User Profiling for the Melvil Knowledge Retrieval System

Johannes Fürnkranz
Christian Holzbaur
Austrian Research Institute for Artificial Intelligence (ÖFAI)
Schottengasse 3, A-1010 Wien, Austria
[juffi,christian]@oefai.at

Robert Temel
uma Information Technology AG
Donau-City-Straße 1, A-1220 Wien, Austria
robert.temel@uma.at

Technical Report OEFAI-TR-2001-29

Abstract

Melvil is an ontology-based knowledge retrieval platform that provides a three-dimensional visualization of search results. The user can tailor the presentation of the search results to her preferences by changing the settings of various parameters on the screen. In this paper, we report on a prototype implementation of a user profiling device that learns to predict appropriate settings for these parameters for the current search results based on previous experiences. In a preliminary study, we evaluated several off-the-shelf machine learning algorithms on parts of the problem. The final implementation required the flexibility of handling both regression and classification problems, being able to deal with set-valued input and output attributes, as well as incorporating Melvil's ontologies for the respective application domain. Thus, we selected a nearest-neighbor approach for the prototype implementation. An evaluation on off-line data collected from several users showed a satisfactory performance.

1 Introduction

Melvil is an ontology-based knowledge retrieval platform that allows to visualize search results in an interactive 3-dimensional representation. The application is currently under development at uma AG. The user interface to the visualization module provides several facilities for changing the presentation of a search result. These include various switches for filtering

groups of documents (e.g., excluding documents from certain domains or servers), adjustable thresholds for filtering documents along some numerical characteristics (e.g., length and recency), and sliders that allow to increase or decrease the weight of query terms (which changes the sorting of the documents in the display). The goal of the work described in this paper is the development of a user profiling component for this visualization component of Melvil. The user profiler should be able to predict suitable positions of these knobs and switches for an individual user based on her past behavior.

For this purpose, we first analyzed the problem, defined potentially relevant variables that could be used for prediction, and collected interaction data from various users (see section 3). The experimental work was done in two separate phases: The first phase was a preliminary exploration aimed at getting a quick grasp of the data material, and a simple comparison of various learning approaches (section 4). For this phase, we relied on existing machine learning software, in particular on the publicly available Weka Data Mining library (Witten and Frank, 2000). Based on our analysis of the problem and the experiences collected in this first experimental study, we decided to implement the prototype based on a nearest-neighbor learning algorithm. This implementation is described in section 5. The final evaluation of the prototype is described in detail in section 6. In the next section, we start with a brief overview of the Melvil knowledge retrieval platform.

2 Overview of Melvil

Finding and analyzing relevant information is becoming more and more critical to the success of every organization. The current rate of data explosion requires new approaches to information management and retrieval. The more data is accessible through digital networks, the more urgent is the demand to find and analyze what you really need. Melvil is a knowledge retrieval system that assists its user in solving this challenging task.

Melvil helps to represent the collective knowledge of an organization and make it accessible as a whole to even a single knowledge worker. Query results are dynamically displayed in a web-based 3D-environment and empower the user to navigate visually through meta and detail information spaces. Retrieving and analyzing information becomes a visual and therefore a more intuitive task.

2.1 Ontology Support

Searching through knowledge-bases by keywords, then sorting through endless lists of documents increasingly frustrates and often does not lead to satisfying results. Sometimes users do not know exactly how to formulate a request or which keywords to use for a successful query. Using Melvil's support for ontologies—hierarchically structured semantic networks of terms—

the user is guided through knowledge areas s/he is not familiar with. This semantic net—created by domain experts—guides a non-expert user in a visual way to help to define the queries that capture his or her information need. Melvil's semantic nets consist of interrelated concepts which are a combination of specific *terms* and matching *term patterns*, such as synonyms and various spellings, in one or more languages.

Retrieved knowledge can only be as meaningful and comprehensive as the underlying sources and semantic nets that build the basis of any current state-of-the-art knowledge retrieval system. Melvil provides a powerful ontology editor which enables a single editor or a widespread network of experts to build an ontology simultaneously online, logically describing a specific knowledge area. Relevant data sources on the user's Intra/Extranet as well as on the Internet can be added to the system, which will scan and index them periodically. Thus Melvil enables knowledge workers to retrieve data from different sources.

2.2 3D Visualization

Melvil provides knowledge workers with several dynamic data filtering tools, that help screening results by parameters including recency, length, domain or top-level-domain. Additional functions such as Melvil's concept weight tool enable the user to give certain concepts more "weight" than others. Melvil's meta-analysis space allows to navigate visually through large groups of documents displayed as graphical, color-coded 3D objects, containing different combinations of the concepts included in the query (see figure 1).

Dynamic data filtering helps the user analyze a set of results before entering Melvil's detail-analysis space to focus on a specific group of documents, all containing the same combination of concepts. To examine one document in detail, one may simply click on the corresponding graphical object and a new browser-window containing this document will be opened. Melvil's long term monitor and weight preference profiler complete the suite of filtering options. The former helps monitoring the change of results for the same query, while the latter is an automated user profiler that assists the user with customizing result sets. Its design is the topic of this paper.

2.3 Implementation and Availability

Requirements of uma's clients differ widely. Therefore Melvil is conceived as a platform-independent, Java-based software solution, that can be easily customized. A modular set of 3D and 2D interfaces of the query, meta-analysis and detail-analysis spaces makes it an easy task to devise customized visualizations.

The Melvil client application is based on Java, HTML and VRML 2.0, and requires the Blaxxun Contact Plug-In. A wholly Java-based version is

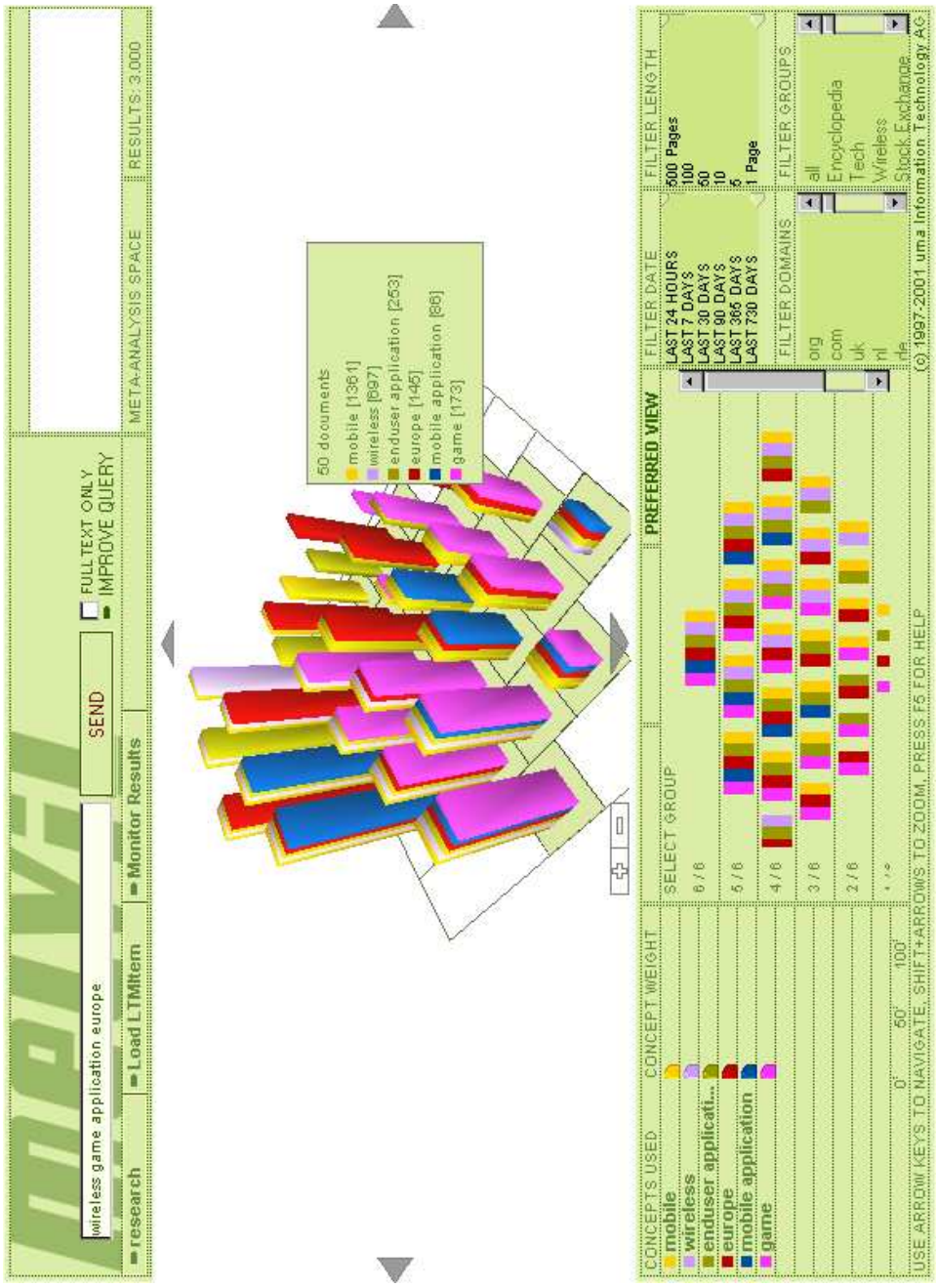

Figure 1: Melvil's 3D visualization of query results.

being developed. The application itself consists of a set of Java applets, that are downloaded from the server. The minimum requirements for Melvil are 64 Mb RAM, and a Pentium II processor or similar. Some advanced features are best viewed with a 3D-accelerated graphics-board, 128 Mb RAM and a Pentium 4 processor or similar. Currently supported OS for the client are Windows98, NT and 2000.

More information, including demonstration software, can be found at `http://www.uma.at/melvil`.

3 Problem Description

As stated above, our goal is to model a user's preferences regarding the visual presentation and the filtering of Melvil's knowledge retrieval engine. The user is able to influence the display by modifying the settings of the parameters shown at the bottom row of panels on the screen (figure 1). In particular, s/he may specify the relative importance of the query terms using the sliders to the right of the query terms (left side of the screen), place constraints on the recency and the length of the retrieved documents using two movable arrows to the right of the respective panels (upper half of the right part), as well as switch various domains and groups on and off by selecting or deselecting their entries (lower right corner). The task of the user profiler is to model a user's preferences with respect to these parameters. As the user's preferred position for the switches and knobs in the interface may depend on her query and on the search result produced by this query, the learning problem is to find a mapping between the query and the returned result to the user's preferences. Different users will have different preference profiles, hence a different user model has to be learned for each individual.

