs/bb_daemons` subdirectory. Currently, these are:

1. `bbs_clean.d`
   marks expired idle entries as zombies and old zombies to be deleted.
2. `bbs_del.d`
   triggers deletion of entries.[3]

B.2.3 Miscellanea

B.2.3.1 A Perl Module for the OySTER blackboard protocol

In order to allow for a more comfortable Tcp/Ip communication using the OySTER blackboard protocol, we provide a special OySTER Perl library. It defines several primitive functions that are frequently used by Perl agents that interact with the OySTER prototype. These are:

[3]Deleting a blackboard entry means to send a request for deletion to the blackboard server. Although deletion can be triggered by any agent, it is recommended to mark entries as "to be wiped out" from the blackboard. This wipe-out is encapsulated in `bbs_del.d` which re-spawns at given time intervals.

1. `__oy_initvars`
   reads from the standard configuration file (`/usr/local/oyster/etc/oyster.conf`) all variable settings and stores them in the global hash tables.

2. `__oy_logmsg`
   logs messages. The log file location is stored in the settings as read by `__oy_initvars`. Arguments:

   (a) (int) Message Level—not used.

   (b) (string) the Message. Usually starting with `[MSG]` or `[ERR]`

3. `__oy_decease`
   throws a `__oy_logmsg` and triggers a Perl *die*. Arguments:

   (a) (string) the Message.

4. `__oy_bbsconnect`
   establishes a client connection to the blackboard server. Arguments:

   (a) (string) Socket name to be used.

5. `__oy_bbsdisconnect`
   closes client connection to the blackboard server. Arguments:

   (a) Socket/Stream name.

6. `__oy_bbssend`
   sends a message to the bb server. Arguments:

   (a) Message type. One out of {`request`,`response`,`ack`}.

   (b) Stream. Where to write to. Usually, "`SOCK`" or "`STDOUT`".

   (c) Reference on a hash of protocol contents. Hash keys are:

      i. `agent`: sender agent name
      ii. `reqtype`: {"r", "a", "d", "x", ...}
      i.e. the command the server agent performs on the blackboard.
      iii. `reqcmd`: *`R/A:AgClass.AgName.AgCmd`*
      iv. `reqarg`
      v. `reqpar`
      vi. `att` : time stamp

   As an example, see figure B.5 on page 196.

7. __oy_bbslisten
   listens for a message; usually for an ack. Arguments:
   (a) Expected Message type. One out of {request, response, ack}; most likely ack.
   (b) Stream. Where to listen. Usually, "SOCK" or "\"STDOUT"
   (c) Reference on a hash of protocol contents. As above.
8. __oy_bbsreceive
   Receives a message from the server. Arguments:
   (a) Socket name to be used.

   Returns a reference on a hash carrying the response.
9. __oy_initdbconn Initializes DB connection. Returns a Pg database handle.

B.2.3.2 Blackboard inspection interface

A WWW interface which displays the current blackboard entries and the last 30 lines of the log file also exists. This CGI script does *not* use the OBBP, but accesses the blackboard database *directly*.
It is located in the cgi subdirectory of the appropriate CGI server. Currently, it can be accessed via http://mir.cl-ki.uni-osnabrueck.de/ocgi/oydbbbi.pl.

B.2.4 Interacting with the blackboard server agent

In this section, we briefly describe the requirements of how requests to the blackboard server agent have to be formulated in order to make it perform the desired actions, namely the *request types* as specified by the ::reqtype tag of each message. In other words, the following paragraphs describe the abstract syntax of commands that can be submitted using the __oy_bbssend Perl routine described in the last section.

Writing. As the command name suggests, it is used to write messages onto the blackboard.
The acknowledgment carries the appended information of the new blackboard message id in the reqpar tag: id:*bbid*. Note, that it is *appended* to the current reqpar with a trailing blank (thus, a blank reqpar starts with a blank).

Adding. Adding something to the blackboard (a) is synonymous to writing (w).

