Personal WebWatcher: design and implementation

Dunja Mladenić
Dpt. for Intelligent Systems, J.Stefan Institute,
Jamova 39, 11000 Ljubljana, Slovenia
Phone: (+38)(61) 1773 272, Fax: (+38)(61) 1258-158
E-mail: Dunja.Mladenic@ijs.si
http://www-ai.ijs.si/DunjaMladenic/home.html
School of Computer Science, Carnegie Mellon University
Pittsburgh, PA, USA

1 Introduction

With the growing availability of information sources, especially non-homogeneous, distributed sources like the World Wide Web, there is also a growing interest in tools that can help in making a good and quick selection of information we are interested in. Recent work that arises at the intersection of Information Retrieval and Machine Learning offers some novel solutions to this problem, as well as work in Intelligent Agents. For example, Armstrong et al. [2] developed WebWatcher, a system that assists user in locating information on the World Wide Web taking keywords from the user, suggesting hyperlinks and receiving evaluation. Balabanovic et al. [3] developed "a system which learns to browse the Internet on behalf of a user". It searches the World Wide Web taking bounded amount of time, selects the best pages and receives an evaluation from the user. The evaluation is used to update the search and selection heuristics. Pazzani et al. [26] collect ratings of the explored Web pages from the user and learn a user profile from them. Pages are separated according to their topic and a separate profile is learned for each topic. Mitchell et al. [25] proposed a system connected to the user's electronic calendar, that generates sets of rules capturing the user's scheduling preferences and some other information about individual attendees of meetings. It uses these rules to provide advice to the user for new, unscheduled meetings. Lang [21] developed a system for electronic news filtering that uses text-learning to generate models of user interests. Krulwich and Burkey [20] proposed "The ContactFinder agent" that reads and responds to bulletin board messages; assists users by referring them to other people who can help them and; categorizes messages and extracts their topic areas. Maes [24] described "interface agents" that learn from the user as well as from other agents. As examples of such agents they developed agents for: electronic mail handling, meeting scheduling, electronic news filtering and entertainment recommendation. Some of these agents use the context of documents and adopt Information Retrieval approaches (eg. news filtering). Others rely on correlation between different users — performing "social learning" (eg. entertainment recommendation). Holte and Drummond [15], Drummond et al. [9] designed a system that assists browsing of software libraries, taking keywords from the user and using a rule-based system with forward chaining inference, assuming that the library consists of one type of item and the user goal is a single item. Etizioni and Weld [11] offer an integrated interface to the Internet combining UNIX shell and the World Wide Web to interact with Internet resources. Their agent accepts high-level user goals and dynamically synthesizes the appropriate sequence of Internet commands to satisfy those goals. Hammond et al. [?] and Burke et al. [6] developed a system that uses a "natural language question-based interface to access distributed text information sources" and helps the user to find answers to her/his question in a databases such as FAQ files.

One of the available information sources is the World Wide Web and it is currently growing quickly, attracting many users with different interests. Since interaction with the World Wide Web (WWW) is by means of computer, one can use computers to "observe" and record user actions and use this information to help users find their way in the WWW.

Personal WebWatcher is a system that observes users of the WWW and suggests pages they might be interested in. It learns user interests from the pages requested by the user. The learned model of user interests is then used to suggest hyperlinks on new HTML-pages requested by and presented to the user via Web browser that enables connection to "proxy" eg. NETSCAPE. Section 2 gives an overview of Personal WebWatcher, its functionality and some directions for further development. Section 3 describes some of the research problems related to Personal WebWatcher's learning. The structure of the system and its implementations are described in Section 4 and Perl code is given in the Appendix. Results of the first experiments are given in Section 5.

2 "Personalizing" WebWatcher

Personal WebWatcher is mainly inspired by WebWatcher: "a Learning Apprentice for the World Wide Web" [2], [16] and some other work related to learning apprentice and learning from text [17], [19], [21], [25]. The idea of a learning apprentice is to automatically customize to individual users, using each user interaction as a training example.

WebWatcher can be described as an agent that assists users in locating information on the WWW. It learns by observing a user on her/his way through the WWW and suggests interesting hyperlinks whenever it is confident enough. The idea is that the user provides a few keywords describing a search goal and WebWatcher highlights related hyperlinks on the current page and/or adds new hyperlinks to the current page. It can also suggest pages related to the current page using information stored in the structure of hypertext without considering the text itself [16], or send an e-mail message to the user whenever specified pages change. The same WebWatcher version is designed to serve all users, collecting information and sharing it between users. For example, if someone recognizes a page as being related to the keywords she/he typed in the system at the beginning of the search, this can be useful for any user searching for similar information.

