Accelerated Focused Crawling through Online Relevance Feedback[*]

Soumen Chakrabarti[†]
IIT Bombay

Kunal Punera
IIT Bombay

Mallela Subramanyam
University of Texas, Austin

Abstract

The organization of HTML into a tag tree structure, which is rendered by browsers as roughly rectangular regions with embedded text and HREF links, greatly helps surfers locate and click on links that best satisfy their information need. Can an automatic program emulate this human behavior and thereby learn to predict the relevance of an unseen HREF target page w.r.t. an information need, based on information limited to the HREF source page? Such a capability would be of great interest in focused crawling and resource discovery, because it can fine-tune the priority of unvisited URLs in the crawl frontier, and reduce the number of irrelevant pages which are fetched and discarded.

We show that there is indeed a great deal of usable information on a HREF source page about the relevance of the target page. This information, encoded suitably, can be exploited by a supervised *apprentice* which takes online lessons from a traditional focused crawler by observing a carefully designed set of features and events associated with the crawler. Once the apprentice gets a sufficient number of examples, the crawler starts consulting it to better prioritize URLs in the crawl frontier. Experiments on a dozen topics using a 482-topic taxonomy from the Open Directory (Dmoz) show that online relevance feedback can reduce false positives by 30% to 90%.

Categories and subject descriptors:
H.5.4 [**Information interfaces and presentation**]: Hypertext/hypermedia; I.5.4 [**Pattern recognition**]: Applications, Text processing; I.2.6 [**Artificial intelligence**]: Learning; I.2.8 [**Artificial intelligence**]: Problem Solving, Control Methods, and Search.

General terms: Algorithms, performance, measurements, experimentation.

Keywords: Focused crawling, Document object model, Reinforcement learning.

1 Introduction

Keyword search and clicking on links are the dominant modes of accessing hypertext on the Web. Support for keyword search through crawlers and search engines is very mature, but the surfing paradigm is not modeled or assisted as well. Support for surfing is limited to the basic interface provided by Web browsers, except for a few notable research prototypes.

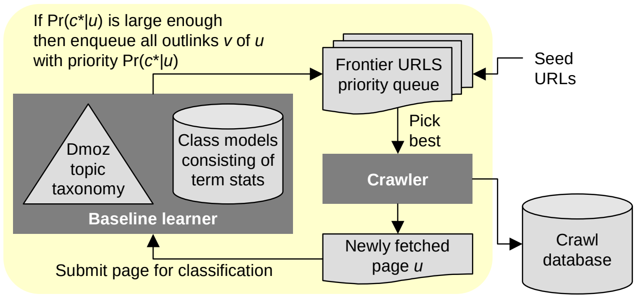

If Pr($c^*|u$) is large enough then enqueue all outlinks v of u with priority Pr($c^*|u$)
Frontier URLS priority queue
Seed URLs
Pick best
Dmoz topic taxonomy
Class models consisting of term stats
Baseline learner
Crawler
Newly fetched page u
Crawl database
Submit page for classification

Figure 1: A basic focused crawler controlled by one topic classifier/learner.

While surfing, the user typically has a topic-specific information need, and explores out from a few known relevant starting points in the Web graph (which may be query responses) to seek new pages relevant to the chosen topic/s. While deciding for or against clicking on a specific link (u, v), humans use a variety of clues on the source page u to estimate the worth of the (unseen) target page v, including the tag tree structure of u, text embedded in various regions of that tag tree, and whether the link is relative or remote. “Every click on a link is a leap of faith” [19], but humans are very good at discriminating between links based on these clues.

Making an educated guess about the worth of clicking on a link (u, v) without knowledge of the target v is central to the surfing activity. Automatic programs which can learn this capability would be valuable for a number of applications which can be broadly characterized as personalized, topic-specific information foragers.

Large-scale, topic-specific information gatherers are called *focused crawlers* [1, 9, 14, 28, 30]. In contrast to giant, all-purpose crawlers which must process large portions of the Web in a centralized manner, a distributed federation of focused crawlers can cover specialized topics in more depth and keep the crawl more fresh, because there is less to cover for each crawler.

In its simplest form, a focused crawler consists of a supervised topic classifier (also called a ‘learner’) controlling the priority of the unvisited frontier of a crawler (see Figure 1). The classifier is trained a priori on document samples embedded in a topic taxonomy such as Yahoo! or Dmoz. It thereby learns to label new documents as belonging to topics in the given taxonomy [2, 5, 21]. The goal of the focused crawler is to start from nodes relevant to a *focus topic* c^* in the Web graph and explore links to selectively collect pages about c^*, while avoiding fetching pages not about c^*.

Suppose the crawler has collected a page u and

[*](Note: The HTML version of this paper is best viewed using Microsoft Internet Explorer. To view the HTML version using Netscape, add the following line to your `~/.Xdefaults` or `~/.Xresources` file:
`Netscape*documentFonts.charset*adobe-fontspecific: iso-8859-1`
For printing use the PDF version, as browsers may not print the mathematics properly.)

[†]Contact author, email `soumen@cse.iitb.ac.in`

WWW2002, May 7–11, 2002, Honolulu, Hawaii, USA.
ACM 1-58113-449-5/02/0005

encountered in u an unvisited link to v. A simple crawler (which we call the *baseline*) will use the relevance of u to topic c^* (which, in a Bayesian setting, we can denote $\Pr(c^*|u)$) as the estimated relevance of the unvisited page v. This reflects our belief that pages across a hyperlink are more similar than two randomly chosen pages on the Web, or, in other words, topics appear clustered in the Web graph [11, 23]. Node v will be added to the crawler's priority queue with priority $\Pr(c^*|u)$. This is essentially a "best-first" crawling strategy. When v comes to the head of the queue and is actually fetched, we can verify if the gamble paid off, by evaluating $\Pr(c^*|v)$. The fraction of relevant pages collected is called the *harvest rate*. If V is the set of nodes collected, the harvest rate is defined as $(1/|V|)\sum_{v\in V}\Pr(c^*|v)$. Alternatively, we can measure the *loss rate*, which is one minus the harvest rate, i.e., the (expected) fraction of fetched pages that must be thrown away. Since the effort on relevant pages is well-spent, reduction in loss rate is the primary goal and the most appropriate figure of merit.

For focused crawling applications to succeed, the "leap of faith" from u to v must pay off frequently. In other words, if $\Pr(c^*|v)$ is often much less than the preliminary estimate $\Pr(c^*|u)$, a great deal of network traffic and CPU cycles are being wasted eliminating bad pages. Experience with random walks on the Web show that as one walks away from a fixed page u_0 relevant to topic c_0, the relevance of successive nodes $u_1, u_2, \ldots$ to c_0 drops dramatically within a few hops [9, 23]. This means that only a fraction of out-links from a page is typically worth following. The average out-degree of the Web graph is about 7 [29]. Therefore, a large number of page fetches may result in disappointment, especially if we wish to push the utility of focused crawling to topic communities which are not very densely linked.

Even w.r.t. topics that are not very narrow, the number of distracting outlinks emerging from even fairly relevant pages has grown substantially since the early days of Web authoring [4]. Template-based authoring, dynamic page generation from semi-structured databases, ad links, navigation panels, and Web rings contribute many irrelevant links which reduce the harvest rate of focused crawlers. Topic-based link discrimination will also reduce these problems.

1.1 Our contribution: Leaping with more faith

In this paper we address the following questions:

How much information about the topic of the HREF target is available and/or latent in the HREF source page, its tag-tree structure, and its text? Can these sources be exploited for accelerating a focused crawler?

Our basic idea is to use **two** classifiers. Earlier, the regular baseline classifier was used to assign priorities to unvisited frontier nodes. This no longer remains its function. The role of assigning priorities to unvisited URLs in the crawl frontier is now assigned to a new learner called the *apprentice*, and the priority of v is specific to the features associated with the (u, v) link which leads to it[1]. The features used by the apprentice are derived from the Document Object Model or

[1]If many u's link to a single v, it is easiest to freeze the priority of v when the first-visited u linking to v is assessed, but combinations of scores are also possible.

