Extracting Article Text from the Web with Maximum Subsequence Segmentation

Jeff Pasternack
University of Illinois at Urbana-Champaign
201 N. Goodwin Avenue
Urbana, Illinois 61801
1 (217) 333-2584
jpaster2@uiuc.edu

Dan Roth
University of Illinois at Urbana-Champaign
201 N. Goodwin Avenue
Urbana, Illinois 61801
1 (217) 244-7068
danr@uiuc.edu

ABSTRACT

Much of the information on the Web is found in articles from online news outlets, magazines, encyclopedias, review collections, and other sources. However, extracting this content from the original HTML document is complicated by the large amount of less informative and typically unrelated material such as navigation menus, forms, user comments, and ads. Existing approaches tend to be either brittle and demand significant expert knowledge and time (manual or tool-assisted generation of rules or code), necessitate labeled examples for every different page structure to be processed (wrapper induction), require relatively uniform layout (template detection), or, as with Visual Page Segmentation (VIPS), are computationally expensive. We introduce maximum subsequence segmentation, a method of global optimization over token-level local classifiers, and apply it to the domain of news websites. Training examples are easy to obtain, both learning and prediction are linear time, and results are excellent (our semi-supervised algorithm yields an overall F_1-score of 97.947%), surpassing even those produced by VIPS with a hypothetical perfect block-selection heuristic. We also evaluate against the recent CleanEval shared task with surprisingly good cross-task performance cleaning general web pages, exceeding the top "text-only" score (based on Levenshtein distance), 87.8% versus 84.1%.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning – *concept learning;* H.3.m [**Information Storage and Retrieval**]: Miscellaneous

General Terms

Algorithms, Performance

Keywords

Text extraction, page segmentation, maximum subsequence

1. INTRODUCTION

Given the ubiquity of articles on the web, it is rather remarkable that there has been no more satisfactory means of isolating article text from the "junk" that appears alongside it on the page. A great deal of knowledge not otherwise available from more structured sources lies within these documents, and they are frequent targets of both information retrieval and direct human consumption. Consequently, article text extraction has a number of important applications, ranging from making web pages more readable on a small screen to removing non-informative noise as a pre-processing step towards further information extraction.

Extracting article text, as with cleaning HTML documents in general, is sometimes dismissed as easy or even trivial. McKeown et al. [17] spend a single sentence describing how they obtained the desired text from a page with a simple heuristic. Unfortunately, this sentiment is antiquated: over time, page layout has become much more complex as style has evolved from the unadorned, flat documents of the early web to basic frames and tables to today's deep, nested structures, with sidebars, inserts, headers, navigation panels, ads, etc., and the rudimentary techniques that worked years ago are inadequate today. Unsurprisingly, their heuristic, as our first baseline, proves to be quite inaccurate. As the first step in an information extraction pipeline, however, cleaning errors propagate through the system and harm overall performance. Accordingly, the most common approach has been to hand-code rules, often as regular expressions; these can be very accurate, but are highly labor intensive and easily broken by changes to a page's structure. This has led to interest to finding a better solution, as exemplified by the recent CleanEval shared task.

This paper first discusses related work and applications before giving a detailed description of the problem to be solved. An overview of the experimental setup and how labeled training examples were semi-automatically obtained is then given. It next presents a small family of methods for performing article text extraction by combining simple local, token-level classifiers with a global maximum subsequence optimization in order to predict the label (the extracted text) of a document. Both supervised and semi-supervised approaches are employed, all of which are linear time in both learning and prediction with only single pass over the training data. Since the required tokenization, feature extraction, local classification and global optimization can all be interleaved, the supervised algorithms also require only a single pass over the documents to be labeled. As experiments demonstrate, however, the more expensive semi-supervised algorithm yields significantly improved results. Performance is analyzed both in terms of precision and recall of the sequence of words extracted versus the correct extraction and the derived F_1-score. It is compared to two baselines, a "largest table cell or <DIV>" heuristic and a basic scoring rule combined with the maximum subsequence optimization, as well as the Visual Page Segmentation (VIPS) system [3, 4]. Because VIPS merely generates a tree in which each node corresponds to a block of content, an imaginary perfect heuristic is assumed that always chooses the block that maximizes

WWW 2009, April 20–24, 2009, Madrid, Spain.
ACM 978-1-60558-487-4/09/04.

the F_1-score. Nevertheless, even when VIPS is run over multiple granularities under this extraordinarily optimistic condition, maximum subsequence segmentation still shows better results. Finally, we apply our supervised article text extraction algorithm to the CleanEval shared task, obtaining better performance cleaning *general* web pages than any other system, and, of course, even higher scores over the dataset's articles alone.

2. BACKGROUND AND RELATED WORK

Extracting information from semi-structured HTML documents has been well-studied in the context of wrappers [12]. While article text is internally unstructured, the pages in which it appears are typically structured to some degree. This is due to both a customary presentation (putting the article text in the center of the page) and the fact that many sites apply one of a small number of templates to each article to give the pages a standardized appearance and functionality. There are, however, numerous exceptions; for example, article text may be placed at one side with equal visual emphasis as a large sidebar or a shorter article may have its text dwarfed by the large amount of surrounding unrelated content.

As noted in the introduction, explicit, hand-crafted web scrapers can often extract the article text embedded in a common template, looking for cues such as particular tags or words in the article, even comments (e.g. <!---article start --->), with rules written as regular expressions, in a traditional programming language such as Java or Perl, or with one of the many specialized languages and tools developed for this purpose, such as NoDoSE [1] and XWRAP [14]. Nevertheless, a different wrapper must be manually created for every data source as different websites will differ in how they arrange and structure information. Websites may also change the structure they employ over time, or employ multiple structures, rendering wrappers brittle and in need of continuous maintenance. Consequently, hand-crafted rules tend to be impractical for more than a handful of sources.

Other approaches attempt to induce a wrapper from a limited number of labeled examples. Kushmerick [11] learns wrappers this way, but is limited to a few predefined structures and cannot handle complex or unexpected information structure in the page. STALKER [18] is more powerful, taking a hierarchical description of the fields to be extracted (the "embedded catalog") along with user-provided field labeling on example documents in order to induce a set of extraction rules. Unfortunately, while these supervised learning systems spare the effort of an expert writing a wrapper manually, they still require up-to-date, tediously labeled examples for each source and remain difficult to scale.

