Finding Advertising Keywords on Web Pages

Wen-tau Yih
Microsoft Research
1 Microsoft Way
Redmond, WA 98052
scottyih@microsoft.com

Joshua Goodman
Microsoft Research
1 Microsoft Way
Redmond, WA 98052
joshuago@microsoft.com

Vitor R. Carvalho
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
vitor@cs.cmu.edu

ABSTRACT

A large and growing number of web pages display contextual advertising based on keywords automatically extracted from the text of the page, and this is a substantial source of revenue supporting the web today. Despite the importance of this area, little formal, published research exists. We describe a system that learns how to extract keywords from web pages for advertisement targeting. The system uses a number of features, such as term frequency of each potential keyword, inverse document frequency, presence in meta-data, and how often the term occurs in search query logs. The system is trained with a set of example pages that have been hand-labeled with "relevant" keywords. Based on this training, it can then extract new keywords from previously unseen pages. Accuracy is substantially better than several baseline systems.

Categories and Subject Descriptors

H.3.1 [**Content Analysis and Indexing**]: Abstracting methods; H.4.m [**Information Systems**]: Miscellaneous

General Terms

Algorithms, experimentation

Keywords

keyword extraction, information extraction, advertising

1. INTRODUCTION

Content-targeted advertising systems, such as Google's AdSense program, and Yahoo's Contextual Match product, are becoming an increasingly important part of the funding for free web services. These programs automatically find relevant keywords on a web page, and then display advertisements based on those keywords. In this paper, we systematically analyze techniques for determining which keywords are relevant. We demonstrate that a learning-based technique using TF×IDF features, web page meta data, and, most surprisingly, information from query log files, substantially outperforms competing methods, and even approaches human levels of performance by at least one measure.

Typical content-targeted advertising systems analyze a web page, such as a blog, a news page, or another source of information, to find prominent keywords on that page. These keywords are then sent to an advertising system, which matches the keywords against a database of ads. Advertising appropriate to the keyword is displayed to the user. Typically, if a user clicks on the ad, the advertiser is charged a fee, most of which is given to the web page owner, with a portion kept by the advertising service.

Picking appropriate keywords helps users in at least two ways. First, choosing appropriate keywords can lead to users seeing ads for products or services they would be interested in purchasing. Second, the better targeted the advertising, the more revenue that is earned by the web page provider, and thus the more interesting the applications that can be supported. For instance, free blogging services and free email accounts with large amounts of storage are both enabled by good advertising systems

From the perspective of the advertiser, it is even more important to pick good keywords. For most areas of research, such as speech recognition, a 10% improvement leads to better products, but the increase in revenue is usually much smaller. For keyword selection, however, a 10% improvement might actually lead to nearly a 10% higher click-through-rate, directly increasing potential revenue and profit.

In this paper, we systematically investigated several different aspects of keyword extraction. First, we compared looking at each occurrence of a word or phrase in a document separately, versus combining all of our information about the word or phrase. We also compared approaches that look at the word or phrase monolithically to approaches that decompose a phrase into separate words. Second, we examined a wide variety of information sources, analyzing which sources were most helpful. These included various meta-tags, title information, and even the words in the URL of the page. One surprisingly useful source of information was query frequency information from MSN Search query logs. That is, knowing the overall query frequency of a particular word or phrase on a page was helpful in determining if that word or phrase was relevant to that page.

We compared these approaches to several different baseline approaches, including a traditional TF×IDF model; a model using TF and IDF features but with learned weights; and the KEA system [7]. KEA is also a machine learning system, but with a simpler learning mechanism and fewer features. As we will show, our system is substantially better than any of these baseline systems. We also compared our system to the maximum achievable given the human labeling, and found that on one measure, our system was in the same range as human levels of performance.

WWW 2006, May 23–26, 2006, Edinburgh, Scotland.
ACM 1-59593-323-9/06/0005.

2. SYSTEM ARCHITECTURE

In this section, we introduce the general architecture of our keyword extraction system, which consists of the following four stages: *preprocessor*, *candidate selector*, *classifier*, and *postprocessor*.

2.1 Preprocessor

The main purpose of the preprocessor is to transform an HTML document into an easy-to-process plain-text based document, while still maintaining important information. In particular, we want to preserve the *blocks* in the original HTML document, but remove the HTML tags. For example, text in the same table should be placed together without tags like `<table>`, `<tr>`, or `<td>`. We also preserve information about which phrases are part of the anchor text of the hypertext links. The *meta* section of an HTML document header is also an important source of useful information, even though most of the fields except the *title* are not displayed by web browsers.

The preprocessor first parses an HTML document, and returns blocks of text in the body, hypertext information, and meta information in the header[1]. Because a keyword should not cross sentence boundaries, we apply a sentence splitter to separate text in the same block into various sentences.

To evaluate whether linguistic information can help keyword extraction, we also apply a state-of-the-art part-of-speech (POS) tagger [6], and record the pos tag of each word. In addition, we have observed that most words or phrases that are relevant are short noun phrases. Therefore, having this information available as a feature would potentially be useful. We thus applied a state-of-the-art chunker to detect the base noun phrases in each document [15][2].

2.2 Candidate Selector

Our system considers each word or phrase (consecutive words) up to length 5 that appears in the document as a candidate keyword. This includes all keywords that appear in the title section, or in meta-tags, as well as words and phrases in the body. As mentioned previously, a phrase is not selected as a candidate if it crosses sentence or block boundaries. This strategy not only eliminates many trivial errors but also speeds up the processing time by considering fewer keyword candidates.

Each phrase can be considered separately, or can be combined with all other occurrences of the same phrase in the same document. In addition, phrases can be considered monolithically, or can be decomposed into their constituent words. Putting together these possibilities, we ended up considering three different candidate selectors.

2.2.1 Monolithic, Separate (MoS)

In the Monolithic Separate candidate selector, fragments that appear in different document locations are considered as different candidates even if their *content* is identical. That is, if the phrase "digital camera" occurred once in the beginning of the document, and once in the end, we would consider them as two separate candidates, with potentially different features. While some features, such as the phrase length and TF values, would all be the same for all of these candidates, others, such as whether the phrase was capitalized, would be different. We call this variation *Separate*.

In this candidate selector, all features of a candidate phrase are based on the phrase as a whole. For example, term frequency counts the number of times the exact phrase occurs in the document, rather than using the the term frequency of individual words. We refer to this as *Monolithic*. To simplify our description, we use **MoS** to refer to this design.