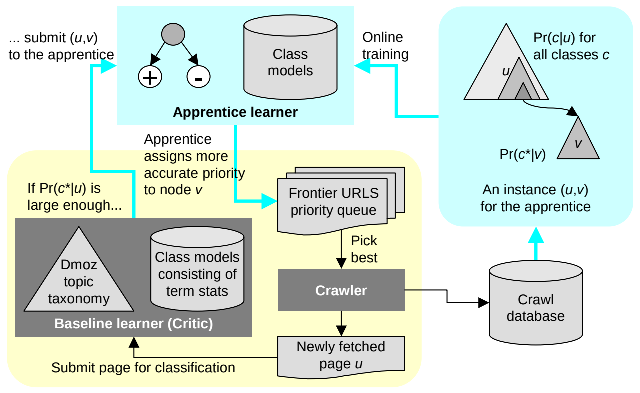

... submit (u,v) to the apprentice
+
-
Class models
Apprentice learner
Online training
Pr(c|u) for all classes c
u
v
Pr(c*|v)
An instance (u,v) for the apprentice
Apprentice assigns more accurate priority to node v
If Pr(c*|u) is large enough...
Frontier URLS priority queue
Pick best
Dmoz topic taxonomy
Class models consisting of term stats
Baseline learner (Critic)
Crawler
Crawl database
Newly fetched page u
Submit page for classification

Figure 2: The apprentice is continually presented with training *cases* (u, v) with suitable features. The apprentice is interposed where new outlinks (u, v) are registered with the priority queue, and helps assign the unvisited node v a better estimate of its relevance.

DOM (`http://www.w3.org/DOM/`) of u. Meanwhile, the role of the baseline classifier becomes one of generating training instances for the apprentice, as shown in Figure 2. We may therefore regard the baseline learner as a *critic* or a *trainer*, which provides feedback to the apprentice so that it can improve "on the job."

The critic-apprentice paradigm is related to reinforcement learning and AI programs that learn to play games [26, §1.2]. We argue that this division of labor is natural and effective. The baseline learner can be regarded as a user specification for *what* kind of content is desired. Although we limit ourselves to a generative statistical model for this specification, this can be an arbitrary black-box predicate. For rich and meaningful distinction between Web communities and topics, the baseline learner needs to be fairly sophisticated, perhaps leveraging off human annotations on the Web (such as topic directories). In contrast, the apprentice specializes in *how* to locate pages to satisfy the baseline learner. Its feature space is more limited, so that it can train fast and adapt nimbly to changing fortunes at following links during a crawl. In Mitchell's words [27], the baseline learner recognizes "global regularity" while the apprentice helps the crawler adapt to "local regularity." This marked asymmetry between the classifiers distinguishes our approach from Blum and Mitchell's co-training technique [3], in which two learners train each other by selecting unlabeled instances.

Using a dozen topics from a topic taxonomy derived from the Open Directory, we compare our enhanced crawler with the baseline crawler. The number of pages that are thrown away (because they are irrelevant), called the *loss* rate, is cut down by 30–90%. We also demonstrate that the fine-grained tag-tree model, together with our synthesis and encoding of features for the apprentice, are superior to simpler alternatives.

1.2 Related work

Optimizing the priority of unvisited URLs on the crawl frontier for specific crawling goals is not new. FishSearch by De Bra et al. [12, 13] and SharkSearch by Hersovici et al. [16] were some of the earliest systems for localized searches in the Web graph for pages with specified keywords.

In another early paper, Cho et al. [10] experimented with a variety of strategies for prioritizing how to fetch unvisited URLs. They used the anchor text as a bag of words to guide link expansion to crawl for pages matching a specified keyword query, which led to some extent of differentiation among out-links, but no trainer-apprentice combination was involved. No notion of supervised topics had emerged at that point, and simple properties like the in-degree or the presence of specified keywords in pages were used to guide the crawler.

Topical locality on the Web has been studied for a few years. Davison made early measurements on a 100000-node Web subgraph [11] collected by the DiscoWeb system. Using the standard notion of vector space TFIDF similarity [31], he found that the endpoints of a hyperlink are much more similar to each other than two random pages, and that HREFs close together on a page link to documents which are more similar than targets which are far apart. Menczer has made similar observations [23]. The HyperClass hypertext classifier also uses such locality patterns for better semi-supervised learning of topics [7], as does IBM's Automatic Resource Compilation (ARC) and Clever topic distillation systems [6, 8].

Two important advances have been made beyond the baseline best-first focused crawler: the use of *context graphs* by Diligenti et al. [14] and the use of *reinforcement learning* by Rennie and McCallum [30]. Both techniques trained a learner with features collected from *paths* leading up to relevant nodes rather than relevant nodes alone. Such paths may be collected by following backlinks.

Diligenti et al. used a classifier (learner) that regressed from the text of u to the estimated *link distance* from u to some relevant page w, rather than the relevance of u or an outlink (u, v), as was the case with the baseline crawler. This lets their system continue expanding u even if the reward for following a link is not immediate, but several links away. However, they *do* favor links whose payoffs are closest. Our work is specifically useful in conjunction with the use of context graphs: when the context graph learner predicts that a goal is several links away, it is crucial to offer additional guidance to the crawler based on local structure in pages, because the fan-out at that radius could be enormous.

Rennie and McCallum [30] also collected paths leading to relevant nodes, but they trained a slightly different classifier, for which:

- An *instance* was a single HREF link like (u, v).
- The *features* were terms from the title and headers (`<h1>...</h1>` etc.) of u, together with the text in and 'near' the anchor (u, v). Directories and pathnames were also used. (We do not know the precise definition of 'near', or how these features were encoded and combined.)
- The *prediction* was a discretized estimate of the number of relevant nodes reachable by following (u, v), where the reward from goals distant from v was geometrically discounted by some factor $\gamma < 1/2$ per hop.

Rennie and McCallum obtained impressive harvests of research papers from four Computer Science department sites, and of pages about officers and directors from 26 company Websites.

Lexical proximity and contextual features have been used extensively in natural language processing for disambiguating word sense [15]. Compared to plain text, DOM trees and hyperlinks give us a richer set of potential features.

Aggarwal et al. have proposed an "intelligent crawling" framework [1] in which only one classifier is used, but similar to our system, that classifier trains as the crawl progresses. They do not use our apprentice-critic approach, and do not exploit features derived from tag-trees to guide the crawler.

The "intelligent agents" literature has brought forth several systems for resource discovery and assistance to browsing [19]. They range between client- and site-level tools. Letizia [18], Powerscout, and WebWatcher [17] are such systems. Menczer and Belew proposed InfoSpiders [24], a collection of autonomous goal-driven crawlers without global control or state, in the style of genetic algorithms. A recent extensive study [25] comparing several topic-driven crawlers including the best-first crawler and InfoSpiders found the best-first approach to show the highest harvest rate (which our new system outperforms).

In all the systems mentioned above, improving the chances of a successful "leap of faith" will clearly reduce the overheads of fetching, filtering, and analyzing pages. Furthermore, whereas we use an automatic first-generation focused crawler to generate the input to train the apprentice, one can envisage specially instrumented browsers being used to monitor users as they seek out information.

We distinguish our work from prior art in the following important ways:

Two classifiers: We use two classifiers. The first one is used to obtain 'enriched' training data for the second one. (A breadth-first or random crawl would have a negligible fraction of positive instances.) The apprentice is a simplified reinforcement learner. It improves the harvest rate, thereby 'enriching' the data collected and labeled by the first learner in turn.

No manual path collection: Our two-classifier framework essentially eliminates the manual effort needed to create reinforcement paths or context graphs. The input needed to start off a focused crawl is just a pre-trained topic taxonomy (easily available from the Web) and a few focus topics.

Online training: Our apprentice trains continually, acquiring ever-larger vocabularies and improving its accuracy as the crawl progresses. This property holds also for the "intelligent crawler" proposed by Aggarwal et al., but they have a *single* learner, whose drift is controlled by precise relevance predicates provided by the user.

No manual feature tuning: Rather than tune ad-hoc notions of proximity between text and hyperlinks, we encode the features of link (u, v) using the DOM-tree of u, and automatically learn a robust definition of 'nearness' of a textual feature to (u, v). In contrast, Aggarwal et al use many tuned constants combining the strength of text- and link-based predictors, and Rennie et al. use domain knowledge to select the paths to goal nodes and the word bags that are submitted to their learner.

2 Methodology and algorithms

We first review the baseline focused crawler and then describe how the enhanced crawler is set up using the apprentice-critic mechanism.

2.1 The baseline focused crawler

The baseline focused crawler has been described in detail elsewhere [9, 14], and has been sketched in Figure 1. Here we review its design and operation briefly.

