Domain-Specific Web Search with Keyword Spices

Satoshi Oyama, Takashi Kokubo, and Toru Ishida, *Fellow*, *IEEE*

Abstract—Domain-specific Web search engines are effective tools for reducing the difficulty experienced when acquiring information from the Web. Existing methods for building domain-specific Web search engines require human expertise or specific facilities. However, we can build a domain-specific search engine simply by adding domain-specific keywords, called "keyword spices," to the user's input query and forwarding it to a general-purpose Web search engine. Keyword spices can be effectively discovered from Web documents using machine learning technologies. This paper will describe domain-specific Web search engines that use keyword spices for locating recipes, restaurants, and used cars.

Index Terms—Domain-specific Web search, query modification, decision tree, information retrieval, machine learning.

1 Introduction

Consider the situation where you want to cook a dish that uses beef and you are looking for a recipe. Using the Web is the most effective way of finding a variety of recipes, so let us challenge Google with the input "beef." [1] You will find few recipes, but many other pages on disease, farming, and trading in the top 20 returned pages. Next, try the query "beef pepper." You will be surprised to find that most of the returned pages are recipes! More surprisingly, adding the keyword "pepper" is useful not only for locating beef recipes, but it is also effective for finding other recipes such as "pork" or "chicken." This indicates the possibility of making a domain-specific search engine that returns only recipe pages simply by adding a keyword, such as "pepper," to the user's query.

Domain-specific search engines are search engines that only return Web pages relevant to certain domains. With general-purpose Web search engines like Google or Altavista,[2] the user can search through all indexed pages, but such search engines can cause the user major problems. Because the Web consists of pages on diverse topics, naive queries by users find matches in many irrelevant pages as described above. Of course, the user will obtain more relevant pages if he can formulate an appropriate query that consists of multiple keywords, but it is difficult for most users because this requires much experience and skill. In fact, up to 70 percent of Web searches use only one keyword [1]. Making full use of more sophisticated search functions like Boolean queries is much more difficult.

This problem is greatly reduced if the user employs a special search engine designed for the topic of interest. For example, special search engines dedicated to recipes are less likely to return irrelevant pages even if the single keyword "beef" is entered.

The most straightforward approach to building domain-specific Web search engines is to collect and index only the relevant pages available on the Web. If the indices are manually constructed, it requires too much cost to build and maintain the indices and it is not a scalable method that can catch up with the rapidly growing Web.

Some domain-specific Web search engines use Web crawlers that collect only domain-specific pages. One example is Cora [2], a domain-specific search engine for computer science research papers. Its crawlers start from the home pages of computer science departments and laboratories and finds research papers effectively using machine learning technologies. SPIRAL [3] or WebKB [4] also use crawlers. These systems offer sophisticated search functions because they establish their own local databases and can apply various machine learning or knowledge representation techniques to the data. Unfortunately, the time and network bandwidth consumed by crawlers are excessive in domains such as personal homepages or cooking pages because these pages are dispersed across many Web sites. This suggests that using crawlers is not an efficient way of developing search engines for these domains.

Reusing the large indices of general-purpose search engines to build domain-specific ones is a clever idea [5]. A domain-specific search engine forwards the user's query to one or more general-purpose search engines and eliminates the irrelevant documents from the returned ones by domain-specific filters. We call this the *filtering model* approach to building domain-specific search engines (Fig. 1). This is a kind of meta-search engine [6] and is the basis of Ahoy! [7], which is a search engine specialized for finding personal homepages. The weakness of the filtering model is its slow response to user's input. It needs to download many irrelevant pages as well as relevant ones and then classify them. Consequently, the response time of the filtering model exceeds that of a crawler-based search engine that has its own document databases.

[1] http://www.google.com
[2] http://www.altavista.com

• S. Oyama and T. Ishida are with the Department of Social Informatics, Graduate School of Informatics, Kyoto University, Yoshida-Honmachi, Sakyo, Kyoto 606-8501, Japan. E-mail: {oyama, ishida}@i.kyoto-u.ac.jp.
• T. Kokubo is with NTT DoCoMo, Inc., 3-5 Hikarinooka, Yokosuka, Kanagawa 239-8536, Japan. E-mail: kokubo@mmd.yrp.nttdocomo.co.jp.

Manuscript received 1 Sept. 2002; revised 1 Apr. 2003; accepted 10 Apr. 2003. For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 118552.

1041-4347/04/$17.00 © 2004 IEEE Published by the IEEE Computer Society

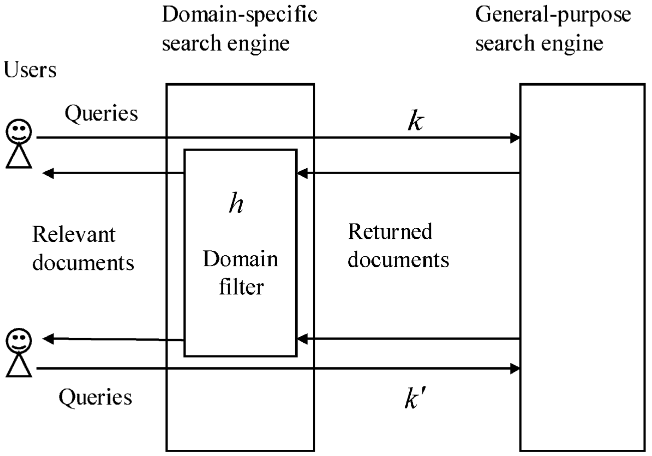

Fig. 1. Filtering model of building domain-specific Web search engines.

