DEADLINER: Building a New Niche Search Engine

A. Kruger, C. L. Giles, F. M. Coetzee, E. Glover, G. W. Flake, S. Lawrence, C. Omlin
NEC Research Institute
4 Independence Way
Princeton, NJ 08540

ABSTRACT

We present DEADLINER, a search engine that catalogs conference and workshop announcements, and ultimately will monitor and extract a wide range of academic convocation material from the web. The system currently extracts speakers, locations, dates, paper submission (and other) deadlines, topics, program committees, abstracts, and affiliations. A user or user agent can perform detailed searches on these fields. DEADLINER was constructed using a methodology for rapid implementation of specialized search engines. This methodology avoids complex hand-tuned text extraction solutions, or natural language processing, by Bayesian integration of simple extractors that exploit loose formatting and keyword conventions. The Bayesian framework further produces a search engine where each user can control the false alarm rate on a field in an intuitive yet rigorous fashion.

Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Search; D.2.2 [Software]:Tools and Techniques

Keywords

Web search, text extraction, Bayesian fusion approaches

1. INTRODUCTION

One of the most exciting benefits resulting from development of the world-wide web is the increased availability of specialized search engines. DEADLINER is a specialized search engine currently under development at NECI, that is aimed at the academic research community. DEADLINER monitors the world-wide web, newsgroups and broadcast e-mail for conference announcements. The system detects relevant documents and extracts speakers, locations, dates, submission and other deadlines, keywords, program committees, abstracts, and affiliations, all of which are stored in a structured database. A user interface is provided that allows the user to construct complicated queries using any of the extracted fields. DEADLINER is being extended to address smaller local events such as seminars, workshops and other colloquia. Figure 1 displays some of the capabilities of DEADLINER.

DEADLINER is an example of a niche search engine; that is, a search engine that is targeted to a specialized web community. In contrast to general purpose engines such as Google, specialized engines scour cyberspace with the goal of indexing only a small subset of documents relevant to the community. The power of customized search engines derives from the fact that the underlying domain is constrained, and documents within this community have common elements. By modeling and extracting these elements, complex queries that take advantage of the sophistication of the community members within their field can be implemented. Specialized search engines are also well suited for use as oracles by user agents. For example, using DEADLINER, a researcher can do more than simply monitor a selected set of conferences; a simple calendar-based agent can easily use DEADLINER to provide alerts when a person of interest appears at a lesser known but local event. Other examples of non-trivial and highly successful niche search engines aimed at the academic community are ResearchIndex [11], and CORA [14] both of which extract citation fields from online publications, and perform full page content and citation analysis.

The major obstacle to creating niche search engines is creating reliable mechanisms capable of detecting relevant documents, and extracting target elements, from a wide range of sources. For example, in DEADLINER, conference related fields must be found in documents of widely varying formats, while construction of ResearchIndex required the encoding of extensive knowledge concerning citation formats. Due to the difficulty of this task, small communities may not have the resources to construct systems such as DEADLINER or ResearchIndex from the ground up. Aside from its functional use, DEADLINER was constructed to investigate suitable methodologies for rapidly building specialized search engines. Ultimately we would like to provide a simple toolkit, that can be used as a framework for rapid implementation by a small user community or interest group. This paper therefore not only serves to introduce DEADLINER, but documents a novel and general methodology for building niche search engines.

The methodology inherent in the architecture of DEADLINER reflects our belief that documents on the world wide web often contain enough structure (e.g., formatting infor-

CIKM 2000, McLean, VA USA

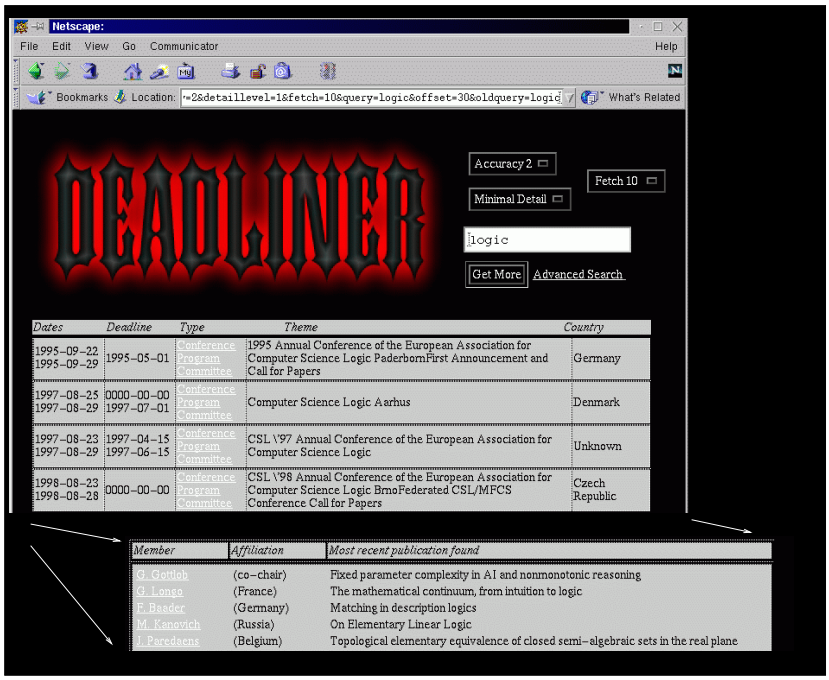

Figure 1: Composite of two DEADLINER interfaces displaying fields automatically extracted from conference announcements. The first screen shows deadlines and themes, while the second screen shows program committee members.

mation, link structure and keyword fields) to allow target fields to be extracted without resorting to natural language processing. We favor machine learning, rather than hand-tuning, of the system. Finally, our architecture emphasizes the integration of multiple partial solutions for extracting text fields rather than a single monolithic solution. Solutions with various degrees of sophistication can readily be produced by different individuals. We defer detailed comparisons with other methodologies to Section 5.