Alternatively, RoadRunner [9] requires neither labeled examples nor programming, taking multiple pages with the same structure and then, based upon where these pages correspond and differ, inducing union-free regular expressions to extract the fields (which are assumed to be where pages disagree). There are significant problems with this approach, though. Errors can be made in detecting field boundaries, and automatically choosing pages with the same structure to learn from may be difficult or impossible, as websites may employ different structures for pages with the same type of content (e.g. a single site may use multiple templates to present article text). Additionally, as the resulting fields are "generic", some mechanism, either human input or a heuristic, must afterwards discern their meaning.

This caveat is shared with another unsupervised system, Visual Page Segmentation (VIPS) [3, 4]. VIPS heuristically segments pages into a tree where nodes are visually grouped blocks. To accomplish this, first the root of the DOM tree is analyzed, and if its "coherence" is low, its children are recursively processed in the same way, until all low-coherence DOM nodes are expanded. Coherence is a real number $\in [0, 1]$ heuristically determined from cues such as the presence of splitter elements (<HR>) or different background colors among the node's children. Expanded nodes are then examined for the vertical and horizontal visual separators (pixel-to-pixel lines between the blocks) that divide them; these are weighted based on features such as thickness (in pixels), co-occurrence with <HR> tags, and differences in font between the split blocks. Blocks divided by separators with weights below a threshold are then combined and a final tree is output. Results presented later demonstrate that if a perfect mechanism is provided to discern the best node in this tree and the right granularity is selected, VIPS can extract article text with a good degree of accuracy, but creating a mechanism to choose among the dozens or hundreds of nodes is a difficult task in itself. Additionally, VIPS must partially render a page to analyze it: while it does not need to actually draw DOM elements to the screen, it still must parse the page, build a DOM tree, and determine element layout (pixel coordinates and size). If external style sheets are used, these must also be retrieved. Consequently, compared to other techniques, VIPS is resource-intensive, which impedes its scalability to larger numbers of documents.

There are also a number of other methods that, like VIPS, attempt to identify generally interesting, informative portions of a document rather than specific fields. Chen, Ye & Li [8] split pages on certain tags into blocks and cluster them based upon style and position; similar clusters among different pages are then identified as part of the template. Yi & Liu [22] and Yi, Liu & Li [23] create a "compressed structure tree" and a "site style tree", respectively, that identify DOM nodes with similar style across pages as uninformative. Ma et al. [15] opt for a simpler approach, dividing a page according to <TABLE> tags and looking for frequently repeated segments. Lin & Ho [13] also partition a page with <TABLE> tags, but uses a more sophisticated method whereby redundant blocks are identified using an entropy measure over a set of word-based features. However, these template detection approaches tend have three major weaknesses: as with RoadRunner, multiple pages known to have the same template are required, and they often incorrectly assume that certain tags always delineate segments of interest or that uninteresting segments are largely repeated across pages (clearly not the case for "junk" like varying text ads or lists of related articles).

Conversely, in the CleanEval shared task, only one or a small handful of pages from any given site is available, requiring a more general approach. Marek, Pecina & Spousta's top-performing system in the task [16] splits pages on their tags into a sequence of blocks and then labels these as "content" or "noise" using conditional random fields with a number of block-level features. This differs substantially from our approach, which applies a global optimization over a simple token-level classifier to obtain a single contiguous passage of text.

3. Applications

There are a number of applications where article text extraction is either essential or can improve overall results. As devices such as PDAs and cell-phones have become commonplace, adapting web

typical news article pattern

pages for small screens has become increasing important. While general methods exist (e.g. [2, 7, 10]), in the context of article pages the article text is the primary (or sole) concern of the user, and, once extracted, this can be easily presented. The same applies to similar adaptation tasks, such as automatically generating RSS feeds from article pages. Eliminating the irrelevant context surrounding the article text is also useful in information retrieval: for example, in naïve keyword-based search an article page's non-article noise may be matched, yielding a false positive. Link analysis may similarly be misled by spurious links outside the article (e.g. in ads or navigation bars) that are unrelated to the main page content; indeed, [5] demonstrates that some gains are made using VIPS to analyze a page's various visual blocks separately. Query expansion benefits for the same reason, as unrelated terms in the page outside the article can be ignored—[24] shows, also using VIPS, that results are improved by segmenting a page and selecting expansion terms from only the "best" blocks. Finally, accurate article text extraction is in general an invaluable preprocessing step whenever the text is to be used as input to subsequent steps a pipeline. For instance, in a simple task where we wish to learn sets of related named entities by their co-occurrence in a document, unrelated (and perhaps mislabeled, as article text and ad text are typically of different domains) entities from outside the article text will confound the results.

4. PROBLEM DESCRIPTION

Given a page, we would like to (1) decide if the page contains an article, (2) if it does, identify a contiguous block of HTML in the page starting with the first word in the article and ending with the last, and finally (3) remove everything other than the article text (and its included markup tags) itself, such as ads, from the extracted block and output the result. As a practical matter, in the case where the first word or last word is nested within one or more pairs of tags (which would otherwise be left open, disrupting the article text's formatting), the relevant opening and ending tags are appended to the beginning and ending of the extracted block, respectively.

4.1 Identifying Articles

Possible features:
does page contain one headline, an author, byline, what is the longest contiguous block of HTML

The first step, determining whether a page contains an article, is a document classification problem. Our evaluation implicitly assumes that such a classifier is provided, since all our testing examples contain articles. No such assumption is made in training, however, and the semi-automatically generated training data may erroneously contain non-articles.

To be specific, by "article" we mean a contiguous, coherent work of prose on a single topic or multiple closely related topics that alone comprises the main informational content of the page—news stories, encyclopedia entries, or a single blog post are considered articles, whereas a collection of headlines with brief summaries, a list of search results, or a set of blog posts are not. For the news domain, a more specific definition is employed, as news websites have many pages that are not commonly considered news articles (such as recipes), but are articles in another domain (such as cookbooks). Thus, in addition to the general requirements for an article, a news article must be a story or report at least two paragraphs and eight total sentences in length. The length requirement serves to exclude those pages that are merely brief summaries (typically with a link to the full article).

