Using URLs and Table Layout for Web Classification Tasks*

Lawrence Kai Shih and David R. Karger
Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Laboratory
200 Technology Square
Cambridge, MA 02139
kai@mit.edu, karger@mit.edu

ABSTRACT

We propose new features and algorithms for automating Web-page classification tasks such as content recommendation and ad blocking. We show that the automated classification of Web pages can be much improved if, instead of looking at their textual content, we consider each links's URL and the visual placement of those links on a referring page. These features are unusual: rather than being scalar measurements like word counts they are *tree structured*—describing the position of the item in a tree. We develop a model and algorithm for machine learning using such tree-structured features. We apply our methods in automated tools for recognizing and blocking Web advertisements and for recommending "interesting" news stories to a reader. Experiments show that our algorithms are both faster and more accurate than those based on the text content of Web documents.

Categories and Subject Descriptors

H.1 [**Information Systems**]: Models and Principles

General Terms

Algorithms, Experimentation

Keywords

Classification, Tree Structures, News Recommendation, Web Applications

1. INTRODUCTION

We propose new features and algorithms for use in automated Web classification tasks, such as content recommendation and ad blocking, that help users cope with the mass of information on the Web. An obvious approach to such classification tasks is to use the extensively available library of text and image classification tools. But in this paper, we argue that two features particular to web documents—their URLs, and the placement of links to them on a referring page—can be used even more effectively for such classification tasks. We pursue the intuition that content providers tend to choose URLs and page layouts that coherently structure their content according to topic, and that such topical structuring can be exploited in classification tasks. For example, even with no understanding of the text of a newspaper, one might guess relationships between articles based on visual groupings alone (for instance, that articles under the same heading are all about the same topic or were of similar importance). The goal of this paper is to formalize such intuitions into a general way of algorithmically predicting the properties of the targets of unvisited links.

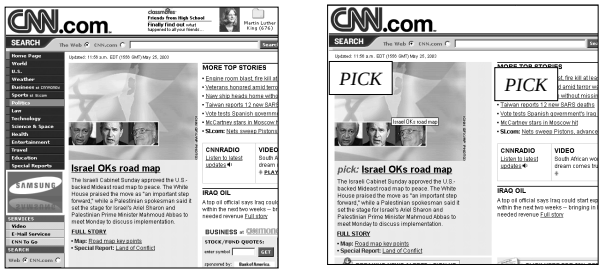

Figure 1: Screen-shots from an original CNN page (left) and the same page viewed through the Daily You (right). Notice the Daily You's version removes the advertisements, some of the navigation boxes, and also writes the word "pick" (emphasized in picture) near recommended news articles.

Two key steps in classification are to select the set of *features* that will be examined and the *decision rule* that will be applied to classify based on those features. In many ad-blocking applications[1], the features include, for example, the dimensions of the image being considered and the decision rules ("an ad is an image 250 by 100 pixels") are laboriously hand-coded. A big disadvantage of such an approach is the need for human effort to create the rules and to write new ones as advertisements evolve. To fix this, systems such as AdEater attempt to apply *machine learning*, automatically generating classification rules by examining a set of labeled *training examples* [13]. In recommendation systems, since different decisions rules work for each user, machine learning is almost always used. Typically, the Web is treated as a large text corpus: the numerous features used are the words in the documents, and standard machine learning algorithms such as Naive Bayes or support vector machines are applied [2].

The Web is more than just text, however: it contains rich, human-oriented structure suitable for learning. In this paper, we argue that

*(Produces the WWW2004-specific release, location and copyright information). For use with www2004-submission.cls V1.0. Supported by ACM.

WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-844-X/04/0005.

[1]for instance, http://webwasher.com

two features particular to Web documents, URLs and the visual placement of links on a page, can be of great value in document classification. We show that machine-learning classifiers based on these features can be simultaneously more efficient and more accurate than those based on the document text.

Our motivating example for these classification problems is *The Daily You*, a tool providing personalized news recommendations from the Web[2]. The Daily You uses URLs and table layout to solve two important classification problems: the blocking of Web advertisements and the page regions and outbound hyper-links predicted to be "interesting" to its user (see Figure 1).

The Daily You's framework is typical of machine learning applications: given a number of *training examples* (documents labeled as advertisements, or as interesting to the user), The Daily You attempts to make predictions about new, unseen items. Intuitively, a good prediction strategy is to find training examples that are "similar" to the new item, and predict that the new example has the same class as those similar training examples.

In typical text classification applications, similarity is measured by word overlap—documents are the same to the extent that they incorporate similar words (or phrases). In this paper, we take a different approach to similarity that stresses the relative position of items in a tree:

- On many Web sites, page URLs are organized in a hierarchy according to subject. For example, this year's articles about space on the CNN Web site have a URL prefixed by cnn.com/2003/tech/space, which can be interpreted as placement in the "space" subtree of the "tech" subtree of the cnn tree. On the natural assumption that a user is typically interested in certain subjects but not others, the location of an article in the URL tree is suggestive of the user's interest in it. Similarly, on many Web sites advertisements often bear links pointing back to a single "ad" subdirectory of the site. Indeed, commercial tools such as WebWasher let users manually specify certain URL "prefixes" as indicators of ads that should be blocked.

- We find that Web-sites often base the visual layout of their index pages on a subject taxonomy. This layout is often hierarchical and reflected in a recursive table layout that can be detected in the (hierarchical) parse tree of the HTML document. For example, the CNN Web-site front page offers a "table of contents" partitioning its stories under a number of labels such as "U.S.," "World," "Travel", and "Education." These placements represent subject classifications that may well be strong indicators of "interestingness" for a reader. Similarly, advertisements often have a specific placement in the page layout.

These two examples suggest the possibility of classifying a document based on its position in some preexisting taxonomy (the class label itself is not part of the taxonomy).

Most classification algorithms deal primarily with features that have been reduced to numbers, such as the number of occurrences of the word "apple" in a document or the length and width of an image. To instead implement the idea of classification using such tree-based features, we need to solve several problems. First, we need to develop a classification model that allows us to train and make predictions using the tree-based feature. Second, we need to show how classification using that model can be implemented efficiently.

[2]publicly accessible at http://daily-you.csail.mit.edu

To design a model, we use the tree as a Bayes net upon which we impose a "mutation" model—nodes in the tree are usually of the same class as their parents, but have some probability of flipping to a different class. To design algorithms, we adapt Bayes-net learning methods to our application domain. Conveniently, these Bayes-net algorithms can be made incremental, such that adding new examples and querying for the likely class of new items (based on their position in the tree) is very fast.

