Adaplix: Towards Adaptive Websites

Nico Jacobs

K.U.Leuven
Dept. of Computer Science
Celestijnenlaan 200 A
B-3001 Heverlee, Belgium
Nico.Jacobs@cs.kuleuven.ac.be

Abstract

Adaptive webpages help the user in finding relevant information by tailoring their content and lay-out specific to the visiting user. In this paper we describe Adaplix, a system we implemented to extend HTML by introducing conditional statements. We describe how this system can be extended with an inductive logic programming component to learn the user's browsing preferences. We also report on some preliminary experiments with this learning component.

1 Introduction

More and more people go on the Internet to look for information. These often unexperienced users have difficulties to find the information they're looking for. There are several techniques that try to alleviate this problem. Search engines (e.g. Altavista) try to index as many webpages as possible. This approach suffers from the fact that a query often results in too many suggested webpages of which many are irrelevant to the subject. Portal sites (e.g. Yahoo!) classify sites in a hierarchical directory of subjects, which solves the problems of the previous approach at the cost of creating and maintaining such a directory.

Once an interesting website has been found, the user still has to navigate through the site to find the information she's looking for. Adaptive webpages can help to solve this problem. The idea behind adaptive webpages is to let a page adapt itself to the user. The techniques mentioned in the previous paragraph try to help the user on a global level (starting from nothing but some keywords, return relevant webpages). Adaptive webpages however help the user on a much more local level: given that the user wants to see this page, how can the information on that page be changed (content as well as lay-out) so that it is as interesting for this user as possible.

In this paper we first introduce the Adaplix system (section 2) which allows the author of a website to use conditional statements in the webpages. In the next step (section 3) we extend this system with a component which learns the browsing preferences of users using the Tilde [3] system and we discuss (section 4) some experiments. In section 5 we briefly compare this approach with other systems and conclude.

2 Adaplix

In the field of adaptive hypermedia there are two major approaches. One consists of letting the author of a document add metadata to the document which describes how the document should be altered based on a model of the user (e.g. don't show this link unless the user is already familiar with concept C). The clear advantage of this approach is that the author can use all his domain knowledge to make the document as useful as possible. Maintaining the user model however is not an easy task. Consider the example above, how do we know the user is familiar with concept C? Usually the author of a document has to provide information describing how the user model should be changed (e.g. when the user is already familiar with concept B and has consulted document A, then the user is familiar with concept C). [5] has more information on this approach. An other approach (used in systems such as WebWatcher [8]) is to use machine learning techniques to learn the interest of a user and then automatically alter each document based on these interests (e.g. highlighting interesting links). The advantage of this approach is that there is no need for an expert to manually denote all the possible changes that can be made when presenting a document, which is particularly interesting when the content of the document changes very often (e.g. a website with news articles). However these systems are only capable of small adaptations of the presented information such as highlighting information or adding links to similar documents.

We developed Adaplix, a prototype which combines both approaches. The author of a webpage can add extra information about how a page can be adapted to the user. Adaplix also tries to learn the browsing preference of the user. In this way, the author can create pages with constructions such as "if the user would be interested in page A but is not interested in page B then show link C" where the learning component decides upon the interest of the user in pages A and B based upon the browsing behaviour of that user in the past. In this section we describe how Adaplix allows the author to alter pages, whereas the learning component is described in the next section.

Information on the World Wide Web (WWW) is offered by http servers. A browser (client) can contact the server to retrieve information. The server will then look up the requested webpage and send it to the client which displays this information. In order to obtain adaptive webpages, one has to alter this scheme in that the static information stored on the server has to be altered before it is presented to the user. The information can be altered either in the server or in the client. Altering the page at the server side is easier to implement and maintain because all programs only need to run on one machine (no need for platform independent implementation plus the possibility for code optimalisation) and all data is stored on one machine. Moreover, the adaptive webpages can be viewed

```
<!-- #adaplang {if (recstring ''BirthYear'') > 1990
  then print ''<BODY BACKGROUND="colordots.gif"''
  else print ''<BODY BACKGROUND="greyrelief.gif"''}>
<P><FONT SIZE=+1>W</FONT>elcome to my homepage.
```

Figure 1: Example of HTML code with Adaplix directive

with any browser. However, it's difficult to scale up this approach to a large group of users. If pages get altered on the client side, it's easier to scale up. Also the user profile resides on the user's computer which is beneficial from a privacy point of view. The main problem however is in writing the Adaplix-agent such that it can run on many computer platforms and browsers.

Because of this problem we implemented Adaplix as a program which resides on the server side. Because we wanted Adaplix to be server independent, we wrote it as a set of common gateway interface (CGI)-scripts in Perl [14]. When the user requests a webpage, this request is redirected to the CGI-script. This script requests the page from the server. Based upon the user model and the directions of the author of the webpage the page is altered and the resulting page is returned to the user[1].