The outline of the paper is as follows. In Section 2 we provide a high-level overview of the architecture of the search engine, and outline the major challenges. We then proceed to address each of the major parts of our general methodology. Section 2.1 describes document retrieval while Section 2.2 describes pre-screening of documents for relevance using Support Vector Machines. Section 2.3 describes detection of target fields, such as deadlines, by Bayesian integration of multiple binary detectors. This section presents the core technical advance of this paper. Section 2.4 discusses setting operating points and presenting results to the users. The filter extractors unique to DEADLINER are described in Section 3, and their performance in Section 4.

2. GENERAL ARCHITECTURE OVERVIEW

The architecture of the specialized search engine produced by our methodology is shown in Figure 2. In the first stage, different retrieval mechanisms locate documents in cyberspace. A second stage performs pre-screening of the documents for relevancy. Relevant documents are then forwarded to a third stage, where multiple extraction filters detect and subsequently extract target fields from the document. Target fields are then stored in an SQL database.

The major characteristic of the system is the focus on a large number of modular extraction filters for each target field, with automatic integration of these filters using learning techniques. This approach resulted from the requirement that the system reliably extract target fields from a wide variety of disparate formats. At first, we tried to extract each field, such as the submission deadline, by constructing monolithic solutions. The complexity and effort required to achieve any degree of reliability proved to be prohibitive. However, we found that it is easy to postulate multiple simple solutions of different specificity, built on regular expressions, suited to extracting target fields from specific formats. Unfortunately, choosing between these different solutions, estimating their robustness, or combining them appropriately, defies human analysis and intuition. However, Bayesian learning techniques offer a solution. The approach we provide automatically integrates the simple regular expression filters in an optimal fashion using relatively modest amounts of labeled data. Improved performance over a single extractor can be achieved, since the integration phase exploits the joint statistics across the different filters.

The Bayesian integration step also addresses a second major problem that bedevils search engine designers; that of selecting an operating point that is acceptable for all users. For visual display, if one has a confidence ranking on query matches, one can simply display results in decreasing order of confidence. However, when an action is to be automatically triggered (for example, an e-mail is sent or a calendar entry created when a specific event is detected), users can exhibit markedly different tolerance and preferences for errors. A key consequence of the Bayesian integration approach of Section 2.3 is that a mechanism results for varying the operating point for each user, even though the operating point of each simple extraction filter is fixed. Performance

variation is monotone and controlled by a single parameter. Further, as new filters are added, the system can evaluate their efficacy and integrate them as appropriate.

2.1 Stage I: Document Retrieval

A large part of the information in DEADLINER is time sensitive, and hence a major problem is ensuring currency without requiring the resources necessary for timely exhaustive crawling of the web. We address this problem using a multi-pronged approach.

First, we have simple polling scripts that download documents from well known sources where seminar and conference related materials commonly appear, such as newsgroups, universities and professional organizations.

To provide coverage of the rest of cyberspace, we borrow systems from other NECI projects. We adopted a Context-Graph Focused Crawler as an input stage, which uses the link structure and contents of documents to improve the retrieval rate of documents related to the training set. A complete discussion of this technique is beyond the scope of this paper. We refer the interested reader to [4].

The last input module finds relevant documents from commercial search engines. For this stage, we extracted query modifiers (essentially a list of keywords) that describe our training data. These query modifiers are periodically submitted to approximately ten commercial search engines via Inquirus 2, a meta-search engine that supports query modifiers. In this way, we leverage the coverage and bandwidth of commercial search engines. Detailed descriptions of this work are available in [6, 7].

2.2 Stage II: Pre-Screening Using SVMs

While some of the retrieval stages (such as the focused crawler), enforce some degree of document relevancy, the fraction of irrelevant documents produced by the input stage is still high. The downloaded documents are therefore prescreened using text classification. We favor SVMs for screening web documents, because of the ability of SVMs to handle large dimensionality vectors while resisting over-fitting. We used the standard SMO SVM algorithm described by Platt [17, 18], and we therefore do not describe the details here. We refer the reader to Kwok [10] and Joachims [9] for detailed discussions comparing SVMs to several different text classifiers, and to Vapnik [20] for a complete technical description. Suffice to say that SVMs embed data vectors in an infinite dimensional kernel space, typically by locating a functional kernel on each data vector. The class vectors are subsequently separated by hyperplanes that maximize the margins between the different classes, which controls complexity and ensures generalization. The following paragraphs detail the construction of the feature vectors we use as input to the SVM.

We convert all the text documents into a set of binary features vectors relative to a fixed vocabulary V. To construct V, we use a labeled training set. We extract all words, bigrams and trigrams from the documents, and calculate the frequency of occurrence of each of these. If a word, bi- or trigram occurs in more than 7.5% of either the true, or the false class documents, it is considered a candidate to be included in V. The candidates are ranked in order of the ratio of their frequency on the true class, to their frequency on the false class documents. The top N (typically 100 to 300) candidates then form the vocabulary V.

A document is represented relative to the vocabulary V by a binary vector x where a 1 occurs in every position corresponding to the appearance of an element of V in the document, with zeroes elsewhere. A target value y of either 1 or 0, respectively, indicates whether the document is relevant or not.

2.3 Stage III: Bayesian Detector Fusion

A major problem in extracting target fields is finding suitable pieces of the document to apply extraction rules. Applying an extraction rule blindly to large portions of the document causes several false alarms. To address this problem, we decided on an approach that first performs reliable detection of relevant sentences or blocks or text, and then performs extraction only on regions of high likelihood.

For every target field we construct a detector by optimally integrating a number of simpler detectors. The problem we address is shown in Figure 3. Consider a set of regular expressions or simple formatting templates for processing text; for example, these could all be aimed at extracting the submission deadline. We refer to these elements as filters. With each filter we associate a binary variable indicating whether a match occurred or not. The combination of filter and match variable will be referred to as a detector, denoted f_i in the figure. At issue is how to integrate these partial detectors. The integration should yield a new detector whose performance exceeds that of any of the constituent detectors, and whose operating point (precision/recall setting) can be easily changed, even though the filters are all fixed.