We then describe our algorithm's application to news recommendation and ad-blocking problems, and summarize experimental data showing that our approach works well. On the ad-blocking front, we show that our ad-blocking algorithm, trained *without human input* (using a simple heuristic for identifying ads) is able to learn to block ads with efficacy matching that of a commercial, hand-coded classifier with numerous human-derived classification rules. On the news recommendation front, we report the results of a study involving 176 users; our tree-based classifier required few training examples and significantly outperformed a state-of-the-art classifier (the support vector machine) in terms of both speed and accuracy applied to either traditional or tree-structured features.

Besides its advantages in learning efficiency and accuracy, our approach has one other important benefit specific to the Web. Unlike text-based classifiers, which must fetch the content of the page being classified, our classifiers do their work by looking at the *pointer* to the page, and need not fetch the page itself at all. Thus, our tree-based classifiers require orders of magnitude less bandwidth than traditional text based classifiers for the same problem. Under current network conditions, this translates into a significantly faster system.

1.1 Related Work

Some related work has been done on prioritizing the spidering for topic-relevant search. Rennie and McCallum [18] used reinforcement learning to map the text surrounding a link to rank pages for spidering. Search engines such as Google [5] are also said to use anchor text in links pointing to a page in order to decide whether that page is relevant to a query. Chakrabarti, et al. [7] built upon that work, creating a more robust set of features by parsing out the structure of the HTML, as we do. However, they choose to treat an HTML document as a linear sequence of (textual) tokens, and emphasize the idea of learning an appropriately-sized and -weighted "window" around the link that should be used to classify the link. Chakrabarti, et al. also propose to use the textual content of the referring page to predict classes *of the referring page*, and to use those predicted classes to make predictions about links on that page. All this work shares our idea of using extrinsic features to evaluate the class of an unseen document, but does not fully take advantage of tree structures as we do.

Trees in the form of taxonomies play a large role in machine learning; however, such taxonomies are generally the target *output* rather than a useful input, as in our work. Many classification algorithms attempt to use traditional features (such as word counts) to *create* taxonomies over the documents, or to insert new documents into a preexisting taxonomy of old documents [15, 12]. Work which aims to use tree position as a feature for learning other classifications seems much less common. Haussler [9] is, to our knowledge, the first to propose using position in a preexisting tree as a feature for classification. The model he proposed is more specific than ours, requiring that the concept to be learned form a conjunction of subtrees. Among other differences with our work, Haussler's model does not capture classes that are *complements* of subtrees. In our terminology, Haussler's work only allows for forward mutations, but does not allow for backward mutations (more

on this discussed below). Agrawal and Srikant [1] have a model for combining catalogs of documents that live in different taxonomies. Their model is simple in that it assumes that the taxonomy has no internal nodes (for more complex taxonomies, they discard all the internal nodes, connecting the root directly to the leaves). This allows for the use of a simpler algorithm (naive Bayes) but discards rich information about the finer-grained relationships between the leaves.

Another type of related work is wrapper induction [14]. In wrapper induction, the goal is to identify the part of a Web page containing a specific piece of information. An example goal would be to find a specific pattern, like a stock's quote on some financial Web page, day after day. Wrapper induction often uses formatting information to identify the right parts of a page, but we are not aware of any that uses URL structure. One could view our work as trying to learn the wrappers that contains things that interest a given user.

Our application resembles that of other prominent Web news services. Most existing work only operates on pre-set sites, while ours allows the users to specify arbitrary target sites. The large commercial services aggregate pages together (`http://my.yahoo.com`), but require pre-set sites (which presumably have been manually configured to feed specific stories to Yahoo!) and do not make user-specific recommendations. Some research applications like NewsDude [4] do make such recommendations, but only from pre-set news sources. NewsDude specifically uses text-based classifiers both to select good articles and to remove articles that seem overly redundant with already seen articles. An earlier attempt at user-profiling [17] also used a Bayesian text model to predict interesting Web pages. Newsblaster [3] and a similar service at Google (`http://news.google.com`) scan multiple pre-set sites and summarize similar articles. Newsblaster uses complex natural language parsing routines to combine articles from multiple sites into one summary article.

Other Web applications allow the user to select their own set of news sources, like the Montage system [2]. Rather than focusing on new information, Montage is more like an "automated book-marks builder." It watches the user to identify pages they visit frequently, and creates collections of links that let the user get to those pages more quickly. It uses traditional text-classification algorithms (SVMs) to break these bookmarks into coherent topic categories. In contrast, our system aims to recommend Web pages that are new but that we believe will be interesting to the user.

2. URL AND TABLE FEATURES

In this section we discuss in greater detail two tree-structured features that are particularly relevant to certain Web classification tasks.

2.1 URL trees

The World Wide Web Consortium argues that document URLs should be opaque (`http://www.w3.org/Axioms.html#opaque`). On this Web page, Tim Berners-Lee writes his axiom of opaque URIs: "... you should not look at the contents of the URI string to gain other information...".

In contrast to those style guidelines, most URLs nowadays have human-oriented meanings that are useful for recommendation problems. Indeed, the guideline's URL contain semantics including authorship (w3.org), that the page is written in HTML, and that the topic relates to an "Axiom about Opaqueness." As the document's URL demonstrates (somewhat ironically), URLs are more than simply pointers: authors and editors assign important meanings to URLs. They do this to make internal organization simpler (authorship rights, file permissions, self-categorization), and sometimes to make that organization scheme clear to readers. Readers often make inferences from URLs, which is why browsers and search engines usually display URLs along with the text description of a link. We can infer from a URL that a document serves a particular function (a specific Web directory might always serve ads); or relates to a topic ('business' stories might be under one directory); or has a certain authorship. Or, we might delete a suffix of an URL in an attempt to move to a more "general" page still related to our starting point. In short, similar documents (as defined by the site's authors) often reside under similar URLs. A good URL structure provides helpful contextual clues for the reader.

URLs are extremely good features for learning. First, they are easy to extract and relatively stable. Each URL maps uniquely to a document, and any fetchable document must have a URL. In contrast, other Web features like anchor text, alt tags, and image sizes, are optional and not unique to a document. Of course, URLs can be obfuscated, hidden or changed in automated fashion; but such changes simultaneously make it difficult for users and search engines to find and return to information. Second, URLs can be read without downloading the target document, which lets us perform classification more quickly. This is a necessary condition for real-time classification tasks like ad-blocking. Third, as we argue below, URLs have an intuitive and simple mapping to certain classification problems. For example, we give empirical evidence in Section 4 that the URL is highly correlated with whether a link is an advertisement or not. Most advertisement clicks are tracked through a small number of programs; these programs are usually contained in subtrees of the URL tree, like `http://doubleclick.net` or `http://nytimes.com/adx/...` .