There are two inputs to the baseline crawler.

- A topic taxonomy or hierarchy with example URLs for each topic.
- One or a few topics in the taxonomy marked as the topic(s) of focus.

Although we will generally use the terms 'taxonomy' and 'hierarchy', a topic tree is not essential; all we really need is a two-way classifier where the classes have the connotations of being 'relevant' or 'irrelevant' to the topic(s) of focus. A topic hierarchy is proposed purely to reduce the tedium of defining new focused crawls. With a two-class classifier, the crawl administrator has to seed positive and negative examples for each crawl. Using a taxonomy, she composes the 'irrelevant' class as the union of all classes that are not relevant. Thanks to extensive hierarchies like Dmoz in the public domain, it should be quite easy to seed topic-based crawls in this way.

The baseline crawler maintains a priority queue on the estimated relevance of nodes v which have not been visited, and keeps removing the highest priority node and visiting it, expanding its outlinks and checking them into the priority queue with the relevance score of v in turn. Despite its extreme simplicity, the best-first crawler has been found to have very high harvest rates in extensive evaluations [25].

Why do we need negative examples and negative classes at all? Instead of using class probabilities, we could maintain a priority queue on, say, the TFIDF cosine similarity between u and the centroid of the seed pages (acting as an estimate for the corresponding similarity between v and the centroid, until v has been fetched). Experience has shown [32] that characterizing a negative class is quite important to prevent the centroid of the crawled documents from drifting away indefinitely from the desired topic profile.

In this paper, the baseline crawler also has the implicit job of gathering instances of successful and unsuccessful "leaps of faith" to submit to the apprentice, discussed next.

2.2 The basic structure of the apprentice learner

In estimating the worth of traversing the HREF (u, v), we will limit our attention to u alone. The page u is modeled as a tag tree (also called the Document Object Model or DOM). In principle, *any* feature from u, even font color and site membership may be perfect predictors of the relevance of v. The total number of potentially predictive features will be quite staggering, so we need to simplify the feature space and massage it into a form suited to conventional learning algorithms. Also note that we specifically study properties of u and not larger contexts such as paths leading to u, meaning that our method may become even more robust and useful in conjunction with context graphs or reinforcement along paths.

Initially, the apprentice has no training data, and passes judgment on (u, v) links according to some fixed prior obtained from a baseline crawl run ahead of time (e.g., see the statistics in §3.3). Ideally, we would like to train the apprentice continuously, but to reduce overheads, we declare a *batch size* between a few hundred and a few thousand pages. After every batch of pages is collected, we check if any page u fetched before the current batch links to some page v in the batch. If such a (u, v) is found, we extract suitable features for (u, v) as described later in this section, and add $\langle (u, v), \Pr(c^*|v)\rangle$ as another instance of the training data for the apprentice. Many apprentices, certainly the simple naive Bayes and linear perceptrons that we have studied, need not start learning from scratch; they can accept the additional training data with a small additional computational cost.

2.2.1 Preprocessing the DOM tree

First, we parse u and form the DOM tree for u. Sadly, much of the HTML available on the Web violates any HTML standards that permit context-free parsing, but a variety of repair heuristics (see, e.g., HTML Tidy, available at `http://www.w3.org/People/Raggett/tidy/`) let us generate reasonable DOM trees from bad HTML.

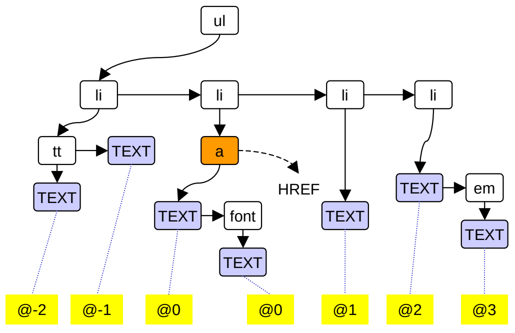


Figure 3: Numbering of DOM leaves used to derive offset attributes for textual tokens. '@' means "is at offset".

Second, we number all leaf nodes consecutively from left to right. For uniformity, we assign numbers even to those DOM leaves which have no text associated with them. The specific `<a href...>` which links to v is actually an internal node a_v, which is the root of the subtree containing the anchor text of the link (u, v). There may be other element tags such as `<em>` or `<b>` in the subtree rooted at a_v. Let the leaf or leaves in this subtree be numbered $\ell(a_v)$ through $r(a_v) \geq \ell(a_v)$. We regard the textual tokens available from any of these leaves as being *at DOM offset zero* w.r.t. the (u, v) link. Text tokens from a leaf numbered μ, to the left of $\ell(a_v)$, are at *negative* DOM offset $\mu - \ell(a_v)$. Likewise, text from a leaf numbered μ to the right of $r(a_v)$ are at *positive* DOM offset $\mu - r(a_v)$. See Figure 3 for an example.

2.2.2 Features derived from the DOM and text tokens

Many related projects mentioned in §1.2 use a linear notion of proximity between a HREF and textual tokens. In the ARC system, there is a crude cut-off distance measured in *bytes* to the left and right of the anchor. In the Clever system, distance is measured in tokens, and the importance attached to a token decays with the distance. In reinforcement learning and intelligent predicate-based crawling, the exact specification of neighborhood text is not known to us. In all cases, some ad-hoc tuning appears to be involved.

We claim (and show in §3.4) that the relation between the relevance of the target v of a HREF (u, v) and the proximity of terms to (u, v) can be learnt automatically. The results are better than ad-hoc tuning of cut-off distances, provided the DOM offset information is encoded as features suitable for the apprentice.

One obvious idea is to extend the Clever model: a page is a linear sequence of tokens. If a token t is distant x from the HREF (u, v) in question, we encode it as a feature $\langle t, x\rangle$. Such features will not be useful because there are too many possible values of x, making the $\langle t, x\rangle$ space too sparse to learn well. (How many HREFS will be *exactly* five tokens from the term 'basketball'?)

Clearly, we need to bucket x into a small number of ranges. Rather than tune arbitrary bucket boundaries by hand, we argue that DOM offsets are a natural bucketing scheme provided by the page author. Using the node numbering scheme described above, each token t on page u can be annotated w.r.t. the link (u, v) (for simplicity assume there is only one such link) as $\langle t, d\rangle$, where d is the DOM offset calculated above. This is the main set of features used by the apprentice. We shall see that the apprentice can learn to limit $|d|$ to less than $d_{\max} = 5$ in most cases, which reduces its vocabulary and saves time.

A variety of other feature encodings suggest themselves. We are experimenting with some in ongoing work (§4), but decided against some others. For example, we do not expect gains from encoding specific HTML tag names owing to the diversity of authoring styles. Authors use `<div>`, `<span>`, `<layer>` and nested tables for layout control in non-standard ways; these are best deflated to a nameless DOM node representation. Similar comments apply to HREF collections embedded in `<ul>`, `<ol>`, `<td>` and `<dd>`. Font and lower/upper case information is useful for search engines, but would make features even sparser for the apprentice. Our representation also flattens two-dimensional tables to their "row-major" representation.

The features we ignore are definitely crucial for other applications, such as information extraction. We did not see any cases where this sloppiness led to a large loss rate. We would be surprised to see tables where relevant links occurred in the third column and irrelevant links in the fifth, or pages where they are rendered systematically in different fonts and colors, but are not otherwise demarcated by the DOM structure.

2.2.3 Non-textual features

Limiting d may lead us to miss features of u that may be useful at the whole-page level. One approach would be to use "$d = \infty$" for all d larger in magnitude than some threshold. But this would make our apprentice as bulky and slow to train as the baseline learner.

Instead, we use the baseline learner to *abstract* u for the apprentice. Specifically, we use a naive Bayes baseline learner to classify u, and use the vector of class probabilities returned as features for the apprentice. These features can help the apprentice discover patterns such as

"Pages about `/Recreation/Boating/Sailing` often link to pages about `/Sports/Canoe_and_Kayaking`."

This also covers for the baseline classifier confusing between classes with related vocabulary, achieving an effect similar to context graphs.

Another kind of feature can be derived from *co-citation*. If v_1 has been fetched and found to be relevant and HREFS (u, v_1) and (u, v_2) are close to each other, v_2 is likely to be relevant. Just like textual tokens were encoded as $\langle t, d\rangle$ pairs, we can represent co-citation features as $\langle \rho, d\rangle$, where ρ is a suitable representation of relevance.