We formalize the problem as follows. Each of the detectors function as a binary classifier, also known as a categorical function, defined as a mapping f from an input space X (the text) to the space $\{0, 1\}$. Given N classifiers $f_i : X_i \rightarrow \{0, 1\}$, we wish to find the classifier $\Gamma : \prod_{i=1}^{N} f_i(X_i) \rightarrow \{0, 1\}$ with the highest probability of detection for a given rate of false alarm.

We use the binary detector outputs as a new feature space. We denote the combined output of the detectors for a given set of inputs x by a bit string, or more formally the ordered pair $y = f(x) = (y_1, y_2, \ldots y_N) = (f_1(x_1), f_2(x_2), \ldots f_N(x_N)), x_i \in X_i, i = 1, 2, \ldots N$. We assume two hypotheses H_0 (false class, or irrelevant) and H_1 (true class, relevant), which yields class conditional probabilities $\mathcal{P}\{f(x) \mid H_0\}$ and $\mathcal{P}\{f(x) \mid H_1\}$ respectively on the feature space. The problem of combining the simple detectors is then equivalent to that of selecting Γ from the set of all categorical mappings on N binary variables. Considering all inputs x yields up to 2^N possible bit strings y. It follows that there are up to 2^{2^N} distinct different classification rules $\Gamma_j, j = 1, 2, \ldots 2^{2^N}$ for combining the simple detectors. Each of these decision rules Γ will yield a particular value of false alarm P_f and of detection P_d, defined by summing over the set of bins $\mathcal{L}(\Gamma)$ labeled as true class by the classifier:

$$P_f(\Gamma) = \sum_{y \in \mathcal{L}(\Gamma)_1} \mathcal{P}\{y \mid H_0\} \qquad P_d(\Gamma) = \sum_{y \in \mathcal{L}(\Gamma)_1} \mathcal{P}\{y \mid H_1\} \quad (1)$$

The set $AOS = \{(P_f(\Gamma_j), P_d(\Gamma_j))\}$ of the operating points defined by the 2^{2^l} binary mappings on a feature set is referred to as the Achievable Operating Set. The Receiver Operating Curve (ROC) is the set of operating points yielding the maximal detection rate for a given false alarm rate. The ROC efficiently summarizes the inherent difficulty of

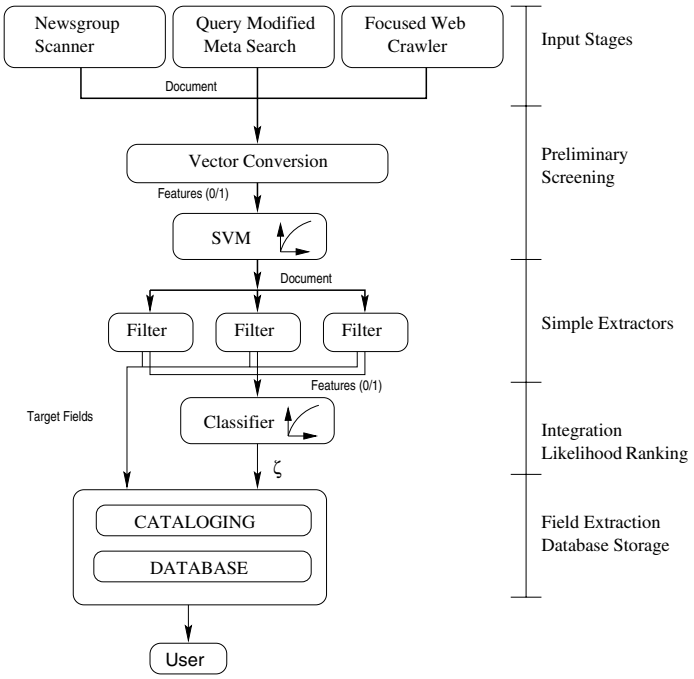

Figure 2: The basic framework of our specialized search engine implementation. Different modules scan the documents. A primary classifier selects individual documents for further processing. A secondary classifier assigns posterior threshold ratios ζ to every document and extracts target elements. The user interacts with the database by selecting a specific operating point for accesses, and providing standard queries.

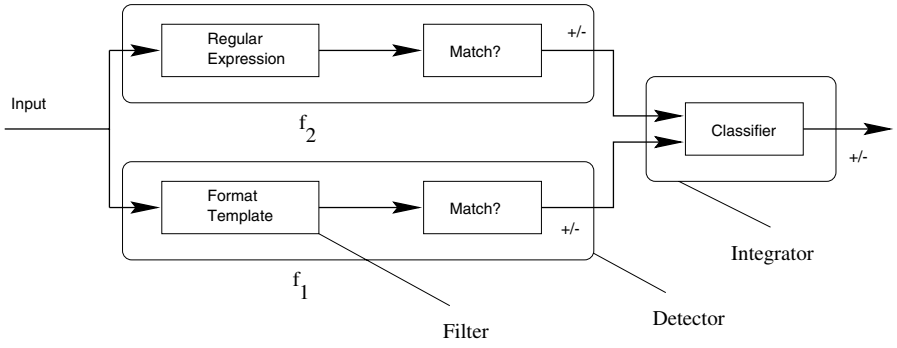

Figure 3: Our approach to probabilistic binary detector integration. Simple fixed filters representing partial solutions are generated using a-priori domain knowledge. The categorical outputs from the filters extractors are automatically recombined using a classifier (the integrator) that exploits the joint statistics.

separating the two classes. The subset of the 2^{2^N} classifiers Γ_j lying on the ROC will be referred to as the ROC support classifiers, and are denoted by the superscripted variables Γ^i.

Exhaustive enumeration of the classifiers Γ to find the ROC is and will be practically impossible except for trivially small cases (even when $N = 5$, $2^{2^N} \simeq 4.3 \times 10^9$). The Neyman-Pearson (NP) design procedure provides the optimal solution to the problem of efficiently obtaining the ROC. This procedure ranks the 2^N possible strings x according to the likelihood ratio function $\zeta : \chi \rightarrow \Re^+$

$$\zeta(x) \quad = \quad \mathcal{P}\{f(x) \mid H_1\}/\mathcal{P}\{f(x) \mid H_0\} \qquad (2)$$

The ROC support classifiers are found in order of increasing false alarm performance by successively assigning strings in decreasing order of likelihood ratio to the true class decision region. Hence, there are 2^N ROC support classifiers.