To convert a URL into a tree-shape, we tokenized the URL by the characters `/`, `?` and `&`. The `/` is a standard delimiter for directories that was continued into Web directories; `?` and `&` are standard delimiters for passing variables into a script. The left-most item (`http:`) becomes the root node of the tree. Successive tokens in the URL (i.e. `nytimes.com`) become the children of the previous token. Note that our construction guarantees we end up with a tree, even if the web site itself is not tree shaped (two pages may point to the same URL, but it is the URL itself that defines the tree location).

2.2 HTML Table Trees

Similarly, the visual layout of a page is typically organized to help a user understand how to use a site. This layout tends to be templated—most pages will retain a 'look and feel' even though the underlying content might be dynamic. For example, different articles on one particular topic might appear in the same place on the page day after day. The page layout is usually controlled by HTML table tags, corresponding to rectangular groupings of text, images and links. Often, one table along the side or top of a page will contain much of the site's navigation. The content of a site might use tables to group together articles by importance (the headline news section of a news-magazine), by subject, or chronologically (newest items typically at the top). Like the URL, this page layout can be used to eliminate certain content (such as the banners at the top of the page); or to focus on other content (the headlines, or the sports section). Like the URL feature, tables make good features for machine learning. For a page to display properly in browsers, the tags have to obey a standardized HTML grammar; this also makes the table feature easy to extract. In the next section, we give the example of a Chinese Web site that might be understood even barring understanding the specifics of the content on the site.

To convert the HTML table structure into a tree-shape, we used a hand-written Perl program that extracted the HTML table tags

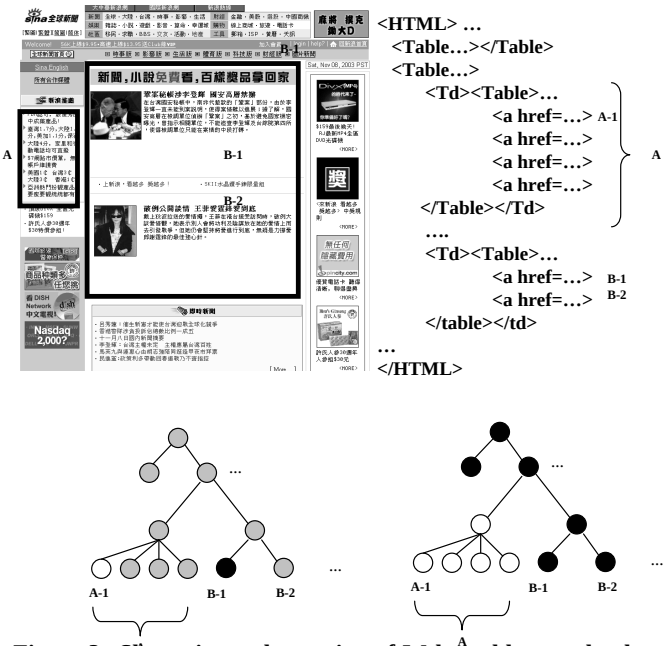

Figure 2: Shown is an abstraction of Web problem to the domain of "tree learning." The top-left shows the original Website. The top-right shows that visual portions of the page are collected in chunks of HTML, which are indented to show the HTML's tree structure. The bottom left shows the abstracted tree, receiving a partial labeling of the page: white ("advertisement"), black ("content") and gray ("unknown") nodes. The bottom right shows one potential generalization of the tree which suggests everything in box A is an advertisement.

(`<table>` and `<td>`). The root of the tree is the entire page's HTML. The children of a node are the next lower level of table elements.

Note that while the same story might be headlined in more than one place on the page (e.g., an article might appear in both "world" and "education"), for the purpose of this paper, we treat those two appearances as two separate leaves on the taxonomy.

3. LEARNING MODEL

In this section, we outline our learning model and algorithm. We classify using a *generative model*. We define a process that enforces our ideas about correlation: items nearby in the (URL or layout) tree usually have the same class. Our training data tells us the classes of some of the items, and our model suggests other areas of the tree that might be similar. More precisely, we model our learning problem using a *Bayes net*, and apply fast algorithms to perform classification on that Bayes net. In the next section, we give a short intuitive description of the types of relationships our algorithm and model might find.

3.1 Intuition Behind Model

Suppose someone was given a Web page, like the one featured at the top left of Figure 2, and asked to recommend a link to a friend. Suppose the recommender could not read any Chinese, had never visited the particular site[3], and generally could not decipher any of the contents of the page. Given that the friend liked the article in B-1, one might reasonably recommend the adjacent article in B-2. Without knowing any specifics of the Web page, most people can make good guesses about some of its semantic attributes.

Figure 2 also shows how a computer algorithm might reflect the earlier-stated intuition that items visually clustered might have similar properties. For example, if one person mentioned that the top link in box A was an advertisement, it might be reasonable to guess that everything in box A was an advertisement too. Similarly, if a person expressed interest in B-1, a natural recommendation might be B-2.

The upper right-hand side of Figure 2 displays snippets of the HTML that describes the site; various visually distinct areas from the Web page are encoded into continuous blocks of HTML. The HTML forms a tree, shown by indentations. That tree is abstracted on the bottom left. Suppose someone labels that node A-1 is an advertisement (white), while node B-1 is content (black). The tree on the bottom right shows a potential generalization of that information in which everything in box A is considered an advertisement. The work described here tries to formalize and build algorithms that automatically make generalizations like these.

3.2 The Generative Model

Our goal is to create a generative model in which nearby items tend to have the same class. More formally, we define a probability distribution over the possible classes of items in the tree. We aim for the probability distribution to reflect our intuition that nearby items usually have the same class. To accomplish this, we use a *Bayes net* over the tree holding the items to be classified. Our assumption that the class to be learned is correlated to the tree position is captured in a model based on mutations. We consider the tree holding the items we wish to classify. The actual items are usually leaves in this tree, but we extend our model to assume that the internal nodes, which may or may not represent actual items, also have "hidden" classes. Some children of internal nodes may "mutate" into different classes. We begin by assigning some class at the tree's root. We then work our way down the tree, declaring that each node is probably of the same class as its parent but has a small chance of being of the other class.

Formally, we consider two classes 0 and 1, and specify a *forward mutation probability* θ and a *backward mutation probability* ϕ (conversely, with probabilities $1-\theta$ and $1-\phi$, respectively, no change occurs). We declare that the class of a node follows probabilistically from the class of its parent. Suppose node x has parent y in the tree. Let N_x and N_y denote the classes of nodes x and y respectively. Then

$$\Pr[N_x = 1 \mid N_y = 0] = \theta \text{ and}$$
$$\Pr[N_x = 0 \mid N_y = 1] = \phi.$$

Applying this rule downward from the root provides a generative model of class assignments to nodes in the tree. Given the root's class, we can flip coins according to the formulas above to determine the children's classes; given these we can generate the grandchildren's classes, and so on.