2.2.2 Monolithic, Combined (MoC)

Since we only care about the ranked list of keywords, not *where* the keywords are extracted, we can reduce the number of candidates by combining *identical* (case ignored) fragments. For instance, "Weather report" in the title and "weather report" in the body are treated as only one candidate. We use **MoC** to refer to this design.

Note that even in the Combined case, word order matters, e.g. the phrase "weather report" is treated as different than the phrase "report weather."

2.2.3 Decomposed, Separate (DeS)

Keyword extraction seems closely related to well-studied areas like information extraction, named-entity recognition and phrase labeling: they all attempt to find important phrases in documents. State-of-the-art information extraction systems (e.g., [5, 20]), named-entity recognition systems (e.g., [21]), and phrase labeling systems (e.g., [3]) typically decompose phrases into individual words, rather than examining them monolithically. It thus seemed worthwhile to examine similar techniques for keyword extraction. Decomposing a phrase into its individual words might have certain advantages. For instance, if the phrase "pet store" occurred only once in a document, but the phrases "pet" and "store" each occurred many times separately, such a decomposed approach would make it easy to use this knowledge.

Instead of selecting phrases directly as candidates, the *decomposed* approach tries to assign a label to each word in a document, as is done in related fields. That is, each of the words in a document is selected as a candidate, with multiple possible labels. The labels can be **B** (*beginning* of a keyphrase, when the following word is also part of the keyphrase), **I** (*inside* a keyphrase, but not the first or last word), **L** (*last* word of a keyphrase), **U** (*unique* word of a keyword of length 1), and finally **O** (*outside* any keyword or keyphrase).

This word-based framework requires a multi-class classifier to assign these 5 labels to a candidate word. In addition, it also needs a somewhat more sophisticated inference procedure to construct a ranked list of keywords in the postprocessing stage. The details will be described in Section 2.4. We use **DeS** to refer to this design.

2.3 Classifier

The core of our keyword extraction system is a classifier trained using machine learning. When a monolithic framework (MoS or MoC) is used, we train a binary classifier. Given a phrase candidate, the classifier predicts whether the word or phrase is a keyword or not. When a decomposed framework (DeS) is used, we train a multi-class classifier. Given a word, the goal is to predict the label as **B**, **I**, **L**, **U**, or **O**. Since whether a phrase is a keyword is ambiguous by nature, instead of a hard prediction, we actually need

[1]We used "Beautiful Soup", which is a python library for HTML parsing. Beautiful Soup can be downloaded from http://www.crummy.com/software/BeautifulSoup/

[2]These natural language processing tools can be downloaded from http://l2r.cs.uiuc.edu/∼cogcomp

the classifier to predict *how likely* it is that candidate has a particular label. In other words, the classifier needs to output some kind of confidence scores or probabilities. The scores or probabilities can then be used later to generate a ranked list of keywords, given a document. We introduce the learning algorithm and the features we used as follows.

2.3.1 Learning Algorithm

We used logistic regression as the learning algorithm in all experiments reported here. Logistic regression models are also called maximum entropy models, and are equivalent to single layer neural networks that are trained to minimize entropy. In related experiments on an email corpus, we tried other learning algorithms, such as linear support-vector machines, decision trees, and naive Bayes, but in every case we found that logistic regression was equally good or better than the other learning algorithms for this type of task.

Formally, we want to predict an output variable Y, given a set of input features, $\overline{X}$. In the monolithic framework, Y would be 1 if a candidate phrase is a relevant keyword, and 0 otherwise. $\overline{X}$ would be a vector of feature values associated with a particular candidate. For example, the vector might include the distance from the start of the document, the term frequency (*TF*) and the document frequency (*DF*) values for the phrase. The model returns the estimated probability, $P(Y = 1|\overline{X} = \overline{x})$. The logistic regression model learns a vector of weights, $\overline{w}$, one for each input feature in $\overline{X}$. Continuing our example, it might learn a weight for the *TF* value, a weight for the *DF* value, and a weight for the distance value. The actual probability returned is

$$P(Y = 1|X = \overline{x}) = \frac{\exp(\overline{x} \cdot \overline{w})}{1 + \exp(\overline{x} \cdot \overline{w})}$$

An interesting property of this model is its ability to simulate some simpler models. For instance, if we take the logarithms of the *TF* and *DF* values as features, then if the corresponding weights are +1 and -1, we end up with

$$\log(TF) - \log(DF) = \log(TF/DF)$$

In other words, by including *TF* and *DF* as features, we can simulate the TF×IDF model, but the logistic regression model also has the option to learn different weightings.

To actually train a logistic regression model, we take a set of training data, and try to find the weight vector $\overline{w}$ that makes it as likely as possible. In our case, the training instances consist of every possible candidate phrases selected from the training documents, with $Y = 1$ if they were labeled relevant, and 0 otherwise. We use the SCGIS method [9] as the actual training method. However, because there is a unique best logistic regression model for any training set, and the space is convex, the actual choice of training algorithm makes relatively little difference.

In the monolithic models, we only try to model one decision – whether a word or phrase is relevant or not, so the variable Y can only take values 0 or 1. But for the decomposed framework, we try to determine for each word whether it is the beginning of a phrase, inside a phrase, etc. with 5 different possibilities (the BILUO labels). In this case, we use a generalized form of the logistic regression model:

$$P(Y = i|\overline{x}) = \frac{\exp(\overline{x} \cdot \overline{w_i})}{\sum_{j=1}^{5} \exp(\overline{x} \cdot \overline{w_j})}$$

That is, there are 5 different sets of weights, one for each possible output value.

Note that in practice, for both forms of logistic regression, we always append a special "always on" feature (i.e. a value of 1) to the $\overline{x}$ vector, that serves as a bias term. In order to prevent overfitting, we also apply a Gaussian prior with variance 0.3 for smoothing [4].

2.3.2 Features

We experimented with various features that are potentially useful. Some of these features are *binary*, taking only the values 0 or 1, such as whether the phrase appears in the title. Others are real-valued, such as the *TF* or *DF* values or their logarithms. Below we describe these features and their variations when used in the monolithic (MoS) and decomposed (DeS) frameworks.

Features used in the monolithic, combined (MoC) framework are basically the same as in the MoS framework. If in the document, a candidate phrase in the MoC framework has several occurrences, which correspond to several candidate phrases in the MoS framework, the features are combined using the following rules.