We make the above discussion concrete by providing a simple example. Consider the hypothetical case where we have two binary features $y = (y_0, y_1)$ indicating matches of two underlying regular expression extractors. Assume that the true class conditional density functions on the data are as follows:

$y_1 y_0(y)$	00(0)	01(1)	10(2)	11(3)
$p(y\|H_1)$	0.30	0.35	0.20	0.15
$p(y\|H_0)$	0.15	0.25	0.40	0.20

The ROC support classifiers $\Gamma^i(y), i = 0, 1, 2, 3$ developed via the NP design are as shown below:

$y_1, y_0(y)$	00(0)	01(1)	10(2)	11(3)	P_f	P_d
$\zeta(y)$	2.00	1.40	0.50	0.75		
$rank(\zeta(y))$	0	1	3	2		
$\Gamma^0(y)$	0	0	0	0	0.00	0.00
$\Gamma^1(y)$	1	0	0	0	0.15	0.30
$\Gamma^2(y)$	1	1	0	0	0.40	0.65
$\Gamma^3(y)$	1	1	0	1	0.60	0.80
$\Gamma^4(y)$	1	1	1	1	1.00	1.00

For example, to achieve a detection rate of 65% at a false alarm rate of 40%, we use classifier $\Gamma^2(x)$, which effectively uses only the regular expression of filter 1. A different operating point, such as produced by $\Gamma^3(y)$ would require arbitrating between the two regular expressions.

An extremely important point to note is that a classifier Γ^j is created by labeling all feature combinations y (i.e. histogram bins) whose likelihood ratio $\zeta(y) \geq \zeta_j$, where ζ_j is the jth ranked likelihood value, as true class. Therefore, when processing a text field, we do not pick a specific classifier and operating point up front. Instead, we map every feature combination (bin y) to its likelihood $\zeta(y)$, which is stored in the database. We can then later easily choose any detection operating point by thresholding the likelihood values. In this way, every user or agent can maintain a different operating point by varying a single personal threshold.

Another important benefit of Neyman-Pearson design is a degree of robustness to changes in the a-priori class conditional probabilities. A change in these probabilities results in a shift of the operating point along the ROC, but does not change the set of ROC classifiers. Therefore, it is possible to account for a change in the environment by adjusting the global user threshold, without a need to re-classify the stored data set. This property is an advantage on the web, where it is a difficult problem to estimate the a-priori class probabilities.

The Neyman-Pearson design approach is a search procedure, where the problem of finding the classifier functions Γ that maximizes the P_d at a given value of P_f is reduced from searching a space of dimension 2^{2^N} to one of searching a space of dimension 2^N, an enormous reduction in complexity.

In practice, the class conditional distributions are unknown and statistics must be estimated from a finite labeled data set. Ensuring that the histogram estimates are accurate is a major subject of investigation. For detailed analyses and solutions to balancing data set size, reliability and number of filters we refer the reader to [3, 2].

However, we note that in the DEADLINER system we have relatively little training data (on the order of 500 conference pages). As such, we cannot estimate histograms combining more than approximately four detectors (16 histogram bins). To take advantage of the large number of filters we therefore have to search for the best combination of four (or fewer) of the N detectors. Performing a full ROC calculation for every such subset is feasible when N is less than approximately thirty, as was the case in our designs.

It is generally not possible to rank ROC curves absolutely over different subsets of detectors, since ROC curves can cross. This crossing simply reflects the fact that different combinations of detectors are desirable for different ranges of operating points. Unfortunately, a severe performance penalty can result from using a single subset of filters over all false alarm rates.

We address this problem by noting that by switching between the outputs of any two classifiers with some probability, any operating point (P_f, P_d) on the line connecting the operating points of the two classifiers can be produced [19]. Hence, any operating point within the convex hull of the ROCs of all the detector combinations can be obtained. We therefore proceed by calculating the ROCs of all the possible combinations of four or fewer detectors. We then construct the convex hull of the ROC curves. The resulting ROC curve is implemented by using different sets of detectors for different operating ranges, and performance exceeds or matches the performance of any of the possible detector combinations. In this way we retain the benefit of having a large number of detectors, while never integrating more features simultaneously than can be supported by the data.

2.4 Stage IV: Presentation and Cataloging

In the previous section we described how we can combine the outputs of simple detectors to detect a region of text that contains a possible match for a target element. However, we note that in order for the system to work, the search engine also has to extract the relevant fields from these text regions. Note that every extractor produces both a feature indicating a match (0 or 1), and an actual estimate of the value of the target element (an associated block of text that triggered a match). The binary features are integrated as previously described, allowing for an overall detection operating point to be set for each target element. A particular setting might overrule one or more of the constituent filters, require a particular combination of features, or enforce some other joint relationship. The text fields extracted by the filters that indicated a match and also have a positive weight in the integration are merged, and then processed for extraction using heuristics (e.g., the smallest common text segment is used).

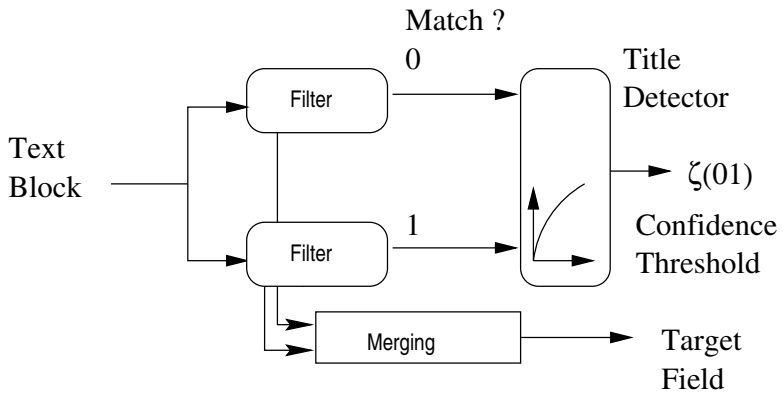

Figure 4: Cataloging section showing extraction of a target field, here the title. A number of different filters attempt to recognize the desired target. Every match results in a binary feature, which is integrated with responses from other detectors. Based on the ROC, every combination of filter matches can be mapped to an associated threshold ζ, ROC support classifier Γ^j and operating point (P_f, P_d). The value of ζ also reflects an overall detection confidence. This value is stored with every field value obtained by merging the fields of the active filters.