To initiate the process we must choose a value for the root class. In the case of ad-blocking it might be natural to argue that the root of a web site is surely not an ad. As this is a general model, we are certainly able to choose a prior biased toward one class or another. However, if we wish to minimize the number of free parameters to only θ and ϕ, we can declare the root class to be a function of those already existing parameters. For the root node r, we declare that

$$\Pr[N_r = 0] = \phi/(\theta + \phi)$$

[3]http://dailynews.sina.com

This formula is useful: if x is a child of the root, then

$$\begin{aligned}\Pr[N_x = 0] =& \Pr[N_x = 0 \mid N_r = 0] \Pr[N_r = 0] + \\ & \Pr[N_x = 0 \mid N_r = 1] \Pr[N_r = 1] \\ =& (1-\theta)\frac{\phi}{\theta+\phi} + \phi\frac{\theta}{\theta+\phi} \\ =& \frac{\phi}{\theta+\phi}.\end{aligned}$$

In other words, with this root probability, all nodes in the tree have the same probability of being class 0, prior to labeling any of the leaves. Later, as we label the leaves, these probabilities will change (see the next sub-section).

If θ and ϕ are small, our model asserts that a node is likely to have the same class as its parent, and thus likely to have the same class as its siblings and other nearby nodes in the tree. As the mutation probabilities increase, the correlation between nearby nodes in the tree decreases. We usually constrain θ and ϕ to be less than .5, which indicates the probability of a mutation is less than the probability of a non-mutation.

To summarize the steps in creating a Bayes net out of the given hierarchy:

- set conditional probabilities for all edges in the tree: for node x with parent y,
  $pr[N_x = 1 \mid N_y = 0] = \theta$
  $pr[N_x = 0 \mid N_y = 1] = \phi$
- set the root prior: $p[N_0 = 1] = \theta/(\phi + \theta)$

Next, we discuss how the probabilities in the tree change when leaves are labeled.

3.3 The Learning Algorithm

With this model, in the absence of any evidence, we make the same prediction about every node x: that it has probability $\Pr[N_x = 0] = \phi/(\theta + \phi)$ of being in class 0. Things become more interesting when we are given some *evidence*. Suppose that there is a set of (leaf) nodes E whose classes are known. This will influence our predictions about other nodes: in particular, we want to compute the $\Pr[N_x = 0 \mid E]$ that node x has class 0 *conditioned* on the evidence E. We can expand this conditional probability as

$$\frac{\Pr[N_x = 0 \text{ and } E]}{\Pr[E]}.$$

Note that both the numerator and denominator are simply the probability of some set of nodes (E in the denominator, $N_x \cup E$ in the numerator) getting certain labels. So if we can compute such a quantity for general evidence, we can compute the desired conditional probability. We focus on the computation of $\Pr[E]$.

We use a standard Bayes-net inference algorithm. Suppose that the root r has two children x and y, and let E_x and E_y denote the labeled leaf nodes in the x and y subtrees, respectively. Because of our generative model, given the class of the root r, the classes of nodes in the two subtrees are *independent* of one another, and their probabilities can be multiplied. That is, $\Pr[E \mid N_r = 0] = \Pr[E_x \mid r = 0] \cdot \Pr[E_y \mid r = 0]$, and similarly for the case $N_r = 1$. But recall that to generate the classes in x's subtree given that $N_r = 0$, our first step is to pick a random class for x, and then assign classes to items below x based on the class of x. Formally, we find that

$$\begin{aligned}\Pr[E_x \mid N_r = 0] =& \Pr[E_x \mid N_x = 0] \Pr[N_x = 0 \mid N_r = 0] \\ & + \Pr[E_x \mid N_x = 1] \Pr[N_x = 1 \mid N_r = 0] \\ =& (1-\theta) \Pr[E_x \mid N_x = 0] + \theta \Pr[E_x \mid N_x = 1]\end{aligned}$$

More generally, let us write $p_{x0} = \Pr[E_x \mid N_x = 0]$ and $p_{x1} = \Pr[E_x \mid N_x = 1]$ (note that these two quantities need not sum to 1). Our analysis above, generalized to an arbitrary number of children, says that for any node x,

$$\begin{aligned}p_{x0} =& \prod_{y \in \text{children}(x)} ((1-\theta)p_{y0} + \theta p_{y1}) \\ p_{x1} =& \prod_{y \in \text{children}(x)} (\phi p_{y0} + (1-\phi)p_{y1})\end{aligned}$$

Technically, the product over children is actually limited to children whose subtrees contain nodes from E.

The above algorithm can compute, working up from the leaves, two quantities p_{x0} and p_{x1} at each node x in the tree. At the end of the recursion we have p_{r0} and p_{r1} and the root. At this point we can compute

$$\begin{aligned}\Pr[E] &= p_{r0} \Pr[N_r = 0] + p_{r1} \Pr[N_r = 1] \\ &= p_{r0}\frac{\phi}{\theta+\phi} + p_{r1}\frac{\theta}{\theta+\phi}\end{aligned}$$

During the computation, each node was "inspected" a constant number of times (to produce values for its unique parent), so the entire computation takes time linear in the size of the tree. As discussed above, we can run the computation once for the training data E, and once for the extended data $E \cup \{N_x\}$, in order to compute $\Pr[N_x \mid E]$.

It would be unfortunate if we had to run a linear time (in the total number of nodes) computation to classify each new candidate item, but fortunately we can do much better. Notice that when a labeled item x is added to E, it only influences the probabilities p_{y0} and p_{y1} for nodes y that are ancestors of the new node x (this follows by induction since a node's values depend only on its children's values). Thus, if we compute and store the quantities p_{y0} and p_{y1} for all nodes y in the initial classified training set, then the computation of $\Pr[E \cup \{N_x\}]$ (and from it $\Pr[N_x \mid E]$) can be done by walking up the tree from x. This takes time proportional to the depth of x, which is effectively constant in our applications (few URLs or table structures have depth exceeding 10). Some small care needs to be applied to maintain this constant time bound when nodes have large numbers of children, but it can be done [20].

In summary, with linear storage space, we can hold a data structure for the training data E that lets us

- compute $\Pr[N_x \mid E]$ for any query node x, or
- add a new labeled node x to the training set E

in time proportional to the depth of the tree, which is in effect constant for most trees. Note that in fact, we need only store parameters for nodes that are ancestors of an evidence node (the rest of the tree can be created on the fly as needed).

4. AD-BLOCKING

We now consider an application of our technique to ad blocking. We show how a learner based on tree features can match the performance of a commercial, hand-coded ad-blocker. Intriguingly,

our learner requires *no human training*—instead, it generates its own training data using a slow but simple heuristic based on link redirection.