Checking. Checking the blackboard is the most important action, since it delivers the desired blackboard message id by acknowledgment.
This id is used for reading, deleting, marking and canceling.
For any such request, a SQL command is generated. Roughly, it is:

```
SELECT * FROM bb WHERE
              type    = 'T'         AND
              address ~ 'command' [AND reqarg]
order by stamp;
```

If `reqarg` is, for example, `state = 'can'`, then it is added to the command:

```
SELECT * FROM bb WHERE
              type    = 'T'         AND
              address ~ 'command'   AND
              state   = 'can'
order by stamp;
```

Reading. After an agent has written something on the blackboard, which is addressed to another agent, the latter one needs to read the message after he has `check`ed for messages The according `read` request will not be acknowledged, but a response is sent from the server, containing all data in the appropriate protocol fields.

Deleting. Although the protocol allows for explicit deletion, deletion shall only be triggered by the blackboard garbage collector and the wiper agents (c.f. B.2.5.2 on page 203 and B.2.5.2 on page 203, respectively). Instead of explicit deletion, corresponding entries should be `mark`ed as to be deleted (see below).
The generated SQL commands are appropriate for specified id's

```
DELETE FROM bb
       WHERE id=reqarg AND state='del';
```

or (if `reqarg` is not specified):

```
DELETE FROM bb
       WHERE reqpar;
```

The acknowledgment carries the appended information of the blackboard message id that has been deleted (redundant, but why not?).

Marking. The `mark`-command is used for toggling the `state` of a blackboard entry. This is used when a `r`equest is being processed and thus changes from `idl` to `prc`, or when the garbage collector combs out zombies (`zom`).
The SQL command that is carried out is

```
UPDATE bb SET
       state = reqpar
WHERE reqarg;
```

Canceling. Canceling (`x`) is synomymous to marking (`m`) with `reqpar` being `can`.

B.2.5 Agents

B.2.5.1 Client agents

Query Processing. Meta search queries are added to the blackboard by interface agents (currently, queries by CGI only). The `bbs/bb_client_agents/ms.qry.d` agent expands any meta search request into a set of search queries which are to be submitted to the utilized search engines.
This agent's functionality allows for a more detailed query processing as, e.g. query expansion, simple meta search or user model filtered meta search. Currently, however, it only supports a simple meta search.
As soon as the request has been read from the blackboard it is being checked as being processed. Then, for each utilized search service, meta search requests are added.

Wrapping. The commands generated by `bbs/bb_client_agents/ms.qry.d` are subject to the wrapper agents `bbs/bb_client_agents/ms.wrp.de`. The `bbs/bb_client_agents/ms.wrp.de` agents (multiple instance running) read the user's search request from the blackboard and receive the search engine that has to be asked. Accordingly, search engine specific HTTP queries are generated and sent to the search services. The returned result is processed (with respect to the search engines HTML layout) and resulting URLs are written to the `oyster_url:mstmpr` cache table.

Updating the URL database. The local database `oyster_url:url_url` is updated with information collected in the `mstmpr` cache. This task is carried out periodically by the `bbs/bb_client_agents/dbi.url_url.upd` agent and thus ensures up–to–date information about URL classifications on the database. Simultaneously, the cache entry is altered such that it includes a back-reference to the newly introduced or updated entry in the local URL database.

Updating the cache `oyster_url:mstmpr`. Both `oyster_url:mstmpr` and `oyster_url:url_url` are mutually updated against each other as shown in figure 3.6 on page 79. Updating the cache is performed by the REBOL agent `bbs/bb_daemon_agents/dbi.mstmpr.upd` every minute. It simply copies the `oyster_url:url_url:id` and classification data from `oyster_url:url_url` for those entries in `mstmpr`, where `oyster_url:mstmpr:urlid` is undefined and the full URL of the entry is known in the local database.

Classifying URLs. Whenever a new URL is encountered and added to `oyster_url:mstmpr`, a request for type and category classifiers is sent to the blackboard as well. Those requests are processed by the document type classifier `bbs/bb_client_agents/db.update.d` and the document category classifier `bbs/bb_client_agents/oyster_cat.r`. Results are written back to `oyster_url:mstmpr` and data in `oyster_url:url_url` is updated by `bbs/bb_client_agents/dbi.url_url.upd`.