Typically we process single paragraphs at a time, and multiple matches refer to the same text segment. However, in long paragraphs, or when considering the document as a whole, a target element can be detected in different places, even though these elements may be subject to restrictions (titles almost always occur only once). Further, the data extracted from the matches may conflict. Note, however, that each integrator provides a confidence estimate in the form of the threshold ζ associated with each detected text segment. If the confidence in a given match is much higher than that of other matches, only the match with the highest confidence is used. However, when multiple matches of equal or almost equal confidence but differing values occur, the system will index all of the matches. In this way, queries might return incorrect results, but in principle no irreversible decisions are made.

3. FILTERS USED IN DEADLINER

The architecture in the previous sections can easily be ported to different community interests by developing relevant extractors (regular expressions), and labeling a set of training data. This section describes these detailed components of DEADLINER.

3.1 Features and Filters

From visual inspection of a number of document sources, we soon found that most conference materials follow a block layout. Major blocks are: the title, a listing of affiliations, abstracts, organizers such as a program committee, discussion topics, venue, scope and objective statements, and a miscellaneous information section (describing topics ranging from the weather to local social events). The block structure significantly simplifies the extraction problem and filters exploiting it produce excellent results without extensive natural-language processing.

The simplest filters in the system perform keyword matching, based on a vocabulary V generated using word frequencies. We also make extensive use of databases of lists of authors (extracted from ResearchIndex), lists of research areas and keywords (extracted from ResearchIndex) and venue and geographical names.

Using these databases, a large number of simple primitives were constructed. The primitives are then used to construct filters that enforce constraints related to the block structure. Table 3.1 shows a subset of the filters that are currently implemented; these filters are aimed at extracting four common elements of conference announcements, namely title, deadline, topic and program committee. The filters are described using an extension of standard regular expression notation that allow for formatting and database lookup: * means zero or more (Kleene closure), + means one or more, <*description*> indicates information obtained outside the regular expression, or is used for clarity. We use ω for the set of alphanumeric characters, and ς for whitespace (newlines, tab-stops, spaces). A range of characters is denoted by [] (e.g., $[A-Z]$ denotes capitals). The function $indentation(text)$ returns a list of numbers indicating the level of indentation for each line in the text, while $max(list)$ and $min(list)$ returns the maximum and minimum of the numbers in the list respectively. The operators *country_names* reflects presence in a list of country names, and *known_names* a match in a list of about 90,000 proper names. We use the notation <*regularly_occurring_separators*> for separators that occur most frequently in a list. The separator may contain whitespace, but does not consist only of whitespace, and is always preceded by a newline. A B indicates a word/non-word boundary, and .. is used to indicate a range. A | means "or" (unification), while () indicates grouping. The operator ! negates a character set, while a full stop (.) matches any character except a newline. Occurrences of a symbol of least n times but not more than m times is denoted by $\{n, m\}$, while #(<*expression*>, <*expression*2>) counts the number of times <*expression*2> occurs in <*expression*>. In (<*expression*>, <*expression*2>) the part <*expression*> is the index of the text block, while $\wedge$ is used to anchor the beginning of text.

Constructing fixed lower level filter parameters using such simple heuristics is the core of our approach; instead of trying to solve the difficult problem of varying parameters internal to each filter, we achieve different operating points by diversifying across different filters.

3.2 Heuristics

Name	Primitive	Target	Meaning
0	$(cur\_par, /\$kwds/)$	title	match keywords in current paragraph
1	$(cur\_par, /\mathbf{B}on\mathbf{B}/)$	title	match "on" in current paragraph
2	$\#(cur\_par, /\omega^+\varsigma^+/) <= 30$	title	count the number of words
3	$\#(cur\_par, /[A-Z]\omega^*\varsigma^+/)/\#(cur\_par, /\omega^+\varsigma^+/) >= 0.75$	title	ratio of capitalized words:words
4	$min(indentation(cur\_par)) \neq min(indentation(prev\_par))$	title	difference in indentation
5	$\#min(indentation(cur\_par)) \neq min(indentation(next\_par))$	title	difference in indentation
6	$(cur\_par, /<date>/)$	title	match a date
7	$(cur\_par..cur\_par+5, /<date>/)$	title	match a date in the next 5 paragraphs
8	$(cur\_par, <= 7)$	title	this is one of the first paragraphs
9	$(cur\_par, /<countryname>/)$	title	match a country name
10	$\#(cur\_par, /[!\omega\varsigma]+/) > 20$	title	count the non-whitespace, non-alphanumeric characters
0	$(cur\_sentence, /deadline/i)$	deadline	match the word "deadline"
1	$(cur\_sentence, /by\|before\|due\|closing/i)$	deadline	match "by" or "before" case insensitive
2	$(cur\_sentence, /later\varsigma^+than/i)$	deadline	match "later than" case insensitive
3	$(cur\_sentence, /on/i)$	deadline	match "on" case insensitively
4	$(cur\_sentence, /<date>/i)$	deadline	match a date
5	$\#(cur\_sentence, /<date>/i) \geq 3$	deadline	there are three or more dates
6	$\#(cur\_sentence, /<date>/i) > 0$	deadline	match a date
7	$(cur\_sentence, /submi/i)$	deadline	match "submi"
8	$(cur\_sentence, /paper/i)$	deadline	match "paper"
9	$(cur\_sentence, /:/i)$	deadline	match ":"
10	$(cur\_sentence, /notify\|notification\|accept\|camera/i)$	deadline	match "deadline qualifiers"
11	$(cur\_sentence, /important\varsigma^+date/i)$	deadline	match "important date"
0	$(prev\_par, /\$kwd/)$	topic	match a keyword
1	$(prev\_par, /:/)$	topic	match a colon
2	$(prev\_par\ or\ cur\_par, /<regularly\_occurring\_separator>/)$	topic	is there a regularly occurring separator?
3	$(cur\_par, /<regularly\_occurring\_separator>/)$	topic	is there a regularly occurring separator?
4	$\#(cur\_par, /./)/\#(cur\_par, /<line>/) < 0.1$	topic	ratio of full stops:number of lines
5	$(cur\_par, /\$kwd/)$	topic	match a keyword
6	$(cur\_par, /limited)or(cur\_par, /limited\varsigma^+to/i)$	topic	match "limited to"
7	$(cur\_par, /includ/i)$	topic	match "includ"
8	$min(indentation(cur\_par)) = min(indentation(next\_par))$	topic	minimum indentation differs
9	$(cur\_par, /interest/i)$	topic	match "interest"
10	$max(indentation(cur\_par)) = max(indentation(next\_par))$	topic	maximum indentation differs
0	$(cur\_par, /\$kwds/)$	program committee	match the keywords
1	$(cur\_par, /univ/)$	program committee	match "univ
2	$(next\_par..next\_par+1, /univ/)$	program committee	match "univ
3	$(cur\_par, /<countryname>/)$	program committee	match country names
4	$(next\_par..next\_par+1, /<country\_names>/)$	program committee	match country names
5	$(cur\_par, /<known\_names>/)$	program committee	match known names
6	$(next\_par..next\_par+1, /<known\_names>/)$	program committee	match known names
7	$(cur\_par, /[!\omega\varsigma]^+/)$	program committee	match non-whitespace, non-alphanumeric chars
8	$(next\_par..next\_par+1, /[!\omega\varsigma]^+/)$	program committee	match non-whitespace, non-alphanumeric chars
9	$\#(cur\_par, /\mathbf{B}[A-Z]\mathbf{B}/)$	program committee	number of single letter words
10	$(next\_par..next\_par+1, /\mathbf{B}[A-Z]\mathbf{B}/)$	program committee	number of single letter words