Finally, a complication arises in determining exactly where an article begins and where it ends. A news article typically follows the pattern "headline → bylines → main text → bylines", where everything other than the main text is optional (bylines after the main text, for example, are less common than those before it, and some articles have neither a headline nor bylines). To resolve this, the true article text is considered to be the main text alone, without the headline or bylines, and this is what the wrappers that generate the training examples (as described in the next section) label as the extraction. However, to be fair in evaluation (especially with regards to VIPS), the evaluation examples are labeled with multiple extractions that correspond with multiple possible interpretations of what constitutes the bounds of an article's text. An extraction may start at (1) the first word of a headline, (2) the first word of an authorship byline before the main text (e.g. "By John Doe") or the organization byline (e.g. "The Associated Press") if no author is given, and (3) the first word of the main text. An extraction may end at (1) the last word of the main text, or (2) the last word of an authorship byline or organization byline appearing after the main text. Articles may thus have up to six possible extractions, while the norm is three. When experimental results are reported, predicted extractions are compared against both the main text alone (the shortest of the possible extractions) and against the "best" extraction, the one that yields the most favorable F_1-score for that particular prediction.

Defines an article

4.2 Cleaning Extracted Blocks

Maximum subsequence segmentation addresses the second part of the problem, identifying the block of HTML containing the article text, but further cleaning may be required to remove extraneous words from embedded ads, lists of links to other stories, images with captions, eye-catching reiterations of quotes appearing in the article, etc. A key observation is that, once the article's HTML block has been identified, removing whatever "junk" remains becomes much simpler. Indeed, after inspecting our evaluation sets we found this could be done using just a few rules with little error. Starting with an extracted block of HTML that starts at the first word and ends at the last word of the main text of the article:

1. Remove the contents of any <IFRAME> or <TABLE> tag pair
2. Remove the contents of any <DIV> tag pair that contains a hyperlink (<A>), <IFRAME>, <TABLE>, <IMG>, <EMBED>, <APPLET> or <OBJECT>

This heuristic works well because the text of most news articles very rarely contains links, and embeds are intended to either show you something (e.g. a picture) or get you to go somewhere (e.g. a link). The few mistakes we did observe were mostly trivial (such as leaving in the word "Advertisement"), though one article would lose most of its text because it contained links and was inside a <DIV>. Outside the news domain we will not always be so fortunate: encyclopedia articles, for example, tend to contain many links, and would require a more sophisticated approach. In practice, however, cleaning the extracted block is far less important than identifying it correctly, since web designers typically place the largest amount of junk text, such as navigation bars or user comments, *around* the article, not inside it, regardless of domain. This is demonstrated by our very strong cross-domain performance cleaning general articles from the CleanEval task despite making no effort to clean the blocks selected by MSS (see section 7.7).

what junk text surrounds an article: banners, navigation comment boxes

5. EXPERIMENTAL SETUP

5.1 Data

Because we are unaware of any suitable preexisting datasets for the article text extraction task, we produced our own, which may be obtained from the author's website[1]. As the news domain was to be used for experimentation, two thousand training examples were semi-automatically gathered from each of twelve news websites, and a total of 450 evaluation examples were gathered from forty-five news websites and tagged by hand. The twelve training sources were picked arbitrarily from among the best-known news sites: abcnews.go.com, cnn.com, foxnews.com, latimes.com, news.bbc.co.uk, news.com.com, news.yahoo.com, nytimes.com, reuters.com, usatoday.com, washingtonpost.com, and wired.com.

The maximum subsequence segmentation approaches investigated require labeled training data (pages where the beginning and the end of the article text are identified), but hand-tagging pages is a very tedious process. Instead, for each of the twelve training domains, a few pages were examined and a wrapper to extract the article text was manually written (in C#) for one or two of the templates used by the site, after which a few more pages were examined to check the wrapper's accuracy. The process took, on average, less than one hour per source. These wrappers were then used to label the pages we had spidered from the source, discarding any pages with labeled extractions that consisted of fewer than fifty words and symbols or were more than 20% tags to eliminate anything that was obviously not a news article. No attempt was made to manually check for or correct errors in the 24,000 examples produced this way.

Incorrect labeling would be unacceptable for evaluation data, so hand-tagging was used to ensure accuracy. The first five evaluation sources were then picked arbitrarily from well-known sites: chicagotribune.com, freep.com, nypost.com, suntimes.com, and techweb.com. This is our "Big 5" set—fifty examples were tagged for each. The "Myriad 40" set consists of the remaining forty sources chosen randomly from a list of several hundred (obtained from the Yahoo! Directory) and contains an international mix of English-language sites of widely varying size and sophistication; we tagged five examples for each. Splitting our evaluation data this way allows us to evaluate our semi-supervised algorithm (which requires a large number of pages for each source) with the Big 5 while the Myriad 40 lets us evaluate our other algorithms against a more diverse, representative sampling of the web.

cleaning the data beforehand keep certain tags but label others as unkown

5.2 Tokenization

When tokenizing pages (breaking up the HTML into a list of tags, words and symbols, where numbers are considered to be "words") a few transformations are applied. A list of "common" tags was created by examining the training data and selecting all tags that occurred in pages of more than one source; any tag not in this list is transformed into a generic <UNKNOWN>, which eliminates the non-standard tags (such as <NEWSELEMENT>) used by some sites. Porter stemming [19] is then applied to non-numeric words, and all numbers are stemmed to "1". Finally, everything between <SCRIPT> and <STYLE> tag pairs is discarded, since scripts and style sheets never contain article text.

[1] http://l2r.cs.uiuc.edu/~cogcomp/Data/MSS/

6. ALGORITHMS

6.1 Maximum Subsequence Optimization

Given a sequence S = (s_1, s_2,…,s_n), where $s_i \in \mathbf{R}$, a maximum subsequence of S is a sequence T = (s_a, s_{a+1},…,s_b) where $1 \le a \le b \le n$ and a and b are given by:

$$(a,b) = \arg\max_{(x,y)} \sum_{i=x}^{y} s_i$$

Note that T is technically constrained to be a substring (not just a subsequence) since we specify that all elements are consecutive, and it would be more accurately termed the "maximum substring", but we adopt this abuse of terminology to match the existing literature. A sequence may have more than one maximum subsequence, and the problem is trivial when all elements of the sequence are non-negative. We will refer to the sum of all the subsequence's elements as its *value*. Figure 1 gives a simple, single pass algorithm for finding the maximum subsequence in linear time. [21] provides a more sophisticated algorithm for finding *all* non-overlapping maximal subsequences in linear time, in which case the time complexity of retrieving an ordered list of the k highest-valued subsequences from a sequence of n elements would just be O(n * log(k)), the time required for sorting.