1. For binary features, the combined feature is the union of the corresponding features. That is, if this feature is active in any of the occurrences, then it is also active in the combined candidate.

2. For real-valued features, the combined feature takes the smallest value of the corresponding features.

To give an example for the binary case, if one occurrence of the term "digital camera" is in the anchor text of a hyperlink, then the anchor text feature is active in the combined candidate. Similarly, for the location feature, which is a real-valued case, the location of the first occurrence of this phrase will be used as the corresponding combined value, since its value is the smallest. In the following description, features are binary unless otherwise noted.

2.3.2.1 **Lin**: *linguistic features.*

The linguistic information used in feature extraction includes: two types of pos tags – *noun* (NN & NNS) and *proper noun* (NNP & NNPS), and one type of chunk – *noun phrase* (NP). The variations used in MoS are: whether the phrase contain these pos tags; whether all the words in that phrase share the same pos tags (either proper noun or noun); and whether the whole candidate phrase is a noun phrase. For DeS, they are: whether the word has the pos tag; whether the word is the beginning of a noun phrase; whether the word is in a noun phrase, but not the first word; and whether the word is outside any noun phrase.

2.3.2.2 **C**: *capitalization.*

Whether a word is capitalized is an indication of being part of a proper noun, or an important word. This set of features for MoS is defined as: whether all the words in the candidate phrase are capitalized; whether the first word of the candidate phrase is capitalized; and whether the candidate phrase has a capitalized word. For DeS, it is simply whether the word is capitalized.

2.3.2.3 **H**: *hypertext.*

Whether a candidate phrase or word is part of the anchor text for a hypertext link is extracted as the following features. For MoS, they are: whether the whole candidate phrase matches exactly the anchortext of a link; whether all the words of the candidate phrase are in the same anchor text; and whether any word of the candidate phrase belongs to the anchor text of a link. For DeS, they are: whether the word is the beginning of the anchor text; whether the word is in the anchor text of a link, but not the first word; and whether the word is outside any anchor text.

2.3.2.4 **Ms**: *meta section features.*

The header of an HTML document may provide additional information embedded in `meta` tags. Although the text in this region is usually not seen by readers, whether a candidate appears in this meta section seems important. For MoS, the features are whether the whole candidate phrase is in the meta section. For DeS, they are: whether the word is the first word in a meta tag; and whether the word occurs somewhere in a meta tag, but not as the first word.

2.3.2.5 **T**: *title.*

The only human readable text in the HTML header is the `TITLE`, which is usually put in the window caption by the browser. For MoS, the feature is whether the whole candidate phrase is in the title. For DeS, the features are: whether the word is the beginning of the title; and whether the word is in the title, but not the first word.

2.3.2.6 **M**: *meta features.*

In addition to `TITLE`, several meta tags are potentially related to keywords, and are used to derive features. In the MoS framework, the features are: whether the whole candidate phrase is in the meta-description; whether the whole candidate phrase is in the meta-keywords; and whether the whole candidate phrase is in the meta-title. For DeS, the features are: whether the word is the beginning of the meta-description; whether the word is in the meta-description, but not the first word; whether the word is the beginning of the meta-keywords; whether the word is in the meta-keywords, but not the first word; whether the word is the beginning of the meta-title; and whether the word is in the meta-title, but not the first word.

2.3.2.7 **U**: *URL.*

A web document has one additional highly useful property – the *name* of the document, which is its `URL`. For MoS, the features are: whether the whole candidate phrase is in part of the URL string; and whether any word of the candidate phrase is in the URL string. In the DeS framework, the feature is whether the word is in the URL string.

2.3.2.8 **IR**: *information retrieval oriented features.*

We consider the TF (term frequency) and DF (document frequency) values of the candidate as real-valued features. The document frequency is derived by counting how many documents in our web page collection that contain the given term. In addition to the original TF and DF frequency numbers, $\log(TF + 1)$ and $\log(DF + 1)$ are also used as features. The features used in the monolithic and the decomposed frameworks are basically the same, where for DeS, the "term" is the candidate word.

2.3.2.9 **Loc**: *relative location of the candidate.*

The beginning of a document often contains an introduction or summary with important words and phrases. Therefore, the location of the occurrence of the word or phrase in the document is also extracted as a feature. Since the length of a document or a sentence varies considerably, we take only the *ratio* of the location instead of the absolute number. For example, if a word appears in the 10th position, while the whole document contains 200 words, the ratio is then 0.05. These features used for the monolithic and decomposed frameworks are the same. When the candidate is a phrase, its first word is used as its location.

There are three different relative locations used as features: *wordRatio* – the relative location of the candidate in the sentence; *sentRatio* – the location of the sentence where the candidate is in divided by the total number of sentences in the document; *wordDocRatio* – the relative location of the candidate in the document. In addition to these 3 real-valued features, we also use their logarithms as features. Specifically, we used $\log(1 + \mathit{wordRatio})$, $\log(1 + \mathit{sentRatio})$, and $\log(1 + \mathit{wordDocRatio})$.

2.3.2.10 **Len**: *sentence and document length.*

The length (in words) of the sentence (sentLen) where the candidate occurs, and the length of the whole document (docLen) (words in the header are not included) are used as features. Similarly, $\log(1 + \mathit{sentLen})$ and $\log(1 + \mathit{docLen})$ are also included.

2.3.2.11 **phLen**: *length of the candidate phrase.*

For the monolithic framework, the length of the candidate phrase (phLen) in words and $\log(1 + \mathit{phLen})$ are included as features. These features are not used in the decomposed framework.

2.3.2.12 **Q**: *query log.*

The query log of a search engine reflects the distribution of the keywords people are most interested in. We use the information to create the following features. For these experiments, unless otherwise mentioned, we used a log file with the most frequent 7.5 million queries.

For the monolithic framework, we consider one binary feature – whether the phrase appears in the query log, and two real-valued features – the frequency with which it appears and the log value, $\log(1 + \mathit{frequency})$. For the decomposed framework, we consider more variations of this information: whether the word appears in the query log file as the first word of a query; whether the word appears in the query log file as an interior word of a query; and whether the word appears in the query log file as the last word of a query. The frequency values of the above features and their log values ($\log(1 + f)$, where f is the corresponding frequency value) are also used as real-valued features. Finally, whether the word never appears in any query log entries is also a feature.

