Learning from hotlists and coldlists:

Towards a WWW information filtering and seeking agent

Michael Pazzani, Larry Nguyen & Stefanus Mantik
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717
pazzani@ics.uci.edu
phone: (949) 824-5888
fax (949) 824-4056
http://www.ics.uci.edu/~pazzani/

Abstract

We describe a software agent that learns to find information on the World Wide Web (WWW), deciding what new pages might interest a user. The agent maintains a separate hotlist (for links that were interesting) and coldlist (for links that were not interesting) for each topic. By analyzing the information immediately accessible from each link, the agent learns the types of information the user is interested in. This can be used to inform the user when a new interesting page becomes available or to order the user's exploration of unseen existing links so that the more promising ones are investigated first. We compare four different learning algorithms on this task. We describe an experiment in which a simple Bayesian classifier acquires a user profile that agrees with a user's judgment over 90% of the time.

1 Introduction

There is a vast amount of information on the World Wide Web (WWW) and more is becoming available daily. How can a user locate information that might be useful to that user? Fortunately, some people have created indices to other information providers. For example, http://golgi.harvard.edu/biopages/all.html contains links to over 300 sites on the topic of Biosciences. However, even for someone interested in Bioscience this is only a partial solution, since it would be time consuming to explore each of the 300 links. In this paper, we investigate how a learning agent could be used to acquire a user interest profile as the user explores some of these links. This user profile may then be used to suggest which other links should be explored or to notify the user when new information is linked to the index.

Most people using Mosaic or other Web browsers maintain a hotlist that contains pointers to favorite Web pages. We are working on an extension to Mosaic that maintains a separate hotlist for each topic a user is interested in. Associated with the topic are a set of URLs such as http://golgi.harvard.edu/biopages/all.html that serves as indices to information providers. The most novel part of the interface is that it maintains a coldlist with pointers to indexed Web pages that the user visited but didn't like. Figure 1 shows an example of the interface for collecting user likes and dislikes.

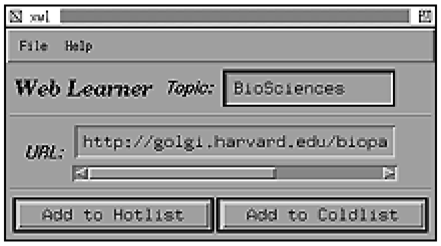

Figure 1. An interface to Mosaic that maintains a separate hotlist and coldlist for each topic.

The information gathered by the interface can be used to learn a user profile. In this paper we focus on the underlying technology for learning a user profile. We show that the learned user profile determines with greater than 90% accuracy whether a user would be interested in a page that hasn't been seen by the learning agent. We begin with a description of how the positive and negative examples may be extracted from the HTML source of the items on the hotlist and coldlist. Next, we describe four different learning algorithms that we have tested. Finally, we describe an experiment that evaluates the accuracy of the learned user profile.

2 Learning from Hotlists and Coldlists

Learning algorithms require a set of positive examples of some concepts (such as web pages one is interested in) and negative examples (such as web pages one is not interested in). The hotlist and coldlist serve as excellent sources of this information. However, learning programs require that the examples be represented as a set of feature vectors. Therefore, we have constructed a method of converting the HTML source of a web page into a Boolean feature vector. Each feature has a Boolean value that indicates whether a particular "word" is present (at least once) or absent in a particular web page. For the purposes of this paper, a word is a sequence of letters, delimited by nonletters. For example, the URL <A HREF= http://golgi.harvard.edu/biopages/all.html> contains nine "words" a, href, http, golgi, harvard, edu, biopages, all, and html. All words are converted to upper case.