Unlike WebWatcher, Personal WebWatcher (PWW) is structured to specialize for a particular user, modeling her/his interests. It "watches over the user's shoulder" the similar way WebWatcher does, but it avoids involving the user in its learning process (it doesn't ask the user for any keywords or opinions about pages). It solely records the addresses of pages requested by the user and highlights hyperlinks that it believes will be of interest. In the learning phase (typically during the night), requested pages are analyzed and a model of user interests is generated/updated. This model is used to give advice for hyperlinks on retrieved HTML-pages requested by and presented to the user via Web browser.

Since each user has her/his own copy of the system - her/his own agent, these agents can communicate and exchange information on a base of similarity between their users, often referred to as collaborative or social learning [22], [31]. There are also some other types of "Personal agents" the user could use, for example, an agent for Calendar Apprentice [25], and these agents can exchange information they have about the same user in different fields/activities. we plan to investigate these in the future work on Personal WebWatcher.

3 PWW "Behind the stage"

There are many research question behind Personal WebWatcher, that wait to be answered. Here, we consider some of them that are related to learning. There are also many others, for example, how to design communication between user and agent and between different agents, how to provide privacy to the user, to which extent agent should involve user to its learning/exploration process, ...

If we concentrate on learning, we first want to know what kind of problem are we faced with, how to represent it to some learning algorithm and which kind of algorithm to use. Currently we restricted our work to text documents (plain text and HTML) so we are faced with the problem of text-learning having mainly short to medium size documents with varying vocabulary. This section gives different approaches related to (1) document representation, (2) feature selection and (3) learning used on text-learning problem. Table 1 summarizes them over some related papers in order to give an idea about current trends.

3.1 Document representation

The frequently used document representation in Information Retrieval and text-learning is the so called $TFIDF$-vector representation. It is a *bag-of-words representation*: all words from the document are taken and no ordering of words or any structure of text is used (see Figure 1). Since most of our documents are in HTML format, there is a well defined underlying structure that could be used. There is also additional information in plain text, for example structure of sentences, position of words or neighboring words. The question is how much can we gain considering additional information in learning (and what information to consider) and what is the price we have to pay for it? There is currently no well studied comparison or directions that we are aware of. There is

Paper reference	Doc. Representation	Feature Selection	Learning
Apté et al. [1]	bag-of-words (frequency)	stop list+ frequency weight	Decision Rules
Armstrong et al. [2]	bag-of-words (Boolean)	mutual info.	TFIDF, Winnow, WordStat
Balabanovic et al. [3]	bag-of-words (frequency)	stop list+stemming+ keep 10 best words	TFIDF
Bartell et al. [4]	bag-of-words (frequency)	latent semantic indexing using SVD	—
Berry et al. [5] Foltz and Dumais [12]	bag-of-words(frequency)	latent semantic indexing using SVD	TFIDF
Cohen [8]	bag-of-words (Boolean)	infrequent words pruned	Decision Rules ILP
Joachims [17]	bag-of-words (frequency)	in/frequent words+ mutual info.	TFIDF, PrTFIDF, Naive Bayes
Lewis et al. [23]	bag-of-words (Boolean)	log likelihood ratio	logistic regression combined with Naive Bayes
Maes [24]	bag-of-words+ header info.	mail/news header info.+ selecting keywords	Memory-Based reasoning
Pazzani et al. [26]	bag-of-words (Boolean)	stop list+ mutual info.	TFIDF, Naive Bayes, Nearest Neighbor, Decision Trees
Sorensen and Mc Elligott [33], [10]	n-gram graph (only bigrams)	weighting graph edges	connectionist combined with Genetic Algorithms
Yang [35]	bag-of-words (Boolean, frequency, TFIDF)	stop list	adapted k-Nearest Neighbor

Table 1: Document representation, feature selection and learning algorithms used in some related work.

some evidence in Information Retrieval research, that for long documents, considering information additional to bag-of-words is not worth efforts. But our documents are mostly HTML-documents on the WWW and they are not especially long!