B.2.5.2 Daemons

As already mentioned in the last section, the blackboard server agent delegates tasks for scheduling deletion and garbage collections to daemon agents for reasons of security.

Blackboard Garbage Collector. This agent collects idle expired requests and expired zombies. Recall, that deletion of entries is not supposed to be triggered by client agent agents. Instead, they are supposed to be *mark*ed as to be deleted. The garbage collector and especially the *Blackboard Wiper* will trigger the actual deletion process.
The Blackboard Garbage Collector is realized by the `bbs/bb_daemons/bbs_clean.d` daemon. It checks the blackboard from time to time, thereby

- turning expired idle entries into zombies and
- turning expired zombies into entries that are to be deleted.

Blackboard Wiper. This agent, implemented by the `bbs/bb_daemons/bbs_del.d` daemon, is *the only component* that actually triggers a physical deletion of blackboard entries. It tells the server to delete entries (thus, the server is the only agent that actually *performs* a physical deletion). Deletion is triggered in fixed time intervals—removing all `del`–tagged entries from the blackboard database table.

Cache Garbage Collector. Search results are stored in the `oyster_url:mstmpr` table, from which result pages are generated upon client HTTP requests. Of course, the `mstmpr` table has to be cleared periodically. This task is performed by the `bbs/bb_daemons/ms.tmpdb.del` agent. Every five minutes it deletes those URLs which already have been synchronized with the local `oyster_url:url_url` database and which are in the cache

for more than 48 hours. URLs which are present for more than 96 hours are deleted regardless as to whether or not they were incorporated into the URL database[4].

B.3 Installation

OySTER is a fully functional meta web search engine which can easily be adapted to special needs. A minor disadvantage of the system are static wrappers which need to be maintained handishly in order to be able to extract results from changing search engine layouts. Currently, the system includes wrappers for eleven different search engines including Google, AltaVista and NorthernLight.
Due to its multi agent architecture, the system can easily be enhanced by special agents or agent families.

B.3.1 System requirements

OySTER currently runs on four Linux servers which share tasks for CGIs, classification, wrapping agents, blackboard management, database serving and user model induction. To install OySTER at least one machine with the following software is required:

- APACHE web server version 1.3.3 and above.
- POSTGRES version 6.4 and above.
- PERL version 5.005_02 and above; including libraries for CGIs and PG interaction.

This does not include the user model induction component, which further needs Progol, SWI–Prolog and a considerable amount of additional scripts.
All code is SMP compatible; two of the currently employed servers are Linux SMP systems. Any newer (SMP–) kernel version (but at least 2.2.14) will do; it is recommended to run OySTER only on hardware that has been proved to work flawlessly in advance. The hardware currently used includes a dual Pentium III–550 (750MB; blackboard, classification, learning), a dual Pentium II–400 (750MB; database, wrapping), a Pentium II–400 (128MB; classification, watchdog) and a Pentium I–300 (128MB; CGI).

B.3.2 System preparation

The whole system needs to be installed in the directory `/usr/local/oyster/` (contents are as directory names indicate):

[4]This is mainly for non–existing URLs which have been reported by utilized search engines due to their outdated indices.

```
martin@soyuz:/usr/local/oyster > ls -al
total 22
drwxrwxr-x   8 martin   oyadm        1024 May 18 11:33 ./
drwxr-xr-x  13 root     root         1024 May 18 11:35 ../
drwxrwxr--   5 martin   oyadm        1024 May 18 17:30 bbs/
drwxr-xr-x   4 wwwrun   oyadm        1024 Nov 19  1999 cgi/
drwxrwxr--   5 martin   oyadm        1024 May 18 18:07 db/
drwxrwxr--   2 martin   oyadm        1024 Apr  6 15:19 etc/
drwxrwxr-x   2 martin   oyadm       14336 May 19 09:00 log/
drwxr-xr-x   2 martin   users        1024 Oct  8  1999 src/
```

If you are running dedicated servers for different tasks, only according directories need to be installed on those servers. Distributed installation includes three tripwires:

1. The configuration file `/usr/local/oyster/etc/oyster.conf` needs to be adjusted on every server.

2. The log directory `/usr/local/oyster/log/` should be cross–mounted by NFS to all involved machines.

3. The Perl module located in `src` should be *copied* to each engine's Perl directory.

Since log files grow rapidly, it is recommended to link the `log` directory to an external volume. In the current prototype version, it is linked to `/var/log/oyster/`.