2.4 Postprocessor

After the classifier predicts the probabilities of the candidates associated with the possible labels, our keyword extraction system generates a list of keywords ranked by the probabilities. When the Monolithic Combined framework is used, we simply return the most probable words or phrases. When the Monolithic Separate (MoS) framework is used, the highest probability of identical fragments is picked as the probability of the phrase.

In the Decomposed Separate (DeS) framework, we need to convert probabilities of individual words being phrase components (Begininning, Inside, etc.) into probabilities of relevance of whole phrases. Typically, in phrase labeling problems like information extraction, this conversion is done with the Viterbi [17] algorithm, to find the most probable assignment of the word label sequence of each sentence [5]. We rejected this method for two reasons. First, because in our training set, only a few words per document tend to be labeled as relevant, our system almost never assigns high probability to a particular phrase, and thus the Viterbi assignment would typically reject *all* phrases. Second, in typical information extraction applications, we not only want to find various entities, we also want to find their relationships and roles. In our application, we simply want a list of potential entities. It is thus fine to extract potentially overlapping strings as potential keywords, something that would not work well if role-assignment was required.

Therefore, we use the following mechanism to estimate the probability of a phrase in the DeS framework. Given a phrase of length n, we calculate the overall probability for the phrase by multiplying by the probability of the individual words being the correct label of the label sequence. For example, if $n = 3$, then the correct label sequence is B, I, L. The probability of this phrase being a keyword, p_1, is derived by $p(w_1 = B) \cdot p(w_2 = I) \cdot p(w_3 = L)$. If the phrase is not a keyword, then the correct label sequence is O, O, O. The corresponding probability, p_0, is then $p(w_1 = O) \cdot p(w_2 = O) \cdot p(w_3 = O)$. The actual probability used for this phrase is then $p_1/(p_0 + p_1)$. Among the the normalization methods we tried, this strategy works the best in practice.

3. EXPERIMENTS

This section reports the experimental results comparing our system with several baseline systems, the comparisons between the variations of our system, the contribution of individual features, and the impact of reducing the search query log size. We first describe how the documents were obtained and annotated, as well as the performance measures.

3.1 Data and Evaluation Criteria

The first step was to obtain and label data, namely a set of web pages. We collected these documents at random from the web, subject to the following criteria:

First, each page was in a crawl of the web from MSN Search. Second, each page displayed content-targeted advertising (detected by the presence of special Javascript). This criterion was designed to make sure we focused on the kind of web pages where content-targeted advertising would be desirable, as opposed to truly random pages. Third, the pages also had to occur in the Internet Archive[3]. We hope to share our labeled corpus, so other researchers can experiment with it. Choosing pages in the Internet Archive may make that sharing easier. Fourth, we selected no more than one page per domain (in part, because of copyright issues, and in part to ensure diversity of the corpus). Fifth, we tried to eliminate foreign language and adult pages. Altogether, we collected 1109 pages.

[3]http://www.archive.org

Next, we selected 30 of these pages at random to be used for inter-annotator agreement measurements. We then randomly split the remaining pages into 8 sets of 120 pages and one set of 119 pages. We gave each set to one of our annotators. For each annotator, the first time they labeled a set of web pages, we also gave them the additional 30 pages for inter-annotator agreement, scattered randomly throughout (so, they received 150 pages total the first time, 120 or 119 on subsequent sets). Several pages were rejected for various reasons, such as one annotator who did not complete the process, several foreign language pages that slipped through our screening process, etc. As a result, 828 web documents had legitimate annotations and were used to train and test the system.

Annotators were instructed to look at each web page, and determine appropriate keywords. In particular, they were given about 3 pages of instructions with actual examples. The core of the instructions was essentially as follows:

> Advertising on the web is often done through keywords. Advertisers pick a keyword, and their ad appears based on that. For instance, an advertiser like Amazon.com would pick a keyword like "book" or "books." If someone searches for the word "book" or "books", an Amazon ad is shown. Similarly, if the keyword "book" is highly prominent on a web page, Amazon would like an ad to appear.
>
> We need to show our computer program examples of web pages, and then tell it which keywords are "highly prominent." That way, it can learn that words like "the" and "click here" are never highly prominent. It might learn that words that appear on the right (or maybe the left) are more likely to be highly prominent, etc. Your task is to create the examples for the system to learn from. We will give you web pages, and you should list the highly prominent words that an advertiser might be interested in.

There was one more important instruction, which was to try to use only words or phrases that actually occurred on the page being labeled. The remaining portion of the instructions gave examples and described technical details of the labeling process. We used a snapshot of the pages, to make sure that the training, testing, and labeling processes all used identical pages. The snapshotting process also had the additional advantage that most images, and all content-targeted advertising, were not displayed to the annotators, preventing them from either selecting terms that occurred only in images, or from being polluted by a third-party keyword selection process.

3.2 Performance measures

We computed three different performance measures of our various systems. The first performance measure is simply the *top-1* score. To compute this measure, we counted the number of times the top output of our system for a given page was in the list of terms described by the annotator for that page. We divided this number by the maximum achievable top-1 score, and multiplied by 100. To get the maximum achievable top-1 score, for each test document, we first removed any annotations that did not occur somewhere in the web page (to eliminate spelling mistakes, and occasional annotator confusion). The best achievable score was then the number of documents that still had at least one annotation. We counted answers as correct if they matched, ignoring case, and with all whitespace collapsed to a single space.

The second performance measure is the *top-10* score, which is similar to the top-1 measure but considers 10 candidates instead of 1. To compute this measure, for each web page, we counted how many of the top 10 outputs of our system were in the list of terms described by the annotator for that page. The sum of these numbers was then divided by the maximum achievable top-10 score, and multiplied by 100.

It is important in an advertising system to not only extract accurate lists of keywords, but to also have accurate probability estimates. For instance, consider two keywords on a particular page, say "digital camera", which might monetize at \$1 per click, and "CD-R media", which monetizes at say, 10 cents per click. If there is a 50% chance that "CD-R media" is relevant, and only a 10% chance that "digital camera" is relevant, overall expected revenue from showing "digital camera" ads will still be higher than from showing "CD-R media" ads. Therefore, accurately estimating probabilities leads to potentially higher advertising revenue.