Not all words that appear in a HTML document are used as features. We use an information-based approach, similar to that used by an early version of the NewsWeeder program (Lang, 1995) to determine which words to use as features. Intuitively, one would like words that occur frequently in pages on the hotlist, but infrequently on pages on the coldlist (or vice versa). This requires finding *E,* the expected information content of a word:

$$E(W) = p(W)I(W) + p(\overline{W})I(\overline{W})$$

where *p(W)* is the probability that a word occurs in the HTML source of a page (on either the hotlist or coldlist), $p(\overline{W})$ is the probability that a word does not appear on a page, and *I(W)* is the information content of the word. This is given by

$$I(V) = -p(Hot\&V)\log_2(p(Hot\&V)) + \\ -p(Cold\&V)\log_2(p(Cold\&V))$$

where *p(Hot&V)* is the probability that the word occurs in the HTML source of a page hotlist (if *V* =1) or the probability it doesn't occur (if *V* = 0).

Using this approach, we find the set of *k* words with the highest information content. In the experiment discussed in Section 4, we use the 128 most informative words. Table 1 shows some of most informative words obtained from a collection of 120 documents on the biosciences. Of these 120 examples, 38 are on the hotlist and the remainder are on the coldlist.

Table 1. Some of the words used as features.

agency animal automated biocomputing classification clearinghouse compound computer contract contracts control data database descriptions detailed documentation funding genetic genome grants graph illness improvements institute legislation mapping netvet org poison poisons predict probe proposal protein quarterly searching sequences simulate statistics webmaster

3 Learning algorithms

Once the HTML source of items on the hotlist and items on the coldlist for a given topic have been converted to positive and negative examples represented as feature vectors, it's possible to run many learning algorithms on the data. We are particularly interested in those algorithms that may be run quickly, so that it would be possible to develop a user profile while the user is browsing. For this reason, we did not investigate neural network algorithms (e.g., Rumelhart, Hinton & Williams, 1986). We concentrated on Bayesian classifiers, two variants of the nearest neighbor algorithm and a decision tree learner.

3.1 Bayesian classifier

The Bayesian classifier (Duda & Hart, 1973) is a probabilistic method for classification. It can be used to determine the probability that an example *j* belongs to class *Ci* given values of attributes of the example:

$$P(C_i|A_1=V_{1_j} \& \ldots \& A_n=V_{n_j})$$

If the attribute values are independent, this probability is proportional to:

$$P(C_i) \prod_k P(A_k=V_{k_j}|C_i)$$

Both $P(A_k=V_{k_j}|C_i)$ and $P(C_i)$ may be estimated from training data. To determine the most likely class of an example, the probability of each class is computed. An example is assigned to the class with the highest probability.

3.2 Nearest Neighbor

The nearest neighbor algorithm operates by storing all examples in the training set. To classify an unseen instance, it assigns it to the class of the most similar example. Since all of the features we use are binary features, the most similar example is the one that has the most feature values in common with a test example.

3.3 PEBLS

In its simplest form, nearest neighbor uses an equal weight for each feature. PEBLS (Cost & Salzberg, 1993) is a nearest neighbor algorithm that makes use of a modification of the value difference metric, MVDM, (Stanfill & Waltz, 1986) for computing the distance between two examples. This distance between two examples is the sum of the value differences of all attributes of the examples. The value difference between two values V_{j_x} and V_{j_y} of attribute Aj is given by

$$d(V_{j_x}, V_{j_y}) = \sum_i \left| \hat{P}(C_i | A_j = V_{j_x}) - \hat{P}(C_i | A_j = V_{j_y}) \right|$$

PEBLS assigns a test example to the class of the training example that has the minimum distance to the test example as measured by the value difference metric. Unlike the "overlap" metric of nearest neighbor, the MVDM metric places more weight on features that are more discriminating.

3.4 Decision Trees

Decision tree learners such as ID3 build a decision tree by recursively partitioning examples into subgroups until those subgroups contain examples of a single class. A partition is formed by a test on some attribute (e.g., is the feature database equal to 0). ID3 selects the test that provides the highest gain in information content, using the equations described in Section 2 for selecting informative features.

