massachusetts institute of technology — artificial intelligence laboratory

Learning Classes Correlated to a Hierarchy

Lawrence Shih and David Karger

AI Memo 2003-013 May 2003

Abstract

Trees are a common way of organizing large amounts of information by placing items with similar characteristics near one another in the tree. We introduce a classification problem where a given tree structure gives us information on the best way to label nearby elements. We suggest there are many practical problems that fall under this domain. We propose a way to map the classification problem onto a standard Bayesian inference problem. We also give a fast, specialized inference algorithm that incrementally updates relevant probabilities. We apply this algorithm to web-classification problems and show that our algorithm empirically works well.

Work funded by the Oxygen Partnership, NTT, and the Packard Foundation.

1 Introduction

Hierarchies are constructed to organize and categorize similar items into common areas of a tree. The semantics of a strict hierarchy are that all elements in any sub-tree share some attribute, while the remaining elements do not have that attribute.

The attributes in a hierarchy often convey human-oriented meanings beyond that of the attribute itself. For example, consider the biological hierarchy of organisms. One sub-tree might be full of animals with the attribute "dog." That attribute "dog" also comes with many characteristics that are not direct attributes of the hierarchy: "have four legs" or "my friend likes them." These characteristics are not exclusive to dogs, and are not attributes of the hierarchy, but are correlated with the location of an element within the hierarchy. "Four-legged animals" are probably in a small number of subtrees in the organism hierarchy. This paper is about learning those characteristics that are correlated with a given hierarchy.

We pose our research as a machine learning problem: given that certain organisms in a hierarchy have four legs, predict which other organisms in the hierarchy have four legs. We propose a Bayesian model of the problem, which reflects intuitions about hierarchies and their correlated characteristics. We follow with an efficient algorithm for performing inference on this problem.

Hierarchies are extremely common organizational tools, so there are ample sources of hierarchical data. Corporate structures, organisms and web directories are examples of diverse domains that are frequently organized into hierarchies. A tremendous amount of work has been invested by human beings in creating hierarchies that capture important attributes of data. While it is clear that hierarchies are common, it is worthwhile emphasizing why they are a particularly good source of data. Most hierarchies are created by humans, for humans, for the express purpose of organization and classification. We can take advantage of these existing hierarchies to improve classification on problems that correlate with the hierarchy.

We have applied our algorithm to some web classification problems. We show how the URLs of web documents can be viewed as a hierarchy and show how our algorithm can learn to automatically distinguish between advertisements and content links. Our algorithm achieves accuracy comparable to a commercial tool. Unlike other systems, including the commercial tool, our ad-blocker can function without human input or labeling, meaning it can adapt to new advertisements (e.g., text ads, or new banner sizes) without the need for time-consuming hand-made software updates. We also briefly discuss the Daily You, a web application that uses the classifier in this paper to recommend interesting web pages to a user.

1.1 Related Work

Hierarchies already play a large role in machine learning. For some, the learning problem is the structure and placement of nodes in the tree itself, such as the sub-field of decision trees (Quinlan, 1987). Others use a pre-defined hierarchy, like the Yahoo! directory, and try to place objects, like a web page, within them (McCallum et al., 1998). Still others use hierarchies to boost an existing algorithm's performance by creating simpler sub-problems according to the hierarchy (Koller & Sahami, 1997). A theoretical examination of hierarchical features in the context of VC dimension is found in Haussler (Haussler, 1988).

Our work is different from the past work. We are given a tree. Like Haussler, we specify an object's position in a tree, and assume that the position in the tree is relevant to our classification problem. Unlike Haussler's work, our contribution is to