The first step is to identify suitable characterizations of the query and the search results. For our study, we agreed upon the descriptors shown in the upper part of table 1. These are mostly aggregate attributes over the individual documents in the search result, such as the minimum, maximum, and average of the document length in the search result, which we thought would bear some relevancy for learning appropriate filter settings for these characteristics. A special role is played by attribute 8, which encodes the frequency of found documents for each term in the query. As different queries contain different query terms (and also a different *number* of query terms), this information cannot be represented by a single value, only by a set of values (one for each term in the query).

The lower part of table 1 shows the variables whose values have to predicted by the learning algorithm. These represent the variables that the user can change with the GUI in order to adjust the presentation of the results to her personal taste. Note that these do not correspond exactly to

No.	attribute	type
Independent Variables		
0	user id	categorical
1	number of found documents	numerical
2	number of found query concepts	numerical
3	number of subspaces	numerical
4	number of document groups	numerical
5	ontology id	categorical
6	number of servers with documents	numerical
7	number of domains with documents	numerical
8	frequency of query terms	numerical / set-valued
9	last modified date / minimum	numerical
10	last modified date / maximum	numerical
11	last modified date / average	numerical
12	document length / minimum	numerical
13	document length / maximum	numerical
14	document length / average	numerical
Dependent Variables		
15	weights of query terms	numerical / set-valued
16	last modified date / minimum threshold	numerical
17	last modified date / maximum threshold	numerical
18	document length / minimum threshold	numerical
19	document length / maximum threshold	numerical
20	server filter	boolean / set-valued
21	domain filter	boolean / set-valued

Table 1: Data specification

the ones shown in the screenshot of figure 1. The reason is that the work described in this paper was performed on an older protoype of Melvil, while the screenshot is taken from a more recent version which incorporated some simplifications in the number and the nature of the parameters that can be modified by the user. Most notably, we were predicting real values for the minimum and maximum thresholds of the desired document length and recency, while the newer version uses a discrete scale.

Note that again several of the attributes are set-valued. Attribute 15 (the importance of each query term for the user) has a one-to-one correspondence to the attribute 8 (the number retrieved documents that contain the query term), i.e., for each value that appears in attribute 8, there is a corresponding concept weight setting in attribute 15. Most problematic are attributes 20 and 21: they specify whether documents from a particular domain or server should be displayed or not. The number of these parameters depends on whether or not documents from these sources are present in the query result.

	Ontology	
user	0	1
chrid	29	0
ck	52	54
hg	52	54
katz	52	50
rs	52	54
rt	52	54
sw	52	54
rt2	154	0

Table 2: Overview of the datasets

The final implementation of the system should work as follows: First, the user formulates a query using a given ontology. An ontology consists of a fixed number of hierarchically organized concepts that can be added to the query via a graphical or text-based query editor. The Melvil system returns the search result for this query and the learned user model decides upon an appropriate filter and concept weight settings by predicting values for the target variables. The user can then improve the presentation of the results by modifying the predicted values. The final values (which are assumed to be the optimal choice for this user for this query) are then returned to the learning system as a new *training example*. It is used by the learner to adjust the user profile so that it is more likely to predict these final values in similar situations (i.e., in situations that are characterized by similar values of the input parameters). In the beginning, the predicted values will follow a default model, but after several interactions with the system, i.e., when more and more training examples come in, the system should adjust the model for this particular user to her individual preferences.

3.1 Data Description

The experiments reported in this paper were performed off-line, using data that were collected from several users that interacted with the system. The first set of data originates from 7 users who were given 106 pre-defined queries, and instructed to visualize the search results for each of the queries according to their liking. Only 5 of the seven completed the entire set of queries. For each test query, we collected the 22 variables shown in table 1.

The test queries were based on two sample ontologies. As it is not possible to re-use the knowledge learned for the concepts in one query within another query (at least not unless an explicit semantic mapping between the concepts in the two hierarchies is defined), the datasets had to be split

```
103 institutionen
     106 ministerien
     107 landesregierungen
     109 agenturen
     108 aemter
     105 kammern
     3   universitaeten
     104 technologiezentren
     110 foerderungsstellen
     111 involvierte fonds
28 taetigkeiten
     29 forschungsprojekt
          30 grundlagenforschung
               31 konzepterstellung
               32 studie
               33 forschungsarbeit
          34 angewandte forschung
               ...
          ...
     ...
...
```

Figure 2: Sample fragment of ontology 0 (BMWA).

according to the ontologies used. These datasets were used for the initial evaluation described in section 4.

At a later stage, we obtained 154 additional examples from user `rt`. This set was mostly used for the experimental evaluation of the prototype described in section 6.

3.2 Ontologies

Domain ontologies are often viewed as the key ingredient for successful IT applications (Fensel, 2001; Staab and Maedche, 2001). They are also at the core of Melvil's knowledge retrieval engine. An ontology in Melvil is a semantic hierarchy that captures expert knowledge about the application domain. Each node in the ontology is associated with textual patterns that are used to index documents under this concept. Both the ontology and the associated patterns are constructed manually using Melvil's ontology editor.[1] For the end-user, Melvil provides a powerful query editor that can be used with a text-based or a graphical interface.

[1]Recent developments for learning ontologies (Maedche and Staab, 2001; Staab et al., 2000; Maedche et al., 2001) and for learning text classifiers that recognize instances of ontologies (Craven et al., 2000; Mladenić, 1998) are considered for future work.

The user query is represented in attribute 8 of our input data, which contains the ontology nodes that have been selected by the user and their frequency of occurrence in the query result (basically the information that is captured in the text box in the middle of figure 1. The set-valued nature of this attribute (each interaction by a user can use an arbitrary set of concepts) is incompatible with the design of most machine learning algorithms, which assume an input vector of a fixed size. Typically, such information is handled by unfolding the set-valued attribute in to a set of n attributes, where n is the number of valid values, in our case the number of different nodes in the ontology. However, this approach has the disadvantage that one needs a large number of training examples in order to ensure that each concept occurs frequently enough. A brief examination of the data contained in the 106 test queries (52 for ontology 0, 54 for ontology 1) showed that most of the concepts occur less than 5 times, many not at all. Table 3 shows the frequencies for the nodes in ontology 0 (the situation for the other ontology was quite similar).

This shows that a straight-forward unfolding of the ontology into binary attributes is not advisable because for most cases there are too few interactions from which future behavior could be predicted. Collecting more data would be helpful in this case, but it would ignore the real-world situation, where the user expects quick learning after a few interactions with the system.

Hence we eventually settled for an approach that exploits the concept hierarchy of the ontology. Figure 2 shows a fragment of ontology 0 (BMWA) which represents knowledge about the Austrian Federal Ministery for Economics and Labor. At the top level, there are nodes like "institutions" and "activities", which are further refined at the levels below. The maximum depth (not shown) is 6. Instead of unfolding each node of the hierarchy into a separate binary attribute, we only consider nodes down to a certain maximum level. Lower nodes are replaced with their predecessors. Thus, we exploit the generality relation that is implicitly defined within the ontology: concept 32 (`studie`) generalizes to 30 (`grundlagenforschung`) which in turn generalizes to 29, and finally to 28 (`taetigkeiten`).

3.3 Evaluation Measures

To allow a comparison of the learning performance under various circumstances, we employed several evaluation measures. We will briefly introduce them here, as they will appear in various places of the paper. In the following, p_i refers to the value predicted by the classifier on example i, while c_i is the correct value. d_i refers to the predictions of a default model. In the case of the experiments reported in section 4, the default model always predicts the mean value of all known values of the dependent variable (regardless of the values of the independent variables). For evaluating the

0	2	**20**	11	**40**	1	**60**	6	**80**	6	**100**	6
1	8	**21**	3	**41**	1	**61**	13	**81**	1	**101**	1
2	0	**22**	2	**42**	3	**62**	6	**82**	13	**102**	6
3	0	**23**	1	**43**	1	**63**	3	**83**	6	**103**	0
4	9	**24**	6	**44**	1	**64**	1	**84**	2	**104**	0
5	4	**25**	0	**45**	2	**65**	2	**85**	8	**105**	0
6	5	**26**	3	**46**	1	**66**	2	**86**	3	**106**	2
7	35	**27**	4	**47**	1	**67**	7	**87**	1	**107**	1
8	16	**28**	0	**48**	15	**68**	2	**88**	1	**108**	0
9	14	**29**	18	**49**	5	**69**	2	**89**	9	**109**	0
10	2	**30**	4	**50**	4	**70**	0	**90**	3	**110**	0
11	17	**31**	3	**51**	1	**71**	27	**91**	3	**111**	0
12	2	**32**	2	**52**	3	**72**	7	**92**	3	**112**	0
13	7	**33**	2	**53**	3	**73**	5	**93**	3	**113**	0
14	3	**34**	6	**54**	10	**74**	2	**94**	0		
15	2	**35**	2	**55**	6	**75**	1	**95**	0		
16	1	**36**	1	**56**	4	**76**	5	**96**	3		
17	2	**37**	4	**57**	14	**77**	1	**97**	2		
18	23	**38**	2	**58**	9	**78**	1	**98**	0		
19	4	**39**	1	**59**	6	**79**	2	**99**	1		

Table 3: Frequencies of concepts occuring in Ontology 0 (BMWA) in the 100 test queries (concept IDs are shown in bold face).

prototype implementation (section 6), we used the default model described in section 5.7, which always leaves all filter settings on (i.e., it always shows the entire search result).

Error:

$$err_i = \left\{ \begin{array}{ll} |p_i - c_i| & \text{for numerical variables} \\ \left.\begin{array}{ll} 0 & \text{if } p_i = c_i \\ 1 & \text{if } p_i \neq c_i \end{array}\right\} & \text{for categorical variables} \end{array} \right. \quad (1)$$

This is basically the difference between predicted and correct value of example i.

Default Error:

$$derr_i = \left\{ \begin{array}{ll} |d_i - c_i| & \text{for numerical variables} \\ \left.\begin{array}{ll} 0 & \text{if } d_i = c_i \\ 1 & \text{if } d_i \neq c_i \end{array}\right\} & \text{for categorical variables} \end{array} \right. \quad (2)$$

The default error $derr_i$ is identical to the error, except that we do not evaluate the predictions p_i of the classifier but the predictions d_i of a default model. Hence, this measure does not evaluate the classifier itself, but it is a useful benchmark comparison for the Error.

Mean Absolute Error:

$$mae = \frac{1}{n}\sum_{i=1}^{n} err_i \tag{3}$$

This measure simply computes the average deviation from the predicted value over all n examples.

Mean Squared Error:

$$mse = \frac{1}{n}\sum_{i=1}^{n} err_i^2 \tag{4}$$

This computes the average squared error over all n examples. It will penalize large deviations on a few examples more severely than small deviations on more examples. Note, however, that for the symbolic predictions (variables 20 and 21) there is no difference between the squared error and the absolute error, because the error can only be 0.0 or 1.0, for which squared values are the same.