4 Experimental Evaluation

To determine whether it is possible to learn user preferences accurately, we asked a user interested in Artificial Intelligence and Medicine to create a hotlist and a coldlist for the BioSciences page located at http://golgi.harvard.edu/biopages/all.html. The hotlist contained 38 items and the coldlist contained 82. We used these pages as training and test data for an experimental evaluation. For an individual trial of an experiment, we randomly selected *k* pages to use as a training set, and reserved the remainder of the data as a test set. From the training set, we found the 128 most informative features, and then recoded the training set as feature vectors to be used by the learning algorithm. We tried four learning algorithms on each training set: A simple Bayesian classifier, PEBLS, Nearest Neighbor (NN) and ID3. The learning algorithm created a representation for the user preferences. Next, the test data was converted to feature vectors using the features found informative on the training set. Finally, the learned user preferences were used to determine whether pages in the test set would interest the user. For each trial, we recorded the accuracy of the learned preferences (i.e., the percent of test examples for which the learned preferences agreed with the user's interest). We ran 50 trials of each algorithm for 7 different training set sizes (10, 15, 25, 35, 50, 75, 100).

Figure 2 shows the average accuracy of each algorithm as a function of the number of training examples. The most striking finding of this experiment is the superior performance of the simple Bayesian classifier. With 15 examples it is 84.5% accurate, with 50 examples it is 89.1% accurate and with 100 examples the accuracy increases to 91.2%. In contrast, none of the other algorithms ever exceed 78% accuracy. One possible explanation for the superior performance of the Bayesian classifier when compared to the decision tree learner is that the Bayesian classifier examines every feature while making a determination, while the decision tree only looks at a few features. The nearest neighbor algorithm also uses all features. However, it does not distinguish between more important similarities (e.g., using the word "grants") and less important ones (e.g., using the word "an"). In contrast, the Bayesian classifier will treat one features as more important than another if the conditional probability of the feature given the class differs substantially between the classes.

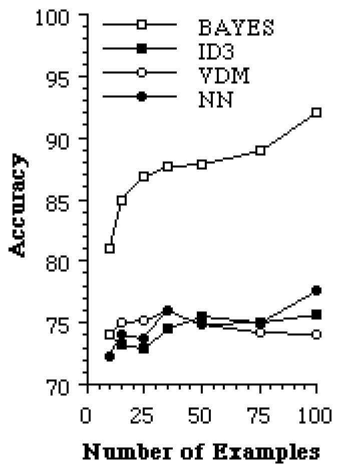

Figure 2. The average accuracy of each learning algorithm at predicting a user's preferences of bioscience pages.

5 Future Work

This work is in its preliminary stages. We have many plans for improving the Web Learner. First, we'd like to reimplement the interface in HTML so that it may be used with any Web browser on any type of computer. This will also give us more control of how information is displayed. For example, we could reorder links so the most promising unseen ones are displayed first or displayed in a different color. We also plan on distinguishing those links that are on a coldlist so that the user is warned not to visit them again.

Since web pages may be changed at anytime, we also plan on doing the necessary bookkeeping of notifying a user when an item on the coldlist has changed so that it now looks like items on the hotlist.

In order to evaluate unseen pages, it is necessary to retrieve the entire HTML to convert the page to a feature vector. We are considering an extension that just searches the first *k* (e.g., 2000) characters rather than the entire document. This may reduce the transmission overhead when using the agent interactively.

We also plan on investigating improvements to the underlying information extraction and machine learning capabilities. For example, some features are different forms of a root word, and we probably should have a single feature for the root word. In addition, some potential features such as "machine" and "learning" may not be very informative individually, but in combination ("machine learning") they may be very predictive of some users interests. Similarly, the most successful learner on this problem, the Bayesian classifier, makes independence assumptions that are probably violated by the data. It might be possible to improve accuracy further by detecting and correcting for such violations (Pazzani, 1995).