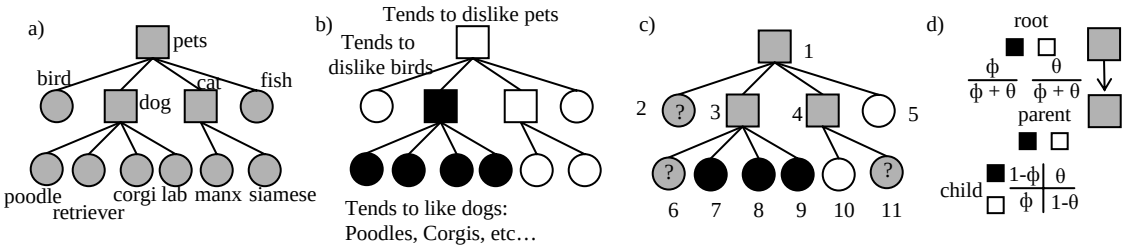

Figure 1: Various stages in the conversion from hierarchy semantics to machine learning problem. A) Hierarchies have semantics that a node is in a subtree iff it has a certain attribute (attributes in labels next to nodes). B) Some characteristics, like "my friend tends to like this type of pet" may be correlated with the hierarchy. Here we show one person's true preference: they dislike pets in general (white nodes), but like most dogs (black nodes). The edge between "pets" and "dogs" is a mutation: where a general dislike for pets becomes a general liking for dogs. C) The machine learning problem is to learn the person's pet preferences (coloration of unlabeled leaves), given that they like various species of dogs (black leaves), but dislike fish and various species of cats (white leaves). D) The formal model, prior to any evidence, sets prior probabilities at the roots and "mutation" probabilities which set the conditional probability tables between elements.

create an appropriate Bayesian model that captures our intuitions about hierarchies, and to provide an efficient algorithm for training and testing over that model.

Many other machine learning techniques are based on probabilistic models, most famously the naive Bayes model (Duda et al., 2000). The naive Bayes model explicitly assumes that features, like sets of words in a document, are conditionally independent from one another, given the class. In the next section, we describe our model of tree learning.

2 A Generative Classifier for Trees

Hierarchies are trees whose nodes have special semantics. In Figure 1a), we show a portion of the pet hierarchy. The meaning of the sample hierarchy is familiar: we can see that every dog is a pet, and every Corgi is a dog, but that no dogs are cats. A node is in a subtree if and only if it shares an attribute with all the other nodes in that subtree. These attributes are taken to be human-oriented and often human-created. The figure places the attributes for the pet hierarchy next to the node that roots the subtree.

Suppose the hierarchy is a tree with m nodes, $N_1 \ldots N_m$, such that each node except the root has exactly one parent. We adopt the convention that parent nodes will have lower-numbered indices than children nodes, so the root is always N_1. Semantically, each node i in a hierarchy specifies some attribute A_i. Any node in the subtree rooted by N_i has attribute A_i, and any node outside of that subtree does not have attribute A_i.

It is trivially possible to construct a hierarchy. For example, a different pet hierarchy might be based on the length of the animal's scientific name. However, most hierarchies are constructed with the intention of conveying human-oriented information beyond the attributes themselves. Dogs have certain common characteristics that most humans can readily define and describe: they are loyal, bark, have four

legs and mostly have good senses of smell.

These characteristics are correlated with the hierarchy: if we know the attribute "dog" we can infer most sub-trees of dog have the "four-legged" characteristic. They are not attributes of the hierarchy: both cats and dogs are four-legged. They are still correlated; we expect siblings on the tree to have a tendency towards having (or not having) four legs. Many types of characteristics might be correlated with a hierarchy. The attribute might be subjective and idiosyncratic: "do you like this type of animal?" Or, it might be objective but in multiple portions of the hierarchy: "Does this animal usually have four legs?"

Figure 1b) shows a mapping from the original pet hierarchy onto a human-oriented problem of subjective preferences. This shows one person liking most dogs but disliking most other types of pets. Other people would have different pet preferences, with black and/or white nodes possibly spanning more than one sub-tree.

Assume we are investigating a characteristic $y \in 0, 1$ that is correlated with the hierarchy. The internal nodes of the figure define whether an attribute A_i found at node N_i is correlated with (black) or against (white) the characteristic y_i. For example, this person does not generally like pets, so a white color is placed at the root. Correlations can "mutate" between a parent and child attribute: this person does like dogs, while not generally liking pets.

2.1 Learning over Hierarchies

In this paper, our goal is to predict whether an element has a characteristic based on its position in a hierarchy. We will formulate this as a standard machine learning problem. A learning algorithm receives a set of nodes in the original hierarchy, along with labels indicating whether it has the chosen characteristic. We want to predict whether other unlabeled nodes have that characteristic.

Figure 1c) gives a simple example of a machine learning problem in this domain. In keeping with our assumption that attributes are correlated with the characteristic (e.g., elements close to one another in the tree are likely to have the same labels), a plausible solution would be to label N_6 black and to label N_{11} white. Our shorthand notation for representing that a node i has a label k will be $N_i = k$. We want to predict the labels for unlabeled test nodes.