B.3.3 Configuration file

The configuration file must be installed in the same directory on any machine `/usr/local/oyster/etc/oyster.conf`. It consists of several sections describing the location of OySTER subservices. It also includes a list of processes/agents that are to be started on the according machine. This list us used by the `/oyster/bbs/mk_bbs` script which generates start and kill scripts for each machine.[5]

```
#------------------------------------------------------------------------
# OySTER configuration file.
# --------------------------
#
# This file holds several OySTER-wide used variables.
# It is read by most OySTER components and agents and
# MUST reside in /usr/local/oyster/etc
#
#------------------------------------------------------------------------
```

[5]Those scripts can be used to (re-) start OySTER upon reboot or as a `cron`–job.

```
#-----------------------------------------------------------------------
# Section: misc
#-----------------------------------------------------------------------

START-SECTION: misc

VERSION:       0.9
ADMIN:         Martin Mueller
MAIL:          Martin.Mueller@cl-ki.uni-osnabrueck.de
OYSTER_DIR:    /usr/local/oyster/

END-SECTION: misc

#-----------------------------------------------------------------------
# Section: hosts
#-----------------------------------------------------------------------

START-SECTION: hosts

CGI: mir.cl-ki.uni-osnabrueck.de
BBS: energia.cl-ki.uni-osnabrueck.de
DBS: soyuz.cl-ki.uni-osnabrueck.de

END-SECTION: hosts

#-----------------------------------------------------------------------
# Section: bbs
#-----------------------------------------------------------------------

START-SECTION: bbs

PORT:            4713
OYSTER_BBS_DIR:  bbs/
OYSTER_BBS_FILE: bb
BB_SERVER:       bb_server
BB_DB_SERVER:    energia.cl-ki.uni-osnabrueck.de
BB_DB_PORT:      5432
BB_DB_NAME:      oyster_bbs


END-SECTION: bbs

#-----------------------------------------------------------------------
# Section: um
#-----------------------------------------------------------------------

START-SECTION: um

UM_DB_SERVER:    energia.cl-ki.uni-osnabrueck.de
```

```
UM_DB_PORT:      5432
UM_DB_NAME:      oyster_um

END-SECTION: um

#-------------------------------------------------------------------------
# Section: wldb
#-------------------------------------------------------------------------

START-SECTION: wldb

DB_SERVER:       energia.cl-ki.uni-osnabrueck.de
DB_PORT:         5432
DB_NAME:         oyster_wl


END-SECTION: wldb

#-------------------------------------------------------------------------
# Section: urldb
#-------------------------------------------------------------------------

START-SECTION: urldb

DB_SERVER:       soyuz.cl-ki.uni-osnabrueck.de
DB_PORT:         5432
DB_NAME:         oyster_url

END-SECTION: urldb

#-------------------------------------------------------------------------
# Section: tmpdb
#-------------------------------------------------------------------------

START-SECTION: tmpdb

DB_SERVER:       soyuz.cl-ki.uni-osnabrueck.de
DB_PORT:         5432
DB_NAME:         oyster_url

END-SECTION: tmpdb

#-------------------------------------------------------------------------
# Section: thishost
#-------------------------------------------------------------------------

START-SECTION: thishost

NAME:          soyuz.cl-ki.uni-osnabrueck.de
```

```
IPN:           141.173.157.177

END-SECTION: thishost

#-------------------------------------------------------------------------
# Section: Runs/Serves
#-------------------------------------------------------------------------

START-SECTION: runs

# server-01: bbs/bb_server_agents/bb_server

client-01: bbs/bb_client_agents/db.update.d
client-03: bbs/bb_client_agents/dbi.url_url.upd
client-04: bbs/bb_client_agents/ms.wrp.d
client-05: bbs/bb_client_agents/ms.qry.d

daemon-01: bbs/bb_daemons/dbi.mstmpr.upd
daemon-02: bbs/bb_daemons/ms.tmpdb.del

#  daemon-03: bbs/bb_daemons/bbs.clean.d
#  daemon-04: bbs/bb_daemons/bbs.del.d

END-SECTION: runs
```

B.3.4 Databases

OySTER's any–time behavior is mainly achieved by extensive usage of databases. Search results are not generated sequentially but are developed in collaboration of different agents. The collaborative, parallel work is co-ordinated through the blackboard, which is realized by a database table as well. The search result is stored in a temporal database, from which it is requested by the according Cgi script.