Another possible feature of the feature vector the longest valued subsequence

```
Given: S = (s1, s2,…,sn)
start = 0
sum = 0
maxSS = (-∞)

for i = 1 to n:
   sum = sum + si
   if sum > value(maxSS):
      maxSS = (sstart, sstart+1,…si)
   if sum < 0:
      start = i + 1
      sum = 0

Output: maxSS
```

Figure 1. Finding the Highest-Valued Subsequence

6.2 Applying Maximum Subsequence

Maximum subsequence segmentation uses local, token-level classifiers to find a score for each token in the document, resulting in a sequence of scores. The k highest-valued subsequences are taken as the k predictions, where the start and end positions of the subsequence in the sequence of scores corresponds to the first and last tokens in the predicted block. In the case of article text extraction, we are looking for a single block of text, so only a single maximum subsequence is sought and our local classifiers will predict either "in article text" or "out of article text" with a degree of certainty that will then be converted into a score.

Because we apply a global optimization over the scores generated by the local classifiers to make our final prediction, the local classifiers themselves need not be very accurate. The classifiers used in the experiments are all Naïve Bayes with only two types of features: trigrams and most-recent-unclosed-tags (a number of other simple features, such as bigrams, unigrams, and most-recent-two-unclosed-tags were tried, but these did not improve performance and were thus dropped). Given a list of n tokens U = (u_1, u_2,…, u_n), the trigram of token i is <u_i, u_{i+1}, u_{i+2}>. The most-recent-unclosed-tag is found by maintaining a stack of opening

tags as the document is examined. When a new opening tag is encountered, it is added to the stack. When a closing tag is found, the stack unwinds until a corresponding opening tag is matched and removed. The most-recent-unclosed-tag is the tag currently at the top of the stack, if any. This usually corresponds to the parent tag of the token in the DOM tree, but since no effort is made to verify that an opening tag is later closed, a "standalone" tag (such as an unclosed <P>) occurring before a token will be its most-recent-unclosed-tag rather than its actual DOM parent.

Could implement as a feature in feature vector (ie the tag that contained the most amount of words) using MMS algorithm

6.2.1 Baseline 1: Longest Cell or <DIV>

The first baseline is adapted from [17], which selects the table cell containing the most characters. We modified this to also consider the contents of <DIV> tags, since not all articles in the evaluation set were embedded in a table.

6.2.2 Baseline 2: Simple MSS

The second baseline used finds a maximum subsequence of a sequence of scores generated with an extremely simple scoring function that assigns a score of -3.25 to every tag and 1 to words and symbols. The score of -3.25 was determined by automatic optimization over the evaluation data to find the score that maximized the overall average F_1-score at 90.79% on the "short" main text extractions in order to give the most favorable performance possible for comparison, although performance was not especially sensitive to this choice: every tag score from -4.99 to -2.14 yields an F_1-score greater than 90%.

6.2.3 The Supervised Approach

The fully supervised setting is straightforward. Every token in the training examples is examined and its trigram and most-recent-unclosed-tag features are extracted, with an "in" or "out" label depending upon whether the particular token is within the article text or not. The features and labels from these tokens are then used to train a Naïve Bayes model as a local classifier (when classifying new tokens, the model will consider an unrecognized trigram t not appearing in the training corpus as equally likely for either label: P(t|in) = P(t|out) = 0.5). The certainty of the classifier's prediction, the probability p that a token is "in", is then transformed into a score (p - 0.5).

While performance is good overall, the supervised approach tends to fail when the local classifier mistakenly believes that a sizable set of tokens outside the article are "in", yielding a region of positive scores that results in both this region and the actual article text being predicted. This happens, for instance, with freep.com's articles because they contain lengthy comment sections that, to the classifier, look like article text. Because the classifier can only generate scores in the range [-0.5, 0.5], even if it can predict with high probability that the tokens between the two regions aren't part of an article this is readily outweighed by the sum of the false-positive scores. However, simply using a non-linear function to boost high-confidence scores, e.g. $(p\text{-}0.5)^3$, is problematic because of the classifier's relatively low accuracy. There are three possible remedies. First, we could improve accuracy by choosing a more expressive feature set or more sophisticated classifier with an accompanying increase in complexity. Second, we can add constraints that exploit our knowledge of the domain. We know, for instance, that articles typically do not contain <HR> tags in the article text but they may separate that text from other content such as comments. This idea can be easily implemented by altering the initially predicted block to remove everything after the first <HR> tag, if any; this shows a substantial overall gain in the experimental results. Third, we can find a way to apply relatively higher scores to the more "important" tokens near the edges of the article text and so prevent the maximum subsequence from exceeding these boundaries to erroneously include other nearby positive-valued subsequences. However, which tokens are near the borders varies from source-to-source; in some sources, for example, the article text is followed by a copyright statements; in others, the article frequently ends with a quote and thus the last few tokens are a quote symbol with an exclamation mark, period or question mark. Therefore, given a document from a previously unseen source, we cannot readily determine which tokens are important from the training examples directly.

6.2.4 The Semi-Supervised Approach

To this end we consider a semi-supervised bootstrapping algorithm, which proceeds as follows:

1. Train a Naïve Bayes local classifier on the provided labeled training examples with trigram and most-recent-unclosed-tag features, as in the supervised approach.
2. Predict extractions for the unlabeled documents U.
3. Iterate:
    a. Choose a portion of the documents in U with the seemingly most likely correct predicted extractions, and call these L.
    b. Find "importance weights" for the trigrams of the documents in L.
    c. Train a new Naïve Bayes local classifier over the documents in L with trigram features.
    d. Predict new extractions for the documents in U.

This can be run for a fixed number of iterations, or until some stopping condition; experimental results show convergence after ten iterations. We are then left with three questions: how to estimate the likelihood a prediction is correct, what number of the most likely correct labeled examples to put into L, and how to find and apply "importance weights" for the token trigrams.