2.2 Vector Models for Classification

One potential way to solve the tree-learning problem is to represent elements of the domain in a vector space, and to attack the problem with standard machine learning algorithms. However, as we will show, the generality of these methods means they cannot take proper advantage of the given tree structure.

To formulate this problem in a vector space, we would be given the tree structure along with n pieces of training data, corresponding to leaf node indices $(x_1 ... x_n)$ with their corresponding labels $y_1 ... y_n \in 0, 1$.

In a vector representation, one would want to maintain information about all the attributes that a particular node has. Thus, a simple vector representation would be a bit-string indicating which attributes, $A_1 \ldots A_m$, a particular leaf has. The bit-string would then have a value of "1" at position i if the path from the root to the chosen leaf included node N_i (and hence attribute A_i), and a "0" otherwise.

Consider the sample problem in Figure 1c). If we use a standard linear classifier, like the linear-SVM or naive Bayes, the final decision rule can be written in terms of a

weight vector and a scalar threshold. Decisions are made by taking the dot product of the weight vector and the test bit-string and comparing it with the threshold. Consider the test example for N_2, whose bit-string is 1100000000, indicating it passes through nodes one and two. The bit-string renders most of the weight vector meaningless; only positions one and two are non-zero. However, nodes one and two have no information in them. The weight vectors for nodes one and two are, respectively, nodes that incorporate every training path and no training paths. Thus the dot-product of the weight vector and the bit-string does not have information relative to our specific test problem; the same is true for the unchanging threshold constant. The best the classifier can do is predict a label equivalent to the majority label of *all* the training points; but this decision rule can be wrong. In our example, there are more black leaves than white, but the fact that N_2 is a sibling of N_5 suggests we might want to label N_2 as white.

2.3 A Model of Characteristic in a Hierarchy

One challenge for this paper is to create a formal model that reflects our intuition that the hierarchy's attributes are correlated with our chosen characteristic, and hence that nearby nodes in the tree should have similar characteristics. Our translation between intuition and model will consist of steps that will convert the tree of nodes into a Bayesian network.

The random variable for a node i in our problem space is the tendency for elements with attribute A_i to have a characteristic $j_i \in 0, 1$. Tendency is the appropriate word here: we are interested in problems where the attribute is correlated with the characteristic, not where the attribute absolutely implies the characteristic. The canonical example in the AI literature is that "birds tend to fly," but "penguins do not fly." In order to capture those nuances of differences between parent and children, we need to use the word "tendency."

The way we formalize this concept of tendency in the model is to use the notion of mutations. Mutations are areas of the tree in which one attribute is positively correlated with our characteristic, but the parent's attribute is negatively correlated, or vice versa. Mutations are generally a rare event, making parent's and children's correlation with the chosen characteristic as similar as possible. Suppose that N_i is N_j's parent, then:

$$p(N_j = 1|N_i = 0) = \theta; 0 < \theta < \frac{1}{2} \tag{1}$$

$$p(N_j = 0|N_i = 1) = \phi; 0 < \phi < \frac{1}{2} \tag{2}$$

Thus the probability of a "forward mutation"—shifting from a negative to a positive correlation between attributes and a characteristic is θ, and the probability of a "backward mutation" is ϕ. Both parameters have a probability below $\frac{1}{2}$, which means that mutations are a rare event.

The ratio of forward to backward mutations has an analog with diffusion models: the non-evidence probability of a deep tree leaf having a characteristic is exactly $\theta/\phi + \theta$. If we had enough data, we might choose to estimate θ and ϕ based on that data. This also suggests a natural probability for the root's prior. If we set $p(N_1 = 1) = \frac{\theta}{\phi+\theta}$ then any node in the tree will have a probability of having the chosen characteristic of $\theta/\phi + \theta$, prior to any evidence.

In standard Bayesian terms, evidence nodes are those nodes whose characteristics are given to us. Thus, our goal is to predict whether node i has a characteristic, given the evidence nodes: $p(N_i = 1|evidence)$. We use maximum likelihood to predict the

presence of the characteristic: if $p(N_i = 1|evidence) \geq \frac{1}{2}$ then we predict it does; otherwise we predict it does not. Of course, other thresholds may be preferable, for example, to minimize the number of false positives.

At this point, we have converted the original, intuitive problem into a formally specified Bayes net with simple assumptions as outlined in Figure 1d).