Many other features can be derived from the DOM tree and added to our feature pool. We discuss some options in §4. In our experience so far, we have found the $\langle t, d\rangle$ features to be most useful. For simplicity, we will limit our subsequent discussion to $\langle t, d\rangle$ features only.

2.3 Choices of learning algorithms for the apprentice

Our feature set is thus an interesting mix of categorical, ordered and continuous features:

- Term tokens $\langle t, d\rangle$ have a categorical component t and a discrete ordered component d (which we may like to smooth somewhat). Term counts are discrete but can be normalized to constant document length, resulting in continuous attribute values.
- Class names are discrete and may be regarded as synthetic terms. The probabilities are continuous.

The output we desire is an estimate of $\Pr(c^*|v)$, given all the observations about u and the neighborhood of (u, v) that we have discussed. Neural networks are a natural choice to accommodate these requirements. We first experimented with a simple linear perceptron, training it with the delta rule (gradient descent) [26]. Even for a linear perceptron, convergence was surprisingly slow, and after convergence, the error rate was rather high. It is likely that local optima were responsible, because stability was generally poor, and got worse if we tried to add hidden layers or sigmoids. In any case, convergence was too slow for use as an online learner. All this was unfortunate, because the direct regression output from a neural network would be convenient, and we were hoping to implement a Kohonen layer for smoothing d.

In contrast, a naive Bayes (NB) classifier worked very well. A NB learner is given a set of training documents, each labeled with one of a finite set of classes/topic. A document or Web page u is modeled as a multiset or bag of words, $\{\langle \tau, n(u, \tau)\rangle\}$ where τ is a feature which occurs $n(u, \tau)$ times in u. In ordinary text classification (such as our baseline learner) the features τ are usually single words. For our apprentice learner, a feature τ is a $\langle t, d\rangle$ pair.

NB classifiers can predict from a discrete set of classes, but our prediction is a continuous (probability) score. To bridge this gap, We used a simple two-bucket (low/high relevance) special case of Torgo and Gama's technique of using classifiers for discrete labels for continuous regression [33], using "equally probable intervals" as far as possible.

Torgo and Gama recommend using a measure of centrality, such as the median, of each interval as the predicted value of that class. Rennie and McCallum [30] corroborate that 2–3 bins are adequate. As will be clear from our experiments, the medians of our 'low' and 'high' classes are very close to zero and one respectively (see Figure 5). Therefore, we simply take the probability of the 'high' class as the prediction from our naive Bayes apprentice.

The prior probability of class c, denoted $\Pr(c)$ is the fraction of training documents labeled with class c. The NB model is parameterized by a set of numbers $\theta_{c,\tau}$ which is roughly the rate of occurrence of feature τ in class c, more exactly,

$$\theta_{c,\tau} \quad = \quad \frac{1+\sum_{u\in V_c} n(u,\tau)}{|T| + \sum_{u,\tau'} n(u,\tau')}, \tag{1}$$

where V_c is the set of Web pages labeled with c and T is the entire vocabulary. The NB learner assumes independence between features, and estimates

$$\Pr(c|u) \quad \propto \quad \Pr(c)\Pr(u|c) \quad \approx \quad \Pr(c)\prod_{\tau\in u}\theta_{c,\tau}^{n(u,\tau)}. \tag{2}$$

Nigam et al. provide further details [22].

3 Experimental study

Our experiments were guided by the following requirements. We wanted to cover a broad variety of topics, some 'easy' and some 'difficult', in terms of the harvest rate of the baseline crawler. Here is a quick preview of our results.

- The apprentice classifier achieves high accuracy in predicting the relevance of unseen pages given $\langle t,d\rangle$ features. It can determine the best value of $d_{\max}$ to use, typically, 4–6.
- Encoding DOM offsets in features improves the accuracy of the apprentice substantially, compared to a bag of ordinary words collected from within the same DOM offset window.
- Compared to a baseline crawler, a crawler that is guided by an apprentice (trained offline) has a 30% to 90% lower loss rate. It finds crawl paths never expanded by the baseline crawler.
- Even if the apprentice-guided crawler is forced to stay within the (inferior) Web graph collected by the baseline crawler, it collects the best pages early on.
- The apprentice is easy to train online. As soon as it starts guiding the crawl, loss rates fall dramatically.
- Compared to $\langle t,d\rangle$ features, topic- or cocitation-based features have negligible effect on the apprentice.

To run so many experiments, we needed three highly optimized and robust modules: a crawler, a HTML-to-DOM converter, and a classifier.

We started with the `w3c-libwww` crawling library from `http://www.w3c.org/Library/`, but replaced it with our own crawler because we could effectively overlap DNS lookup, HTTP access, and disk access using a `select` over all socket/file descriptors, and prevent memory leaks visible in `w3c-libwww`. With three caching DNS servers, we could achieve over 90% utilization of a 2Mbps dedicated ISP connection.

We used the HTML parser `libxml2` library to extract the DOM from HTML, but this library has memory leaks, and does not always handle poorly written HTML well. We had some stability problems with HTML Tidy (`http://www.w3.org/People/Raggett/tidy/`), the well-known HTML cleaner which is very robust to bad HTML. At present we are using `libxml2` and are rolling our own HTML parser and cleaner for future work.

We intend to make our crawler and HTML parser code available in the public domain for research use.

For both the baseline and apprentice classifier we used the public domain BOW toolkit and the Rainbow naive Bayes classifier created by McCallum and others [20]. Bow and Rainbow are very fast C implementations which let us classify pages in real time as they were being crawled.

3.1 Design of the topic taxonomy

We downloaded from the Open Directory (`http://dmoz.org/`) an RDF file with over **271954** topics arranged in a tree hierarchy with depth at least **6**, containing a total of about **1697266** sample URLs. The distribution of samples over topics was quite non-uniform. Interpreting the tree as an **is-a** hierarchy meant that internal nodes inherited all examples from descendants, but they also had their own examples. Since the set of topics was very large and many topics had scarce training data, we pruned the Dmoz tree to a manageable frontier by following these steps:

1. Initially we placed example URLs in both internal and leaf nodes, as given by Dmoz.
2. We fixed a minimum per-class training set size of $k = 300$ documents.
3. We iteratively performed the following step as long as possible: we found a leaf node with less than k example URLs, moved all its examples to its parent, and deleted the leaf.
4. To each internal node c, we attached a leaf subdirectory called `Other`. Examples associated directly with c were moved to this `Other` subdirectory.
5. Some topics were populated out of proportion, either at the beginning or through the above process. We made the class priors more balanced by sampling down the large classes so that each class had at most 300 examples.

The resulting taxonomy had **482** leaf nodes and a total of **144859** sample URLs. Out of these we could successfully fetch about **120000** URLs. At this point we discarded the tree structure and considered only the leaf topics. Training time for the baseline classifier was about about **two hours** on a 729MHz Pentium III with 256kB cache and 512MB RAM. This was very fast, given that 1.4GB of HTML text had to be processed through Rainbow. The complete listing of topics can be obtained from the authors.

3.2 Choice of topics

Depending on the focus topic and prioritization strategy, focused crawlers may achieve diverse harvest rates. Our early prototype [9] yielded harvest rates typically between 0.25 and 0.6. Rennie and McCallum [30] reported recall and not harvest rates. Diligenti et al. [14] focused on very specific topics where the harvest rate was very low, 4–6%. Obviously, the maximum gains shown by a new idea in focused crawling can be sensitive to the baseline harvest rate.

To avoid showing our new system in an unduly positive or negative light, we picked a set of topics which were fairly diverse, and appeared to be neither too broad to be useful (e.g., `/Arts`, `/Science`) nor too narrow for the baseline crawler to be a reasonable adversary. We list our topics in Figure 4. We chose the topics without prior estimates of how well our new system would work, and froze the list of topics. All topics that we experimented with showed visible improvements, and none of them showed deteriorated performance.

3.3 Baseline crawl results

We will skip the results of breadth-first or random crawling in our commentary, because it is known from earlier work on focused crawling that our baseline crawls are already far better than breadth-first or random crawls. Figure 5 shows, for most of the topics listed above, the distribution of page relevance after running the baseline crawler to collect roughly 15000 to 25000 pages per topic. The baseline crawler used a standard naive Bayes classifier on the ordinary term space of whole pages. We see that the relevance distribution is bimodal, with most pages being very relevant or not at all. This is partly, but only partly, a result of using a multinomial naive Bayes model. The naive Bayes classifier assumes term independence and multiplies together many (small) term probabilities, with the result that the winning class usually beats all others by a large margin in probability. But it is also true that many outlinks lead to pages with completely irrelevant topics. Figure 5 gives a clear indication of how much improvement we can expect for each topic from our new algorithm.