Ad-blocking is a difficult problem because advertisements change over time, in response to new advertising campaigns or new ad-blocking technology. This makes it quite burdensome to maintain an ad-blocker based on hand-coded rules: when new advertisements arrive, a static ad-blocker's performance suffers; typically engineers must code a new set of rules, and users must periodically download those rules.

As we have mentioned, our approach is to have the computer learn sets of rules; such an approach means less work for the engineers and fewer rule updates for the users. A typical approach would be for engineers to label some small set of training examples, and have the classifier learn a classification rule based on them. An example of this training-based approach is AdEater [13]. AdEater got a 97% accuracy (the only metric reported) on their ad blocking experiments, though it requires 3,300 hand labeled images to achieve that accuracy. AdEater uses humans to perform the relatively simple task of labeling images as ads or not; the machine learning takes on the more difficult task of discovering rules. However, AdEater suffers in the same way that most ad-blocking programs do: it takes considerable human effort and time to produce an update; hence its accuracy decreases as new ad forms surface.

Of course, hand labeling large numbers of images is also time consuming. So we actually go one step further, using *no* human training at all. In order to avoid human training, we use a slow, but reasonably accurate, heuristic to label links on a page as either advertisements or not (of course, AdEater could have used the same training methodology). Instead of using high-quality human data, we use a heuristic called *Redirect*. Redirect labels a link as an ad if fetching the link yields a redirection to another site. The redirect heuristic makes sense because it captures the normal process that advertisers use: tracking the click, then sending the user to the advertised site. Notice that this is much more of a "content" based heuristic than image sizes which are "form" heuristics; ads can take any shape and size, but most current advertisements incorporate some type of tracking mechanism.

As we shall see, the Redirect rule is about as accurate as WebWasher at identifying ads. But there is a big barrier to using Redirect itself as an ad blocker: to decide whether to block an ad, Redirect must fetch it first. This generates a significant overhead in additional connections and downloads—one which in today's network environment makes the Redirect heuristic too slow to use in real time ad blocking. However, we will show that our tree-based learner can predict a redirect without actually trying it—this gives us an approximation to the redirect heuristic that *can* be used for real-time ad blocking.

In practice, click-through tracking requires back-end infrastructure like databases and CGI scripts. Therefore, ads tend to be located together under a small number of URL directories per site (i.e., under xyz.com/adserver). It is rare for a site to have advertisements and content in the same leaves of the URL tree. Therefore, we use our tree learning algorithms to associate existing URLs with an 'ad' or 'not ad' label provided by the Redirect heuristic. As a new page is loaded, the learning model predicts whether new URLs are 'ads' or 'not.' A big advantage of the redirect heuristic is that training examples can be provided *automatically* by an off-line algorithm that, when the user is doing other things, visits sites and takes the time to check and follow redirects. In other words, we can train without any human input.

4.1 Ad-blocking Framework

In order to label the advertisements on a given page, we downloaded the page and saved the HTML into a cached file. The cached file was necessary so that every algorithm would be viewing exactly the same Web page. We went through the cached file locating links on the page. We used a Perl expression to find all links on a page, with a regular expression that matched `<a href= ... /a>`. This corresponds to a standard HTML definition of links on pages.

From the list of links we found on the cached page, we only used links that contained images. This is because, at the time of our experiments, the commercial ad-blocker (WebWasher) only blocked image-based advertisements. To detect images embedded within links, we only took the subset of links whose anchors contained the text `<img`, which is a standard HTML definition for displaying images. For a given page, we call this subset of the links on a page *image-links*.

Given this list of image-links, we used four systems to label whether a link contained an advertisement or not. The four systems are listed below:

• *WebWasher* is a commercial product with handwritten rules that uses many features like the dimensions of the ad, the URL and the text within an image. It is in use by four million users. WebWasher can be downloaded free for educational and personal use[4]. We used the WebWasher Linux version, which was installed as a proxy. The WebWasher proxy reads in the requested Web page, erases the advertisements, then forward the modified page to the Web client.

In order to have WebWasher label a page, we ran our cached file through the WebWasher application. Webwasher removed all the image-based links it deemed advertisements. Any link that was in our original list of image-links but was not present in the WebWasher-processed page was deemed an advertisement by the WebWasher method.

• *Redirect* is the simple heuristic mentioned above that monitors third-party redirects. As we mentioned, it is a somewhat noisy heuristic, meaning its accuracy is less than humans'; but it can run in the background without human input. Redirect used a small amount of Perl code, along with the lynx program [5] to check for redirects that went to top-level domains different from the originating page.

• *Learn-WW* is our tree-based URL algorithm, *trained on WebWasher's output*. The URL provided the tree-structure as discussed in Section 2.1, and WebWasher provided the labeling for the leaf nodes.

• *Learn-RD* is our tree-based URL algorithm, *trained on Redirect*. The method for training Learn-RD is identical to that of Learn-WW, except that the RD (third-party redirect) heuristic was used in place of WebWasher to label the leaf nodes.

As we have mentioned before, RD alone is a good heuristic, but cannot operate in real-time. Our hope is to train a classifier like Learn-RD off-line periodically with RD, allowing for a fast ad-blocker that has been trained by the RD heuristic. We might also hope that the learner could find a general rule that ignored certain RD errors so became more accurate than the heuristic.

4.2 Testing and Training Data

In the previous sub-section we described a variety of algorithms designed to label whether the links on a page contained advertisements or not. In this section, we describe how we acquired training and testing data.

[4]http://www.Webwasher.com/client/download/private_use/index.html
[5]http://lynx.org

	Top 25 Sites	weather	look smart	euniverse
Web-Washer	.935 (.136)	.907	.857	.517
Learn-WW	.931 (.118)	.860	1	.517
Redirect	.933 (.08)	.842	.857	1
Learn-RD	.934 (.08)	.837	1	1

Table 1: Shown are the Web blocking accuracies for several Web sites, along with standard deviation information for the top 25 Web sites.

We generated a dataset from Media Metrix's largest 25 Web properties as of January 2002[6]. Empirical evidence shows the average user spends all their time on a small number of the largest sites. We felt that blocking ads of the largest 25 Web properties would be both representative and beneficial to many, if not most, users.

We crawled through a given Web site, randomly picking eleven pages linked from the front page that shared their top-level domain with the front page.

The links on those eleven pages, along with the links on the front page, were divided randomly into a six-page training and a six-page test set. Each Web site went through the random training and testing twice. WebWasher and Redirect classified each linked image in the training group, and those classifications, along with the link URL, were used to train Learn-WW and Learn-RD respectively.

Next, all four classifiers were applied to the linked images on the test group. If all four classifiers agreed that an image was either an ad or all agreed it was not an ad, they were all deemed to have classified the image correctly. Spot-checks suggested that agreement between all methods almost always lead to the correct prediction. If one technique disagreed with the others, we manually judged whether the image was really an ad or not.