The most commonly used measure of the accuracy of probability estimates is *entropy.* For a given term t and a given web page p, our system computes the probability that the word is relevant to the page, $P(t|p)$ (i.e., the probability that the annotator listed the word on the relevant list). The entropy of a given prediction is

$$\begin{array}{ll} -\log_2 P(t|p) & \textit{if } t \textit{ is relevant to } p \\ -\log_2(1 - P(t|p)) & \textit{if } t \textit{ is not relevant to } p \end{array}$$

Lower entropies correspond to better probability estimates, with 0 being ideal. When we report the entropy, we will report the average entropy across all words and phrases up to length 5 in all web pages in the test set, with duplicates within a page counted as one occurrence. The DeS framework was not easily amenable to this kind of measurement, so we will only report the entropy measure for the MoS and MoC methods.

3.3 Inter-annotator Agreement

We wanted to compute some form of inter-annotator agreement, where the typical choice for this measure is the *kappa* measure. However, that measure is designed for annotations with a small, fixed number of choices, where a prior probability of selecting each choice can be determined. It was not clear how to apply it for a problem with thousands of possible annotations, with the possible annotations different on every page.

Instead, we used our inter-annotator agreement data to compute an upper-bound for the possible performance on our top-1 measure. In particular, we designed an accuracy measure called *committee-top-1.* Essentially this number measures roughly how well a committee of four people could do on this task, by taking the most common answer selected by the four annotators.

To compute this, we performed the following steps. First, we selected a page at random from the set of 30 labeled web pages for inter-annotator agreement. Then, we randomly selected one of the five annotators as *arbitrator*, whose results we called "correct." We then merged the results of the remaining four annotators, and found the single most frequent one. Finally, if that result was on the "correct" keywords list, we considered it correct; otherwise we considered it wrong. The average committee-top-1 score from 1000 samples was 25.8. This gives a general feeling for the difficulty of the problem. Due to the small number of documents used (30) and the small number of annotators (5), there is considerable uncertainty in this estimate. In fact, in some cases, our experimental results are slightly better than this number, which we attribute to the small size of the sample used for inter-annotator agreement.

3.4 Results

All experiments were performed using 10-way cross validation. In most cases, we picked an appropriate baseline, and computed statistical significance of differences between results and this baseline. We indicate the baseline with a ^{b} symbol. We indicate significance at the 95% level with a † symbol, and at the 99% level with a ‡ symbol. Significance tests were computed with a two-tailed paired t-test.

3.4.1 Overall Performance

We begin by comparing the overall performance of different systems and configurations. The top-1 and top-10 scores of these different systems are listed in Table 1. Among the system configurations we tested, the monolithic combined (MoC) framework is in general the best. In particular, when using all but the linguistic features, this configuration achieves the highest top-1 and top-10 scores in all the experiments, just slightly (and not statistically significantly) better than the same framework with all features.

The monolithic separate (MoS) system with all features performs worse for both top-1 and top-10, although only the top-10 result was statistically significant. Despite its prevalence in the information extraction community, for this task, the decomposed separate (DeS) framework is significantly worse than the monolithic approach. We hypothesize that the advantages of using features based on phrases as a whole (used in MoS and MoC) outweigh the advantages of features that combine information across words.

In addition to the different configurations of our systems, we also compare with a state-of-the-art keyword extraction system – KEA [7], by providing it with our preprocessed, plain-text documents. KEA is also a machine learning-based system. Although it relies on simpler features, it uses more sophisticated information retrieval techniques like removing stop words and stemming for preprocessing. Our best system is substantially better than KEA, with relative improvements of 27.5% and 22.9% on top-1 and top-10 scores, respectively. We tried a very simple baseline system, MoC TFIDF, which simply uses traditional TF$\times$IDF scores. This system only reaches 13.01 in top-1 and 19.03 in top-10. We also tried allowing using TF and IDF features, but allowing the weights to be trained wiht logistic regression (MoC IR). Training slightly improved top-1 to 13.63, and substantially improved the top-10 score to 25.67.

Notice that the top-1 score of our best system actually exceeds the committee-top-1 inter-annotator agreement score. There was considerable uncertainty in that number due to the small number of annotators and small number of documents for inter-annotator agreement, but we interpret these results to mean that our best system is in the same general range as human performance, at least on top-1 score.

system	top-1	top-10
MoC (Monolithic, Combined), *-Lin*	30.06^b	46.97^b
MoC (Monolithic, Combined), *All*	29.94	46.45
MoS (Monolithic, Separate), *All*	27.95	$44.13^{\ddagger}$
DeS (Decomposed, Separate), *All*	$24.25^{\ddagger}$	$39.11^{\ddagger}$
KEA [7]	$23.57^{\ddagger}$	$38.21^{\ddagger}$
MoC (Monolithic, Combined), *IR*	$13.63^{\ddagger}$	$25.67^{\ddagger}$
MoC (Monolithic, Combined), *TFIDF*	$13.01^{\ddagger}$	$19.03^{\ddagger}$

Table 1: Performance of different systems

3.4.2 Feature Contribution

One important and interesting question is the contribution of different types of features. Namely, how important is a particular type of information to the keyword extraction task. We studied this problem by conducting a series of ablation studies: removing features of a specific type to see how much they contribute. We repeated these experiments in the three different candidate selector frameworks of our keyword extraction system to see if the same information affects the system differently when the framework changes.

Tables 2, 3, and 4 show the detailed results. Each row is either the baseline system which uses all features, or the compared system that uses all except one specific kind of feature. In addition to the top-1 and top-10 scores, we also compare the entropies of the various systems using the same framework. Recall that lower entropies are better. Entropy is a somewhat smoother, more sensitive measure than top-n scores, making it easier to see differences between different systems. Note that because the numbers of candidates are different, entropies from systems using different frameworks are not comparable.

features		top-1	top-10	entropy
A	all	27.95^b	44.13^b	0.0120040^b
-C	capitalization	27.39	43.50	$0.0120945^{\ddagger}$
-H	hypertext	27.39	43.82	$0.0120751^{\ddagger}$
-IR	IR	$18.77^{\ddagger}$	$33.60^{\ddagger}$	$0.0149899^{\ddagger}$
-Len	length	27.10	$42.45^{\dagger}$	0.0121040
-Lin	linguistic	28.05	44.89	$0.0122166^{\ddagger}$
-Loc	location	27.24	$42.64^{\ddagger}$	$0.0121860^{\ddagger}$
-M	meta	27.81	44.05	0.0120080
-Ms	meta section	27.52	$43.09^{\dagger}$	0.0120390
-Q	query log	$20.68^{\ddagger}$	$39.10^{\ddagger}$	$0.0129330^{\ddagger}$
-T	title	27.81	44.25	0.0120040
-U	URL	$26.09^{\dagger}$	44.14	$0.0121409^{\dagger}$