Relative Mean Absolute Error:

$$rmae = \frac{\sum_{i=1}^{n} err_i}{\sum_{i=1}^{n} derr_i} \tag{5}$$

This is the relative average error between two different classifiers. A value < 1 signals an improvement over the default model.

Relative Mean Squared Error:

$$rmse = \frac{\sum_{i=1}^{n} err_i^2}{\sum_{i=1}^{n} derr_i^2} \tag{6}$$

This definition is analogous to the previous one, only for squared errors.

Correlation:

$$corr = (n-1)\frac{\sum_{i=1}^{n}(p_i - \bar{p})(c_i - \bar{c})}{\sum_{i=1}^{n}(p_i - \bar{p})^2 \sum_{i=1}^{n}(c_i - \bar{c})^2} \tag{7}$$

where $\bar{p}$ is the mean value of all predictions p_i and $\bar{c}$ is the mean value of the correct values. The correlation coefficient is close to 1 if there is a good correlation between the predictions and the correct values, it is close to 0 if there is no correlation (and it may be close to -1 if there is a negative correlation, i.e., when one variable increases when the other decreases and vice versa).

Run-Time: *time* shows the run-time of the algorithms in CPU secs. user time on a SPARC Ultra-2 station running Solaris 5.6.

4 First Steps: Evaluation of Different Learning Algorithms

After collecting the datasets shown in Table 2, we decided to perform a quick evaluation of the datasets with publicly available machine learning techniques. The casual reader may skip the details of this section and fast-forward to its summary in section 4.4.

For the purposes of these first experiments, we chose to work on variables 16–19 because they have single target values, which can be straightforwardly dealt with by commonly available algorithms. The remaining targets (15, 20, and 21) are set-valued variables, which cannot be handled by available machine learning algorithms without a transformation of the data. We deferred their evaluation until an implementation of the prototype was available, which was able to handle this particular type of problem (see section 5).

The main goal of the evaluation was primarily to get a "feel" for the learning problem and for possible ways to address it. First of all, we wanted to see whether learning is possible at all, i.e., whether learning algorithms are in fact able to improve classification performance over time. Besides, we hoped to get some insights about which learning algorithms are applicable, and how we could deal with the peculiar properties of these data, most notably with the ontologies.

4.1 Algorithms

We decided to try the following learning algorithms for numerical target attributes (*regression algorithms*):

kNN: Instance-based learning was popularized in machine learning through the work of David Aha (Kibler et al., 1989; Aha et al., 1991), but it has its roots much earlier in the *k-nearest-neighbor classifier* (see (Dasarathy, 1991) for a collection of early papers in that area). The kNN algorithm does not learn an explicit model of the data, but simply stores all past experiences and uses them as the current model. To classify a new example, it scans its database to find the k most similar matches, and derives a prediction for the new example from the stored values of these k neighbors. In this section, we used the simplest form with $k = 1$, i.e., the dependent variables of the nearest neighbor are directly used for predicting the dependent variables of the new example. A more detailed description can be found later in this paper (section 5.1).

LR: *Linear Regression* is a well-known technique from statistics. It finds a linear combimation

$$y = \lambda_0 + \sum_{i=1}^{n} \lambda_i x_i \tag{8}$$

which computes a prediction y from the input variables x_i. The training data are used for determining optimal values for the λ_i ($i = 0 \ldots n$), where optimality is defined as minimizing the mean squared error of the predicted values (see section 3.3).

LWR: *Locally Weighted Regression* is a technique that tries to unify some of the properties of kNN and LR: the prediction is made using a linear equation such as (8), but the coefficients λ_i are determined in a way that gives higher weights to errors that are near the example to classifiy, i.e., each example is classified with a separate model that is derived at classification time for this particular example. A survey on this and related so-called "local" learning techniques can be found in (Atkeson et al., 1997).

M5': M5' learns a so-called *regression tree* (or *model tree*). Contrary to an ordinary decision tree (or classification tree), a regression tree can be used for predicting numeric variables, but it shares with them the tree-structure which is incrementally refined during learning. Regression trees have been introduced by (Breiman et al., 1984) and (Quinlan, 1992).

Default (Mean Prediction): This is not a learning algorithm in the strict sense (or at least not a very clever one). It simply predicts the mean value of all past observations of the independent variables as the value for the new observations. This prediction is the constant value that minimizes the mean squared error estimate. The method is mainly included as a benchmark value for the other algorithms: if they are unable to achieve a better performance than this method, they cannot be said to learn something useful.

For experimentation, we used the Weka Data Mining library, which provides publicly available, state-of-the-art implementations of the above (and other) learning algorithms (Witten and Frank, 2000). This choice also allowed us to perform a fair comparison of the run-times because all algorithms are implemented in the same framework, in the same programming language (Java) and with the same overhead for data handling.

Each of the above algorithms was tried on each of the datasets for each of the target variables. We also implemented an option that allows to prune the ontology to various degrees (described below). In total, we performed 1120 experiments (7 users $\times$ 2 ontologies $\times$ 4 target variables $\times$ 5 learning

Measure	kNN	LR	LWR	M5'	Default
	Variable 16				
mae	3.38e+07	3.81e+07	3.23e+07	1.66e+07	3.56e+08
mse	1.14e+16	5.35e+15	3.79e+15	1.90e+15	1.41e+17
rmae	0.162	0.297	0.244	0.179	1.000
rmse	0.126	0.100	0.079	0.052	1.000
corr	0.858	0.875	0.889	0.872	0.000
time	2.04	57.17	539.18	2.28	0.27
	Variable 17				
mae	1.60e+07	2.35e+07	2.71e+07	1.20e+07	1.64e+07
mse	6.20e+14	1.69e+15	2.12e+15	3.88e+14	5.76e+14
rmae	0.955	1.333	1.470	0.664	1.000
rmse	1.100	2.791	3.129	0.560	1.000
corr	0.480	0.547	0.604	0.669	0.000
time	2.06	62.96	566.20	5.12	0.29
	Variable 18				
mae	7.94e+03	8.46e+03	9.81e+03	5.55e+03	9.56e+03
mse	1.52e+08	1.66e+08	2.31e+08	6.25e+07	1.60e+08
rmae	0.787	0.768	0.912	0.495	1.000
rmse	0.874	0.798	1.167	0.296	1.000
corr	0.458	0.701	0.699	0.731	0.000
time	2.06	58.85	540.35	4.68	0.29
	Variable 19				
mae	4.73e+04	3.70e+04	3.82e+04	2.09e+04	7.21e+04
mse	5.61e+09	2.76e+09	3.04e+09	9.95e+08	8.63e+09
rmae	0.694	0.584	0.657	0.354	1.000
rmse	0.732	0.416	0.545	0.171	1.000
corr	0.557	0.722	0.735	0.793	0.000
time	2.05	56.86	523.05	4.53	0.28

Table 4: Average Results over all experiments

algorithms $\times$ 4 ontology pruning settings). Each experiment in turn performed a 10-fold cross-validation for estimation of the evaluation measures, i.e., the reported results are the average of 10 runs, where in each of them one tenth of the data is left out in training and reserved for testing, in a way that ensures that each example appears exactly once in a test set.

4.2 Comparison of Learning Algorithms

An evaluation of such a five-dimensional results matrix (users $\times$ ontologies $\times$ target variables $\times$ learning algorithms $\times$ ontology pruning settings) is a challenging task itself. We decided to average the performance over the other

dimensions when evaluating a particular dimension. We also decided to ignore the results for the users `chrid` and `katz` who did not complete all 106 test queries. Table 4 shows the average results for each of the algorithms on each of the four target variables. The six evaluation measures are described in section 3.3.

The most important result is that all four learning algorithms outperform the default classifier in variables 16, 18 and 19. In all three cases, the values of *rmae* and *rmse* are well below 1.0, which means that they improve upon the default model. Likewise, the correlation coefficients indicate a positive correlation (although not always a strong one) between the predicted and the correct values. However, on variable 17, only one of the algorithms, namely the model tree M5' seems to achieve a better squared error rate than the default classifier, although all of the algorithms achieve at least a better correlation than the default classifier. kNN is also slightly better than default classification with respect to (relative) mean absolute error.

The obvious next question is, which algorithm performed best and should be implemented in the prototype. Looking at the run-times of the algorithms, we see that kNN and M5' are one order of magnitude faster than linear regression, which in turn is one order of magnitude faster than locally weighted regression. From these results we ruled out regression algorithms entirely because their performance was not better than other choices but their run-time was much worse.

Eventually, we had to decide whether to use a nearest-neighbor classifier or a tree-based algorithm for the implementation of the prototype. The above results are clearly in favor of M5'. However, there are several pragmatic reasons that strongly suggest the use of a nearest-neighbor approach. These are:

- kNN algorithms work *incrementally*, i.e., they can adapt their model after each individual example without having to be re-trained. In fact, the learning step consists simply of adding the example to the case base of that particular model.

- kNN algorithms are able to predict both, numerical and categorical target values. Typically, machine learning algorithms specialize on one or the other, so that we would have had to integrate and coordinate at least two different types of learning algorithms, which is a considerable overhead.

- The core element of a kNN algorithm is a similarity function, which can be defined in a flexible way by defining similarities on individual attributes (see section 5.1). For example, this could allow to define a flexible similarity between query terms on the basis of the hierarchical ontology. In the current prototype the handling of the ontology is performed as a pre-processing step but a dynamic integration into the

similarity function could turn out to be more desirable for the final system.

Finally, response time is crucial for our application, and here kNN's flexibility once again proves to be advantageous: In tree learning, a separate model has to be learned for each of the target variables after each incoming example. Instance-based learning approaches have to identify the distances of a new example to all stored examples once, and can then re-use this information for deriving a prediction for arbitrarily many target variables (unless we use a differently weighted similarity function for different target values, see section 5.1). In effect, this means that for learning all four target variables shown in table 4, M5' would need the sum of the times for each individual variable. kNN, however, has to perform the same operation for each variable (computing the distance to all neighbors), which can also be seen from the (almost) identical run-times of kNN on each of the four tasks. Hence, the total run-time of an efficient implementation would be equivalent to the maximum (and not the sum) of the run-times on the four variables.[2] This problem will be much more severe for the set-valued target variables because, as we will see in section 5.4, each of them will be unfolded into a set of single-valued target variables. In this case, in can be expected that kNN is orders of magnitude faster than M5'.

However, a slow algorithm with a good classification performance may, of course, still be preferrable to a fast algorithm that is much worse. Hence, the question is whether the performance of the nearest neighbor algorithm can be improved. Fortunately, this turned out to be the case: in the above experiments, we had only used a single neighbor for making the predictions. After seeing the result, we made a few additional experiments to probe whether there is a chance to improve the performance by using more than one neighbor, which proved to be the case (see section 6.5).