3.4 DOM window size and feature selection

A key concern for us was how to limit the maximum window width so that the total number of synthesized $\langle t, d\rangle$ features remains much smaller than the training data for the baseline classifier, enabling the apprentice to be trained or upgraded in a very short time. At the same time, we did not want to lose out on medium- to long-range dependencies between significant tokens on a page and the topic of HREF targets in the vicinity. We eventually settled for a maximum DOM window size of 5. We made this choice through the following experiments.

The easiest initial approach was an end-to-end cross-validation of the apprentice for various topics while increasing $d_{\max}$. We observed an initial increase in the validation accuracy when the DOM window size was increased beyond 0. However, the early increase leveled off or even reversed after the DOM window size was increased beyond 5. The graphs in Figure 6 display these results. We see that in the Chess category, though the validation accuracy increases monotonically, the gains are less pronounced after $d_{\max}$ exceeds 5. For the AI category, accuracy fell beyond $d_{\max} = 4$.

Topic	#Good	#Bad
`/Arts/Music/Styles/Classical/Composers`	24000	13000
`/Arts/Performing_Arts/Dance/Folk_Dancing`	7410	8300
`/Business/Industries.../Livestock/Horses...`	17000	7600
`/Computers/Artificial_Intelligence`	7701	14309
`/Computers/Software/Operating_Systems/Linux`	17500	9300
`/Games/Board_Games/C/Chess`	17000	4600
`/Health/Conditions_and_Diseases/Cancer`	14700	5300
`/Home/Recipes/Soups_and_Stews`	20000	3600
`/Recreation/Outdoors/Fishing/Fly_Fishing`	12000	13300
`/Recreation/Outdoors/Speleology`	6717	14890
`/Science/Astronomy`	14961	5332
`/Science/Earth_Sciences/Meteorology`	19205	8705
`/Sports/Basketball`	26700	2588
`/Sports/Canoe_and_Kayaking`	12000	12700
`/Sports/Hockey/Ice_Hockey`	17500	17900

Figure 4: We chose a variety of topics which were neither too broad nor too narrow, so that the baseline crawler was a reasonable adversary. #Good (#Bad) show the approximate number of pages collected by the baseline crawler which have relevance above (below) 0.5, which indicates the relative difficulty of the crawling task.

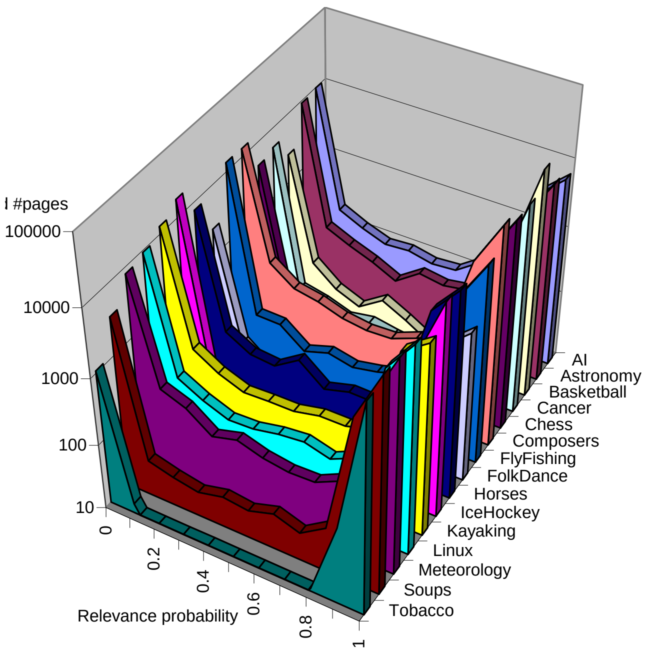
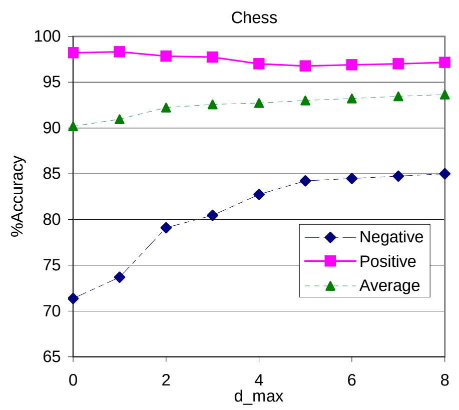


Figure 5: All of the baseline classifiers have harvest rates between 0.25 and 0.6, and all show strongly bimodal relevance score distribution: most of the pages fetched are very relevant or not at all.

It is important to notice that the improvement in accuracy is almost entirely because with increasing number of available features, the apprentice can reject negative (low relevance) instances more accurately, although the accuracy for positive instances *decreases* slightly. Rejecting unpromising outlinks is critical to the success of the enhanced crawler. Therefore we would rather lose a little accuracy for positive instances rather than do poorly on the negative instances. We therefore chose $d_{\max}$ to be either 4 or 5 for all the experiments.

We verified that adding offset information to text tokens was better than simply using plain text near the link [8]. One sample result is shown in Figure 7. The apprentice accuracy decreases with $d_{\max}$ if only text is used, whereas it increases if offset information is provided. This highlights



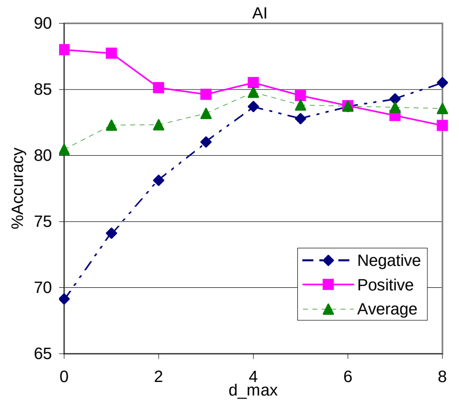


Figure 6: There is visible improvement in the accuracy of the apprentice if $d_{\max}$ is made larger, up to about 5–7 depending on topic. The effect is more pronounced on the the ability to correctly reject negative (low relevance) outlink instances. 'Average' is the microaverage over all test instances for the apprentice, not the arithmetic mean of 'Positive' and 'Negative'.

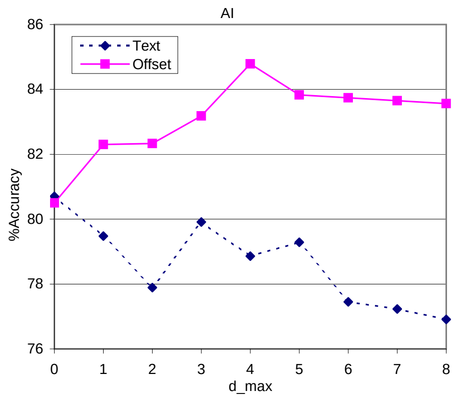
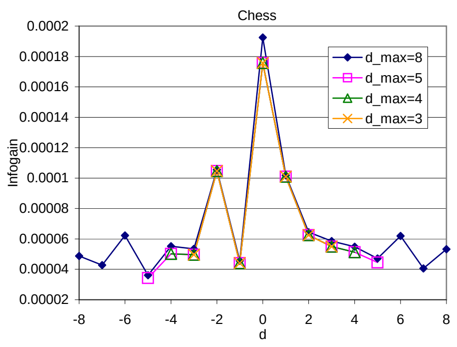


Figure 7: Encoding DOM offset information with textual features boosts the accuracy of the apprentice substantially.

the importance of designing proper features.