The keyword spice model does not filter documents returned by a general-purpose search engine. Instead, it extends the user's input query with a domain-specific Boolean expression (keyword spice) that better classifies the domain documents and passes the extended query to a general-purpose search engine (Fig. 2). This model is just the reverse of the filtering model.

The merit of the keyword spice model is its simplicity. High response performance is easy to achieve because the system can assume that all returned pages are domain pages and so simply displays all of them with no further processing. On the other hand, in the filtering model, the system has to analyze the results to eliminate the irrelevant pages. A very short program that adds the keyword spices to the user's input and forwards it to a general-purpose search engine can be written and embedded into the Web page. This method simplifies the construction of many domain-specific search engines.

There are several remaining questions. How can we find the most effective keywords for the domain? Can we find similar effective keywords for domains other than recipes? This paper addresses these issues and pursues a general method for building domain-specific search engines in various domains by using keyword spices.

The remainder of this paper is organized as follows: Section 2 presents the idea of building domain-specific search engines that use keyword spices. We formulate the domain-specific Web search as a classification problem and present that the problem of collecting training examples, which is a barrier to applying previous methods of text classification, is settled by our method. Section 3 describes a machine learning algorithm for discovering keyword spices that are highly effective but small enough to enter the commercial search engines. Section 4 evaluates our method in the domains of recipes, restaurants, and used cars. Section 5 describes related work and Section 6 provides discussion on future work, especially in terms of reducing the cost of labeling training examples. Our conclusions are given in Section 7.

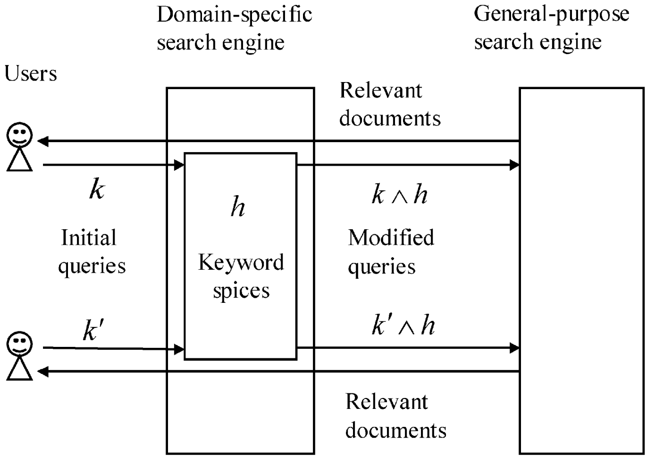

Fig. 2. Keyword spice model of building domain-specific Web search engines.

2 Domain-Specific Web Search with Keyword Spices

2.1 Domain-Specific Web Search as a Text Classification Problem

The above discussion of the filtering model indicates that the problem of building a domain-specific Web search engine can be regarded as the problem of classifying pages as either domain relevant or irrelevant. Unfortunately, human expertise is required to make a good domain-filter that can correctly classify domain-pages. Ahoy! has a learning mechanism to assess the patterns of relevant URLs from previous successful searches, but the filter basically depends on human heuristic knowledge.

One solution to the above problem is to make domain filters automatically from sample documents. Automatic text filtering, which classifies documents into relevant and nonrelevant ones, has been a major research topic in both information retrieval [8] and machine learning [9].

Here, we reuse some of the notations set in [9] to define the machine learning problem. We let $\mathcal{D}$ denote the set of all Web documents; $\mathcal{D}_t$ denotes the set of documents relevant to a certain domain. The target function (an ideal domain filter) that correctly classifies any document $d \in \mathcal{D}$ is given as

$$f(d) = \begin{cases} 1 & \text{if } d \in \mathcal{D}_t \\ 0 & \text{otherwise.} \end{cases}$$

We let $\mathcal{K}$ be the set of all keywords in the domain and let $\mathcal{H}$ be the hypothesis space composed of all Boolean expressions where any keyword $k \in \mathcal{K}$ is regarded as a Boolean variable. We adopt the Boolean hypothesis space because most commercial search engines can accept queries written in Boolean expressions.