4.3 Experimental Results

In total, 2696 images were classified, and all four classifiers ended up with an average classification accuracy across all sites of within a quarter percent of 93.25% (see Table 4.3). The first column contains average error rates for the 25 sites, with standard deviations in parentheses. Standard deviations are based on site-to-site comparisons. The overall false negative rates (labeling an ad as content) and false positive rates (labeling content as ads) were approximately 26% and 1%, respectively, and that was fairly consistent between all the classifiers. Note that the mistakes were biased toward false negatives, which means the classifiers let through some ads, but rarely blocked content. This is probably the appropriate behavior for an ad-blocker.

Table 4.3 also shows data for specific sites. On various sites, trainers beat learners (weather.com); and learners beat their trainers (looksmart.com). Thus learners are not exact imitations of their trainers, but on average end up with the same accuracy rates.

The standard deviations of the four classifiers are relatively large because errors tend to cluster around a few Web sites. WebWasher, for example, does poorly marking ads on euniverse.com, which uses different dimensions for its ad images than many sites. Redirect does poorly on portals and internal ads. For example, Redirect would not label a *New York Times* advertisement for its own newspaper as an advertisement.

It is possible for the learners to generalize to better performance than the trainers. For example, all of the New York Times advertisements are in a few URL directories. Thus, a few incorrect examples from Redirect are ignored in favor of the larger number of correct examples; the learner correctly "overrides" Redirect. Conversely, the learners can also generalize incorrectly; if the trainers are a little more wrong than right, the learners end up generalizing in the wrong direction and consistently mis-labeling the links. Such problems happened on the weather.com Web site.

The Redirect heuristic worked well; its accuracy was not statistically different from the commercial ad-blocking program. Redirect's simplicity suggests that, for now, that it is correctly understanding the mechanisms by which most sites serve up ads (i.e. through third party redirects). That mechanism can certainly change, but such changes would be harder for advertisers to make than simply changing the advertisement sizes, which can foil WebWasher at times.

The URL tree seemed to be a good feature for ad-blocking. Both WebWasher's and Redirect's rules seemed to be captured in many cases by the presence or absence of a single specific URL prefix. Had the URL been a poor feature, one would expect the tree-learning based ad-blockers to form more complex hypotheses and have much lower accuracies.

A spot-check of random sites (using a random link generator like www.mangle.ca) suggests that the URL feature works even better on smaller sites than larger sites, because smaller sites tend to use third party advertisers whose URLs are almost entirely used to serve advertisements, like doubleclick.com.

We wish to make a few points about our results. First, our Learn-RD algorithm achieved performance comparable to a commercial ad-blocker, without needing complex, hand-written rules. In fact our system performs better in some respects. Most ad-blocking systems do not remove text-based ads (ads, for example, placed within search engine results), while both Redirect and our learner trained on Redirect acts no differently for image-based ads and text-based ads. Second, we argue that our ad-blocking classifier will adapt in the long term better than a static version of WebWasher, since it can update rules nightly without any human input. Of course, in the adversarial world of advertisements, it is probable that this system, too, would be defeated if it became widespread: an advertiser could deliberately obfuscate their URLs. Third, we point out that unlike the black and white "redirect" heuristic, our learner gives pages a range of "blackness" scores. It can thus be tuned to trade off false positives and false negatives depending on the user's preference.

5. RECOMMENDATION EXPERIMENTS

In the previous section on Ad blocking, we showed that machine learning on tree-based features could match or outperform hand-coded rules (WebWasher) and an inefficient heuristic (Redirect). In this section, we demonstrate that tree-based learning can also outperform classification algorithms based on traditional textual (and other) features. As we mentioned in the introduction, our goal is to build real-world Web systems that save time and effort on behalf of users. One of those systems was a recommendation system designed to find interesting links customized to individual users. In this section, we describe the system and discuss experiments we performed to judge the effectiveness of such a system.

We ran a user study on 176 Web participants to get real news story data from users. We asked participants to click on any stories they normally would read, and conducted experiments into the best algorithms for predicting test clicks on new pages given training clicks on previous pages.

Overall our two tree algorithms performed well, even against the support vector machine which is commonly thought of as one of the best general-purpose learning algorithms. We showed that our features, alone or coupled with our algorithms, can increase the accuracy of classification. In the next sub-section (5.1) we describe

[6]http://www.jmm.com/xp/jmm/press/MediaMetrixTop50.xml

the set-up of the user study, followed by a sub-section (5.2) on the algorithms we tried and their advantages and disadvantages. That is followed with empirical results and analysis of our algorithms (5.3).

5.1 User-Study Set-Up

We performed a study involving 176 users. Users were presented with 5 pages containing links to stories, and asked to click on the links they would normally click on. The pages were visually unaltered replicas of pages downloaded from the Web, consisting of 5 consecutive days worth of *The New York Times* front page (September 15th through 19th, 2003). The pages and user study are available from our server[7]. Based on the submitted email addresses of the participants, the users encompassed a broad range of people from throughout the world.

In order to encourage a large number of participants, we made the experiments easy to complete and gave financial incentives for doing the experiments. The experiment was entirely conducted in one session on the Web, by using cached copies of the five pages mentioned above.

Some users did not complete the study, and a small number of others did not click on any links; these were discarded from the study and are not considered part of the 176-user sample set.

Each click on a link within the user study would place a checkmark in a check-box corresponding to that link, and when the user submitted the page, a list of all the clicked (and un-clicked) links was noted on the server. The clicked (and un-clicked) links were used to generate positive and negative examples with a variety of features (next subsection) which were then made available to various learning algorithms for prediction of unseen clicks.

We chose *The New York Times* as one of the most popular news sites in the world, meaning that the content of those pages was broadly focused. We believe it is a representative of news sites that many users read.

5.1.1 Features

For each link on each page, we collected a variety of features for use in our algorithms, to the extent practicable. For each link, we have a copy of the link's URL, the anchor text of the link, the position of that link in the table structure of the page, and the full text of the article. In total there were 1105 links across the five pages[8]. There are some difficulties with collecting the data for every link, particularly the textual data.

For example, not every link has anchor text inside it, since some links are only images. Therefore, when available, we took the "alt" tags to stand for the text. Even so, there are several image-only links that have no available anchor text. In total, approximately 1% of the links did not have anchor text we could parse within them. We will refer to data based on the anchor text in the link as "Anchor."

It is even more difficult to fetch all of the pages behind the links. For example, many sites (including the ones we chose) only allow registered users, which generally means that the fetching agent needs to understand and respond to cookies. Harder still is following the many types of redirects and translating properly between absolute and relative links. Also many pages (again, including the pages we chose), have Javascript links that require a Javascript interpreter to understand. We used Lynx as a browser, and wrote special routines to follow redirects, to automatically log in to the site via cookies, and to translate between absolute and relative links. While not perfect, we believe we gathered as many of the target pages as was reasonably possible. We did not follow Javascript-based links because we were not aware of any reasonable way to place a Javascript parser within Lynx.