To corroborate the useful ranges of $d_{\max}$ above, we compared the value of average mutual information gain for terms found at various distances from the target HREF. The experiments revealed that the information gain of terms found further away from the target HREF was generally lower than those that were found closer, but this reduction was *not* monotonic. For instance, the average information



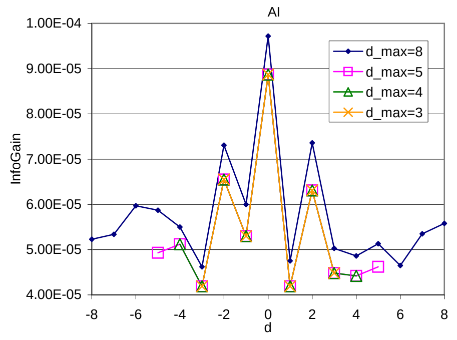


Figure 8: Information gain variation plotted against distance from the target HREF for various DOM window sizes. We observe that the information gain is insensitive to $d_{\max}$.

gain at $d = -2$ was higher than that at $d = -1$; see Figure 8. For each DOM window size, we observe that the information gain varies in a *sawtooth* fashion; this intriguing observation is explained shortly. The average information gain settled to an almost constant value after distance of 5 from the target URL. We were initially concerned that to keep the computation cost manageable, we would need some cap on $d_{\max}$ even while measuring information gain, but luckily, the variation of information gain is insensitive to $d_{\max}$, as Figure 8 shows. These observations made our final choice of $d_{\max}$ easy.

In a bid to explain the occurrence of the unexpected saw-tooth form in Figure 8 we measured the rate $\theta_{\langle t,d\rangle}$ at which term t occurred at offset d, relative to the total count of all terms occurring at offset d. (They are roughly the multinomial naive Bayes term probability parameters.) For fixed values of d, we calculated the sum of θ values of terms found at those offsets from the target HREF. Figure 9(a) shows the plot of these sums to the distance(d) for various categories. The θ values showed a general decrease as the distances from the target HREF increased, but this decrease, like that of information gain, was not monotonic. The θ values of the terms at odd numbered distances from the target HREF were found to be lower than those of the terms present at the even positions. For instance, the sum of θ values of terms occurring at distance -2 were higher than that of terms at position -1. This observation was explained by observing the HTML tags that are present at various distances from the target HREF. We observed that tags located at odd d are mostly non-text tags, thanks to authoring idioms such as `<li><a...><li><a...>` and `<a...><br><a...><br>` etc. A plot of the frequency of HTML tags against the distance from the HREF at which

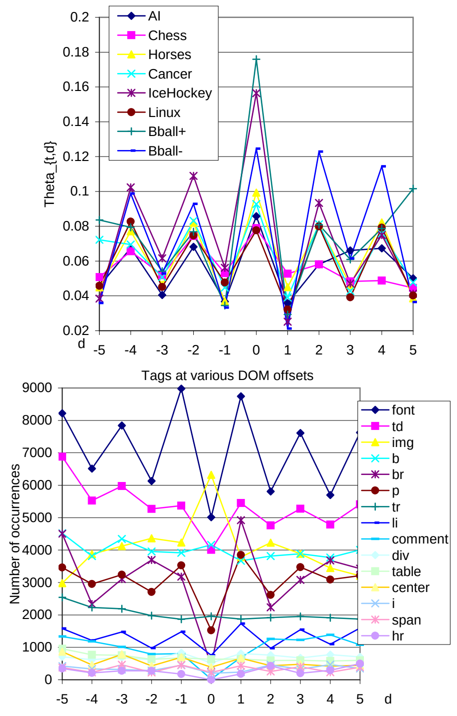


Figure 9: Variation of (a) relative term frequencies and (b) frequencies of HTML tags plotted against d.

they were found is shown in Figure 9(b). (The `<a...>` tag obviously has the highest frequency and has been removed for clarity.)

These were important DOM idioms, spanning many diverse Web sites and authoring styles, that we did not anticipate ahead of time. Learning to recognize these idioms was valuable for boosting the harvest of the enhanced crawler. Yet, it would be unreasonable for the user-supplied baseline black-box predicate or learner to capture crawling strategies at such a low level. This is the ideal job of the apprentice. The apprentice took only **3–10 minutes** to train on its (u, v) instances from scratch, despite a simple implementation that wrote a small file to disk for each instance of the apprentice. Contrast this with several hours taken by the baseline learner to learn general term distribution for topics.

3.5 Crawling with the apprentice trained off-line

In this section we subject the apprentice to a "field test" as part of the crawler, as shown in Figure 2. To do this we follow these steps:

1. Fix a topic and start the baseline crawler from all example URLs available from the given topic.
2. Run the baseline crawler until roughly 20000–25000 pages have been fetched.
3. For all pages (u, v) such that both u and v have been fetched by the baseline crawler, prepare an instance from (u, v) and add to the training set of the apprentice.
4. Train the apprentice. Set a suitable value for $d_{\max}$.

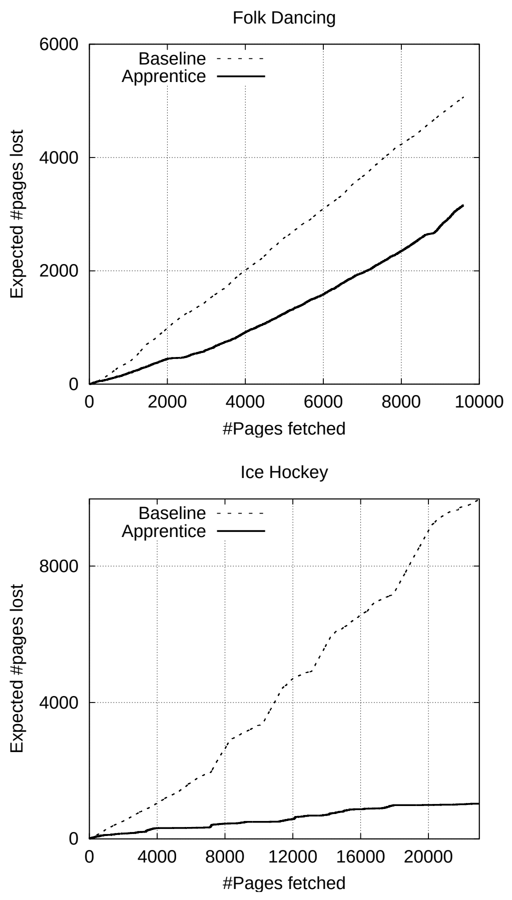


Figure 10: Guidance from the apprentice significantly reduces the loss rate of the focused crawler.

5. Start the enhanced crawler from the *same* set of pages that the baseline crawler had started from.
6. Run the enhanced crawler to fetch about the same number of pages as the baseline crawler.
7. Compare the loss rates of the two crawlers.

Unlike with the reinforcement learner studied by Rennie and McCallum, we have no predetermined universe of URLs which constitute the relevant set; our crawler must go forth into the open Web and collect relevant pages from an unspecified number of sites. Therefore, measuring recall w.r.t. the baseline is not very meaningful (although we do report such numbers, for completeness, in §3.6). Instead, we measure the *loss* (the number of pages fetched which had to be thrown away owing to poor relevance) at various epochs in the crawl, where time is measured as the number of pages fetched (to elide fluctuating network delay and bandwidth). At epoch n, if the pages fetched are $v_1, \ldots, v_n$, then the total expected loss is $(1/n)\sum_i(1 - \Pr(c^*|v_i))$.

Figure 10 shows the loss plotted against the number of pages crawled for two topics: Folk dancing and Ice hockey. The behavior for Folk dancing is typical; Ice hockey is one of the best examples. In both cases, the loss goes up substantially faster with each crawled page for the baseline crawler than for the enhanced crawler. The reduction of loss for these topics are 40% and 90% respectively; typically, this number is between 30% and 60%. In other words, for most topics, the apprentice reduces the number of useless pages fetched by one-third to two-thirds.

In a sense, comparing loss rates is the most meaningful evaluation in our setting, because the network cost of fetching relevant pages has to be paid anyway, and can be regarded as a fixed cost. Diligenti et al. show significant improvements in harvest rate, but for their topics, the loss rate for both the baseline crawler as well as the context-focused crawler were much higher than ours.

3.6 URL overlap and recall

The reader may feel that the apprentice crawler has an unfair advantage because it is first trained on DOM-derived features from the *same* set of pages that it has to crawl again. We claim that the set of pages visited by the baseline crawler and the (off-line trained) enhanced crawler have small overlap, and the superior results for the crawler guided by the apprentice are in large part because of generalizable learning. This can be seen from the examples in Figure 11.