In addition to the search results collected from the search engines, we need to locally store document classification data. This is done by a special Url database.

Finally, all user information that is needed to induce user models is stored in a distinct database, too.

B.3.4.1 Blackboard

The blackboard is realized by the table `oyster_bbs:bb`. It can be accessed exclusively by the `bb_server` agent.

A detailed description of the database design is available with the OySTER manuals.

B.3.4.2 URL database

The URL database `oster_url` mainly contains a table for all URLs and their respective classifications as well as the concept hierarchies. The main table is the `url_url` table. It contains links to several URL component tables which describe protocol, server, path and filename of the document behind the URL.
For reasons of brevity, we only include a description of the most important tables. Information conceerning the rest of the database design can be taken from the OySTER manuals.
Data concerning all URLs and information about the document behind it is stored in the table `url_url`.

Table: URL_URL (`oyster_url`) URL_URL

This table actually contains all URL / document relevant information

Field name: `id`
Description: Unique id for URLs (indexed)
Type: `int4` from `id_url_url_seq`

The following five fields can be used to simultaneously build up an indexed database on URL databases. Every URL is decomposed into protocol, server, path, file and extension and each of those is stored separately. Furthermore, the path itself becomes decomposed into directory names. The according tables then were used to induce document type classifiers that took into account only URL components instead of the documents themselves.

Field name: `url_protocol`
Description: First URL component: Protocol id
Type: `int4` from `url_protocol.id`

Field name: `url_server`
Description: Second URL component: Server id (indexed)
Type: `int4` from `url_server.id`

Field name: `url_path`
Description: Third URL component: Path id (indexed)
Type: `int4` from `url_path.id`

Field name: `url_file`
Description: Fourth URL component: Filename id (indexed)
Type: `int4` from `url_file.id`

Field name: `url_extension`
Description: Fifth URL component: Filename extension id
Type: `int4` from `url_extension.id`

Field name: `full_url`
Description: Contains full URL for indexing purposes (indexed)
Type: `text`

Field name: `full_url_lcsp`
Description: Contains id to URL with longest common server postfix (for compression purposes in future versions)[deprecated]
Type: `int4` from `url_url.id`

Field name: `full_url_lcpp`
Description: Contains id to URL with longest common path prefix (for compression purposes in future versions)[deprecated]
Type: `int4` from `url_url.id`

The timestamp fields are used for URL maintenance. They allow for updating the local database against the current state of the URL. For example, if a search engine delivers a different title for an URL as stored here, the `lastchange` attribute can be used to determine, whether an update of this entry is reasonable. Furthermore, any URL with a `lastrequest` that is long ago should be checked whether it is still existent at all.

Field name: `lastvisit`
Description: Timestamp for last check
Type: `timestamp`

Field name: `lastchange`
Description: Timestamp for last detected file change
Type: `timestamp`

Field name: `lastrequest`
Description: Timestamp for last OySTER URL request
Type: `timestamp`

Field name: `title`
Description: contains the title of the document as specified in its HTML `title` tag. If the tag is not specified, it is empty (i.e. it does not contain the URL.)
Type: `text`

Field name: `snippet`
Description: May contain a snippet (including HTML tags).
Type: `text`

Field name: `meta_author`
Description: May contain information from the `content` field of HTML `meta` tag where `name` is `author` or equivalent.
Type: `text`

Field name:	`meta_keywords`
Description:	May contain information from the `content` field of HTML `meta` tag where `name` is `keywords` or equivalent.
Type:	`text`

Field name:	`doc_type`
Description:	Contains a single expression (sort) of the document type as provided by the hierarchy in `url_dtype`. Determined by agents. (indexed)
Type:	`int4` from `url_dtype.id`

Field name:	`doc_type_ag`
Description:	Contains id of agent which determined `doc_type`. For simulated URLs we use a special reserved id 77.
Type:	`int4`

Field name:	`doc_type_time`
Description:	Contains timestamp of when `doc_type` was last changed.
Type:	`int4`

Field name:	`doc_type_confidence`
Description:	Contains confidence; `int2` range to be interpreted in [0..1].
Type:	`int2`

This section was initially motivated by the idea that a search request also could be specified by the author of a web page; i.e. a user looks for a page of type t that is about c and which was written by a person t'.