In the process of using text to learn how to give advice for a hyperlink, different approaches can be used to decide which part of text to use and how to represent it. Personal WebWatcher uses an approach similar to that of WebWatcher. In WebWatcher the bag-of-words representation is used, where considered text consists of underlined words, words in the sentence containing the hyperlink, words in all the headings above the hyperlink and words given as keywords by the user [2]. Some later versions of the WebWatcher system change slightly the way of constructing text for learning, eg. adding words in the document retrieved behind hyperlink. Many current systems that learn on text use the bag-of-words representation using either Boolean features indicating if specific word occurred in document (eg. [2], [8], [23], [26], [35]) or frequency of word in a given document (eg. [1], [3] [4], [5], [17], [35]). There is also some work that uses additional information such as word position [8] or word tuples called n-grams [33].

We decided to use the bag-of-words representation using frequency of word and observe success of given advice (whether user selected the advised hyperlink). In case of poor system performance, some additional information from HTML-structure could be added, for example, frequency of word in headlines of a given document. We would

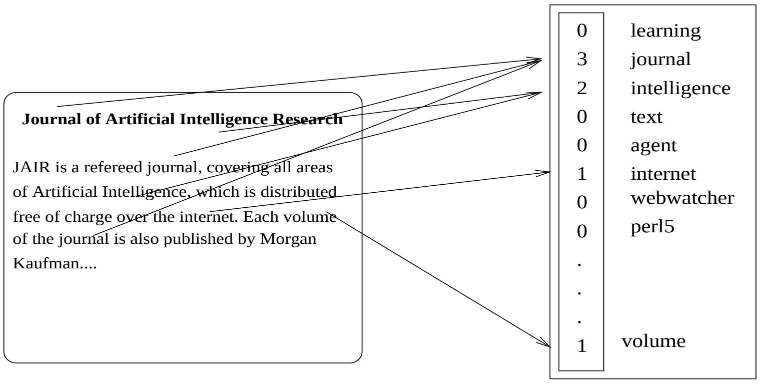


Figure 1: Bag-of-words representation using frequency vector.

like to spend time on extracting and using this additional information only when highly probable that we will gain a decent improvement in system performance. This is currently under research.

3.2 Feature selection

If we decide to ignore all the additional information and use the bag-of-words approach, we still end up with several tens of thousands of different words that occur in our documents. Not only is using all these words time-consuming but also many of them are not really important for our learning task. One of the approaches to reduce number of different words is to use "stop-list" containing common English words (eg. a, the, with) or pruning the infrequent and/or very frequent words ([8], [17]). There is also the possibility of word stemming. Many approaches introduce some sort of word weighting and select only the best words (eg. [1], [2], [3], [17], [23], [26]) or reduce dimensionality using latent semantic indexing with singular value decomposition (eg. [4], [5], [12]). Some of the Machine Learning techniques for feature selection could also be used (eg. [7], [32], [18]) but most of them take too long in situations with several tens of thousands of features.

In the current implementation of PWW we weight words using mutual information between word occurance and class value [34], the same way as used in [2] or [17]. The research topic is the number of best words to consider and we observe its influence on classification accuracy and precision of the best suggested hyperlink (see Section 5). Mutual information assigns higher weight to the words that make better distinction between interesting and uninteresting documents. One of the practical problems during classification is that a new document often contains very few or even non of the selected words. Since we are more interested in positive class (interesting documents) and we want to have words that are frequent, it might be better to include in the weighting formula the probability of a word occurring in the positive class or frequency of the word ([23]) $TF(w)\log(\frac{P(w|c)}{P(w|\bar{c})}$, where w is a selected word, c is the positive class and $TF(w)$ is the frequency of word w.

We plan to make additional experiments using the proposed combined weighting as well as using some other weighting methods. For example, combining a stop list with weighting words by their frequency and keeping the most frequent words [35] or using word weighting used in the odds ratio method [27]. Odds ratio is the method of document ranking according to their relevance for a given problem (eg. being interesting for user). Ranking of documents is defined as $ranking(d, c) = \log \frac{P(c|d)}{P(\bar{c}|d)} = \log \frac{P(c)P(d|c)}{P(\bar{c})P(d|\bar{c})} = \log \frac{P(c)\Pi_{w \in d} P(w|c)}{P(\bar{c})\Pi_{w \in d} P(w|\bar{c})} = \sum_{w \in d} weight(w) + const.$ Word weight that it defines can be used for feature selection, weighting all words and selecting words with the highest weight. This word weight is defined as

$$weight(w) = \log \frac{p(w)(1 - \bar{p}(w))}{(1 - p(w))\bar{p}(w)}$$

where $p(w) = \frac{TF(w,c) + const.}{\#docs(c) + 1}$, where $docs(c)$ is number of documents in class c and $\bar{p}(w)$ is the same as $p(w)$ except that c is substituted with $\bar{c}$. Shaw [28] proposed special handling of singularities in the above formulas for $p(w)$ and $\bar{p}(w)$, namely, $p(w) = \frac{1}{\#docs^2}$ when $TF(w, c) = 0$ and $p(w) = 1 - \frac{1}{\#docs^2}$ when $TF(w, c) = \#docs(c)$