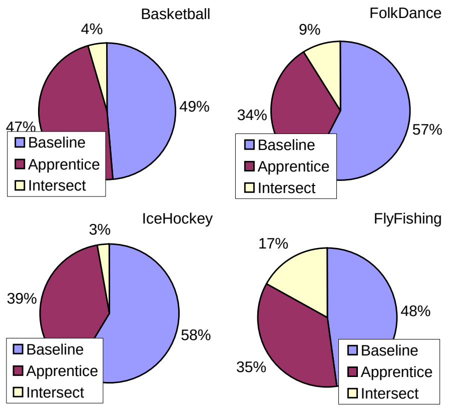


Figure 11: The apprentice-guided crawler follows paths which are quite different from the baseline crawler because of its superior priority estimation technique. As a result there is little overlap between the URLs harvested by these two crawlers.

Given that the overlap between the baseline and the enhanced crawlers is small, which is 'better'? As per the verdict of the baseline classifier, clearly the enhanced crawler is better. Even so, we report the loss rate of a different version of the enhanced crawler which is restricted to visiting *only* those pages which were visited by the baseline learner. We call this crawler the *recall* crawler. This means that in the end, both crawlers have collected exactly the same set of pages, and therefore have the same total loss. The test then is how long can the enhanced learner prevent the loss from approaching the baseline loss. These experiments are a rough analog of the 'recall' experiments done by Rennie and McCallum. We note that for these recall experiments, the apprentice *does* get the benefit of not having to generalize, so the gap between baseline loss and recall loss could be optimistic. Figure 12 compares the expected total loss of the baseline crawler, the recall crawler, and the apprentice-guided crawler (which is free to wander outside the baseline collection) plotted against the number of pages fetched, for a few topics. As expected, the recall crawler has loss generally somewhere between the loss of the baseline and the enhanced crawler.

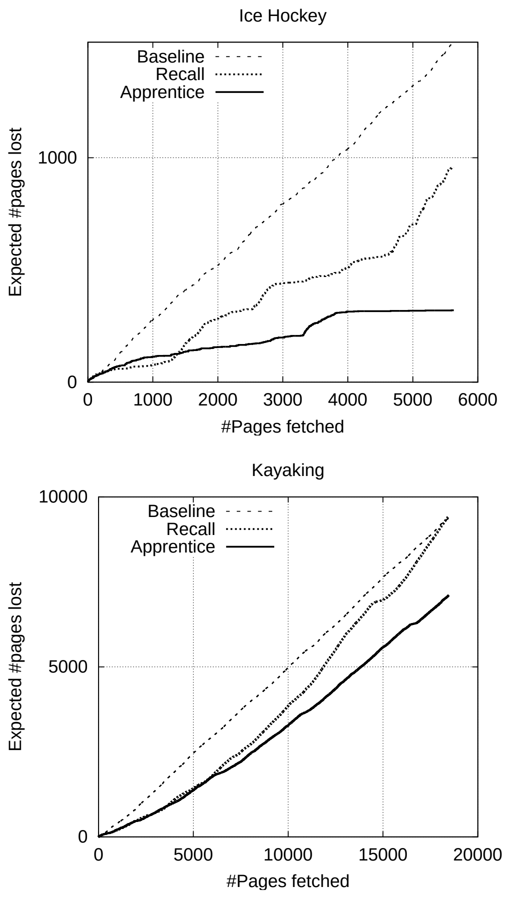


Figure 12: Recall for a crawler using the apprentice but limited to the set of pages crawled earlier by the baseline crawler.

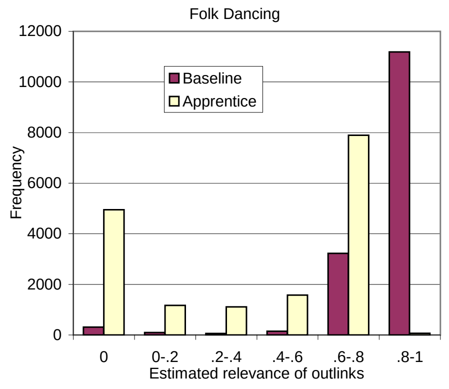
3.7 Effect of training the apprentice online

Next we observe the effect of a mid-flight correction when the apprentice is trained some way into a baseline and switched into the circuit. The precise steps were:

1. Run the baseline crawler for the first n page fetches, then stop it.
2. Prepare instances and train the apprentice.
3. Re-evaluate the priorities of all unvisited pages v in the frontier table using the apprentice.
4. Switch in the apprentice and resume an enhanced crawl.

We report our experience with "Folk Dancing." The baseline crawl was stopped after 5200 pages were fetched. Re-evaluating the priority of frontier nodes led to radical changes in their individual ranks as well as the priority distributions. As shown in Figure 13(a), the baseline learner is overly optimistic about the yield it expects from the frontier, whereas the apprentice already abandons a large fraction of frontier outlinks, and is less optimistic about

(a)



(b)

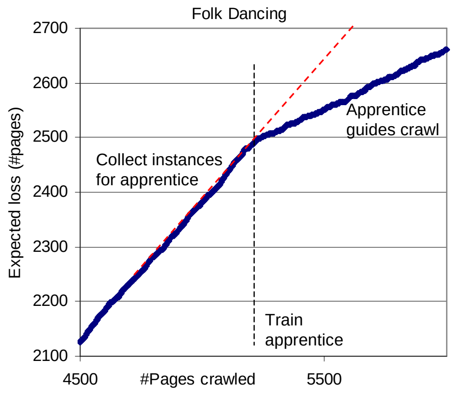


Figure 13: The effect of online training of the apprentice. (a) The apprentice makes sweeping changes in the estimated promise of unvisited nodes in the crawl frontier. (b) Resuming the crawl under the guidance of the apprentice immediately shows significant reduction in the loss accumulation rate.

the others, which appears more accurate from the Bayesian perspective.

Figure 13(b) shows the effect of resuming an enhanced crawl guided by the trained apprentice. The new (u, v) instances are *all* guaranteed to be unknown to the apprentice now. It is clear that the apprentice's prioritization immediately starts reducing the loss rate. Figure 14 shows an even more impressive example. There are additional mild gains from retraining the apprentice at later points. It may be possible to show a more gradual online learning effect by retraining the classifier at a finer interval, e.g., every 100 page fetches, similar to Aggarwal et al. In our context, however, losing a thousand pages at the outset because of the baseline crawler's limitation is not a disaster, so we need not bother.

3.8 Effect of other features

We experimented with two other kinds of feature, which we call *topic* and *cocitation* features.

Our limiting $d_{\max}$ to 5 may deprive the apprentice of important features in the source page u which are far from the link (u, v). One indirect way to reveal such features to the apprentice is to classify u, and to add the *names* of some of the top-scoring classes for u to the instance (u, v). §2.2.3 explains why this may help. This modification resulted in a 1% increase in the accuracy of the apprentice. A further increase of 1% was observed if we added all

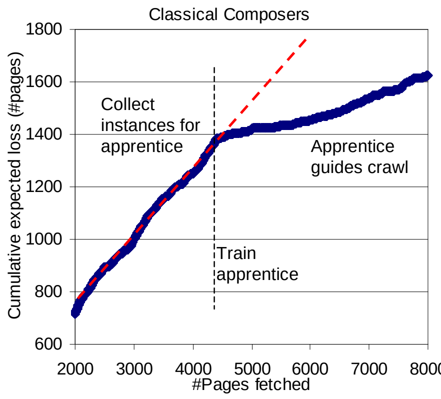


Figure 14: Another example of training the apprentice online followed by starting to use it for crawl guidance. Before guidance, loss accumulation rate is over 30%, after, it drops to only 6%.

prefixes of the class name. For example, the full name for the Linux category is `/Computers/Software/Operating_Systems/Linux`. We added all of the following to the feature set of the source page: `/`, `/Computers`, `/Computers/Software`, `/Computers/Software/Operating_Systems` and `/Computers/Software/Operating_Systems/Linux`. We also noted that various class names and some of their prefixes appeared amongst the best discriminants of the positive and negative classes.

Cocitation features for the link (u, v) are constructed by looking for other links (u, w) within a DOM distance of $d_{\max}$ such that w has already been fetched, so that $\Pr(c^*|w)$ is known. We discretize $\Pr(c^*|w)$ to two values HIGH and LOW as in §2.3, and encode the feature as $\langle \text{LOW}, d \rangle$ or $\langle \text{HIGH}, d \rangle$. The use of cocitation features did not improve the accuracy of the apprentice to any appreciable extent.

For both kinds of features, we estimated that random variations in crawling behavior (because of fluctuating network load and tie-breaking frontier scores) may prevent us from measuring an actual benefit to crawling under realistic operating conditions. We note that these ideas may be useful in other settings.