To estimate if a prediction is correct, we have a number of alternatives. Remembering that documents are of varying lengths, we might try obtaining a [0, 1] score v by:

$$v = \sqrt[n]{\prod_{i=1}^{j-1}(1-p_i)\cdot\prod_{i=j}^{k}(p_i)\cdot\prod_{i=k+1}^{n}(1-p_i)}$$

where p_i is the probability given by the Naïve Bayes predictor that a token is "in", j is the index of the first token of the predicted article text, k is the index of the last token, and n is the total number of tokens in the document. One can equivalently consider log(v), which is the arithmetic mean of the logarithm of the probability Naïve Bayes gave for the token labels assigned after the maximum subsequence optimization. A simpler variant is:

$$v = \sqrt[n]{\prod_{i=j}^{k}(p_i)}$$

Here we use the probabilities assigned to the tokens of the predicted article text only. However, neither of these approaches works well in practice. This is partly due to the rather weak classifier used; if Naïve Bayes predicts with certainty (p = 1 or 0)

that a token is "in" and this is later corrected after finding the maximum subsequence, v will be 0. Although this may be alleviated somewhat by smoothing (to ensure that Naïve Bayes is never certain), a remaining problem is that tokens far from the boundaries of the article text are weighted equally with those closer, even though the accurate local classification of tokens near a boundary is much more important for finding a correct extraction. This suggests a windowed approach:

$$v = \sqrt[n]{\prod_{i=j-w}^{j-1}(1-p_i)\cdot\prod_{i=j}^{j+w-1}(p_i)\cdot\prod_{i=k-w+1}^{k}(p_i)\cdot\prod_{i=k+1}^{k+w}(1-p_i)}$$

where w is the window radius. A window radius of 10 was found to work well and is the parameter used in the experiments.

Once every prediction is scored, a number of the highest-scoring are selected. In the experiments, only a limited number of unlabeled examples are available (100 from each of the five evaluation sources) and there is a key tradeoff between the quality and quantity of examples, with 60% (60 from each source) demonstrating the best performance. It is worth noting that when the number of unlabeled examples is large, the algorithm can afford to be more selective.

Finally, we need a way to accomplish "importance weighting" so more important tokens near the borders can exert greater influence on the final maximum subsequence found. We do this by directly calculating a weight for each trigram feature based upon where tokens with that feature are relative to the edges of the article text. Let T_h be the set of all tokens with trigram h, and let j and k be the indices of the first and last token in the article text, respectively. Then the "importance weight" of trigram h, m_h, is given by:

$$m_h = \sum_{t\in T_h}\frac{d - Max\left(d, Min\left(\begin{matrix}|index(t)-(j-0.5)|,\\ |index(t)-(k+0.5)|\end{matrix}\right)\right)-0.5}{d}$$

where d is the distance cutoff. Importance weights vary from 1 (a trigram always appears immediately adjacent to the border between article text and non-article text) to 0 (a trigram is always found d or more tokens away from the border). If p_i is the probability of "in" produced by Naïve Bayes for a token i with trigram h, then the score can be calculated as $(p_i\text{-}0.5)(m_h \cdot c + 1)$, where $c \in \mathbf{R}$ is a $[0,\infty)$ constant. We hope that the increased scores of the "important" tokens will ensure that the maximum subsequence includes everything up to, but not past, the boundaries of the article text. Notice that importance weighting serves a function somewhat similar to an "open/close" classifier, complementing our existing "in/out" classification with better boundary detection. Tuning was used to select the constants d = 64 and c = 24.

6.2.5 *Efficiency*

Tokenization, feature extraction, local classification, and maximum subsequence optimization may be interleaved so that the supervised method is not only linear time but also single pass. The semi-supervised approach, while still linear time, requires several passes over the documents to be labeled. Both algorithms require only a single pass over the labeled training data, after which it can be discarded—this may be a significant practical advantage since, given the appropriate wrapper, training on a source can occur while spidering, without ever having to persist pages to storage.

6.2.6 *Discussion*

In the terminology of [20], the supervised approach presented is a Learning plus Inference (L+I) classifier, in which the local classifier (Naïve Bayes) has its output corrected by the inference step (maximum subsequence). The semi-supervised approach, conversely, is an Inference Based Training (IBT) classifier, because feedback from the inference step is used to iteratively correct and train the local classifier. Other, more direct IBT approaches are also possible. For instance, one can locally predict over a training example's tokens, find the maximum subsequence, and then correct the classifier only where the *global* classification is mistaken (such as with a weight update for Perceptron). However, in the case of Perceptron over trigrams and most-recently-unclosed-tags, we found this ineffective. Perceptron tends to find very large weights that, while they allow it to fit the training examples very well, hurt performance on new examples (e.g. by assigning a high score to a token outside the article text). Other, more sophisticated IBT approaches using the maximum subsequence optimization may still yield improvement, but we leave this as future work.

7. Results

As previously explained, the algorithms were trained on 24,000 labeled training examples taken from twelve sources and evaluated against 450 hand-labeled examples split into the Big 5 set (five sources with fifty examples each) and the Myriad 40 set (forty sources with five examples each). To evaluate the semi-supervised algorithm, we also employ an additional 50 unlabeled pages for each of the Big 5 sources so that the set U used for iterative learning contains a total of 500 examples, 100 from each source. During evaluation, the precision, recall, and F_1-score are calculated by comparing the prediction with both the article's main text alone (the *short extraction*) and by comparing against all the possible extraction labels and choosing the one with the highest F_1-score (the *best extraction*). If we take W_P as the set of words in the predicted extraction, and W_L as the set of words in the labeled extraction, then precision and recall are given by:

$$P = \frac{|W_P \cap W_L|}{|W_P|}, R = \frac{|W_P \cap W_L|}{|W_L|}$$

F_1-score is computed as normal, and Big 5, Myriad 40, and overall results are calculated by averaging each of the three metrics over all relevant examples. We also present results for each of the Big 5 sources individually. Note that every word in the document is considered distinct, even if two words share the same text. The exception to this is when we report the results from VIPS, as unfortunately the current implementation VIPS sometimes moves or removes text (even within a block) from its output, which makes it impossible to align with the original page and forces us to treat W_P and W_L as bags of words, where two words are considered the same if they share the same text. As this is more lenient, VIPS scores are likely to be slightly inflated.