Table 2: The system performance by removing one set of features in the MoS framework

By examining these results, we found that *IR* features and *Query Log* features are the most helpful consistently in all three frameworks. The system performance dropped significantly after removing either of them. However, the impact of other features was not so clear. The location feature also seems to be important, but it affects the top-10 score much more than the top-1 score. Surprisingly, unlike the cases in other typical phrase labeling problems, linguistic features don't seem to help in this keyword extraction domain. In fact, removing this set of features in the decomposed separate framework improves the top-1 score a little. We also observe more statistically significant differences in the entropy metric.

features		top-1	top-10	entropy
A	all	29.94^b	46.45^b	0.0113732^b
-C	capitalization	30.11	46.27	$0.0114219^{\dagger}$
-H	hypertext	30.79	$45.85^{\dagger}$	0.0114370
-IR	IR	$25.42^{\ddagger}$	$42.26^{\ddagger}$	$0.0119463^{\ddagger}$
-Len	length	30.49	$44.74^{\dagger}$	$0.0119803^{\ddagger}$
-Lin	linguistic	30.06	46.97	$0.0114853^{\ddagger}$
-Loc	location	29.52	$44.63^{\dagger}$	$0.0116400^{\ddagger}$
-M	meta	30.10	46.78	$0.0113633^{\ddagger}$
-Ms	meta section	29.33	46.33	0.0114031
-Q	query log	$24.82^{\dagger}$	$42.30^{\ddagger}$	$0.0121417^{\ddagger}$
-T	title	28.83	46.94	0.0114020
-U	URL	30.53	46.39	0.0114310

Table 3: The system performance by removing one set of features in the MoC framework

features		top-1	top-10
A	all	24.25	39.11
-C	capitalization	24.26	39.20
-H	hypertext	24.96	39.67
-IR	IR	$19.10^{\ddagger}$	$30.56^{\ddagger}$
-Lin	linguistic	$25.96^{\dagger}$	39.36
-Loc	location	24.84	37.93
-M	meta	24.69	38.91
-Ms	meta section	24.68	38.71
-Q	query log	$21.27^{\ddagger}$	$33.95^{\ddagger}$
-T	title	24.40	38.89
-U	URL	25.21	38.97

Table 4: The system performance by removing one set of features in the DeS framework

Evaluating the contribution of a feature by removing it from the system does not always show its value. For instance, if two types of features have similar effects, then the system performance may not change by eliminating only one of them. We thus conducted a different set of experiments by adding features to a baseline system.

We chose a system in the monolithic combined framework (MoC), using only IR features, as the baseline system. We then built different systems by adding one additional set of features, and comparing with the baseline system. Table 5 shows performance on the top-1, top-10, and entropy metrics, ordered by difference in top-1.

Interestingly, all the features seem to be helpful when they are combined with the baseline IR features, including the linguistic features. We thus conclude that the linguistic features were not really useless, but instead were redundant with other features like capitalization (which helps detect proper nouns) and the Query Log features (which helps detect linguistically appropriate phrases.) The Query Log features are still the most effective among all the features we experimented with. However, the performance gap between our best system and a simple system using only IR and Query Log features is still quite large.

3.4.3 Different Query Log Sizes

The query log feature were some of the most helpful features, second only to the IR features. These features used the top 7.5 million English queries from MSN Search. In

	features	top-1	top-10	entropy
	IR	13.63^b	25.67^b	0.0163299^b
+Q	query log	$22.36^‡$	$35.88^‡$	$0.0134891^‡$
+T	title	$19.90^‡$	$34.17^‡$	$0.0152316^‡$
+Len	length	$19.22^‡$	$33.43^‡$	$0.0134298^‡$
+Ms	meta section	$19.02^‡$	$31.90^‡$	$0.0154484^‡$
+H	hypertext	$18.46^‡$	$30.36^‡$	$0.0150824^‡$
+Lin	linguistic	$18.20^‡$	$32.26^‡$	$0.0146324^‡$
+C	capitalization	$17.41^‡$	$33.16^‡$	$0.0146999^‡$
+Loc	location	$17.01^‡$	$32.76^‡$	$0.0154064^‡$
+U	URL	$16.72^†$	$28.63^‡$	$0.0157466^‡$
+M	meta	$16.71^†$	$28.19^‡$	$0.0160472^†$

Table 5: The system performance by adding one set of features in the MoC framework

practice, deploying systems using this many features may be problematic. For instance, if we want to use 20 languages, and if each query entry uses about 20 bytes, the query log files would use 3 GB. This amount of memory usage would not affect servers dedicated to keyword extraction, but might be problematic for servers that perform other functions (e.g. serving ads, or as part of a more general blogging or news system). In scenarios where keyword extraction is done on the client side, the query log file size is particularly important.

Query logs may also help speed up our system. One strategy we wanted to try is to consider only the phrases that appeared in the query log file as potential matches for keyword extraction. In this case, smaller query logs lead to even larger speedups. We thus ran a set of experiments under the MoC framework, using different query log file sizes (as measured by a frequency threshold cutoff). The log file sizes were very roughly inversely proportional to the cutoff. The leftmost point, with a cutoff of 18, corresponds to our 7.5 million query log file.

The graph in Figure 1 shows both the top-1 and top-10 scores at different sizes. "Res. top-1" and "Res. top-10" are the results where we restricted the candidate phrases by eliminating those that do not appear in the query log file. As a comparison, the graph also shows the non-restricting version (i.e., top-1 and top-10). Note that of course the top-1 scores cannot be compared to the top-10 scores.

When the frequency cutoff threshold is small (i.e., large query log file size), this restricting strategy in fact improves the top-1 score slightly, but hurts the top-10 score. This phenomenon is not surprising since restricting candidates may eliminate some keywords. This tends to not affect the top-1 score since when using the query log file as an input, the most probable keywords in a document almost always appear in the keyword file, if the keyword file is large. Less probable, but still good keywords, are less likely to appear in the keyword file. As the query log file size becomes smaller, the negative effect becomes more significant. However, if only the top-1 score is relevant in the application, then a reasonable cutoff point may be 1000, where the query log file has less than 100,000 entries.

On the other hand, the top-1 and top-10 scores in the non-restricting version decrease gradually as the size of the query log file decreases. Fairly small files can be used with the non-restricting version if space is at a premium. After all, the extreme case is when no query log file is used, and as we have seen in Table 3, in that case the top-1 and top-10 scores are still 24.82 and 42.30, respectively.