Table 1: Primitive operators used for constructing filters. Each primitive is aimed at extraction of a particular target concept. We also associate a specific area of text with each filter.

As noted earlier, following detection of a target element, the option exists to use heuristics to combine the text fields extracted by the active filters. For example, a detector might return a sentence with additional text that has to be trimmed. We briefly describe three elements to provide an indication of the complexity of the approach:

Program Committees: The text obtained from the detectors is split into tokens and matched against a dictionary of known author names and possible affiliations obtained from ResearchIndex papers (there are approximately 90,000 distinct dictionary elements), then against a list of common dictionary words, and ultimately a dictionary of country and place names. All matching words are replaced by symbols denoting the dictionary in which they were found. We then use regular expression templates to match the resulting symbolic strings. A regular expression is then constructed for all the positive matches, and re-applied to the text to find persons not in the dictionary.

Deadlines: We match standard date formats in sentences and tables. Extraction of the surrounding or immediately preceding text is used to determine the type of deadline (e.g., abstract submission date).

Titles: A title usually contains at least two of the following: (i) country name, (ii) city/state name, (iii) date of meeting, (iv) deadline, (v) list of sponsors, (vi) name, (vii) acronym for the conference, and (viii) theme/summary of the conference. We enforce the presence of at least two elements. Most elements are recognized by matching entries from a database. The summary is generated by removing elements (i) through (viii) from the match.

4. PERFORMANCE

SVM Performance: The data used to train the SVM in Section 2.2 was a set of 592 manually classified Calls for Papers (CFPs), and 2269 negative examples, consisting of several "random" URLs from the Inquirus 2 logs, and about 850 conference related pages. To qualify as a CFP we require a title describing the event, a list of topics, a program committee, deadlines and submission information. CFPs were collected by combining URLs from documents containing lists of CFPs, and by combining searches from various search engines (For example, search for "Calls for Papers" in a normal search engine).

The training set consisted of 249 positive and 1250 negative documents (randomly selected, with a limit of 20 pages from any one domain to prevent bias). The remaining 343 positive and 1019 negative examples formed the test set.

Table 2 summarizes the results on the test set. As can be seen, we obtain excellent results using extremely limited structural processing. From our evaluation of the test set, we noted that non-English or multilingual sites represent a major problem for our system, as expected from the biases of our dictionaries. While a Gaussian kernel in the SVM produces noticeable improvements over a linear SVM, the significant extra overhead does not warrant its use at this moment. We therefore use the linear classifier.

Extractor Performance: We had 500 documents from DB-World (http://www.cs.wisc.edu/dbworld/) for training, and 100 documents from DIKU (http://www.diku.dk/research-groups/topps/Conferences.html) for testing of the feature extractors. We made the deliberate decision not to gener-

Collection	Type	No. Pos	No. Neg	Pos. Accuracy	Neg. Accuracy
CFP Test	Gauss	343	1019	95.9%	98.6%
CFP Test	Linear	343	1019	88.1%	98.7 %

Table 2: Summarized results for the SVM call for paper classifier. There was no overlap with the training set.

ate test and training data by sampling from both data sets. On the web we are extremely likely to process data sources that we have had no representative data for. We wanted to see how fragile our feature extractors are when the class distributions in deployment differ significantly from that in training.

These data sets were labeled to indicate desired target fields. The DBWorld documents contain 208 lists of interesting topics, 338 conference titles, 906 deadlines and 197 program committees. We chose not to exclude announcements that strictly are not calls for papers, since our SVM classifier misclassifies a small percentage of web documents, and we therefore need fairly robust detectors.