There is also a problem of rights for visiting Web sites. Many Web sites (including those we chose) do not allow spiders on the pages underneath the home page. Also, standard etiquette says you may not download more than one page every five seconds. We followed the second rule but not the first since the first makes it impossible to grab the full text of articles. One occasional consequence of having a slight delay between downloading Web pages is that sometimes the pages disappear before downloading. Though we tried to get as much of the pages as possible, around 5% of them were not download-able for the reasons given above. We will refer to the data based on the full text of the fetched document as "Doc."

The URL of the link is comparatively easy to read, as it is a pre-requisite to having a usable link (and a pre-requisite to being able to download any pages, for both our robot and for a user). The table structure the links sits in is slightly more difficult, because it requires parsing the page into table elements, but every link has a position in the table structure. Both of these datasets were complete: i.e., for every link a person could click on, there is a corresponding URL "URL" and table element "Table" that link sat within.

To summarize, there were four basic features used:

Anchor: the anchor text within the link.

Doc: the full text of the words in the linked document

Url: the URL in a form that retains its path through the tree

Table: the location of the link in the table, in a form that retains its path through a tree.

5.2 Recommendation Algorithms

We tried several algorithms with various datasets labeled with each user's click data. That is, for each of the 1105 data points, 176 different combinations of positive and negative labellings were found based on the user's empirical news selection.

We used two "general purpose" classifiers, Naive Bayes "NB", [8] and the support vector machine "SVM" [6], across all four of the feature sets, plus our tree learner "TL" on the two tree-structure features. That is, some of our feature/classifier pairs gave the SVM and NB classifiers the *same* URL and Table features the tree algorithm used.

We choose the SVM because it is considered to be one of the best general-purpose discriminant classifiers [11]. We used a standard, fast, publicly available implementation called SVMFu [19].

In order to give the various non-tree algorithms (SVM, Naive Bayes) a chance to learn based on the URL and Table tree-features, we took each node from the tree and translated it into a 'word' that contained information about the path to that node. For example, the URL `http://nytimes.com/business` was tokenized into three tokens: `http://`, `http://nytimes.com`, and `http://nytimes.com/business`. We chose this "nested" tokenization instead of the obvious splitting up of the URL into "words" so that the representation would still convey the exact position of a node in the tree, giving the non-tree algorithms the opportunity to generalize in the same way that our tree-algorithms might. In particular, two nodes that are distant in the tree but happen to have the same words will, under our scheme, still have completely different tokens. We believe that this hierarchical tokenization gives the non-tree algorithms a fair chance to exploit the same features as our tree. For example, if all "relevant" links are in a specific subtree, the SVM can learn that class by identifying and classifying based on the token representing the root of that subtree.

[7] http://daily-you.csail.mit.edu/data3

[8] The raw recommendation data and features are available at http://www.ai.mit.edu/ kai/recommendation_dataset.txt

Classifier	Top Rec	Top 3 Recs	Top 5 Recs	Top 10
1. Random	29	104	162	330
2. Perfect	857	2488	3899	6093
3. NB-Doc	81	232	342	594
4. NB-Anchor	302	713	1021	1615
5. NB-Table	72	180	278	576
6. NB-Url	80	319	507	1036
7. SVM-Doc	59	156	257	520
8. SVM-Anchor	220	614	913	1438
9. SVM-Table	177	472	662	934
10. SVM-URL	308	839	1268	1953
11. TL-Table	**401**	900	1176	1512
12. TL-URL	385	**979**	**1388**	**2149**

Table 2: Summary of all the classifiers and features on *The New York Times* datasets. The numbers represent the number of clicked recommendations each classifier would get, if they (columns) recommended the top, top three, top five, and top ten highest scoring links. Bolded items have higher accuracies than non-bolded items for a given column (statistical significance methodology is reported in the Appendix).

5.2.1 Test Environment Parameters

Overall we tried a wide variety of algorithms on a cross-validated set of *The New York Times* data. From our five documents, we used the links on 4 of them for training and the links on the remaining one for testing, across all of the 176 users. In total, this corresponds to 182,325 classified links per experiment (1105 *Times* links x 176 users).

We tried a total of 10 different algorithms corresponding to mixtures of different algorithms and features:

Support Vector Machine (C=1): SVM-Anchor SVM-Doc SVM-Url SVM-Table

Naive Bayes ($\alpha = 1$): NB-Anchor NB-Doc NB-Url NB-Table

Tree Learning ($\theta = .2$ $\phi = .05$):TL-Url TL-Table

The algorithms we used required specific parameter settings that we describe here so the experiments can be replicated. We chose Naive Bayes because it is a fast, common model-based classifier whose model does not work well with tree structures. Specifically, NB assumes independence between features whereas a tree implies certain strict dependencies. For NB we used an α smoothing parameter of 1 [8].

The SVM required the setting of a "C" parameter that measures the outlier penalty. We used a smaller portion of the test set (2 pages) to determine an optimal C parameter for the four different feature sets. We tried C values of 1, 3, 5 and 10. Across the four feature types (Link, Doc, Url, Table), C did not substantially change the results, so we used a C parameter of 1. Other parameters included setting the number of cache rows to 3000 (this changed speed but not accuracy) and setting the kernel to linear. Both the kernel and the data were represented as floating point numbers.

We used our Tree Learning (TL) algorithm on the two tree-structured features, Url and Table. We set the backward mutation rate to .2, and cross validated the forward mutation rate on the same 2-page training set we used for the SVM. We tried forward mutation rates of .05, .1, .15, .2, .25, and .3. In general, lower forward mutation rates worked better, so we used .05 in successive experiments.

5.3 Recommendation Results

On average, users selected nine links a day (out of an average of 221 possibilities each day). Each of the classifiers produced a ranked list of recommendations for each user and each page (one page represents one day of stories). Results are written in terms of the number of correct recommendations across all users and links within the top recommendation, top three recommendations, top five recommendations, and top ten recommendations. Complete results are shown in Table 2. A perfect classifier (second row) would have achieved 857 correct recommendations when producing one recommendation per day per user (176 users times 5 days; a small number of users did not click on any recommendations for some days). A random classifier on the other hand, would have done a (statistically) significantly worse job than any of the trained classifiers. Statistical significance is reported in terms of a non-parametric statistical test found in Hollander and Wolfe [10] and detailed in Shih [20].

Overall the best performing algorithm was the tree learning algorithm applied to the URL feature, which had the best scores for three out of the four categories. The second best algorithm was a tie between the SVM applied to the URL feature and the tree learning algorithm applied to the table feature. Each of these performed well in different categories.