4.3 Different Ontology Depths

Most machine learning algorithms are restricted to the use of rectangular data, i.e., data where each example is described with a fixed number of values for a fixed number of attributes. Hence we had to implement a data conversion routine that maps the set-valued attribute 8, which captures information about the ontology, to such a representation. The basic idea of the mapping is to encode each possible value that may occur in the set as a separate binary variable (the value is present in the original set-valued attribute or it is not). As the underlying ontologies are rather big, it may take

[2] There will be some small additional overhead for actually computing the predictions for each variable, but the expensive operation is the computation of the distances to the neighbors, and this only has to be performed once.

Measure	Ontology Depth			
	0	1	2	∞
	Ontology 0			
rmae	0.630	0.658	0.692	0.796
rmse	0.579	0.623	0.753	0.979
corr	0.638	0.603	0.541	0.472
time	0.84	1.16	1.97	5.58
	Ontology 1			
rmae	0.545	0.563	0.574	0.741
rmse	0.446	0.464	0.495	0.865
corr	0.678	0.672	0.647	0.455
time	0.87	0.90	1.10	4.00

Table 5: Results of kNN with different settings of the ontology pruning parameter

many training examples before the same term occurs another time. However, with the use of the ontologies, it should be possible to use information about the similarity of two different concepts for improving the frequency of similarity matches.

Our approach maps all concepts in the ontologies to their ancestors at a specified level l and uses only the concepts at this level in a binary coding as described above. More precisely, all concepts with depth $d > l$ are replaced with their ancestor node at depth l. For each of the remaining concepts (those with depth $d \leq l$) we generate one binary attribute. In effect, l specifies the deepest level of the ontology that will be used during classification. A setting of $l = 0$ means that only the top node of the ontology is used (which basically amounts to not using it at all, because all values are treated the same). The setting $l = \infty$ represents the setting where all concepts are used as specified and are not generalized (l exceeds the maximum depth of the ontology).

Table 5 shows the results for different settings of the ontology depth parameter. We only show the results for the kNN algorithm, but the results for the other algorithms were qualitatively the same. It can be seen that the use of the entire ontology (last column, ∞) performs much worse than generalizing to the first or second level. Moreover, the best performance (albeit only by a small and insignificant margin) is achieved by reducing the ontology to a single node. There are at least two possible explanations for this phenomenon: first, it may actually be the case that the ontology information is in fact irrelevant for the prediction of the variables 16–19. This would mean that the decision on the filter settings regarding the recency and length of the retrieved documents mainly depends on the character-

istics of the retrieved batch of documents and not so much on the query concepts. This does not sound entirely implausible. Another explanation is that increasing the number of used concepts in the ontology will increase the number of binary variables that are used in the internal representation of the data, which in turn means that the number of possible concepts increases considerably. As the number of available examples does not increase at the same time, there is more room for overfitting the data, i.e., for learning a concept that fits the available dataset very well but generalizes poorly to unseen examples. Also note that the run-time increases considerably when using the entire ontology, which is mostly due to the increasing number of variables that have to be matched.

However, it should be noted that from these results we cannot infer that we do not need the ontology. In particular for variable 15—the weight setting for the query terms—the information about the query terms will be of importance, but it can also be expected to help to filter the retrieved documents based on their origin (server (20) and domain (21)). A possible remedy for the overfitting problem mentioned above might be to use an adaptive procedure that successively increases the ontology depth with the number of stored examples. Thus one could avoid the overfitting problem with few examples, but could gradually shift to deeper and deeper concepts in the ontology as soon as sufficiently many training examples are available for a particular user.

4.4 Summary

In summary, the results showed that learning is indeed possible on the sample user data, and that a nearest-neighbor approach is the best choice with respect to run-time behavior and flexibility. On the other hand, its performance in terms of predicitive accuracy is inferior to a tree-based approach, if we only use a single neighbor for deriving the prediction. However, we also conducted informal experiments that showed that increasing the number of neighbors will improve the performance considerably, an issue that will be explored in detail in section 6.5. Consequently, we decided to implement the nearest-neighbor approach in the final prototype.

With respect to the ontology, our experiments seemed to indicate that the information contained in the ontologies is not crucial for predicting variables 16–19. In fact, if the concepts are not generalized, the predictive accuracy may even decrease, most likely because of overfitting the low amount of data. However, while it can be imagined that the concepts that appeared in the query do not have much influence on a user's filtering settings for the desirable recency and length of retrieved documents, the settings for the origin of the document (server and domain) and, in particular, for the weights of the query terms will certainly depend on these values.

We also did informal experiments with a logarithmic encoding of the data with the goal of trying to exploit the fact that differences in time and length are usually more crucial when their respective values are low. For example, it may matter more whether a document is one week or two weeks old than whether it is 101 or 102 weeks old. However, a few preliminary trials did not produce conclusive evidence that this in fact improves performance, and we decided to not further pursue this issue. Note that the new version of Melvil's user interface (figure 1) uses a discrete set of values that encode a logarithmic scale.

5 Prototype Implementation

For reasons discussed above (section 4.2), we decided to implement a nearest-neighbor algorithm at the core of our prototype. Overall, this seemed to be the most feasible and most flexible approach. We will review the basics of this algorithm in the next section 5.1. However, the particularities of this dataset—the presence of more than one dependent variable and the set-valued nature of several attributes (in particular of the three target variables—forced us to develop a version that conforms to these requirements. These modifications are described in subsequent sections.

5.1 The k Nearest Neighbor Algorithm

The k nearest neighbor (kNN) algorithm is one of the oldest and most popular algorithms that is currently used in machine learning. Early descriptions in the statistical literature go back to the 70's (Duda and Hart, 1973), a collection of reprints of articles from that period appeared as (Dasarathy, 1991). Within the Machine Learning community, it gained popularity with the rise of *Case-Based Reasoning* (CBR), the idea that problem solving is mostly based on retrieving and adapting past experiences (Riesbeck and Schank, 1989; Kolodner, 1992). Within the field of machine learning, the application of this framework resulted in the development of *instance-based learning* algorithms (Aha et al., 1991), which have quickly gained popularity among researchers and pracitioners. A collection of recent research and survey papers can be found in (Aha, 1997).

Instance-based learning algorithms are generalisations of the basic kNN algorithm to the propositional symbolic learning framework that is predominantly used in machine learning. In this framework, the program is given a database of past observations (described with a fixed number of measurements x_i, so-called *features* or *attributes*) and a designated *dependent* variable y. The goal is to find a mapping f that is able to compute the class value $y = f(x_1, \ldots, x_n)$ from the feature values of new, previously unseen observations. Both the features and the dependent variable can either be numerical or categorical (i.e., its values are unordered symbols). Depending

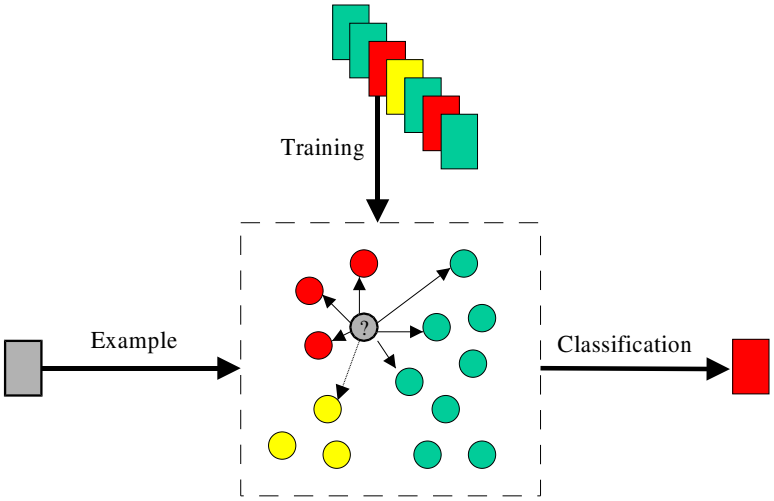

Figure 3: The nearest neighbor algorithm

on the nature of the dependent variable, one discerns *classification* (symbolic prediction) or *regression* (numeric prediction) problems.

The basic idea of nearest neighbor algorithms is straightforward (see figure 3): given a new example for which we need to predict the value of its dependent variable (represented by different color/shading), we compute the distance of the example to all previously encountered examples in the database. The k examples that are closest are selected for computing the prediction of the new example. k is a parameter of the algorithm. In the simplest case ($k = 1$) the value of the dependent variable of the nearest neighbor is directly used as the prediction of the dependent variable of the example to classify. If $k > 1$, the predictions of the nearest neighbors have to be combined, e.g., by taking a majority vote (for classification problems) or computing the average (for regression problems). In both approaches, the predictions of the neighbors are often weighted with the inverse of their distance to the example (or a similar function), so that nearer neighbors are given more importance than farther neighbors.

At the core of the nearest neighbor algorithm is the distance function, which determines which of the past experiences are considered closest to the new example. In the simplest case, the Euclidean distance is used, which is defined as

$$d(e_n, e_m) = \sqrt{\sum_i (x_{in} - x_{im})^2} \tag{9}$$

where e_n and e_m are two examples, and x_{in} is the value of the i-th attribute of the n-th example. As this distance is only defined for numerical attributes, it has to be generalized for the use with categorical attributes:

$$d(e_n, e_m) = \left(\sum_i d_i(x_{in}, x_{im})^o \right)^{\frac{1}{o}} \tag{10}$$

where d_i is a function that computes the distance between different values of the i-th attribute and o determines the order of the distance function (e.g., $o = 2$ corresponds to a Euclidean distance). The task of defining a distance function d for examples is now reduced to defining distance functions d_i for all attributes. The most common realisation is

$$d_i(x_{in}, x_{im}) = \left\{ \begin{array}{ll} \frac{|x_{in} - x_{im}|}{\max_j x_{ij} - \min_j x_{ij}} & \text{for numerical values} \\ \left.\begin{array}{ll} 0 & \text{if } x_{in} = x_{im} \\ 1 & \text{if } x_{in} \neq x_{im} \end{array}\right\} & \text{for categorical values} \end{array} \right. \tag{11}$$

It is necessary to scale the numerical distances so that $0 \leq d_i(x_{in}, x_{im}) \leq 1$ holds for all distances. This avoids the undesirable effect that attributes with large values (e.g., document length) may dominate the distance function.

Another common enhancement is the use of a *weighted* distance function:

$$d(e_n, e_m) = \left(\sum_i w_i \times d_i(x_{in}, x_{im})^o \right)^{\frac{1}{o}} \tag{12}$$

The weights w_i can either set by the user (in case she has some prior knowledge about the application domain) or determined automatically with a variety of techniques. In particular, *feature subset selection* (i.e., the removal of irrelevant or redundant attributes from the training examples) can be considered as a special case of these techniques, where the w_i are restricted to the values 0 or 1. For a survey of automated feature weighting techniques see (Wettschereck et al., 1997).