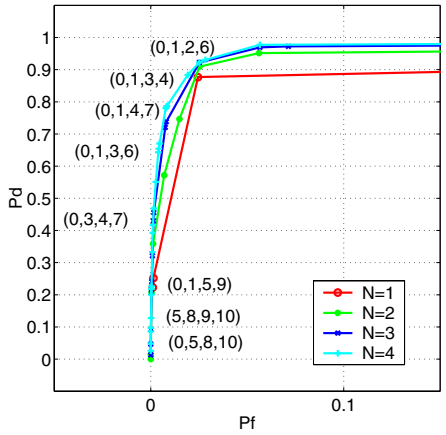

Figure 5: Detection performance for the target field "deadlines" for different numbers of combined detectors N. Also shown are the optimal indices of the detectors when $N = 4$. Integrating multiple detectors provides a significant performance improvement. For example, at a false alarm rate of 2%, integrating $N = 4$ detectors results in an improvement of almost 20% over the best individual detector.

Figures 5-7 contain the ROC curves obtained on the DBWorld data for different integrators that detect the deadlines, conference topics and program committee members. Note that these curves are statistically accurate, conditioned on the class distribution of new data matching that of the DBWorld data. In each case we show the performance obtained by the best individual detectors ($N = 1$), as well as combinations of from two through four detectors. In each case a significant performance improvement results from integrating multiple detectors. For example, for deadline extraction, at a false alarm rate of 2%, integrating $N = 4$ detectors results in an improvement of almost 20% over the best individual detector. The curves also show the features that are preferably integrated as a function of the desired operating point, for $N = 4$. The values clearly show that different combinations are preferred in different ranges, although some filters, such as the keyword filters, are always

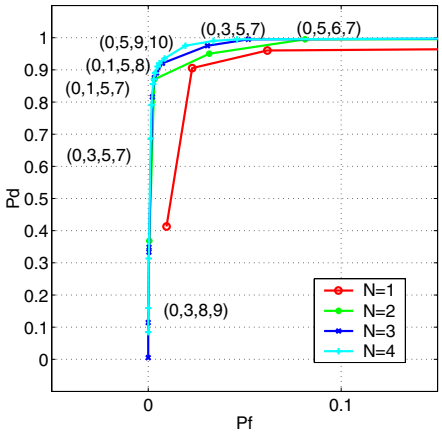

Figure 6: Detection performance for the target field "program committee" for different numbers of combined detectors N. Also shown are the optimal indices of the detectors when $N = 4$. Integrating multiple detectors provides a significant performance improvement.

desirable.

Having detected the target field, the actual target field value was extracted using heuristics. These results are contained in the DIKU data, and are shown in Tables 3-4.

The first row of Table 3 shows the results on deadline extraction. In each case we selected an operating point with a target rate of P_f of around 5%. The document set contained a total of 300 deadline date fields. Of these, 214 deadlines were correctly detected and extracted, while 2 dates were detected, but incorrectly extracted. A total of 31 non-deadline dates were detected and extracted (usually dates from a program announcement), and are considered extraneous. Note that these dates will be indexed but the user can resolve the ambiguity. We required that the text that describe the deadlines had to be perfectly extracted: 86% of the time the text was considered to be correct. Our overall deadline extraction is therefore approximately 70%. By evaluating the errors, we found that errors usually result when dates are given in tables; we aim to develop new parsers to improve processing in this domain.

The second row of Table 3 summarizes the performance on the program committee extraction task. Of the 1455 program committee members, we found 1252 with our system. This performance corresponds to an 87% accuracy.

Table 4 shows the performance when extracting parts of the title. Due to the fact that title composition varies widely, we provide these results as percentages. While we expected difficulty with the date fields, we had relatively disappointing performance on country and city names. We believe that an expansion of our dictionaries will improve performance. Better performance was achieved in identifying the theme and type of conference; we correctly identified almost 90%

Target	Total	Detected/Extracted	Detected/Not Extracted	Extraneous
Deadline	300	214	2	31
Committee & Affiliation	1455	1252	72	136

Table 3: Extraction results for target concepts "Deadline" and "Program Committee".

Start Dates	End Date	Theme/Name	Country	Type of Meeting
73%	71%	81%	77.5%	85%

Table 4: Extraction results for target concept "Title", with associated subfields.

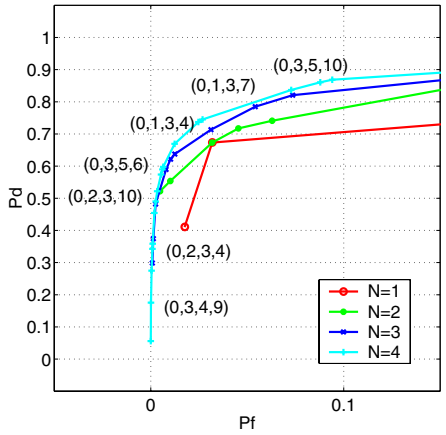

Figure 7: Detection performance for the target field "topics" for different numbers of combined detectors N. Also shown are the optimal indices of the detectors when $N = 4$. Integrating multiple detectors provides a significant performance improvement.

of conference names and the type of meeting.

As expected, there was a marked increase in false detection when the DBWorld filters are used on the DIKU data source (the error usually doubling from that expected). This error shows the challenges inherent in developing systems that can deal with the different formats on the web, and of obtaining representative data for web documents. The start and end dates were sometimes confused (especially when one of the two was not found). However, the results are adequate for DEADLINER to be a useful tool. As it is used, we expect to receive additional labeled data from user feedback, which will be used to improve the system.

5. RELATED AND FUTURE WORK

At this stage it is informative to compare DEADLINER against two other niche search engines that perform extensive field extraction, namely ResearchIndex [11] and CORA [14].

CORA uses Hidden Markov Models (HMMs) to extract information in citations. ResearchIndex, in contrast, uses hand-constructed algorithms and heuristics, where the most uniform features are first parsed and syntactic relationships are used to predict other fields. Both these systems required significant effort to construct, which could prove problematic when porting their architectures to new applications. DEADLINER has a simpler and we believe more flexible approach, which uses simple filters to select appropriate text blocks to narrow the text space and then applies simple heuristics.