7.1 Baseline 1: Longest Cell or <DIV>

Baseline 1 performed very poorly. Table cell and boundaries did not typically encapsulate the article text, and merely selecting the longest of these results in, as one might expect, low precision, but high recall. Only results against best

extractions are shown, in Table 1. Results on short extractions were similarly bad, with an overall F_1-score of 61.852%. Notice, however, that the F_1-score over the Myriad 40 set is 13.218% higher than that for the Big 5; this may be because Myriad 40 contains many smaller sources with outdated or simple layouts, while Big 5 sources are all mainstream and relatively modern.

Table 1. Baseline 1 Results Scored Against Best Extractions

Source	Precision	Recall	F_1-Score
chicagotribune.com	80.248%	99.996%	88.108%
freep.com	35.204%	100.000%	50.342%
nypost.com	45.790%	100.000%	61.637%
suntimes.com	36.022%	72.000%	47.591%
techweb.com	28.686%	89.886%	42.158%
Big 5	**45.190%**	**92.376%**	**57.967%**
Myriad 40	**60.411%**	**97.934%**	**71.185%**
Overall	**51.955%**	**94.847%**	**63.842%**

7.2 Baseline 2: Simple MSS

There is a high degree of variability in performance among the sources tested—an F_1-score of 99.5% for nypost.com versus 68.4% for suntimes.com, as shown below in Table 2. Interestingly, with the optimal tag penalty, simple MSS has performance on the Myriad 40 that is equivalent to the supervised algorithm, but note that, in practice, the optimal penalty will not be known, so supervised learning remains preferable. However, overall results are good over a wide range of tag penalties (F_1-score > 90% from -4.99 to -2.14), suggesting that this algorithm is a very reasonable method for extracting article text when labeled examples are not available. Results for short extractions were similar, with an overall F1-score of 90.907%.

just on this hurestic alone had a overall 90 percent article extraction rate

Table 2. Baseline 2 Results Scored Against Best Extractions

Source	Precision	Recall	F_1-Score
chicagotribune.com	98.974%	90.170%	93.301%
freep.com	81.081%	95.283%	84.181%
nypost.com	99.666%	99.434%	99.537%
suntimes.com	55.493%	95.661%	68.364%
techweb.com	95.323%	94.027%	94.391%
Big 5	**86.107%**	**94.915%**	**87.955%**
Myriad 40	**95.159%**	**97.011%**	**95.056%**
Overall	**90.130%**	**95.847%**	**91.111%**

7.3 Supervised Algorithm

Tables 3 and 4 present the results from the supervised algorithm, which runs maximum subsequence over the scores produced by a local Naïve Bayes classifier trained with trigram and most-recent-unclosed-tag (MRUT) features. Looking at individual sources, we find that freep.com suffers from low precision because of lengthy comment sections that the local classifier mistakes for article text. We also wished to determine how varying the number of sources would affect these results, and so trained on each of the 4095 possible non-empty subsets of the set of twelve training sources, evaluated against short extractions and averaged the F_1-score at each cardinality, as shown in Figure 2. We surpass simple MSS with four sources (8000 examples), and continue to improve with diminishing returns thereafter. With few sources performance suffers because so many trigrams seen during evaluation are unseen in the training data. Though in our experiments the number of examples per source is fixed, there is an interesting potential tradeoff between adding easily-obtained examples from an existing source and writing wrappers to use new sources.

Table 3. Supervised Results with Trigrams and MRUT Scored Against Short Extractions

Source	Precision	Recall	F_1-Score
chicagotribune.com	98.379%	97.084%	97.441%
freep.com	75.355%	99.979%	83.120%
nypost.com	98.533%	99.425%	98.940%
suntimes.com	99.136%	99.452%	99.278%
techweb.com	95.428%	99.672%	96.878%
Big 5	**93.366%**	**99.122%**	**95.131%**
Myriad 40	**92.597%**	**99.285%**	**94.637%**
Overall	**93.024%**	**99.194%**	**94.912%**

Table 4. Supervised Results with Trigrams and MRUT Scored Against Best Extractions

Source	Precision	Recall	F_1-Score
chicagotribune.com	98.615%	97.084%	97.565%
freep.com	78.268%	99.368%	84.788%
nypost.com	99.141%	99.429%	99.264%
suntimes.com	99.370%	99.446%	99.394%
techweb.com	96.776%	99.455%	97.517%
Big 5	**94.434%**	**98.956%**	**95.705%**
Myriad 40	**93.508%**	**99.107%**	**95.052%**
Overall	**94.023%**	**99.023%**	**95.415%**

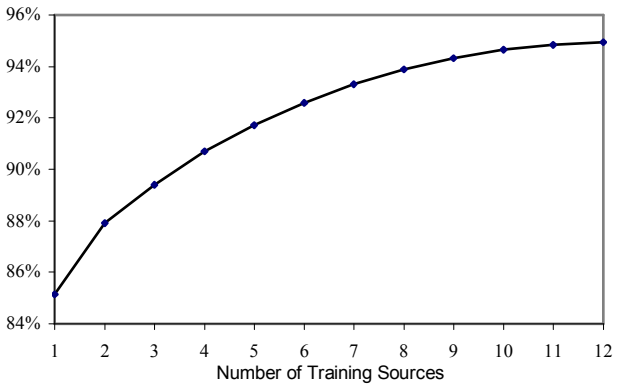

Figure 2. Overall Supervised F_1-Score Against Shortest Extractions for Increasing Number of Training Sources

7.4 Supervised + <HR> Constraint Algorithm

One way to counter the mistakes made by the vanilla supervised algorithm is to apply our knowledge of the domain and create heuristics that can provide another layer of global inference to correct mistakes. A simple implementation of this idea is to add a constraint that articles cannot contain an <HR> (horizontal rule) tag, truncating any extraction when one is found. As shown in Tables 5 and 6, though performance on chicagotribune.com and techweb.com declines, the overall F_1-score improves by approximately 1.2%, demonstrating that constraints can be a straightforward and efficient way to improve performance by leveraging existing domain knowledge.