2.4 A Generative Classifier for Trees

In order to classify a new query, we simply need to do the standard inference procedure, which is to calculate $argmax_c\ p(N_i = c|evidence)$, the probability of having a characteristic given the evidence.

In this section, we will discuss a fast algorithm specialized for several unusual aspects of our learning problem. First, new nodes—evidence or queries—might appear at any time, which is atypical in Bayesian networks. For example, a new species might be discovered, or a web site might add new page. Therefore, our focus is on fast incremental updates rather than one slower pass across an entire fixed network. Second, hierarchies typically have a large number of nodes, but a relatively small (or constant) depth. Our algorithm therefore provides an update rule that is proportional to the depth of the tree.

Two common inference procedures are described in Russell and Norvig (1994). Backward chaining is simpler, and recursively moves from the query node outwards to the evidence nodes. Forward chaining is more complicated, and moves from the evidence nodes to the query node, caching reusable probabilities. Our algorithm is similar to forward chaining, but is specialized for the classification problem we have presented.

The main unit of calculation we will use is the probability of a collection of evidence; we can also use this to calculate conditional probabilities. Let E_i indicate the evidence nodes in the sub-tree rooted at node N_i, and let $p(E_i)$ be the probability of those evidence nodes. We will repeatedly calculate the conditional probability of the evidence given a node's value. We use shorthand to represent this:

$$p_{ik} \equiv p(E_i|N_i = k) \tag{3}$$

where i indexes the node, and k can be 0 or 1 corresponding to correlations of attribute A_i with the characteristic. The independence assumptions embedded in the Bayesian network, gives us the following recursive relationship by summing over values of the children:

$$p_{i0} = \prod_{j \in \text{children(i)}} \sum_{k \in 0,1} p_{jk} p(N_j = k|N_i = 0) \tag{4}$$

$$p_{i1} = \prod_{j \in \text{children(i)}} \sum_{k \in 0,1} p_{jk} p(N_j = k|N_i = 1). \tag{5}$$

We can expand this out replacing the conditional probabilities with the mutation parameters:

$$p_{i0} = \prod_{j \in \text{children(i)}} p_{j0}(1 - \theta) + p_{j1}\theta \tag{6}$$

$$p_{i1} = \prod_{j \in \text{children(i)}} p_{j0}\phi + p_{j1}(1 - \phi). \tag{7}$$

If we recursively move from the leaves to the root applying this recursive definition, at the root N_1 we will have calculated p_{10} and p_{11} or the probability of all the

function $Calculate\_Evidence(\vec{L}, tree)$

1: Initialization: mark all evidence nodes found in $\vec{L}$
2: Upward pass from leaves to root:
3: **for** node index $i = max(i)$ to 1 **do**
4: **if** Node N_i has a descendant (other than itself) with evidence **then**
5: calculate and save p_{i1} and $pi0$ (see eq. 6, 7)
6: **end if**
7: **end for**
8: return $p(E_1) = \frac{p_{10}\phi}{\phi+\theta} + \frac{p_{11}\theta}{\phi+\theta}$

evidence conditioned on the root's prior. We multiply out the prior, and get the probability of all the evidence E_1:

$$p(E_1) = p_{10}p(N_1 = 0) + p_{11}p(N_1 = 1) \tag{8}$$

$$= \frac{p_{10}\phi}{\phi+\theta} + \frac{p_{11}\theta}{\phi+\theta} \tag{9}$$

The pseudo-code for calculating the probability of all evidence is below.

Suppose that an application wants to query a node. We have already calculated the evidence, so it's easy to use this form of the conditional:

$$p(N_t = 1|E_1) = \frac{p(N_t = 1, E_1)}{p(E_1)}. \tag{10}$$

We could calculate the numerator, $p(N_t = 1, E_1)$ using the Calculate Evidence procedure, which would require another full pass through the tree. But notice that calculating $p(E_1)$ and $p(N_t = 1, E_1)$ are very similar; we can re-use some of the saved data to make the algorithm more efficient.

Recalling that $p(E_i)$ is the probability of evidence that are descendents of node N_i, the only values of p_{i0} that require updating are when the test node t is a descendant of i. In other words, we can update the tree's probabilities by simply updating all the probabilities on the path between the test node and the root.