5.2 Distance Computation for Set-Valued Attribute

The core of a kNN algorithm is the distance computation. In principle, our implementation uses the distance function shown in (10) and (11) with $o = 2$ for computing similarities.[3] However, this cannot be directly applied to attribute 8, which consists of a *set* of numerical values, one for each

[3]Strictly speaking, we implemented a version of the weighted distance function shown in (12) but, in the prototype version, we did not incorporate any means for automatically setting these weights (all weights are set to 1.0, which reduces (12) to (10)). The main reason why we did not implement feature weighting is that at the moment we do not have a clear idea how this can be performed incrementally and for all tasks simultaneously (we use the same similarity computation for all predictions, a case of multi-task learning (Caruana, 1997)). We performed a few informal experiments for adjusting the weights with batch data for each individual user, but this did not yield any significant results. Clearly this is on our list for future work.

query term. As the maximum number of query terms is finite and known in advance, a straight-forward approach to handling this problem is to turn the set-valued attribute into a number of numerical attributes, one for each possible query term. If a query term appears in the original attribute, its associated value is copied as the value of the new attribute that corresponds to this query term. The values for all query terms that were not used are set to 0. Most of the query terms will be 0 in all examples, so that for each pair of examples most of the attribute distances will be 0 as well (in fact, one only needs to compute the distance for the union of the non-0 value in each pair).

However, this approach has some shortcomings: the sparsity of non-0 attribute values will be particularly bad when there are only a few training examples available, because we need a significant number of previous examples for each possible query term in order to have a fair chance of finding a neighbor that is reasonably close. To solve this problem we make use of the hierarchical ontology which is available for the query terms. In analogy to the technique describe in section 4.3, the similarity computation takes a parameter l with value 0 to ∞, which specifies the maximum depth of the hierarchy that will be used. The default value of the parameter is $l = 2$. The value could be incrementally enlarged as soon as the model has incorporated enough examples to use bigger parts of the hierarchy.

The parameter allows to reduce the number of derived attributes to the number of query terms that appear in depths $d \leq l$. In effect, this allows to compute attribute similarities for query terms that are different, but are mapped to a common ancestor at the ontology depth level. This enables the system—to some extent—to incorporate information about the similarity of query terms into its similarity function.

5.3 Multiple Target Attributes

In principle, the user profiler learns one user model for each user and for each ontology. The latter is necessary because the information contained in one ontology can in general not be mapped to a different ontology in a meaningful way (different ontologies contain different query terms). Thus, the program uses attributes 0 and 5 to decide which model has to be used. Each model consists of a case base of the examples of the previous experiences of that user with that particular ontology. Naturally, all values of attributes 0 and 5 are identical for one model and can be ignored.

Actually, each model consists of several models—at least one for each target variable (the set-valued targets need more than one model, see below). However, in the case of a kNN classifier, these models are all the same as long as the distance function and the training set is the same for all variables.[4]

[4]When switching to weighted distance functions (see previous footnote) it may turn out that the use of different weight sets for different target variables is beneficial. On the one

This is the case for the numerical target variables 16–19: the algorithm finds the k nearest neighbors in the full set of previous cases for that user and that ontology. These values of the variables 16–19 of these neighbors are then used to determine the predictions for the corresponding variables of the current example. The numeric prediction for each of these variables is computed by the formula

$$x_{in} = \frac{\sum_{j \in N} w_j \times x_{ij}}{\sum_{j \in N} w_j} \qquad \text{where} \quad w_j = \frac{1}{1 + d(x_{in}, x_{ij})} \tag{13}$$

N is the set of indices of the nearest neighbors and i is the index of the target attribute ($i \in 16 \ldots 19$). Hence, the prediction is the average of the prediction of the nearest neighbors, weighted by the inverse of the distance of the neighbor to the example to classify. To prevent division by 0, we simply add 1 to each distance, which makes the average a little smoother. As the distances do not necessarily add up to 1, we have to normalize the prediction by dividing it by the sum of the weights.

5.4 Set-Valued Target Attributes

Additional problems are caused by the set-valued nature of the target attributes 15, 20, and 21. Variables 20 and 21 can be handled with a technique similar to the one described above: we unfold the variable into a set of new attributes, one for each possible value of the variable. The new attributes are binary-valued, i.e., they specify whether the switch that corresponds to this value is turned on and off. For example, for each of the different servers that can possibly appear in the value set of feature 20, we construct a new attribute that specifies whether documents from this server should be filtered or not.

As described in the previous section, we have to learn a different classifier for each of these new attributes. However, contrary to the variables 16–19, in this case we cannot simply use the same nearest neighbors for predicting the value of the target attribute. The reason for this is simply that the documents might not contain a prediction for this target value. This may happen when no documents that are relevant for the query were found on one (or more) of the servers (attribute 20) or domains (attribute 21).

Thus, for predicting these variables, we must not consider the k nearest neighbors to the query, but the k nearest neighbors that actually predict a value for the sought target attribute. This is realized by sorting the examples in the current model by their distance to the example to classify and

hand, this would allow to tailor the weight setting for each target attribute independently, but on the other hand it would also cost run-time, because the nearest-neighbor search has to be performed for each attribute. As already mentioned, trying to optimize one set of weights that can be useful for all tasks is an—to our knowledge—unsolved case of multi-task learning (Caruana, 1997).

progressing the list in order until we have found k examples that predict a value for the variable in question. Of course, the sorted list of neighbors is the same that is used for computing the k nearest neigbors for the variables 16–19, which saves additional computation time.

The list of values for which a prediction is needed is only known at classification time, because it depends on the result of the query (we only need a filter setting for domains and servers that actually appear in the search result). These *request lists* have to be communicated to the classifier as additional arguments. This solution is unproblematic as long as there are not many possible values of the attribute (as in attribute 20). For attribute 21, the number of possible values (the possible domains) can be numerous, which may cause similar problems as discussed above for variable 8. In the prototype implementation we have ignored this problem because in the sample data we have received, this option was rarely used (i.e., only rarely a user chose to filter out documents based on their domain of origin). If the high number of resulting variables turns into a problem, we may have to find a solution similar to the handling of variable 8, i.e., using some sort of hierarchical decomposition of domain names to generalize the domains to groups and thus reduce the number of the variables.

5.5 Prediction of Concept Weights

The most complicated case is the prediction of variable 15. Here, we need to predict a weight for each query term. In principle, this could be handled in the same way the variables 20 and 21, i.e., learn a separate classifier for each query term (as discussed in the previous section). However, this treatment would neglect an important piece of information, namely the frequency of this query term in the search result, which is encoded in variable 8. Although we can compute similarities over this variable (see section 5.2), these similarities are often aggregating information (for low settings of the maximum ontology depth l). More importantly, however, we also loose the information which frequency is the one that corresponds to the query term for which a prediction is needed.

We solved this problem by dynamically adding a new input variable for each query term classifier. This attribute simply contains the frequency (as given in variable 8) of the query term for which we want to predict the weight setting. One can imagine this situation as turning each training example into q new training examples (q being the number of query terms), where all new training examples are identical except for the addition of the new attribute. The computation of the similarities of the new examples can be performed incrementally: We already have the distances to all examples given the original attributes. Hence, all that needs to be done is to add

the attribute distance of the new attribute to each of the old distances and re-sort the examples according to the resulting value.[5]

Also note that it does not make sense to compute the similarities between frequencies of different query terms (at least not always, see below). Consequently, only past cases where exactly the same query term was present are considered. Once more, this may cause the problem that too few past cases that satisfy this constraint are found. Hence we introduce a second ontology depth parameter that controls the similarity computation for this variable by specifying the ontology depth below which all entries are considered equal. While a setting of ∞ will only consider previous examples that contain exactly the same query term as candidates for the similarity computation, lower settings will enlarge the set of candidates. A setting of l, for example, will consider all examples as candidates that contain a query term whose lowest common ancestor with the current query term can be found at an ontology depth of l or higher. If an example contains multiple such terms, each one of them is considered. Thus, contrary to the computation of the matching of variable 8 (section 5.2), changing this parameter does not change the number of attributes, but changes the number of examples that are effectively been considered as candidates for the nearest neighbor computation of a particular query term.

5.6 Relative Encoding of Target Attributes

The value range of the variables 16–19 is determined by attributes 9, 10, 12, and 13. For example, the maximum threshold for the document length (19) cannot be set to a value that is smaller than the maximum document length that is actually encountered in the search result (which is encoded in attribute 13). However, there is no guarantee that the document lengths in the search results of the nearest neighbors conform to that bound. It may well be the case that the value of variable 19 in all nearest neighbors is higher than the value of attribute 13 of the example to classify. Hence, the prediction for variable 19 may be out of range. Usually, this will not cause a problem because specifying too large a threshold will simply leave all documents in, but we felt that this solution is not quite clean.

Our alternative approach is to allow an option which scales these variables in a relative way. If the the user leaves the minimum threshold for document length unchanged from the minimum encountered in the search result (12), we encode a value of 0.0 for variable 18. Likewise, the maximum (19) is encoded as 1.0 if it is not changed from the maximum value

[5] The actual implementation is a bit more complicated and more efficient. It performs a cut-off in the sorted list of neighbors when the value of the old distance is already bigger than the value of the k-th neighbor of the new distances. A similar mechanism is used for values of $o > 1$ (equation (10)).

encountered in the search result. The same situation applies to the variables concerning the last modified date.

We also believe that this encoding is intuitively more understandable: in cases where the user decides not to filter according to the document length, i.e., the absolute values of variables 18 and 19 are identical to the values of the input variables 12 and 13, the relative encoding of these variables would be simply 0.0 and 1.0 respectively. While the absolute values are quite likely to be different for different search results, their relative encoding is identical. This makes the structure of the data more visible and facilitates learning.

5.7 Default Model

In several places, we need a default prediction for the variables. For example, we have to rely on a default prediction when no previous experience is available. This is the case when a new model is started, i.e., when a new user is entered or a known user switches to a new ontology. Another case for a default model is when the prediction for a previously unseen value for the variables 15, 20, or 21 is requested.

The current set of default predictions opts for minimal filtering, i.e., it sets the values in a way that the user sees all items of the search result. The maximum and minimum thresholds for the document length and the last modified date are set to the corresponding values in the input, which are their natural boundaries. All switches in the variables 20 and 21 are turned on, and all server weights are (arbitrarily) set to 0.

As mentioned above (section 3.3), we also used this default model as a benchmark for comparing the performance of the learners. In addition to providing a fair comparison between the filter settings with and without learning, we felt that the prediction of the mean is not a fair comparison because it does not take into account that the range of values for variables 16–19 is limited by the respective minimum and maximum variables specified in the input variables 9–14. Besides, we cannot compute mean values for categorical variables 20 and 21, and would have had to use a different evaluation measure for these cases.