Table 5. Supervised + <HR> Constraint Results with Trigrams and MRUT Scored Against Short Extractions

Source	Precision	Recall	F_1-Score
chicagotribune.com	98.385%	94.147%	95.125%
freep.com	89.826%	99.979%	93.467%
nypost.com	98.533%	99.425%	98.940%
suntimes.com	99.136%	99.452%	99.278%
techweb.com	95.428%	99.672%	96.878%
Big 5	**96.262%**	**98.535%**	**96.737%**
Myriad 40	**93.550%**	**99.285%**	**95.433%**
Overall	**95.057%**	**98.868%**	**96.158%**

Table 6. Supervised + <HR> Constraint Results with Trigrams and MRUT Scored Against Best Extractions

Source	Precision	Recall	F_1-Score
chicagotribune.com	98.621%	94.147%	95.248%
freep.com	93.332%	99.368%	95.217%
nypost.com	99.141%	99.429%	99.264%
suntimes.com	99.370%	99.446%	99.394%
techweb.com	96.776%	99.455%	97.517%
Big 5	**97.448%**	**98.369%**	**97.328%**
Myriad 40	**94.494%**	**99.107%**	**95.860%**
Overall	**96.135%**	**98.697%**	**96.675%**

7.5 Semi-Supervised Algorithm

The semi-supervised algorithm is more computationally expensive than the supervised methods but, as Tables 7 and 8 show, yields notably improved performance. Because the algorithm requires a large number of examples from each source, only the Big 5 sources are tested. Both sets of results are taken after the tenth iteration of the algorithm; later iterations alternated between overall F_1-scores of 97.615% and 97.581% (short) and 97.947% and 97.915% (best). High precision relative to the other algorithms is exhibited, and results are also more uniformly good over all sources. The parameters used are a window size (w) of 10, a distance cutoff (d) of 64, and a distance scoring constant (c) of 24, with 60% of the seemingly best-labeled examples from each source chosen for L.

Table 7. Semi-Supervised Results after 10 Iterations Scored Against Short Extractions

Source	Precision	Recall	F_1-Score
chicagotribune.com	98.999%	95.051%	96.628%
freep.com	97.094%	99.321%	98.177%
nypost.com	98.658%	99.452%	99.020%
suntimes.com	99.517%	97.482%	98.424%
techweb.com	98.541%	94.584%	95.824%
Big 5	**98.562%**	**97.178%**	**97.615%**

Table 8. Semi-Supervised Results after 10 Iterations Scored Against Best Extractions

Source	Precision	Recall	F_1-Score
chicagotribune.com	99.016%	95.034%	96.628%
freep.com	98.867%	99.231%	99.036%
nypost.com	99.163%	99.342%	99.230%
suntimes.com	99.531%	97.483%	98.431%
techweb.com	99.840%	94.454%	96.411%
Big 5	**99.283%**	**97.109%**	**97.947%**

7.6 VIPS

Visual Page Segmentation takes as input a page and produces a tree whose leaves are blocks of HTML, and whose higher nodes thus correspond to sets of their descendants' blocks. In testing, a hypothetical perfect heuristic is assumed, one that always picks the node yielding the highest F_1-score; this is done to give VIPS the most favorable results possible. In contrast, a simple heuristic that evaluated blocks by giving one point for every word and subtracting two points for every tag yielded a top F_1-score of only 89.705%.

VIPS can be run at different "granularities" that determine the thresholds for splitting blocks. Increasing granularities, however, does not simply expand some of the leaves found in a lower-granularity tree, but can alter its entire structure, and we found that the best results were obtained at granularity 8, as shown in Tables 9 and 10.

Table 9. VIPS at Granularity 8 Scored Against Short Extractions

Source	Precision	Recall	F_1-Score
chicagotribune.com	98.702%	94.824%	96.147%
freep.com	97.603%	99.510%	98.472%
nypost.com	98.448%	90.179%	93.776%
suntimes.com	97.806%	99.971%	98.867%
techweb.com	84.985%	99.711%	91.020%
Big 5	**95.509%**	**96.839%**	**95.657%**
Myriad 40	**90.163%**	**96.913%**	**92.974%**
Overall	**90.831%**	**98.439%**	**94.092%**

Table 10. VIPS at Granularity 8 Scored Against Best Extractions

Source	Precision	Recall	F_1-Score
chicagotribune.com	98.587%	95.006%	96.264%
freep.com	98.364%	99.517%	98.873%
nypost.com	99.134%	90.487%	94.307%
suntimes.com	99.480%	99.974%	99.722%
techweb.com	86.557%	99.711%	92.001%
Big 5	**96.424%**	**96.939%**	**96.233%**
Myriad 40	**93.001%**	**96.791%**	**94.516%**
Overall	**93.831%**	**98.428%**	**95.758%**

We see that VIPS' performance on the Big 5 is good, but still more than 2% lower than the semi-supervised algorithm. Further, VIPS does relatively poorly on the Myriad 40, lower than the supervised algorithm. It should also be noted that, at granularity 8, a real heuristic for VIPS would be quite challenged with an average of 104 nodes in each tree.

7.7 CleanEval

We now consider the CleanEval shared task for cleaning arbitrary web pages. There are two datasets: a 58 document development set and a 674 document evaluation set. For each document, both the original HTML and the annotator-cleaned text are provided. The task considered both a "text-only" score and a "text-and-markup" score. The text-only score is based on Levenshtein distance from a cleaning algorithm's output to the annotator cleaned text (the number of insertions and removals of words necessary to align the annotator's text with the algorithm's output; substitutions are not allowed). The full formula is:

$$\text{textOnlyScore}(a,b) = 1 - \frac{\text{distance}(a,b)}{\text{alignmentLength}(a,b)}$$

where alignmentLength(a,b) is the number of operations (insert, remove, or align) required to align to sequences of words; for instance, the text-only score for "the dog barked" and "the dog" would be 1 – (1/3), since it takes three operations (two aligns and one insert) to align them. This metric is also relatively expensive to compute, taking O(|a| * |b|) time, significant when |a| and |b| can each be well over 10,000.

The "text-and-markup" score, in addition to considering Levenshtein distance, also compares "segment validity", how well markup added by the algorithm identifies lists, paragraphs, and headers (contestants fared relatively poorly in this); as our algorithm does not add such markup, we do not consider it further.