The updated probabilities p'_{i0} and p'_{i1} can be written in terms of the cached probabilities from the Calculate Evidence procedure, combined with the contribution stemming from new node t. Let g index nodes along the path from N_t's parent to N_1 (the root) and let N_j be N_g's child. Then the algorithm moves from leaf to root with the following update rule:

$$p'_{g0} = p_{g0}(p'_{j0}(1-\theta) + p'_{j1}\theta)/(p_{j0}(1-\theta) + p_{j1}\theta) \tag{11}$$

$$p'_{g1} = p_{g1}(p'_{j0}\phi + p'_{j1}(1-\phi))/(p_{j0}\phi + p_{j1}(1-\phi). \tag{12}$$

This says you can update the probability in the tree by multiplying in the "new" contribution of N_j on N_g, and dividing out the "old" contribution of N_j on N_g. This same technique can either update a new piece of evidence or to find the conditional probability of a test node given evidence.

Consider the algorithmic time bounds for the standard inference algorithm compared with the caching, incremental algorithm we propose. If there are n nodes in the tree and m queries, then a standard backward-chaining algorithm will traverse through all n nodes for each new query, so it takes time $O(nm)$. With our proposed algorithm, calculating the evidence node requires traversing through each node once; then each test node requires a further pass up the tree. Assuming the

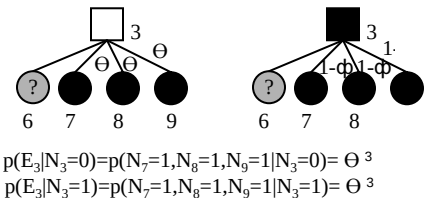

Figure 2: The algorithm iteratively calculates $p(E_i|N_i = 1)$ for various nodes from the leaves to the root. Shown are the graphical representations and probability calculations for $i = 3$.

tree's maximum depth is d, then our algorithm takes time $O(n + dm)$. In many hierarchies, d is a low, fixed constant. The phylogenetic tree classifies all known organisms into a tree seven nodes deep; the Yahoo hierarchy classifies numerous web pages into a tree perhaps ten nodes deep. Therefore, in many problems where the hierarchical depth is a constant, our algorithm only takes $O(n+m)$ as compared to backward-chaining which would take $O(nm)$.

2.5 Extensions

Since our model is probabilistic, we can easily add assumptions to the model to fit our specific problem. Such assumptions would not significantly change the algorithm, but would allow us flexibility in creating domain-specific models. We are not constrained to having one θ and ϕ; we are free to assume different mutation rates for different links. For example, we might have the mutation rate a function of the node depth, causing the solution to prefer changes near the root of the tree. We might make assumptions about probable (but not observed) values for certain leaf nodes. That might be useful when there is non-conclusive evidence for a leaf having a certain value.

For a subjective learning problem like preferences, we might use a learning technique based on past user's preferences. Given past data, it is simple to calculate a prior (empirical) probability on each of the links. For example, past data might indicate that forward mutations are common along the "dog" - "Corgi" edge, indicating that many people like Corgis even when they do not like other dogs. In this way, prior mutation probabilities can be placed throughout the tree so that learning for a new user would be weighted towards the empirical probability distributions of past users.

3 Applications of Tree Learning

3.1 Data Sources

Corporate structures, organisms and web directories are examples of diverse domains that are frequently organized into hierarchies. Our experimental results will focus on one easily accessed hierarchical domain—the domain of URLs on the world wide web.

URLs have human-oriented meanings that are useful for recommendation problems. URLs are more than simply pointers: authors and editors assign important semantics to URLs, and readers make inferences from them. We can infer from a URL that a document serves a particular function (that nytimes.com/adx/ contains an ad); or relates to a topic (cnn.com/tech/). We parse the URL into features by using

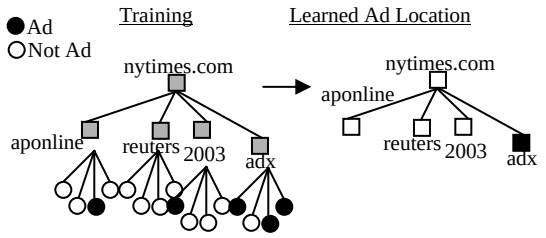

Figure 3: (Top) A portion of the parsed New York times hierarchy, as applied to the ad blocking problem. Notice that a portion of the real news (Aponline) is incorrectly labeled as an ad. (Bottom) Shown is the algorithm's maximum likelihood guess of the color of internal nodes. Despite some mis-lablings, the algorithm quickly identifies the "Adx" sub-directory as filled with ads.

the '/' or '?' to delimit between different levels of the tree.