Rows 3 and 4 (NB) and 7 and 8 (SVM) shows a comparison between two of the text features (Anchor and Doc) against two algorithms (NB and SVM). In both cases, the anchor text provides a much better feature than the fetched document's text. This may be because the anchor text is supposed to contain a summary of the document, and it is easier for the classifier to understand these summaries than the documents themselves (perhaps the summaries use a smaller feature set than the full documents so require less training data). This is a positive result for text classifiers because the anchor text is much easier to fetch than the target document's text (which, as mentioned before, can be slow and can be difficult technically to download). The differences are all statistically significant.

Rows 5, 9 and 11 (Table feature) and 6, 10 and 11 (URL feature) shows the difference between the various algorithms on the tree structured features. As mentioned previously, we 'tree-ified' the features for the benefit of NB and the SVM so each algorithm would be aware of the position in the tree. The Bayes-net algorithm does the best job of taking advantage of the tree structured features. This may be partially due to its domain knowledge of the problem, coupled with a relatively small amount of training data. In those situations, model-based classifiers often perform better than discriminative classifiers [16]. This is particularly true on the table features, which tend to be much flatter than the URL: the depth of the table tree is much more constant than the depth of the URL tree. The next best performing algorithm is the SVM, which generalizes well considering it has no domain knowledge. The worst classifier on tree-structured features is NB. We believe this is because NB assumes independence between features while a tree actually generates highly dependent features (a given child always has the same parent). The differences for 3, 5 and 10 recommendations are all statistically significant between the classifiers.

Rows 4, 10 and 12 shows the best feature given the various algorithms on our data sets. Naive Bayes needs to use the text-based features (since it is unsuitable for the tree-based features), and as discussed before, the best text-based feature was the anchor text. Both the SVM and the tree-learner did best on the URL feature, suggesting that the URL feature is a good feature for use on recommendation problems like the one we posed. The differences for 3, 5 and 10 recommendations are all statistically significant between the classifiers.

An important caveat in our results is that our system was aiming to learn what links a user *would click on*, not what stories a user *would enjoy reading*. It is perhaps unsurprising that the anchor text

is a better predictor of user clicks than the body, since the anchor text is what users actually see when making their click decision. It is an interesting open question to evaluate the ability of our tree learner to identify stories the user would enjoy reading; this evaluation however would involve significantly more overhead since it would require test subjects to actually read all the articles being consider, rather than just reading their anchors.

6. CONCLUSIONS

We began with the observation that the proper choice of features can have a significant impact on the performance of classification algorithms. Since Web site authors have an incentive to organize their materials, for the sake of both the author and their audience, we hypothesized that the URL of documents and the physical placement of elements on a page could provide clues into the Web site's fundamental organization.

To take advantage of this observation, we noted that URLs and table structure can both be viewed as trees, and this facilitated certain machine learning algorithms. These learning techniques try to find correlations between certain properties (i.e., this document is an ad) with the document's location in either the URL or table tree.

We argued that our new features and learning would produce fast, good classification schemes. We gave empirical results on two Web applications: an ad-blocker and a recommendation system. We showed that the ad-blocker could achieve commercial-grade accuracy without requiring any human inputs. The recommendation results showed that our tree-learning approach outperformed conventional techniques and features on real world news recommendation data.

Our tree-based algorithms exhibit a well-recognized tradeoff between specificity and accuracy. The text-based classifiers we compared our work to are very general: the set of words that characterizes documents "interesting" to a given user is not (very) specific to any particular site. Our URL classifier is site specific—what it learns about the URLs of ads on one site will not generalize to other sites. Thus, it needs to train separately on each site, and will make no useful recommendations until such training is complete. Our table-based classifier is even more specific, as it focuses on the layout of a specific page, and requires training data specifically from that page in order to make recommendations.

7. ACKNOWLEDGMENTS

This work was supported by the MIT Oxygen Partnership. We thank the early users of the system for their feedback on the system, Mike Masnick for his remarks on user interface, Mark Ackerman for his help editing earlier versions of this document and Leslie Kaelbling and Jason Rennie for their help on the machine learning aspects of the system.

8. REFERENCES

[1] R. Agrawal and R. Srikant. On integrating catalogs. In *Proceedings of 10th Intl. Conference on the World Wide Web*, pages 603–612, Hong Kong, CN, 2001. ACM Press, New York, US.

[2] C. R. Anderson and E. Horvitz. Web montage: a dynamic personalized start page. In *Proceedings of the Eleventh Intl. Conference on World Wide Web*, pages 704–712. ACM Press, 2002.

[3] R. Barzilay, N. Elhadad, and K. R. McKeown. Inferring strategies for sentence ordering in multidocument news summarization. *Journal of Artificial Intelligence Research*, 17:35–55, 2002.

[4] D. Billsus and M. J. Pazzani. A hybrid user model for news story classification. In *Proceedings of the Seventh Intl. Conference on User Modeling*, pages 99–108. Springer-Verlag New York, Inc., 1999.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[6] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[7] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *Proceedings of the Eleventh Intl. Conference on World Wide Web*, pages 148–159. ACM Press, 2002.

[8] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley and Sons, Inc., 1973.

[9] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36(2):177–221, Sept. 1988.

[10] M. Hollander and D. A. Wolfe. *Nonparametric Statistical Methods*. John Wiley and Sons, 1973.

[11] T. Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. In *Proceedings of the Fourteenth Intl. Conference on Machine Learning*, 1997.

[12] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *Proceedings of the 14th Intl. Conference on Machine Learning*, pages 170–178, 1997.

[13] N. Kushmerick. Learning to remove internet advertisement. In O. Etzioni, J. P. Müller, and J. M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 175–181, Seattle, WA, USA, 1999. ACM Press.

[14] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[15] A. K. McCallum, R. Rosenfeld, T. M. Mitchell, and A. Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In J. W. Shavlik, editor, *Proceedings of the 15th Intl. Conference on Machine Learning*, pages 359–367, Madison, US, 1998. Morgan Kaufmann Publishers, San Francisco, US.

[16] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems 14 (NIPS*01)*, 2002.

[17] M. J. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331, 1997.

[18] J. Rennie and A. K. McCallum. Using reinforcement learning to spider the Web efficiently. In I. Bratko and S. Dzeroski, editors, *Proceedings of ICML-99, 16th International Conference on Machine Learning*, pages 335–343, Bled, SL, 1999. Morgan Kaufmann Publishers, San Francisco, US.

[19] R. Rifkin. Svmfu. http://five-percent-nation.mit.edu/SvmFu/, 2000.

[20] L. K. Shih. *Machine Learning of Web Documents*. PhD thesis, Massachusetts Institute of Technology, Feb. 2004.