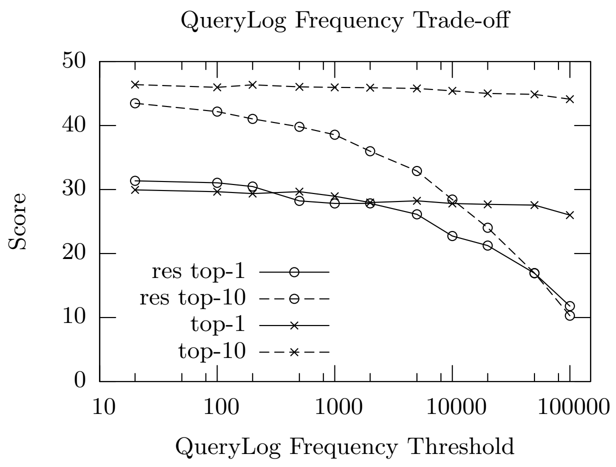

Figure 1: The performance of using different sizes of the query log file

4. RELATED WORK

There has been a moderate amount of previous work that is directly relevant to keyword extraction. In most cases, the keyword extraction was not applied to web pages, but instead was applied to a different text domain. Most previous systems were fairly simple, using either a small number of features, a simple learning method (Naive Bayes), or both. In this section, we describe this previous research in more detail.

4.1 GenEx

One of the best known programs for keyword extraction is Turney's GenEx system [22]. GenEx is a rule-based keyphrase extraction system with 12 parameters tuned using a Genetic Algorithm. GenEx has been trained and evaluated on a collection of 652 documents of three different types: journal articles, email messages, and web pages. Precisions of top 5 phrases and top 15 phrases are reported as evaluation metrics, where the numbers are 0.239 and 0.128 respectively for all documents. Turney showed GenEx's superiority by comparing it with an earlier keyphrase extraction system trained by C4.5 [16] with several complicated features [22].

4.2 KEA and Variations

Concurrently with the development of GenEx, Frank *et al.* developed the KEA keyphrase extraction algorithm using a simple machine learning approach [7]. KEA first processes documents by removing stopwords and by stemming. The candidate phrases are represented using only three features – TF×IDF, distance (number of words before the first occurrence of the phrase, divided by the number of words in the whole document), and keyphrase-frequency, which is the number of times the candidate phrase occurs in other documents. The classifier is trained using the naive Bayes learning algorithm [14]. Frank et al. compared KEA with GenEx, and showed that KEA is slightly better in general, but the difference is not statistically significant [7].

KEA's performance was improved later by adding Web related features [23]. In short, the number of documents returned by a search engine using the keyphrase as query terms is used as additional information. This feature is particularly useful when training and testing data are from different domains. In both *intra* and *inter* domain evaluations, the relative performance gain in precision is about 10%.

Kelleher and Luz also reported an enhanced version of KEA for web documents [13]. They exploited the link information of a web document by adding a "semantic ratio" feature, which is the frequency of the candidate phrase P in the original document D, divided by the frequency of P in documented linked by D.

Using the Meta Keyword HTML tag as the source of annotations, they experimented with their approach on four sets of documents, where documents are connected by hyperlinks in each collection. Among three of them, they reported significant improvement (45% to 52%) compared to the original version of KEA. Adding the semantic ratio to our system would be an interesting area of future research. It should, however, be noted that using the semantic ratio requires downloading and parsing several times as many web pages, which would introduce substantial load. Also, in practice, following arbitrary links from a page could result in simulating clicks on, e.g., unsubscribe links, leading to unexpected or undesirable results.

The use of linguistic information for keyword extraction was first studied by Hulth [12]. In this work, noun phrases and predefined part-of-speech tag patterns were used to help select phrase candidates. In addition, whether the candidate phrase has certain POS tags is used as features. Along with the three features in the KEA system, Hulth applied *bagging* [1] as the learning algorithm to train the system. The experimental results showed different degrees of improvements compared with systems that did not use linguistic information. However, direct comparison to KEA was not reported.

4.3 Information Extraction

Analogous to keyword extraction, *information extraction* is also a problem that aims to extract or label phrases given a document [8, 2, 19, 5]. Unlike keyword extraction, information extraction tasks are usually associated with predefined semantic templates. The goal of the extraction tasks is to find certain phrases in the documents to fill the templates. For example, given a seminar announcement, the task may be finding the *name* of the speaker, or the *starting time* of the seminar. Similarly, the *named entity recognition* [21] task is to label phrases as semantic entities like *person*, *location*, or *organization*.

Information extraction is in many ways similar to keyword extraction, but the techniques used for information extraction are typically different. While most keyword extraction systems use the monolithic combined (MoC) framework, typically, information extraction systems use the DeS framework, or sometimes MoS. Since identical phrases may have different labels (e.g., "Washington" can be either a person or a location even in the same document), candidates in a document are never combined. The choice of features is also very different. These systems typically use lexical features (the identity of specific words in or around the phrase), linguistic features, and even conjunctions of these features. In general, the feature space in these problems is huge – often hundreds of thousands features. Lexical features for keyword extraction would be an interesting area of future research, although our intuition is that these features are less likely to be useful in this case.

4.4 Impedance Coupling

Ribeiro-Neto *et al.* [18] describe an Impedance Coupling technique for content-targeted advertising. Their work is perhaps the most extensive previously published work specifically on content-targeted advertising.

However, Ribeiro-Neto *et al.'s* work is quite a bit different from ours. Most importantly, their work focused not on finding keywords on web pages, but on directly matching advertisements to those web pages. They used a variety of information, including the text of the advertisements, the destination web page of the ad, and the full set of keywords tied to a particular ad (as opposed to considering keywords one at a time.) They then looked at the cosine similarity of these measures to potential destination pages.