Finally, we intend on running many more experiments with additional users and additional topics to see if the promising results of the initial experiment can be achieved for other users and other problems.

6. Related work

The methods developed for our learning agent are related to work information retrieval and relevance feedback (e.g., Salton & Buckey, 1990; Croft & Harper, 1979). However, rather than learning to adapt user queries, we are developing a user profile that may be used for filtering new information as it becomes available.

There are several other agents designed to perform tasks similar to ours. The WebWatcher (Armstrong, Freitag, Joachims, and Mitchell, 1995) system is designed to help a user retrieve information from Web sites. When given a description of a goal (such as retrieving a paper by a particular author), it suggests which links to follow to get from a starting location to a goal location. It learns by watching a user traverse the WWW and it helps the user when similar goals occur in the future. The WebWatcher and the work described here serve different goals. In particular, the user preference profile may be used to suggest new information sources related to ones the user is interested in.

Like our work, WebHound (Lashkari, 1995) is designed to suggest new Web pages that may interest a user. WebHound uses a collaborative approach to filtering. In this approach, a user submits a list of pages together with ratings of these pages. The agent finds other users with similar ratings and suggests unread pages that are liked by others with similar interests. One drawback of the collaborative filtering approach is that when new information becomes available, others must first read and rate this information before it may be recommended. In contrast, by learning a user profile, our approach can determine whether a user is likely to be interested in new information without relying on the opinions of other users.

There have been a variety of software agents developed that perform tasks such as managing a calendar (Mitchell, Caruana, Freitag, McDermott & Zabowski, 1994) or electronic mail (Maes, 1994). Although similar in spirit to our learning agent, learning user preferences from HTML presents additional challenges and opportunities for software agents.

7 Conclusions

We have introduced an agent that collects user evaluations of the interestingness of pages on the World Wide Web. We have shown that a user profile may be learned from this information and that this user profile can be used to determine what other pages might interest the user.

Acknowledgments

The research reported here was supported in part by NSF grant IRI-9310413 and ARPA grant F49620-92-J-0430 monitored by AFOSR. Wed like to thank Lisa Tellier for comments on an earlier draft of this paper.

References

Armstrong, R. Freitag, D., Joachims, T., and Mitchell, T. (1995). WebWatcher: A learning apprentice for the World Wide Web. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-6/webagent/www/webagent-lus/webagent-plus.html

Cost, S. & Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features *Machine Learning, 10,* 57-78.

Croft, W.B. & Harper, D. (1979). Using probabilistic models of document retrieval without relevance. *Journal of Documentation, 35,* 285-295.

Duda, R. & Hart, P. (1973). *Pattern classification and scene analysis.* New York: John Wiley & Sons.

Kononenko, I. (1990). Comparison of inductive and naive Bayesian learning approaches to automatic knowledge acquisition. In B. Wielinga (Eds..), *Current trends in knowledge acquisition.* Amsterdam: IOS Press.

Lang, K. (1995). NewsWeeder: Learning to filter news. *Proceedings of the Twelfth International Conference on Machine Learning.* Lake Tahoe, CA.

Lashkari, Y. (1995). The WebHound Personalized Document Filtering System.

Maes, P. (1994). Agents that reduce work and information overload. *Communications of the ACM, 37*.

Mitchell, T., Caruana, R., Freitag, D. McDermott , J. & Zabowski, D. (1994) . Experiences with a learning personal assistant. *Communications of the ACM, 37*.

Pazzani, M. (1995). Searching for dependencies in Bayesian classifiers. *Artificial Intelligence and Statistics Workshop*.

Quinlan, J.R. (1986). Induction of decision trees. Machine Learning, 1, 81-106.

Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning internal representations by error propagation. In D. Rumelhart and J. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations,* (pp 318-362). Cambridge, MA: MIT Press.

Salton, G. & Buckley, C. (1990). Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science, 41,* 288-297.