3.3 Learning algorithm

One of the well-established techniques for text in Information Retrieval is to represent each document as a $TFIDF$-vector in the space of words that appeared in training documents [30], sum all interesting document vectors and use the resulting vector as a model for classification (based on [29] relevance feedback method). Each component of a document vector $d^{(i)} = TF(w_i, d)IDF(w_i)$ is calculated as the product of TF (Term Frequency — number of times word w_i occurred in document) and $IDF = log \frac{D}{DF(w_i)}$ (Inverse document Frequency), where D is the number of documents and document frequency $DF(w_i)$ is the number of documents word w_i occurred in at least once. The exact formulas used in different approaches may slightly vary (some factors are added, normalization performed [30]) but the idea remains the same. A new document is then represented as a vector in the same vector space as the generated model and the distance between them is measured (usually defined as a cosine of angle between vectors) in order to classify the document. This technique has already been used in Machine Learning experiments on World Wide Web data (eg. [2], [3], [5], [17], [26]).

There are also some other techniques for model generation that have been used in text-learning. Armstrong et al. [2] used a statistical approach they called WordStat that assumes mutual independence of words and defines probability of class c as $P(c) = 1 - \Pi_w(1 - P(c/w))$. Pazzani et al. [26] used a Naive (Simple) Bayesian classifier on Boolean vectors, that assumes independence of words and defines probability of class c for given document doc that contains words w as proportional to $P(c)\Pi_w P(c/w)$. They also used Nearest Neighbor and symbolic learning using Decision Trees. A variant of k-Nearest Neighbor was also used by Yang [35], where relevance of class c given document doc is defined as $rel(c/doc) = \sum_{i=1}^{k} similarity(doc, D_i)P(c/D_i)$ and $similarity$ is measured by cosine between vectors and $P(c/D_i) = \frac{\#D_i Inc}{\#D_i Intrainingdata}$ (same document may occur several times being classified in different categories). Joachims [17] introduced Probabilistic $TFIDF$ that takes into account document representation Θ and defines probability of class c for given document doc that contains words w as $P(c/doc) = \sum_w \frac{P(c)P(w/c)}{\sum_i P(c_i)P(w/c_i)} P(w/doc, \Theta)$ where $P(w/c) = \frac{TF(w,c)}{\sum_i TF(w_i,c)}$ and $P(w/doc, \Theta) = \frac{TF(w,doc)}{\sum_i TF(w_i,doc)}$. He also used Naive (Simple) Bayesian classifier on frequency vectors, the same as we used in PWW. It assumes independence of words and defines probability of class c for given document doc that contains words w as

$$P(c/doc) = \frac{P(c)\Pi_w P(w/c)^{TF(w,doc)}}{\sum_i P(c_i)\Pi_w P(w/c_i)^{TF(w,doc)}}$$

where $P(w/c) = \frac{1+TF(w,c)}{\#words+\sum_i TF(w_i,c)}$; $TF(w,c)$ denotes frequency of word w in documents of class c and $TF(w, doc)$ denotes frequency of word w in document doc.

Apté et al. [1] used Decision Rules and observed that in case of different topics being categories, it is better to select features for each given topic (using stop-list and frequency weighting) than for all topics at once, even if the set of features is additionally reduced for each topic using entropy-based measure to weight features. Cohen [8] used Decision Rules and the Inductive Logic Programming systems FOIL and FLIPPER. Lewis and Gale [23] used a combination of a Naive Bayesian classifier and logistic regression defining probability of class c for given document doc that contains words w as $P(c/doc) = \frac{exp(a+b\sum_w log\frac{P(w/c)}{P(w/\bar{c})})}{1+exp(a+b\sum_w log\frac{P(w/c)}{P(w/\bar{c})})}$.

Maes [24] used Memory-Based reasoning, McElligot and Sorensen [10], [33] used a connectionist approach combined with Genetic Algorithms.

We decided to test different learning algorithms on PWW data (see Section 5), since it is not clear which algorithm is the most appropriate. The current version of PWW uses a Naive (Simple) Bayesian classifier on frequency vectors to generate a model of user interests, that is used for advising hyperlinks.