There are several reasons we do not compare directly to the work of Ribeiro-Neto *et al.* Most importantly, we, like most researchers, do not have a database of advertisements (as opposed to just keywords) that we could use for experiments of this sort. In addition, they are solving a somewhat different problem than we are. Our goal is to find the most appropriate keywords for a specific page. This dovetails well with how contextual advertising is sold today: advertisers bid on specific keywords, and their bids on different keywords may be very different. For instance, a purveyor of digital cameras might use the same ad for "camera", "digital camera", "digital camera reviews", or "digital camera prices." The advertiser's bids will be very different in the different cases, because the probability that a click on such an ad leads to a sale will be very different. Ribeiro-Neto *et al.'s* goal is not to extract keywords, but to match web pages to advertisements. They do not try to determine which particular keyword on a page is a match, and in some cases, they match web pages to advertisements even when the web page does not contain any keywords chosen by an advertiser, which could make pricing difficult. Their techniques are also somewhat time-consuming, because they compute the cosine similarity between each web page and a bundle of words associated with each ad. They used only a small database (100,000 ads) applied to a small number of web pages (100), making such time-consuming comparisons possible. Real world application would be far more challenging: optimizations are possible, but would need to be an area of additional research. In contrast, our methods are directly applicable today.

4.5 News Query Extraction

Henzinger *et al.* [11] explored the domain of keyword extraction from a news source, to automatically drive queries. In particular, they extracted query terms from the closed captioning of television news stories, to drive a search system that would retrieve related online news articles. Their basic system used TF×IDF to score individual phrases, and achieved slight improvements from using TF×IDF^2. They tried stemming, with mixed results. Because of the nature of broadcast news programs, where boundaries between topics are not explicit, they found improvements by using a history feature that automatically detected topic boundaries. They achieved their largest improvements by postprocessing articles to remove those that seemed too different from the news broadcast. This seems closely related to the previously mentioned work of Ribeiro-Neto *et al.*, who found analogous comparisons between documents and advertisements helpful.

4.6 Email Query Extraction

Goodman and Carvalho [10] previously applied similar techniques to the ones described here for query extraction for email. Their goal was somewhat different than our goal here, namely to find good search queries, to drive traffic to search engines, although the same technology they used could be applied to find keywords for advertising. Much of their research focused on email-specific features, such as word occurrence in subject lines, and distinguishing new parts of an email message from earlier, "in-reply-to" sections.

In contrast, our work focuses on many web-page-specific features, such as keywords in the URL, and in meta-data. Our research goes beyond theirs in a number of other ways. Most importantly, we tried both information-extraction inspired methods (DeS) and linguistic features. Here, we also examine the tradeoff between query file size and accuracy, showing that large query files are not necessary for near-optimal performance, if the restriction on words occurring in the query file is removed. Goodman *et al.* compare only to simple TF×IDF style baselines, while in our research, we compare to KEA. We also compute a form of inter-annotator agreement, something that was not previously done. The improvement over KEA and the near human performance measurements are very important for demonstrating the high quality of our results.

5. CONCLUSIONS

The better we can monetize the web, the more features that can be provided. To give two examples, the explosion of free blogging tools and of free web-based email systems with large quotas, have both been supported in large part by advertising, much of it using content-targeting. Better targeted ads are less annoying for users, and more likely to inform them of products that deliver real value to them. Better advertising thus creates a win-win-win situation: better web-based features, less annoying ads, and more information for users.

Our results demonstrate a large improvement over Kea. We attribute this improvement to the large number of helpful features we employed. While Kea employs only three features, we employed 12 different sets of features; since each set contained multiple features, we actually had about 40 features overall. As we showed in Table 5, every one of these sets was helpful, although as we also showed, some of them were redundant with each other. GenEx, with 12 features, works about as well as Kea: the choice of features and the learning algorithm are also important. Our most important new feature was the query frequency file from MSN Search.

6. ACKNOWLEDGMENTS

We thank Emily Cook, Natalie Jennings, Samantha Dickson, Krystal Paige and Katie Neely, who annotated the data. We are also grateful to Alexei Bocharov, who helped us measure inter-annotator agreement.

7. REFERENCES

[1] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[2] M. Califf and R. Mooney. Bottom-up relational learning of pattern matching rules for information extraction. *JMLR*, 4:177–210, 2003.

[3] X. Carreras, L. Màrquez, and J. Castro. Filtering-ranking perceptron learning for partial parsing. *Machine Learning*, 60(1–3):41–71, 2005.

[4] S. F. Chen and R. Rosenfeld. A gaussian prior for smoothing maximum entropy models. Technical Report CMU-CS-99-108, CMU, 1999.

[5] H. Chieu and H. Ng. A maximum entropy approach to information extraction from semi-structure and free text. In *Proc. of AAAI-02*, pages 786–791, 2002.

[6] Y. Even-Zohar and D. Roth. A sequential model for multi class classification. In *EMNLP-01*, 2001.

[7] E. Frank, G. W. Paynter, I. H. Witten, C. Gutwin, and C. G. Nevill-Manning. Domain-specific keyphrase extraction. In *Proc. of IJCAI-99*, pages 668–673, 1999.

[8] D. Freitag. Machine learning for information extraction in informal domains. *Machine Learning*, 39(2/3):169–202, 2000.

[9] J. Goodman. Sequential conditional generalized iterative scaling. In *ACL '02*, 2002.

[10] J. Goodman and V. R. Carvalho. Implicit queries for email. In *CEAS-05*, 2005.

[11] M. Henzinger, B. Chang, B. Milch, and S. Brin. Query-free news search. In *Proceedings of the 12th World Wide Web Conference*, pages 1–10, 2003.

[12] A. Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proc. of EMNLP-03*, pages 216–223, 2003.

[13] D. Kelleher and S. Luz. Automatic hypertext keyphrase detection. In *IJCAI-05*, 2005.

[14] T. Mitchell. Tutorial on machine learning over natural language documents, 1997. Available from `http://www.cs.cmu.edu/~tom/text-learning.ps`.

[15] V. Punyakanok and D. Roth. The use of classifiers in sequential inference. In *NIPS-00*, 2001.

[16] J. R. Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[17] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), February 1989.

[18] B. Ribeiro-Neto, M. Cristo, P. B. Golgher, and E. S. de Moura. Impedance coupling in content-targeted advertising. In *SIGIR-05*, pages 496–503, 2005.

[19] D. Roth and W. Yih. Relational learning via propositional algorithms: An information extraction case study. In *IJCAI-01*, pages 1257–1263, 2001.

[20] C. Sutton and A. McCallum. Composition of conditional random fields for transfer learning. In *Proceedings of HLT/EMLNLP-05*, 2005.

[21] E. F. Tjong Kim Sang. Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *CoNLL-02*, 2002.

[22] P. D. Turney. Learning algorithms for keyphrase extraction. *Information Retrieval*, 2(4):303–336, 2000.

[23] P. D. Turney. Coherent keyphrase extraction via web mining. In *Proc. of IJCAI-03*, pages 434–439, 2003.