In our development of extractors using the DEADLINER framework most of the filters were simple regular expressions. The option exists to use the wealth of existing approaches for learning regular expressions, simple rules or HMMs automatically [15, 12, 5]. In particular, these approaches are especially useful in modeling structured documents when markup elements are present[1]. Some notable approaches have tried to automate extraction using these techniques. For example, Hsu[8] trains a finite state tranducer (SOFTMEALY) for token extraction. Hsu utilizes a heuristic to prevent non-determinism in the FST. Contextual rules are produced by an induction algorithm. The FST's obtained are applied to HTML pages. Similarly, XWrap[13], is a wrapper construction system; XWrap transforms HTML pages into XML. Rules are generated and applied to HTML, and interesting document regions are identified via an interactive interface. The same is done for semantic tokens, followed by a hierarchy determination for the content, resulting in a context free grammar. One of the goals of this system is minimal user interaction.

Stalker[16] is another algorithm that uses landmark automata to generate wrappers. Stalker is a greedy sequential covering algorithm, and tries to form a landmark automaton that accepts only true positives by iterating until it finds a perfect disjunct or runs out of training examples, where the best disjunct is the one that covers the most positive examples. New disjuncts are added iteratively to cover uncovered positive candidates.

While most of the methods above can exploit unlabeled data (as long as the corpus is known to consist of relevant pages), fully automated learning techniques require a large amount of data. Especially the labeling of sufficient data at the target field level for these approaches to work is a significant effort. In DEADLINER we focus on having human designers develop simple rule expressions, and using a more modest data set for performing the integration.

6. SUMMARY

We presented a new research tool, DEADLINER. DEADLINER currently catalogs conference and workshop announcements, extracting deadlines, topics and program committees. These elements already allow researchers or their agents to find conferences covering relevant topics in fields outside their core disciplines, and monitor smaller workshops. We hope that DEADLINER will ultimately be able to extract from the web a wide range of academic convocation and seminar related materials.

DEADLINER was constructed in a modular fashion and can be reconfigured to create niche search engines of other web communities. The architecture reflects our belief that within a domain (especially on the web), simple formatting

conventions, specialized dictionaries and key phrases contain enough information to allow for field extraction without the need for extensive natural language processing. Good performance can be achieved with a reasonable amount of effort by pre-screening documents using term frequency classifiers, and then integrating multiple simple detectors to tag text sections for target field extraction. We presented a general method for integrating a set of partial extraction solutions, such as regular expressions, to perform the detection and provide a confidence estimate for the extraction. Further, by combining multiple partial solutions a classifier can be constructed spanning an ROC curve. In this way, different operating points can be selected by the user or an agent in an intuitive yet rigorous manner by varying a single monotone parameter, as opposed to having the user manipulate multiple poorly-behaved filter parameters.

7. REFERENCES

[1] B. Adelberg. Nodose - a tool for semi-automatically extracting structured and semistructured data from text documents. In Proc. SIGMOD '98, 1998.

[2] F. Coetzee, E. Glover, S. Lawrence, and C. L. Giles. Feature selection in web applications using ROC inflections and power set pruning. Technical Report 2000-028, NEC Research Institute, 2000.

[3] F. Coetzee, S. Lawrence, and C. L. Giles. Bayesian classification and feature selection from finite data sets. In Proceedings UAI2000, Stanford, CA, July 2000. UAI. Preprint available at http://www.neci.nj.nec.com/homepages/coetzee/.

[4] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In Proc. Very Large Data Bases 2000 (VLDB 2000), September 2000. To appear.

[5] L. Firoiu, T. Oates, and P. Cohen. Learning regular languages from positive evidence. In Twentieth Annual Meeting of the Cognitive Science Society, pages 350–355, 1998.

[6] E. J. Glover, S. Lawrence, W. P. Birmingham, and C. L. Giles. Architecture of a metasearch engine that supports user information needs. In Eighth International Conference on Information and Knowledge Management (CIKM'99), pages 210–216, Kansas City, MO, November 1999. ACM Press.

[7] E. J. Glover, S. Lawrence, M. D. Gordon, W. P. Birmingham, and C. L. Giles. Web search – your way. Communications of the ACM, To appear.

[8] C. Hsu. Initial results on wrapping semistructured web pages with finite-state transducers and contextual rules. In "Papers from the 1998 Workshop on AI and Information Integration". AAAI Press, Madison, WI, 1998.

[9] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In Tenth European Conference on Machine Learning ECML-98, pages 137–142, 1999.

[10] J. T.-Y. Kwok. Automated text categorization using support vector machine. In Proceedings of the International Conference on Neural Information Processing (ICONIP), pages 347–351, Kitakyushu, Japan, 1999.

[11] S. Lawrence, C. L. Giles, and K. Bollacker. Digital libraries and autonomous citation indexing. IEEE Computer, 32(6):67–71, 1999.

[12] T. R. Leek. Information extraction using hidden Markov models. Master's thesis, UC San Diego, 1997.

[13] L. Liu, C. Pu, and W. Han. Xwrap: An xml-enabled wrapper construction system for web information sources. In Proc. International Conference on Data Engineering (ICDE), pages 611–621, 2000.

[14] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Building domain-specific search engines with machine learning techniques. In Proc. AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace, 1999., 1999.

[15] D. Miller, T. Leek, and R. Schwartz. BBN at TREC-7: Using hidden Markov models for information retrieval. In The Seventh Text Retrieval Conference, TREC-7. NIST Special Publications, 1999.

[16] I. Muslea, S. Minton, and C. Knoblock. Stalker: Learning extraction rules for semistructured, web-based information sources. In Proceedings of AAAI-98 Workshop on AI and Information Integration. AAAI Press, 1998.

[17] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Scholkopf, C. Burges, and A. Smola, editors, Advances in kernel methods - support vector learning. MIT Press, 1998.

[18] J. Platt. Using sparseness and analytic QP to speed training of support vector machines. In Advances in Neural Information Processing Systems, 1999.

[19] H. L. Van Trees. Detection Estimation and Modulation Theory, volume 1-3. Wiley and Sons, 1971.

[20] V. Vapnik. The Nature of Statistical Learning Theory. Springer Verlag, New York, 1995.