In order to be able to adapt a page to the user's interest Adaplix needs information about the user as well as directions about how to change the page. The information Adaplix collects about the user comes from two sources. When a user first logs in into the Adaplix system, the systems asks some questions (minimal an identifier and a password but the website designer can extend this list of requested information). Furthermore each time the user visits a page on the website, Adaplix logs the URL and the time the page was visited.

This information can be used by the author of the webpage to indicate how the webpage can be adapted to the user. This can be done by writing condition HTML statements: if the user satisfies certain conditions use HTML code H_t else use HTML code H_f. Because HTML code can specify lay-out as well as content of a page, this simple schema is also very flexible: the author can use it to change the background, the font size, the text and/or pictures displayed etc. Tests can be based on the information the user provides during the first log in as well as on information extracted from the logs (has the user already visited page X more than N times, has the user visited page Y before date D,...). Figure 1 shows an example of a piece of HTML code with Adaplix conditional tests. It tells Adaplix to use colordots.gif as background file if the user is born after 1990, greyrelief.gif otherwise.

3 Learning Browsing Preferences

The Adaplix system logs all pages visited by a user together with the time of visit. Based on this information we can calculate how long a user visited a page. We use this as an indication of the interest the user has for this page. If we can identify a

[1] all links in the page are replaced by links to the CGI-script

correlation between attributes of a page and the duration of the visit of a certain user to that page, we can use this information to predict how long the user will visit an yet unvisited page (and hence — if our hypothesis that duration of visit is an indication for interest holds — we can predict the interest of a user in an yet unvisited page).

So the task that remains is identifying correlations between webpages and the duration of the visit to each of these pages. This can be seen as an inductive learning task. Each visit to a webpage can be seen as one example, and the task is to induce from a set of examples one general hypothesis predicting the duration of a visit to a page based upon the attributes of that page. Since we're only interested in a rough approximation of the duration of the visit, we can discretise the duration values into three classes. In this way we transform the browsing preferences learning task into the task of inductively building a classifier.

The input of the system consists of classified examples. Each example consists of the following information:

- title: each word occurring in the title of the document
- counts: for each page we count the number of images, level 1, 2, 3 and 4 headers, frames, http-links, ftp-links, email-links and the global number of words occurring on that page
- words: the system keeps a vector of frequently occurring words [2]. Each example contains a list of words from this vector that occur in the document, together with their frequency
- class: the discretised duration of the visit to the page
- previous page: link to the information about the page visited before the current page

A link to the information of the previous page is included because the duration of a visit to a page can be dependent on previous pages visited, e.g. a user who doesn't like pages with lots of links on it, unless it is a page visited right after a page which had the word "homepage" in the title.

We use inductive logic programming (ILP) [10, 6] as a learning methodology because this is a very flexible technique which is able to learn hypothesis which take into account relations between objects (such as the relation between pages visited in one session, words in one document, ...). ILP systems are also able to use 'background information'[3] in the learning process. By this we can for example use a lexicon (such as WordNet [9]) in the learning process.

In the experiments the ILP-system Tilde [3] was used because it is very efficient in constructing classifiers in the ILP setting. Moreover, experiments show that it can handle large datasets [4]. Tilde builds a first order equivalent of a decision tree. Figure 2 shows an example of a decision tree built by Tilde. If the page contains ftp-links the duration of the visit will be medium. Else it depends on

[2] stop words like 'the', 'a', ... are left out

[3] general information which holds for all examples

```
mid(A)
ftprefcnt(A,B) , B < 1 ?
+--yes: wordcnt(A,C) , C < 56 ?
|       +--yes: short_visit
|       +--no:  long_visit
+--no: medium_visit
```

Figure 2: Part of a decision tree produced by Tilde

the number of words in the document: with less than 56 words the prediction is a short visit, else a long visit.

4 Experimental Results

The system described in the previous section was implemented and used to analyze the visits to an experimental site. This site contains 181 pages on nine different subjects. A total of 69 users visited this site, resulting in 1496 visits to pages. The duration was discretised in three intervals (short, medium and long visit). This data was used in the following experiments. All reported testing accuracies are obtained from a six-fold cross-validation.

A first problem to solve is the discretisation of the numeric values (number of images, headers, links, ...). Tilde is able to discretise automatically. However one has to specify the number of intervals by hand. With three intervals Tilde gets a 10% better test accuracy than with two intervals. Further increasing the number of intervals doesn't increase the accuracy (it only increases the time to build the decision tree).

Next we analyze whether the system can predict better than just always predicting the main class. The main class is `short visit`, with 41% of the examples belonging to this class. The decision tree built by Tilde for an individual user[4] however has a testing accuracy of 62%, which is significantly better. We also tried to build a tree without distinguishing between individual users which resulted in a testing accuracy of 58%.