4 Conclusion

We have presented a simple enhancement to a focused crawler that helps assign better priorities to the unvisited URLs in the crawl frontier. This leads to a higher rate of fetching pages relevant to the focus topic and fewer false positives which must be discarded after spending network, CPU and storage resources processing them. There is no need to manually train the system with paths leading to relevant pages. The key idea is an apprentice learner which can accurately predict the worth of fetching a page using DOM features on pages that link to it. We show that the DOM features we use are superior to simpler alternatives. Using topics from Dmoz, we show that our new system can cut down the fraction of false positives by 30–90%.

We are exploring several directions in ongoing work. We wish to revisit continuous regression techniques for the apprentice, as well as more extensive features derived from the DOM. For example, we can associate with a token t the length ℓ of the DOM path from the text node containing t to the HREF to v, or the depth of their least common ancestor in the DOM tree. We cannot use these in lieu of DOM offset, because regions which are far apart lexically may be close to each other along a DOM path. $\langle t, \ell, d\rangle$ features will be more numerous and sparser than $\langle t, d\rangle$ features, and could be harder to learn. The introduction of large numbers of strongly dependent features may even reduce the accuracy of the apprentice. Finally, we wish to implement some form of active learning where only those instances (u, v) with the largest $|\Pr(c^*|u) - \Pr(c^*|v)|$ are chosen as training instances for the apprentice.

Acknowledgments: Thanks to the referees for suggesting that we present Figure 7.

References

[1] C. C. Aggarwal, F. Al-Garawi, and P. S. Yu. Intelligent crawling on the World Wide Web with arbitrary predicates. In *WWW2001*, Hong Kong, May 2001. ACM. Online at `http://www10.org/cdrom/papers/110/`.

[2] C. Apte, F. Damerau, and S. M. Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 1994. IBM Research Report RC18879.

[3] A. Blum and T. M. Mitchell. Combining labeled and unlabeled data with co-training. In *Computational Learning Theory*, pages 92–100, 1998.

[4] S. Chakrabarti. Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction. In *WWW 10*, Hong Kong, May 2001. Online at `http://www10.org/cdrom/papers/489`.

[5] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *VLDB Journal*, Aug. 1998. Online at `http://www.cs.berkeley.edu/~soumen/VLDB54_3.PDF`.

[6] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource compilation by analyzing hyperlink structure and associated text. In *7th World-wide web conference (WWW7)*, 1998. Online at `http://www7.scu.edu.au/programme/fullpapers/1898/com1898.html`.

[7] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *SIGMOD Conference*. ACM, 1998. Online at `http://www.cs.berkeley.edu/~soumen/sigmod98.ps`.

[8] S. Chakrabarti, B. E. Dom, D. A. Gibson, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Topic distillation and spectral filtering. *Artificial Intelligence Review*, 13(5–6):409–435, 1999.

[9] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31:1623–1640, 1999. First appeared in the 8th International World Wide Web Conference, Toronto, May 1999. Available online at `http://www8.org/w8-papers/5a-search-query/crawling/index.html`.

[10] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. In *7th World Wide Web Conference*, Brisbane, Australia, Apr. 1998. Online at `http://www7.scu.edu.au/programme/fullpapers/1919/com1919.htm`.

[11] B. D. Davison. Topical locality in the Web. In *Proceedings of the 23rd Annual International Conference on Research and Development in Information Retrieval (SIGIR 2000)*, pages 272–279, Athens, Greece, July 2000. ACM. Online at `http://www.cs.rutgers.edu/~davison/pubs/2000/sigir/`.

[12] P. M. E. De Bra and R. D. J. Post. Information retrieval in the world-wide web: Making client-based searching feasible. In *Proceedings of the First International World Wide Web Conference*, Geneva, Switzerland, 1994. Online at `http://www1.cern.ch/PapersWWW94/reinpost.ps`.

[13] P. M. E. De Bra and R. D. J. Post. Searching for arbitrary information in the WWW: The fish search for Mosaic. In *Second World Wide Web Conference '94: Mosaic and the Web*, Chicago, Oct. 1994. Online at `http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/debra/article.html` and `http://citeseer.nj.nec.com/172936.html`.

[14] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 527–534. Morgan Kaufmann, 2000. Online at `http://www.neci.nec.com/~lawrence/papers/focus-vldb00/focus-vldb00.pdf`.

[15] W. A. Gale, K. W. Church, and D. Yarowsky. A method for disambiguating word senses in a large corpus. *Computer and the Humanities*, 26:415–439, 1993.

[16] M. Hersovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur. The shark-search algorithm—an application: Tailored Web site mapping. In *WWW7*, 1998. Online at `http://www7.scu.edu.au/programme/fullpapers/1849/com1849.htm`.

[17] T. Joachims, D. Freitag, and T. Mitchell. WebWatcher: A tour guide for the web. In *IJCAI*, Aug. 1997. Online at `http://www.cs.cmu.edu/~webwatcher/ijcai97.ps`.

[18] H. Leiberman. Letizia: An agent that assists Web browsing. In *International Joint Conference on Artificial Intelligence (IJCAI)*, Montreal, Aug. 1995. See Website at `http://lieber.www.media.mit.edu/people/lieber/Lieberary/Letizia/Letizia.html`.

[19] H. Leiberman, C. Fry, and L. Weitzman. Exploring the Web with reconnaissance agents. *CACM*, 44(8):69–75, Aug. 2001. `http://www.acm.org/cacm`.

[20] A. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. Software available from `http://www.cs.cmu.edu/~mccallum/bow/`, 1998.

[21] A. McCallum and K. Nigam. A comparison of event models for naive Bayes text classification. In *AAAI/ICML-98 Workshop on Learning for Text Categorization*, pages 41–48. AAAI Press, 1998. Online at `http://www.cs.cmu.edu/~knigam/`.

[22] A. McCallum and K. Nigam. A comparison of event models for naive Bayes text classification. In *AAAI/ICML-98 Workshop on Learning for Text Categorization*, pages 41–48. AAAI Press, 1998. Also technical report WS-98-05, CMU; online at `http://www.cs.cmu.edu/~knigam/papers/multinomial-aaaiws98.pdf`.

[23] F. Menczer. Links tell us about lexical and semantic Web content. Technical Report Computer Science Abstract CS.IR/0108004, arXiv.org, Aug. 2001. Online at `http://arxiv.org/abs/cs.IR/0108004`.

[24] F. Menczer and R. K. Belew. Adaptive retrieval agents: Internalizing local context and scaling up to the Web. *Machine Learning*, 39(2/3):203–242, 2000. Longer version available as Technical Report CS98-579, `http://dollar.biz.uiowa.edu/~fil/Papers/MLJ.ps`, University of California, San Diego.

[25] F. Menczer, G. Pant, M. Ruiz, and P. Srinivasan. Evaluating topic-driven Web crawlers. In *SIGIR*, New Orleans, Sept. 2001. ACM. Online at `http://dollar.biz.uiowa.edu/~fil/Papers/sigir-01.pdf`.

[26] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[27] T. Mitchell. Mining the Web. In *SIGIR 2001*, Sept. 2001. Invited talk.

[28] S. Mukherjea. WTMS: a system for collecting and analyzing topic-specific Web information. *WWW9/Computer Networks*, 33(1–6):457–471, 2000. Online at `http://www9.org/w9cdrom/293/293.html`.

[29] S. RaviKumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the Web graph. In *FOCS*, volume 41, pages 57–65. IEEE, nov 2000. Online at `http://www.cs.brown.edu/people/eli/papers/focs00.ps`.

[30] J. Rennie and A. McCallum. Using reinforcement learning to spider the web efficiently. In *ICML*, 1999. Online at `http://www.cs.cmu.edu/~mccallum/papers/rlspider-icml99s.ps.gz`.

[31] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[32] M. Subramanyam, G. V. R. Phanindra, M. Tiwari, and M. Jain. Focused crawling using TFIDF centroid. *Hypertext Retrieval and Mining* (CS610) class project, Apr. 2001. Details available from `manyam@cs.utexas.edu`.

[33] L. Torgo and J. Gama. Regression by classification. In D. Borges and C. Kaestner, editors, *Brasilian AI Symposium*, volume 1159 of *Lecture Notes in Artificial Intelligence*, Curitiba, Brazil, 1996. Springer-Verlag. Online at `http://www.ncc.up.pt/~ltorgo/Papers/list_pub.html`.