Field name:	`doc_type_author`
Description:	Contains a single expression (sort) of the document author as provided by the hierarchy in `url_dtype`. Determined by agents.
Type:	`int4` from `url_dtype.id`

Field name:	`doc_type_author_ag`
Description:	Contains id of agent which determined `doc_type_author`. For simulated URLs we use a special reserved id 77.
Type:	`int4`

Field name:	`doc_type_author_time`
Description:	Contains timestamp of when `doc_type_author` was last changed.
Type:	`int4`

Field name:	`doc_type_author_confidence`
Description:	Contains confidence; `int2` range to be interpreted in [0..1].
Type:	`int2`

Most important to the user model induction process is the document category classification data. Document categories c_1, c_2 and c_3 are stored here together with their confidences p_1, p_2 and p_3, respectively.

Field name: `doc_category, doc_category_2, doc_category_3`
Description: Contains document category id as determined by agents.
Type: `text`

Field name: `doc_category_ag`
Description: Contains id of agent which determined `doc_category`. For simulated URLs we use a special reserved id `77`.
Type: `int4`

Field name: `doc_category_confidence, doc_category_confidence_2, doc_category_confidence_3`
Description: Contains confidence; `int2` range to be interpreted in [0..1].
Type: `int2`

Field name: `doc_category_time`
Description: Contains timestamp of when `doc_category` was last changed.
Type: `int4`

Field name: `url_ftype`
Description: Pointer to file type id
Type: `int4` from `url_ftype`

Field name: `source`
Description: Source of URL. There are different versions: `CMU` indicates that the URL and its classification has been taken from the CMU Dataset. `CH_wget` identifies URLs that were collected by Christian Heißing during his work on document type classifiers. For artificially generated URLs that were used during the evaluation of the user model induction process, we used the key `testdata`. URL that have been added during the normal meta search operation are labeled `Oy msqry (uid:`*`id`*`)`, where *`id`* is the Blackboard id of the last request which delivered this URL.
Type: `text`

Field name: `url_fserver`
Description: full server name
Type: `text`

Field name: `url_fpath`
Description: full path
Type: `text`

Field name: `url_file`
Description: full file name
Type: `text`

For the meta search we need a temporary URL Cache which is realized by the `mstmpr` table.

Table: MSTMPR (`oyster_url`) MSTMPR

This table contains URLs that have been found by recent search requests.

Field name: `uid`
Description: User id
Type: `int4`

Field name: `rid`
Description: Request id as noted on the blackboard
Type: `int4`

Field name: `uip`
Description: User IP-number
Type: `text`

Field name: `url`
Description: URL
Type: `text`

Field name: `urlid`
Description: URL id as id from `url_url`
Type: `int4`

Field name: `host`
Description: URL of the URL host system.
Type: `text`

Field name: `hostid`
Description: URL id of the URL host system.
Type: `int4`

Field name: `foundby`
Description: Key word(s) for result source.
Type: `text` from `url_extension.id`

Field name: `foundtimes`
Description: Integer describing how many resources reported that URL
Type: `int4`

Field name: `hostorurl`
Description: [deprecated]
Type:

Field name: `listed_at`
Description: Unix timestamp for adding this URL to `mstmpr`
Type: `int4`

Field name: `added_at`
Description: Unix timestamp for adding this URL to `url_url`
Type: `int4`

Field name: `last_request`
Description: Unix timestamp for last client query for URL
Type: `int4`

Field name: `rank`
Description: Aggregate rank value
Type: `float8`

B.3.4.3 User model database

The database `oyster_um` is used to store user data, user profiles and user feedback from which samples are derived.
We here only give an overview of the table holding all data about user proided and simulated feedback.