In testing against CleanEval, we used the same training data, but switched our features from trigrams to unigrams; we found that as our training examples come from news sources, too many trigrams in the CleanEval development set were previously unseen, hurting performance. We also applied smoothing by taking both of a token's neighbors as features when that token was not seen in the training data. Finally, to boost recall, we biased all scores from our Naïve Bayes classifiers by adding 0.14, a value obtained by tuning over the development set. Testing on the evaluation set yielded a text-only score of **87.832%** (with a 95% confidence interval from 86.388% to 89.286%), a precision of **90.268%**, a recall of **96.516%**, and an F_1-score of **91.983%**. Because the alignment between the annotator's text and the original HTML are not provided, we measured precision and recall using bags of words (as we did with VIPS), and these scores are thus also likely to be slightly inflated.

We also categorized all the documents in the evaluation set as either "article" or "non-article", identifying 411 articles (61% of the set) and tested on these alone, for a text-only score of **92.582%**, a precision of **93.825%**, a recall of **98.298%**, and an F_1-score of **95.717%**.

Interestingly, our text-only score is higher (with statistical significance) than the top CleanEval contestant's 84.1% [16]. We believe this is for two reasons: first, our cross-domain performance on articles of all types (very few were news) was very good, and second, because many non-articles such as sets of blog posts have little "junk" between the start and end of the content, by biasing the scores from our classifier to favor larger extractions we boosted recall sufficiently to outweigh the loss in precision.

8. Conclusion

The effectiveness of efficient algorithms utilizing maximum subsequence segmentation to extract the text of articles from HTML documents has been demonstrated, with the best of these, the semi-supervised method, well exceeding the performance of the much more computationally expensive VIPS. While the semi-supervised results are significantly better than the supervised, constraints may be able to bridge the gap somewhat, and the supervised methods are faster (by a constant factor) as well as single-pass. The semi-supervised algorithm also requires several tunable parameters which must be retuned in a new domain.

Though we focused on news articles, we were also able to achieve very strong cross-domain performance on general articles from CleanEval while still using our original news training data. Even running cross-task to clean arbitrary, non-article web pages, maximum subsequence segmentation still managed to surpass the best existing system.

While hand-written web scrapers are likely to persist for some time, MSS provides a fast, accurate method for extracting article text from a large number of sources in a variety of domains with relatively little effort, computational or otherwise; even our simple MSS baseline was able to perform quite respectably with a single parameter and no training data whatsoever.

9. REFERENCES

[1] Adelberg, B. NoDoSE--a tool for semi-automatically extracting structured and semistructured data from text documents. SIGMOD 1998.

[2] Buyukkokten, O., Garcia-Molina, H. and Paepcke, A. Accordion summarization for end-game browsing on PDAs and cellular phones. SIGCHI 2001.

[3] Cai, D., Yu, S., Wen, J.R. and Ma, W.Y. Extracting content structure for web pages based on visual representation. APWeb 2003.

[4] Cai, D., Yu, S., Wen, J.R. and Ma, W.Y. VIPS: a Vision-based Page Segmentation Algorithm. Microsoft Technical Report (MSR-TR-2003-79),2003

[5] Cai, D., He, X., Wen, J.R. and Ma, W.Y. "Block-level Link Analysis", SIGIR 2004.

[6] Cai, D., Yu, S., Wen, J.R. and Ma, W.Y. "Block-based Web Search", SIGIR 2004.

[7] Chen, J., Zhou, B., Shi, J., Zhang, H. and Fengwu, Q. Function-based object model towards website adaptation. WWW2001.

[8] Chen, L., Ye, S. and Li, X. Template detection for large scale search engines. SAC 2006.

[9] Crescenzi, V., Mecca, G. and Merialdo, P. Roadrunner: Towards automatic data extraction from large web sites. VLDB 2001.

[10] Fox, A., Goldberg, I., Gribble, S.D., Lee, D.C., Polito, A. and Brewer, E.A. Experience With Top Gun Wingman: A Proxy-Based Graphical Web Browser for the 3Com PalmPilot. Middleware 1998.

[11] Kushmerick, N. Wrapper induction: Efficiency and expressiveness. Artificial Intelligence, 118, 2000.

[12] Laender, A., Ribeiro-Neto, B., Silva, A. and Teixeira, J. A Brief Survey of Web Data Extraction Tools, SIGMOD Record, Volume 31, Number 2, 2002.

[13] Lin, S.H. and Ho, J.M. Discovering informative content blocks from Web documents. KDD 2002.

[14] Liu, L., Pu, C. and Han, W. XWRAP: an XML-enabled wrapper construction system for Web information sources. ICDE 2000.

[15] Ma, L., Goharian, N., Chowdhury, A. and Chung, M. Extracting unstructured data from template generated web documents. CIKM 2003.

[16] Marek, M., Pecina, P., Spousta, M. Web Page cleaning with Conditional Random Fields. WAC3 2007.

[17] McKeown, K.R., Barzilay, R., Evans, D., Hatzivassiloglou, V., Klavans, J.L., Nenkova, A., Sable, C., Schiffman, B. and Sigelman, S. Tracking and summarizing news on a daily basis with Columbia's Newsblaster. HLT 2002.

[18] Muslea, I., Minton, S. and Knoblock, C.A. Hierarchical Wrapper Induction for Semistructured Information Sources. Autonomous Agents and Multi-Agent Systems, 2001.

[19] Porter, M.F. An algorithm for suffix stripping. Program, vol. 14, no. 3, pp. 130–137, 1980.

[20] Punyakanok, V., Roth, D., Yih, W. and Zimak, D. Learning and inference over constrained output. IJCAI 2005.

[21] Ruzzo, W.L. and Tompa, M. A linear time algorithm for finding all maximal scoring subsequences. ISMB 1999.

[22] Yi, L. and Liu, B. Web Page Cleaning for Web Mining through Feature Weighting. IJCAI-03.

[23] Yi, L., Liu, B. and Li, X. Eliminating noisy information in Web pages for data mining. KDD 2003.

[24] Yu, S., Cai, D., Wen, J.R. and Ma, W.Y. "Improving Pseudo-Relevance Feedback in Web Information Retrieval Using Web Page Segmentation", WWW2003