3.2 The Daily You

We have implemented a content recommendation system ("The Daily You") for the web that uses several hierarchies to learn about user interests. The Daily You is available for public use: `http://falcon.ai.mit.edu`.

For space reasons, we cannot describe the full functionality of the Daily You. However, a brief overview of the system helps describe multiple human-oriented domains for our tree learning algorithm. One portion of the system uses the layout information of a web page—codified as tree structured "table" tags—to find regions of a web page that interest the user. The intuition is that most pages are templated so that important sections (i.e., top stories, sports) appear in the same place every day. When we learn over the table tree, we can understand something about the user's interests, without needing to understand anything about the actual content. Similarly, we can use the URL to find user interests, like a preference for users to click on articles under the "nytimes.com/business" URL. We will give detailed results for another aspect of the Daily You, ad-blocking, in the next sub-section.

3.3 Ad Blocking

Many web sites make revenue by placing distracting advertisements throughout their site. In response, several projects and companies like WebWasher[1] and Muffin[2] have sought to block those ads. In this section, we try to show the empirical efficacy of our tree-learning approach by building an ad-blocker based on the URL feature. We do not claim our approach is full proof; a sufficiently motivated company could design new ads in response to this or any other ad-blocking algorithm.

Conventional wisdom says blocking ads is as simple as blocking certain fixed aspect ratios on incoming images. This is simplistic, and belies the fact that entire companies have formed around the problem of ad-blocking—and advertising. Advertisers constantly change the form of advertisements: they are becoming larger, shaped more like traditional images, or even come in the form of text or colored tables. A static ad-blocker goes out of date in so-called "Internet time." The typical way to adapt to advertising changes is to have humans write new rules blocking the new

[1] `http://webwasher.com`
[2] `http://muffin.doit.org/`

ad image sizes, new IP addresses that serve ads, and so forth.

Our research is different: machine learning finds the new rules, eliminating the need for human input whatsoever. Our goal is to see whether an entirely self-directed computer can achieve accuracies comparable to teams of humans working on the identical problem. If so, our ad-blocker can learn new rules nightly, adapting daily to new ad forms.

Our work is similar in spirit to the AdEater system which also learns ad-blocking rules (Kushmerick, 1999). AdEater got a remarkable 97% accuracy 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 simply 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 to produce an update; hence its accuracy decreases as new ad forms surface.

We want to label our data for training our classifier. One alternative is to use human labeled data, the way AdEater does. However, this is time consuming and mitigates the benefits of having the computer learn ad-blocking rules. Instead, we use a heuristic called Redirect to label the data. Redirect labels ads if a link goes to one site then redirects to a third-party site. Note that Redirect cannot be used for real-time ad blocking because it takes minutes to follow all the links on a page. Similarly, a real-time system using the text of the advertisements would also be implausible. We have chosen the redirect heuristic because it matches the normal process that advertisers use: tracking the click, then sending the user to the advertising site. Notice that this is much more of a "content" based heuristic than image sizes which are "form" heuristics.

In practice, click through tracking requires back-end infrastructure like databases and CGI scripts. Therefore, they tend to be located together under a small number of URL directories per site (i.e. under xyz.com/adserver). We use our tree learning algorithms to associate existing URLs with an 'ad' or 'not ad' label provided by the Redirect heuristic. Then the learning model predicts whether new URLs are 'ads' or 'not.' Figure 3 shows how our algorithm might find ads on the New York Times web site.

What follows is an experiment that shows the performance of our ad-blocker versus a commercial system. We compared two control ad-blocking 'trainers' with two ad-blocking 'learners.'

• *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.

• *Redirect* is the simple heuristic mentioned above that monitors third party redirects.

• *Learn-WW* is our tree-based URL algorithm, *trained on WebWasher*. We want to show its single feature can learn WebWasher's complex rules.

• *Learn-RD* is our tree-based URL algorithm, *trained on Redirect*. We suggest this is a very adaptable ad-blocker, because it learns from Redirect, so can be trained automatically each night.

We generated a dataset from Media Metrix's largest 25 web properties, Jan 2002[3]. Empirical evidence shows the average user spends all their time on a small number

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

Table 1: Ad-Blocking Experiments (accuracy)

	Top 25 Sites	weather	look smart	euniverse
Web-Wash	93.5 (.136)	90.7	85.7	51.7
Learn-WW	93.1 (.118)	86.0	100	51.7
Redirect	93.3 (.08)	84.2	85.7	100
Learn-RD	93.4 (.08)	83.7	100	100

Table 2: shown are the web blocking accuracies for several web sites, along with standard deviation information for the top 25 web sites.