Table: USERFEEDBACK (`oyster_um`) USERFEEDBACK

The `userfeedback` table is the most important part for the user model induction process. Here, feedback of the user with respect to distinct aspect

Field name: `uid`
Description: User id as defined in `userdata`
Type: `int4`

Field name: `urlid`
Description: URL id as defined in `oyster_url:userdata`
Type: `int4`

Field name: `time`
Description: Local Unix time of feedback submission
Type: `int4`

Field name: `aspect`
Description: Aspect id as defined in `oyster_um:aspects:id`. Feedback is given with respect to aspects.
Type: `int4`

Field name:	`feedback`
Description:	Feedback value from [-2, 2].
Type:	`int4`

Field name:	`type`
Description:	Contains optional keys in order to identify `learn`- and `test`data which was generated during the evaluation of user model induction.
Type:	`text`

Field name:	`centroid`
Description:	This is an additional optional field which was used for generation of artificial data. It contains document category id's (`oyster_url:url_dcat:id`) which were considered as so–called centroids during the process of simulating feedback. See section 6.1.2.
Type:	`int4`

Field name:	`sign`
Description:	Optional feature which was used to discriminate `p`ositive and `n`egative interest centroid. Only used in the first test series (c.f. section 6.2)
Type:	`char`

B.3.4.4 Word lists

In addition to the databases mentioned above, the training of classifiers required a database for word lists.
All data concerning word lists and classifier training is stored in the database `oyster_wl`.

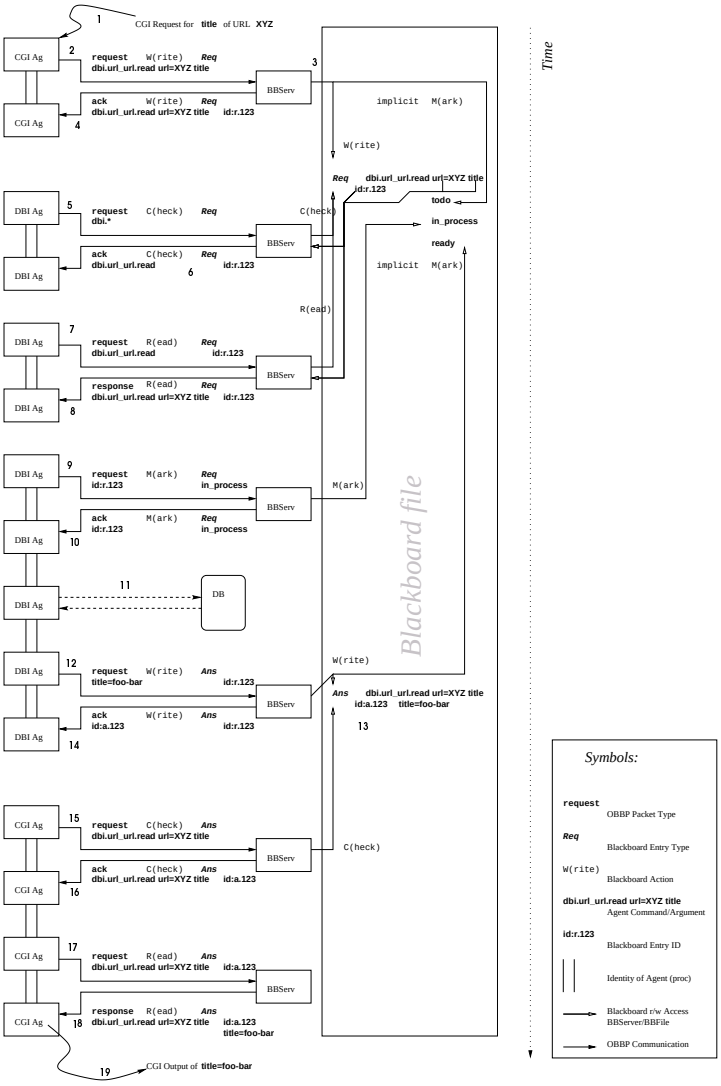


Figure B.6: A sample communication flow.