When we analyse the importance of the used attributes it turns out that not using the words occurring in the title of a page results in a tree with the same accuracy as when one doesn't use the words in the vector of frequent words or when one uses both. Also surprising is the fact that allowing the system to use information about the previous visited page doesn't result in a more accurate tree.

More information about these experiments can be found in [7].

5 Conclusion

In this paper we described the Adaplix system. The goal of Adaplix is to provide a system to website developers to enable them to make their site adapt itself to

[4] we used the user that visited the highest number of pages

the visitors. To this extend we first created a system which transforms HTML extended with conditional statements into plain HTML. The conditional statements contain tests based on certain properties of the user. For the system to be able to test these properties, it has to log the user's browsing behaviour. Since we already have this information, our next step is to use it to learn a profile of the user's browsing preferences. This is treated as a inductive classification tree building problem for which the Tilde system is used. In this way we combine two common approaches to adaptive websites: defining the possible changes manually or using machine learning techniques to find useful information for the user. We also investigated whether inductive logic programming can be used to learn user preferences in web browsing. As the preliminary results show Tilde was able to accurately predict the time the user spends at a page (accuracy comparable with WebWatcher).

This approach can be compared with systems like Webwatcher [11, 8], Fab [2, 1] and Do-I-Care [13, 12]. The main difference with the above systems is the combination of machine learning techniques together with the classic approach of adaption predefined by the author of the document. We also use ILP, a very flexible learning technique which allows for the use of many useful types of background knowledge (such as a lexicon, translations and domain specific knowledge).

Future work will be to integrate the learning component in the Adaplix system so that the learned knowledge can be effectively used when building adaptive websites. We use almost no background knowledge in these experiments. Further experiments can indicate the effect of the use of the types of background knowledge mentioned above.

Also on the learning aspect further extensions are possible. We only learned from individual users. Techniques from the domain of collaborative filtering can be integrated in the system to learn from other users as well. Since ILP is a very flexible learning technique this can easily be integrated.

Acknowledgements: The author wishes to thank Lieven Desmet, Jos Devloo, Roel Hertoghs and Guy Wellens for implementing the basic Adaplix system. Tom Ghelen implemented and experimented with the learning component. Hendrik Blockeel helped with the Tilde system. Thanks also go to Luc De Raedt and Denise Pilar da Silva for the interesting and stimulating discussions. Nico Jacobs is financed by a specialization grant of the Flemish Institute for the promotion of scientific and technological research in the industry (IWT).

References

[1] M. Balabanovic and Y. Shoham. Fab: content-based collaborative recommendation. *Communications of the ACM*, 40(3):66–72, March 1997.

[2] Marko Balabanovic. An adaptive web page recommendation service. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 378–385, New York, February 5–8, 1997. ACM Press.

[3] H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June 1998.

[4] H. Blockeel, L. De Raedt, N. Jacobs, and B. Demoen. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, 3(1):59–93, 1999.

[5] Peter Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2-3):87–129, 1996.

[6] L. De Raedt. Logical settings for concept learning. *Artificial Intelligence*, 95:187–201, 1997.

[7] T. Ghelen. Adaplix: Leren van gebruikersinteresses uit bezochte webpagina's. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1999.

[8] Thorsten Joachims, Dayne Freitag, and Tom Mitchell. WebWatcher: A tour guide for the World Wide Web. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 770–777, San Francisco, August 23–29 1997. Morgan Kaufmann Publishers.

[9] G. A. Miller, Beckwith R., C. Fellbaum, D. Gross, and K. J. Miller. Introduction to wordnet: An on-line lexical database. *Journal of Lexicography*, 3(4):234–244, 1990.

[10] S. Muggleton and C. D. Page. A learnability model for universal representations. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, pages 139–160, Sankt Augustin, Germany, 1994. GMD.

[11] T. Joachims R. Armstrong, D. Freitag and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. In *AAAI Spring Symposium on Information Gathering*, 1995.

[12] Brian Starr, Mark S. Ackerman, and Michael Pazzani. The do-I-care agent: Effective social discovery and filtering on the web. Technical Report ICS-TR-96-50, University of California, Irvine, Department of Information and Computer Science, August 1996.

[13] Brian Starr, Mark S. Ackerman, and Michael Pazzani. Do-I-Care: Tell me what's changed on the Web. In *Proceedings of the AAAI Spring Symposium on Machine Learning in Information Access Technical Papers*, March 1996.

[14] Larry Wall, Randal L. Schwartz, Tom Christiansen, and Stephen Potter. *Programming Perl*. Nutshell Handbook. O'Reilly & Associates, 2nd edition, 1996.