We also kept the default model as a separate module so that it can be easily changed in later versions. For example, one could set the values to averages over all data from all users (but from the same ontology) or try to find the nearest neighbor in all data. This would implement a form of *generic* user modelling. One can even think of dynamically combining specific and generic user models for deriving predictions (e.g., by starting to use a generic model and successively shifting the weight to a user-specific model, which can take over entirely as soon as enough data for a reliable performance of that model have been collected).

5.8 Implementation details

The user profiling prototype is implemented in `perl`. Its functionality provided platform independently and transparently on the internet through a client/server architecture written in Perl. Clients access the kNN service(s) through TCP/IP sockets. The server listens on a TCP/IP port and deals with requests issued by the clients. A typical interaction starts with a CLASSIFY request by the client, followed by a transmission of variables 0 – 14 of the data specification (Table 1). In addition, the client has to communicate which of the sever filters (variable 20) and domain filters (variable 21) are visible in the display, i.e., for which filter variables the user profiler is expected to predict a setting. As a reply, the servers sends back the predictions for variables 15 - 21. New examples can be incorporated via a LEARN request, followed by a complete set of variables 0 - 21.

A process communication module makes the kNN algorithm available to possibly many clients simultaneously without unintended interference in the data base of learned examples. In the presence of multiple simultaneous clients, reading and writing of these files is subject to locking to prevent inconsistent views of the data in different processes. On top of the locking scheme we need a mechanism to signal the modification of the example data base to other processes which may hold cached previous versions. In principle this could be done via the modification times of the files as provided by the file system. However, this solution is plagued by the low time resolution of one second and other potential latency problems caused in particular by file systems mounted via NFS. Our solution is to have each process touch a file `user.ontology.recent.`p, where p is the number of the active process, to signal that it just brought itself up to date. Any process invalidating the cache(s) by writing to `user.onto` unlinks (deletes) all but its own `user.ontology.recent.`p files. Thus any process can check the recency of its cache by verifying the existence of its `recent` file.

6 Evaluation of the prototype

For evaluating the prototype system, we mainly relied on the 154 example data set `rt2` (table 2). We performed a few trials with other datasets as well, but, as will become apparent from the learning curves presented below, 50 examples are typically not enough for a conclusive evaluation. Note that the fact that we only used one ontology (BMWA) for testing should not have a major influence. The program is unable to capture the semantics of an ontology: it only sees a tree structure of symbols. This property is shared among all ontologies. Thus an approach that works for one ontology, should also work for different ontologies (although, of course, learning has to occur separately for different ontologies).

6.1 Testing Strategy

As a testing strategy, we decided to adopt a real-world scenario: each experiment starts from scratch. The training examples are processed on-line, i.e. the learner receives them one by one (not in a single batch as is commonly the case in data mining). Each example is classified by the current model, i.e., using all examples that have been seen so far: example e_i may use all examples $e_1 ... e_{i-1}$ for deriving its prediction. The first example e_1 is classified by the default model (section 5.7). Thus, as more and more examples come in, performance should improve because more and more information is available for the learner. We visualize this by plotting performance measures (section 3.3) over the number of examples.

6.2 Prediction of real-valued targets

Figure 4 shows several error graphs for variable 16. The first graph (upper left) shows the raw error rates plotted over the number of examples classified. It can be clearly seen, that the size of the errors as well as their frequency are decreasing. This is particularly obvious when one compares this graph to the one on the right, which shows the error rates of the default model that does not filter anything (cf. section 5.7). In this graph, errors occur much more frequently, and their size does not exhibit a decreasing tendency. Thus, from looking at these two graphs, one can already infer that learning occurs in the left one, while it does not occur in the right one.

This becomes clearer if we look at the two graphs in the second row. They show the mean error and the mean error of the default model plotted over the training set size. They may simply be viewed as graphs that plot the average of all values $\leq x$ in the graph above as the value of x. One can immediately see that the left curve has a clearly decreasing trend, while the right one is increasing in the beginning but stabilizes then. The left graph shows learning, while the right one simply exhibits fluctuations of the mean value which originate from the inherent instability of statistical estimates from only a few values. With increasing sample sizes, the estimate converges towards a stable value, and this value is one order of magnitude above the one of the graph to the left (also note the different scales of the y-axes of the two graphs!).

Recall that the values that are averaged are computed incrementally, i.e., after predicting the dependent variables for a new example, the error measures are adapted to incorporate the new results, and the example is subsequently added to the training set. Thus the prediction for the next example is made with more information. Although these estimates become more stable over time in both graphs, the influence of individual examples is still visible. For example, the relatively big error that occurred around example 100 (cf. the upper left graph) is clearly visible in the graph below.

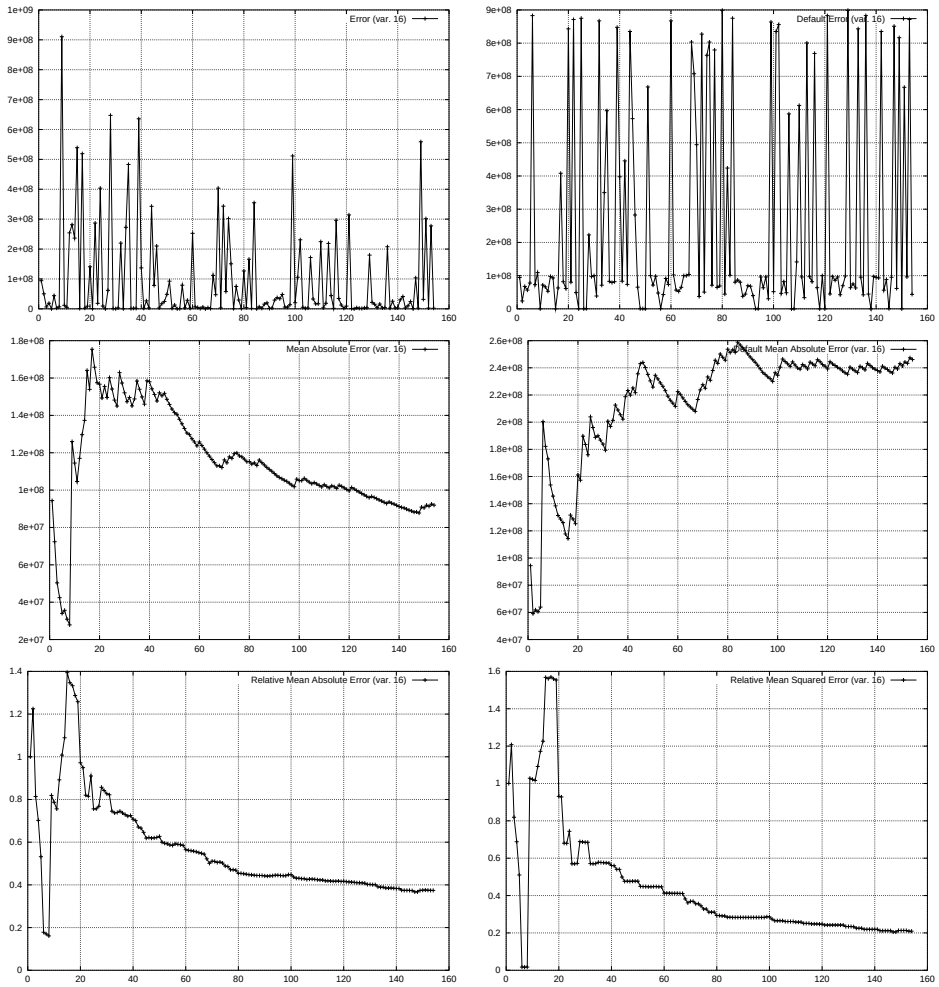

Figure 4: Error curves for variable 16 (min threshold of last modification).

So far we always had to look at two graphs in order to get an idea on how the learner performs with respect to the default model. The two graphs in the bottom row show the ratio of the mean (squared) error of the learner and the default model. If this ratio is > 1, the default model performs better than the learner (which is bad), while a ration < 1 indicates a good performance by the learner. Both graphs (mean error ratios on the left, mean squared error ratios on the right) soon converge towards values below 1.0. The effect of the ratio can also be seen at the above-mentioned local peaks (such as the one near 100): these are not visible in this graph, because the default error at this place is also considerably high. In fact, performance improves near 100, because some of the errors that the default model makes in this neighborhood are not made by the learner.

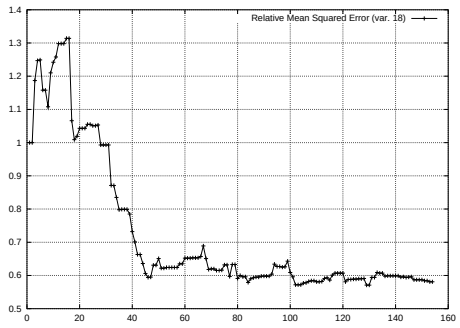

Figure 5: RMSE curve for variable 18.

Also visible is a clear difference between the mean error estimates on the left and the squared mean error estimates on the right: the squared estimates converge towards a much lower value than the absolute error estimates, meaning that the variance of the learner's errors is considerably smaller than those of the default model, i.e., the mistakes of the learner deviate considerably less from the correct value.

It should be noted that after about 50 examples (which would be the training set sizes for all other users (table 2)), the performance is still in a relatively steep descent, i.e., adding additional examples to the user model may still cause significant performance improvements. After the about 150 examples of this user (`rt2`), the performance increase has flattened considerably. Small improvements can still be expected if the curve is extended to the right (i.e., more interactions of this user with this ontology are observed) but the number of additional examples needed for such an increase will increase considerably. The performance is probably already close to the peak performance that can be expected with the chosen learning architecture.

The situation for predicting variable 18 is quite similar. Figure 5 shows the relative mean squared error curves for this variable. Again, the learner clearly exhibits a learning behavior, although in this case, the performance limit is somewhat below the one for variable 16 (0.6 instead of 0.2 for the $rmse$).

Figure 6 shows the errors committed by the learner and the default model for variable 17. We again see a clear decrease in the frequency and amplitude of the errors for the learner, which indicates a good learning performance. However, in comparison to the default model, the learner appears to be quite bad. In fact, the average performance after all interactions (not shown) is around 2.3, which shows that the performance of the default model is more than two times more accurate than the performance of the learner. While there is some truth to that, a closer inspection of the other curves reveals the reason for this: the user hardly ever changed the default settings. In

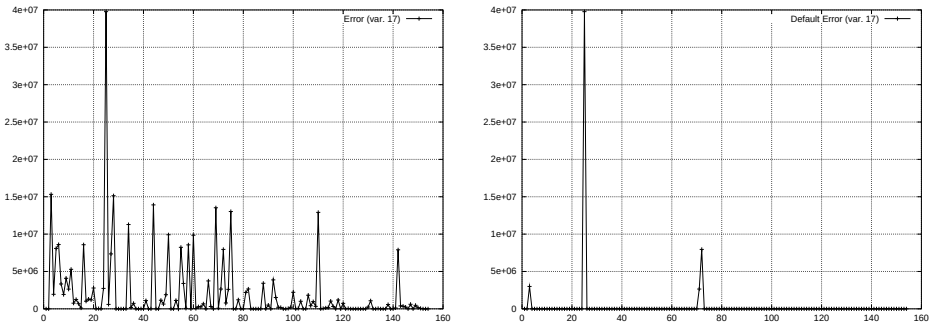

Figure 6: Absolute error of learner and default model for variable 17.

fact, the default model, which always predicts that all the filter settings are left unchanged from their defaults, commits only 4 mistakes on the entire set of 154 examples! (This can be clearly seen from the three peaks on the right curve; note that one peak consists of two non-zero values.) Clearly, such a model is very hard to beat. For variable 19, the situation was similar (only 14 mistakes by the default model).