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 off the front page. Those eleven pages, along with 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, for statistical significance reasons.

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. Otherwise, a human was used to judge whether the image was really an ad or not.

In total, 2696 images were classified, and, surprisingly, all four classifiers ended up with an average classification accuracy across all sites of within a quarter percent of 93.25% (see Table 1). Errors were much more likely to be false positives (labeling something an ad when it wasn't) then false negatives; but overall false negative and false positive rates (approximately 26% and 1%, respectively) were fairly consistent between all the classifiers.

Table 1 also shows data for specific sites. The first column contains average error rates for the 25 sites, with standard deviations in parenthesis. 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 confidence intervals 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, such as the Times advertisement for a paper subscription.

In some cases, Learn-WW (the learner) does better than its own trainer, WebWasher. It generalizes all of WebWasher's many features down to one location in a tree, which at times is a perfectly predictive feature, such as when it labels everything under /adserver.cgi/ as an ad, for example on the looksmart.com site.

We wish to make two 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 our system removes them by identifying their URL and removing the link. 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.

3.4 Redirect Learning

Our next experiment tries to show that shifting from the 'independent features' view of naive Bayes to a 'hierarchical features' view of the world can improve classification power on a simple classification problem.

To keep our experiments objective, we had two algorithms predict whether a URL redirects or not, over the top 25 web sites. This was simply a follow-up experiment to the previous subsection, without needing a human to judge correctness. The first classifier simply uses the hierarchical URL classifier; the second is a naive Bayes classifier that assumes each 'word' in the URL is independent.

Both classifiers were given the exact same feature data-the ordered 'words' in the URL, and the same training data. Furthermore, both classifiers were probabilistic. The sole differences were the underlying models: one uses the hierarchy explicitly, and the other assumes independence between the features.

Overall the hierarchical model performs significantly better than the naive Bayes model. Overall classification accuracy-percentage of all links classified correctly-was 93% (.09 standard deviation) for the hierarchical model vs. 40% (.28 standard deviation) for naive Bayes.

Our results showed that naive Bayes classified every element in the same way on 23 of the 25 sites in this experiment, using a maximum likelihood decision process. These results strongly suggest that naive Bayes performs poorly on the tree-learning problem.

4 Conclusions

Hierarchies are a common way of organizing information, and this paper poses a classification problem where features indicate an element's position in a tree. We introduced a generative model that accommodates our intuition that nearby nodes in a tree will have similar properties. We then converted the model into a Bayesian Network and gave an efficient algorithm for solving arbitrary classification problems.

In general, our approach can take advantage of the fact that hierarchies are common and contain information about certain classification problems. In our paper, we have looked at problems that discriminative learners might have on such a domain; have used our algorithm to mimic commercial ad blocking software; and showed an example where the algorithm outperforms naive Bayes.

Acknowledgements The authors wish to thank Leslie Kaelbling, Yu-Han Chang, Jason Rennie and Terran Lane for their helpful comments, and the Oxygen Partnership, NTT, and the Packard Foundation for funding.

References

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern classification.* Wiley-Interscience.

Haussler, D. (1988). Quantifying inductive bias: AI learning algorithms and valiant's learning framework. *Artificial Intelligence*, *36*, 177–221.

Koller, D., & Sahami, M. (1997). Hierarchically classifying documents using very few words. *Proceedings of the 14th International Conference on Machine Learning (ICML-97)* (pp. 170–178).

Kushmerick, N. (1999). Learning to remove internet advertisement. *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)* (pp. 175–181). Seattle, WA, USA: ACM Press.

McCallum, A. K., Rosenfeld, R., Mitchell, T. M., & Ng, A. Y. (1998). Improving text classification by shrinkage in a hierarchy of classes. *Proceedings of ICML-98, 15th International Conference on Machine Learning* (pp. 359–367). Madison, US: Morgan Kaufmann Publishers, San Francisco, US.

Quinlan, J. R. (1987). Generating production rules from decision trees. *Proc. of the 10th IJCAI* (pp. 304–307). Milan, Italy.

Russell, S. J., & Norvig, P. (1994). *Artificial intelligence: A modern approach*. Prentice Hall.