4 Structure of Personal WebWatcher

Personal WebWatcher consists of two main parts: a **proxy server** that interacts with the user via Web browser and a **learner** that provides the user-model to the server (see Figure 2). Communication between them is via disk; the **proxy** saves addresses of visited documents (URLs) and the **learner** uses them to generate model of user interests. The whole system is implemented in approximately 2500 lines of Perl code and 1500 lines of C^{++} code.

The **proxy server** consists of three main parts, each implemented as a Perl script: **proxy** (additionally calls external **fetcher** code to fetch the page), **adviser** and **classifier** (calls external C^{++} code for classification). **Proxy** waits in an infinite loop for a page request from browser. On request, it fetches the requested document and; if it is an HMTL-document adds advice and; forwards the document to the user. To add advice **proxy** forwards the page to **adviser**, that extracts hyperlinks from document and calls external code for **classification** that uses generated user-model. A limited number of hyperlinks that are scored above some threshold are recommended to the

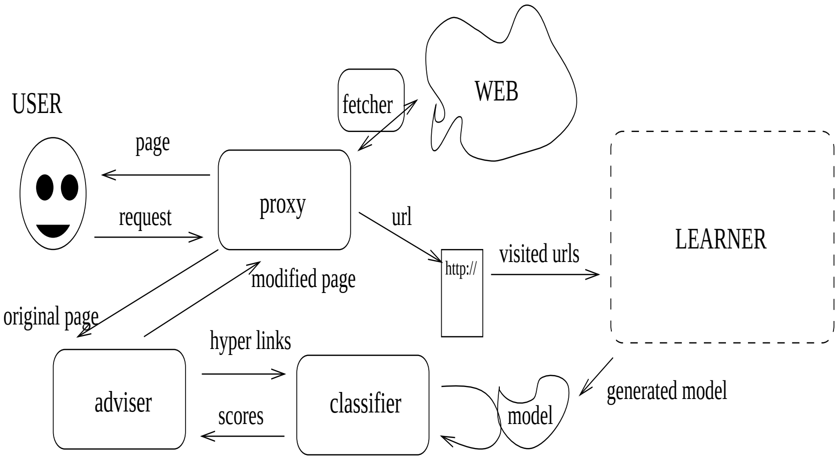


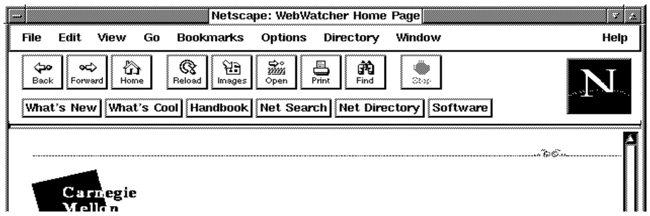
Figure 2: Structure of Personal WebWatcher. The learning part is described separately .

user, indicating their scores using graphical symbols placed around each advised hyperlink. For example, in Figure 3 three hyperlinks are suggested by PWW: “Machine Learning Information Services” and two project members (Dayne Freitag, Thorsten Joachims). There is a banner at the top of the page showing that PWW is “watching over the user’s shoulder”.

4.1 Structure of the learning module

Learner works in two versions: learning a new model from scratch (LEARNER) or updating an existing model (UPDATER) as shown in Figure 4. The difference is that the first one has to define the domain (words to be used) and starts learning with an empty model, while the second one has already defined which words to use in representing documents as frequency vectors and has an existing model to modify. Both versions fetch visited documents and documents one step behind the hyperlinks of visited documents and store them as positive or negative examples of user interests, depending whether the user visited the document or not (getDoc in Figure 4). Hyperlinks from visited HTML-documents are extracted and stored in an extended hyperlink format, the same that is used by **adviser**. Each hyperlink is represented in extended format, taking into account underlined words, words in a window around a hyperlink and words in all the headings above the hyperlink. Using hyperlinks represented only with underlined words is often a bad idea, eg. click here...

Hyperlinks whose documents were visited by the user are considered to be positive examples, and all the other to be negative examples of the user interests. The idea is that all hyperlinks were presented to the user and the user chose to visit some of them that meet her/his interests. This simplification is introduced to minimize users


Netscape: WebWatcher Home Page
File
Edit
View
Go
Bookmarks
Options
Directory
Window
Help
Back
Forward
Home
Reload
Images
Open
Print
Find
Stop
What's New
What's Cool
Handbook
Net Search
Net Directory
Software
Carnegie
Mellon