One the one hand, these results reflect the fact that - in this set of experiments - we did not use a relative coding for the target variables for the learner, while it is implicitly captured by the default model. We discuss this in more depth in section 6.3. On the other hand, this result raises the question whether such a behavior is typical for users or not. If users tend to leave certain parameters unchanged, the learner should provide means for a faster convergence in such cases. For example, a simple classification scheme could be implemented that monitors the performance of both the default model and the learned user model and classifies incoming examples with the model that has worked best in the past. One could even imagine to implement several different default "personalities", and constantly compare the performance to a learned model and choose the best. We are currently working on this issue.

6.3 Relative Encoding

As discussed in section 5.6, we also implemented an option that allows to use a relative encoding of the numerical variables 16–19. In the raw data, these encode the absolute value of thresholds that are used for filtering out documents according to their minimum or maximum criteria for length or recency. A relative encoding restricts these four variables to the range [0..1], where a value of 0 means that the threshold is identical to the minimum value that occurred in the search result, while a value of 1 means that the threshold is identical to the maximum value. Values inbetween are linearly interpolated. This encoding is completely transparent to the user because

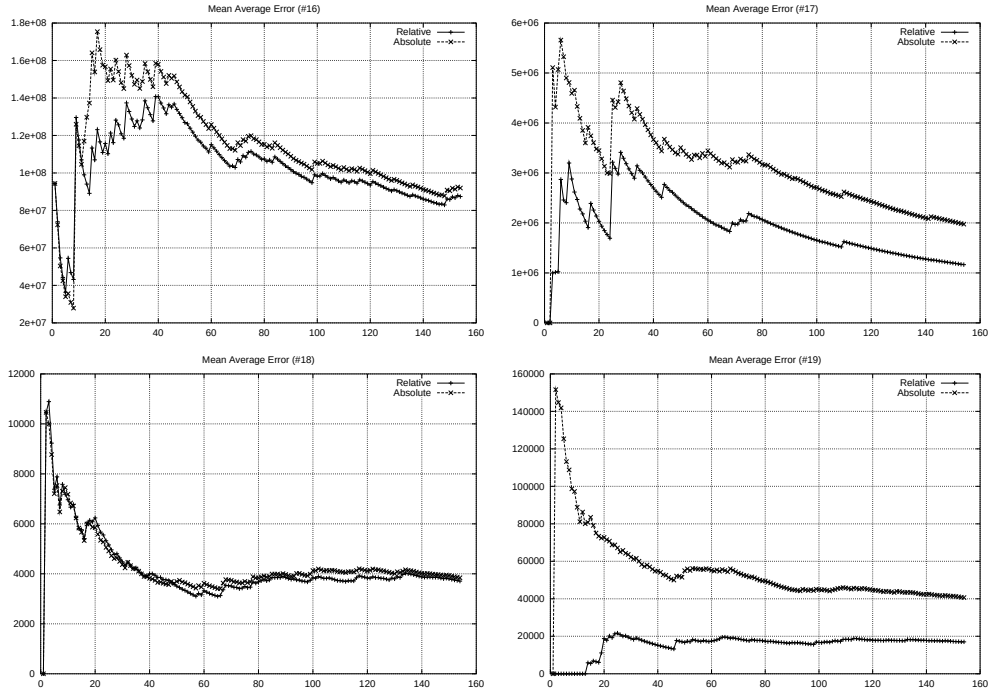

Figure 7: The influence of absolute vs. relative encoding of variables 16–19 on the average error. x-curves denote absolute encoding, while +-curves denote relative encoding.

the examples are always stored in the absolute format and the predicted values are mapped back to their respective absolute scale. However, the internal re-coding has the effect that predicting the minimum (maximum) of the search result is aequivalent to predicting the value 0 (1), which can be expected to be easier than the prediction of the exact absolute value.

The results in figure 7 show that this is indeed the case. In all graphs, the relative curves are below the absolute curves, signaling that the relative encoding does indeed provide better results. This difference is particularly clear for the maximum-threshold variables 17 and 19, where the relative error is almost cut in half. In part, this effect is due to a considerably better performance of the relative encoding for cases where we have only few examples (as we expected in section 6.2).

However, part of the improvement could also be due to the different nature of these variables. For example, the minimum modified date variable (16, upper left) shows only a very small gain for the relative encoding. The reason is simply that recency has a mean value close to today. Hence the linear transformation that is performed by the relative encoding maps values close to 0 to values close by 0, and the performance difference is only minor. However, the maximum threshold for the last modified date can become ar-

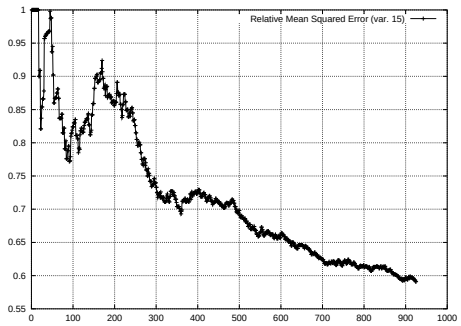

Figure 8: Relative mean squared error curve for variable 15.

bitrarily large, so that the relative encoding captures important information. Hence its performance is clearly better than an absolute encoding, which is witnessed by the lower error rates for the maximum-threshold variables 17 and 19.

6.4 Prediction of set-valued targets

As discussed in section 5.4, variables 15, 20, and 21 differ from variables 16–19 in that they are set-valued, i.e., each of them can contain an arbitrary number of items for which a prediction is required. Variables 15 requires a separate (numeric) prediction for the weight of each term that occurred in the user's query. Variables 20 and 21 are used for filtering out which servers or domains the user wants to see (binary predictions). Consequently, for each example, multiple predictions have to be made. We chose to plot these results by simply concatenating all predictions, which means that we have many more predictions than examples.

Figure 8 shows the relative mean squared error curves over the 925 predictions that were made for the concept weights (variable 15) of the 154 examples in the set `rt2`. The results are quite satisfactory: there is a clear improvement over time, and the final classifier is considerably better than the default model. This is despite the fact that the individual concepts have a fairly low frequency (see section 3.1). Thus the generalization within the ontology that is implicitly performed by the classifier seems to work.

With variable 20 (server filters), we had a simlar problem as with variables 17 and 19. In only two out of 2698 server filter settings, the user decided to filter out a server. In all other cases, the server was left on (not filtered). However, in this case, the learner's performance was considerably better than with the real-valued prediction of variables 17 and 19. The learner made three mistakes in the beginning: naturally, it did not recognize the first occurrence of a filtered server because at that point it had not yet encountered a case where this had happened. However, after it had seen

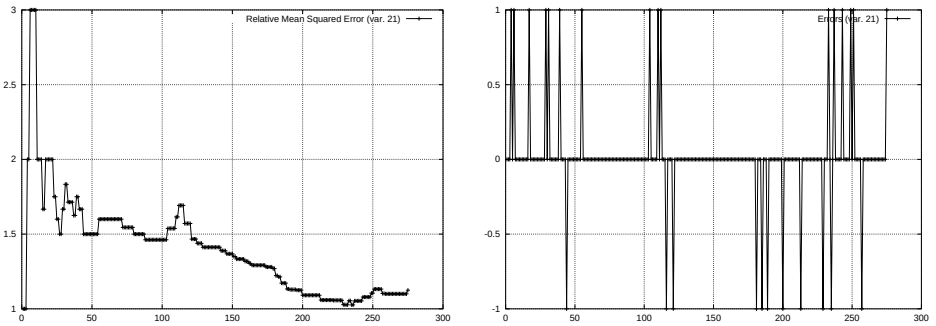

Figure 9: Error curves for variable 21. The lower curves shows the errors that are commited only by the learner (pointing upwards) and only by the default model (pointing downwards).

the first occurrence in interaction 91, it was subsequently over-estimating the probability of filtering and was erroneously predicting switches to be off for predictions 119 and 214. It then recognized that these events practically never occur, and, consequently missed the next (and final) occurrence at prediction 1543. In total, it made 4 mistakes, as opposed to 2 mistakes that were made by the default model (always predict on). We interpret this as evidence that in a classification setting, the nearest-neighbor algorithm adapts much faster towards a consistent behavior of the user than in the regression setting that we used for variables 16–19. Note that in the latest version of Melvil, these variables are also encoded as categorical (see figure 1), which lets us hope for additional performance improvements once these changes are reflected in the learning architecture (this is subject to current work).

Finally, the relative error curve for variable 21 (domain filters) is shown in the left half of Figure 9. For this variable, the user has changed the settings about 15% of time (in 15% of all interactions with domain filter settings they were turned off). The learner produces a classifier that misclassifies about 15% of the data, i.e., its performance is no better than that of the default model which always predicts that the filter settings are left unchanged. This is somewhat disappointing and we are not sure whether this indicates that the learning architecture is inadequate or whether there simply is no relation between the input data and the dependent domain filter settings. In the right half of Figure 9 we plotted the error difference between the learner and the default model. This gives us an upward pointing peak in cases where only the learner is wrong, and a downward pointing peak in cases where only the default model is wrong. Cases where both models produce the same prediction (either both are right or both are wrong) are plotted as 0. The interesting result is that the learner makes most of its unique mistakes in the beginning, while towards the end, there is a phase where the default

	k						
target	1	3	5	7	10	15	20
15	0.725	0.591	0.573	0.571	0.570	**0.567**	**0.567**
16	0.328	0.209	0.192	0.188	**0.185**	0.190	0.197
17	4.060	2.297	1.983	**1.981**	2.013	2.191	2.391
18	0.868	**0.581**	0.603	0.599	0.589	0.609	0.621
19	4.356	**3.112**	3.537	3.657	3.860	4.128	4.211
20	3.000	2.000	**1.500**	**1.500**	**1.500**	**1.500**	**1.500**
21	1.325	**1.125**	1.250	1.250	1.300	1.175	**1.125**

Table 6: Average cumulative RMSE after predicting all 154 examples.

model makes considerably more mistakes than the learner. Interestingly, however, there comes a short period at the very end where the learner again performs considerably worse than the default model, which can be seen from the upward pointing peaks in the right graph and the rise of the error curve in the left graph. It is hard to say whether this behavior is simply a random deviation or whether these examples near the end are systematically different from the ones before, which would not allow the learner to predict them correctly.

6.5 Number of Neighbors

As we already discussed in section 4, it can be expected that the number of neighbors that are considered for predicting the value of a target variable has a considerable influence on the quality of the results. The use of only one neighbor brings the danger that this neighbor is noisy, erroneous, or simply not chosen well. The use of more examples allows—in theory—to correct for badly chosen nearest neighbors, because the next-to-nearest neighbors may unanimously over-rule the prediction of the nearest neighbor (or, in the case of a numeric target attribute, correct the prediction by adding their votes to the weighted sum). However, the use of too many neigbors may result in too general predictions, as can be seen from the limiting case where all the examples are used. In this case, the predictions will be quite similar for all examples and the predictive performance will typically decrease.[6]

Table 6 shows some results for different settings of k, i.e., for different numbers of neighbors. The numbers are the relative mean squared error (rmse) after seeing all 154 examples (i.e., the end points of the error curves). In each line, the best performing setting of k is emphasized in bold face. The

[6]Note, however, that not all examples are classified identically if we are using weights that are proportional to the distances between examples. The reason is simply that each example has a different distance to the examples in the training set, and hence the predictions of the training examples will be weighted differently.

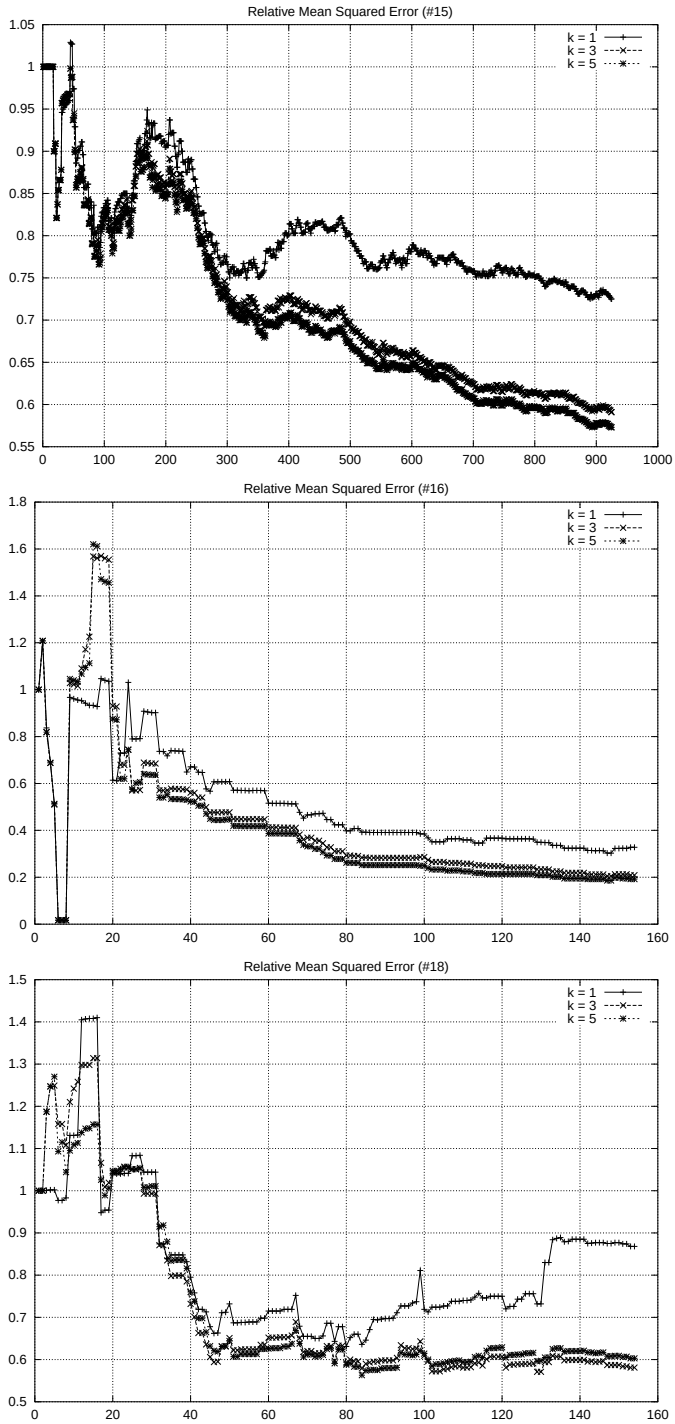

Figure 10: Learning curves (rmse) for different values of k ($k = 1, 3, 5$) for variables 15, 16, and 18.

default setting, $k = 3$, seems to perform reaonably well over all sets, but for some variables much higher settings are more appropriate. However, the differences between settings of $k \geq 3$ are only minor compared to the differences between $k = 1$ and the others. Using only a single neighbor is uniformly the worst approach of all.

Figure 10 shows three learning curves for the rmse of variables 15, 16, and 18. Each graph contains three learning curves, for $k = 1$, $k = 3$, and $k = 5$ respectively. The graphs confirm that the difference between $k = 1$ and other settings is very clear. They also show that this difference is particularly pronounced near the end of the learning curve, when a sufficiently large number of examples is already present in the training set. Then it starts to make sense to use more neighbors for classification. In the beginning, the approaches seem to be more or less equivalent, with frequently changing "leaders".

There are at least two possible explanations for this:

- The data is noisy. User behavior is at times inconsistent: focussing on the values provided by one particular neighbor (as in the case $k = 1$) is probably too crude. Using more than one neighbor is able to correct outliers and has a higher resilience against noise in the data (Aha, 1992).

- This effect can only kick in if there is a sufficient number of data points in the training set. In the beginning, the use of many neighbors will typically result in using almost all of the examples in the training set, thereby paying too much attention to distant examples.

The current implementation is only able to use a uniform setting for all variables (the default setting is $k = 3$). The above results demonstrate that it might be worth-while to use different values of k for different variables. (The same argument may also apply to other parameters of the implementation, like the ontology depth, see section 4.3.)

7 Conclusions

uma and ÖFAI are currently collaborating with two organisations to adapt the prototype implementation of Melvil into a marketable product in their respective area of expertise. The results of the study reported in this paper have shown us where the main efforts with respect to improving its user profiling component should go. We determined that a nearest-neighbor algorithm is the most natural and most flexible choice for this problem, and investigated several design options—including different ontology depths, relative vs. absolute encodings of prediction variables, and a varying number of neighbors—and determined suitable settings for these parameters. We

believe that it will be particularly important to focus on the role of the ontology. Our current implementation allows to focus on the upper levels of an ontology in order to improve learning behavior. However, we currently use the same setting for all interactions. It can be exptected that a gradual shift that allows to incorporate information from deeper levels of the ontologies may be beneficial once there are enough user interactions available.

We also realized that for all variables, the learner makes considerably more mistakes towards the beginning than toward the end. This is natural because in the beginning the learner has much less information about the user's behavior than towards the end. It cannot be prevented that a model that is learned from only one or two examples is necessarily incorrect because the generalizations that it makes from these few interactions cannot be based on a sufficient amount of data. However, it might be an option to consider alternative techniques like using a default model (or multiple default "personalities") for classifying the first few examples and passing control over to the learned user model as soon as the latter proves to be more reliable. Alternatively, one could also consider to use the interactions stored in the models of other users as a first guess for the behavior of a new user, while, as soon as there are more examples for this user coming in, the control may gradually shift towards giving more weight to the predictions of the user's own model.

In addition to the findings of this study, we hope that our current collaboration with those two organisations will provide us with valuable feedback with respect to the current architecture. In particular, we plan a redesign of the input variables. These define the characteristics of the query and the search results and should ideally capture all factors that might influence a user's behavior. Our current set seems to provide reasonable results but there is certainly room for improvement. Finally, this collaboration will provide us with the opportunity to evaluate fielded implementations of Melvil on real interaction data, which will be the ultimate test for our design choices.

Acknowledgements

This research was supported by the Austrian *Forschungsförderungsfonds für die gewerbliche Wirtschaft (FFF)* under grant no. ZL.802702/6522k/Sa. The current development of the software is supported by an European IST programme as trial project no. IST-2000-29583. The Austrian Research Institute for Artificial Intelligence is supported by the Austrian Federal Ministry of Education, Science and Culture.

References

D. W. Aha. Tolerating noisey, irrelevant, and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies*, 36(2):267–287, 1992.

D. W. Aha, editor. *Lazy Learning*. Kluwer Academic Publishers, Dordrecht, 1997. ISBN 0-7923-4584-3. Reprinted from *Artificial Intelligence Review*, 11:1-5.

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

C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.

L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth & Brooks, Pacific Grove, CA, 1984.

R. Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.

M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence*, 118(1-2):69–114, 2000.

B. V. Dasarathy, editor. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, CA, 1991.

R. O. Duda and P. E. Hart. *Pattern Recognition and Scene Analysis*. John Wiley and Sons, 1973.

D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, Berlin, 2001.

D. Kibler, D. W. Aha, and M. K. Albert. Instance-based prediction of real-valued attributes. *Computational Intelligence*, 5:51–57, 1989.

J. L. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1992.

A. Maedche, C. Nédellec, S. Staab, and E. Hovy, editors. *Proceedings of the 2nd Workshop on Ontology Learning (OL-2001)*, volume 38 of *CEUR Workshop Proceedings*, Seattle, WA, 2001. IJCAI-01. URL `http://ceur-ws.org/Vol-38/`.

A. Maedche and S. Staab. Learning ontologies for the semantic web. *IEEE Intelligent Systems*, 16(2), 2001.

D. Mladenić. Turning Yahoo into an automatic web-page classifier. In H. Prade, editor, *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 473–474, Brighton, U.K., 1998. Wiley.

J. R. Quinlan. Learning with continuous classes. In N. Adams and L. Sterling, editors, *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, Hobart, Tasmania, 1992. World Scientific.

C. K. Riesbeck and R. C. Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.

S. Staab and A. Maedche. Knowledge portals — ontologies at work. *AI Magazine*, 21(2):63–75, Summer 2001.

S. Staab, A. Maedche, C. Néellec, and P. Wiemer-Hastings, editors. *Proceedings of the 1st Workshop on Ontology Learning (OL-2000)*, volume 31 of *CEUR Workshop Proceedings*, Berlin, 2000. ECAI-00. URL `http://ceur-ws.org/Vol-31/`.

D. Wettschereck, D. W. Aha, and T. Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11(1–5):273–314, 1997.

I. H. Witten and E. Frank. *Data Mining — Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann Publishers, 2000.