A Boolean expression of keywords can be regarded as a function from $\mathcal{D}$ to $\{0, 1\}$ when we assign 1 (true) to a keyword (Boolean variable) if the keyword is contained in the document and 0 (false) otherwise. In the filtering model, the problem of building a domain filter is equivalent to finding hypothesis h that minimizes the error rate:

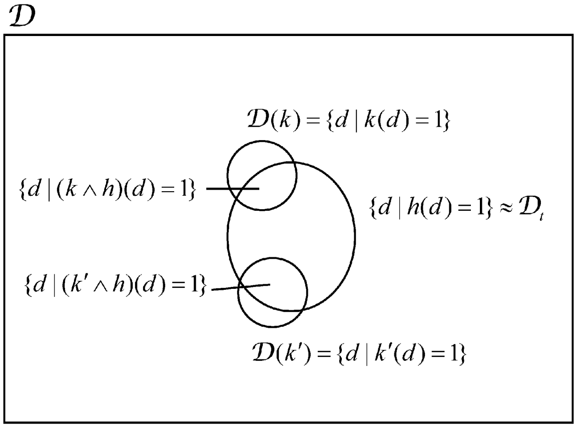
$\mathcal{D}$
$\mathcal{D}(k)=\{d \mid k(d)=1\}$
$\{d \mid (k \wedge h)(d)=1\}$
$\{d \mid h(d)=1\} \approx \mathcal{D}_t$
$\{d \mid (k' \wedge h)(d)=1\}$
$\mathcal{D}(k')=\{d \mid k'(d)=1\}$

Fig. 3. Sampling with input keywords to increase the ratio of positive examples.

$$\frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}}\delta(h(d),f(d)).$$

Note: Quantity $\delta(h(d),f(d))$ is 1 if $h(d)\neq f(d)$, 0 otherwise.

We can use various machine learning algorithms to find such filters if the training examples, which consist of documents randomly sampled from the Web together with their manual classification, are available. Unfortunately, making such training examples is the real barrier because the Web is very large and randomly sampling the Web will provide only a small likelihood of encountering the domain in question. In fact, most studies on text classification have been applied to e-mail, net news, or Web documents at limited sites, where the ratio of positive examples is rather high. Various machine learning methods used for text classification, such as decision trees [10], naive Bayesian classifiers [11], support vector machines [12], are difficult to directly apply to this problem because we need to solve the problem of sampling training examples from the Web beforehand.

2.2 Collecting Sample Web Pages According to the Assumption of User's Input

Our method is based on the idea that when we build a domain-specific Web search engine, we need consider only those Web pages that contain the user's input query keywords, not all Web pages. Obviously, the user always inputs at least one keyword if he wants to use a search engine. This insight into the nature of domain-specific Web search eliminates the problem of finding positive examples and enables us to make domain-specific search engines at reasonable cost. By entering a collection of keywords that users would be likely enter when accessing a specific domain to a general-purpose search engine, the resulting set of documents contains a fairer percentage of relevant pages than the complete Web. The returned Web pages can be classified by humans and the results assessed as training examples by most of the existing text classification methods to create a domain filter that can classify future pages.

As described in Fig. 3, the scope of sampling is reduced from set $\mathcal{D}$, all Web documents, to $\mathcal{D}(k)$, the set of Web pages that contain input keyword k; this increases the ratio of positive examples $\{d|(k\wedge h)(d)=1\}$. This idea makes it easier to create training sets and it becomes easier to build a domain filter, which is difficult with random sampling because of the sparseness of positive examples.

It would be best to collect training examples according to $p(k)$, the probability of that a user will input keyword k to a domain-specific search engine. However, we do not know $p(k)$ before the domain-specific search engine completes. We have to start with some reasonable value of $p(k)$ and modify the value as statistics on input keywords are collected. In practice, we can somehow estimate users' input when we design a domain-specific search engine. For example, in the recipe domain, we can use the names of ingredients such as "beef," "salmon," "potato," etc. as sample keywords and download Web pages containing these keywords from a general-purpose search engine. In this paper, we choose several input keyword candidates for each domain. We assume that all candidates have the same probability of occurrence and collect the same number of documents for each keyword.

By using domain filter h, we modify the user's input query k to $k\wedge h$, so the returned documents contain k and are included in the domain. In short, h is the *keyword spice* for the domain.

3 Algorithm for Extracting Keyword Spices

3.1 Identifying Keyword Spices

In this section, we describe an algorithm for extracting keyword spices [13]. First, collected sample pages are classified into two classes T (relevant to the domain) or F (irrelevant to the domain) by hand. We remove html tags from initially collected Web pages and extract nouns as keywords. We then split the examples into two disjoint subsets, the training set $\mathcal{D}_{training}$ (used for identifying initial keyword spices) and the validation set $\mathcal{D}_{validation}$, to simplify the keyword spices described in Section 3.2.

We apply a decision tree learning algorithm to discover keyword spices because it is easy to convert a tree into Boolean expressions, which are accepted by most commercial search engines. In this decision tree learning step, each keyword is used as an attribute whose value is 1 (when the document contains this keyword) or 0 (otherwise).

Fig. 4 shows an example of a simple decision tree that classifies documents. Each node is an attribute, the value of a branch indicates the value of the attribute, and each leaf is a class. In order to classify a document, we start at the root of the tree, examine whether the document contains the attribute (keyword) or not, and take the corresponding branch. The process continues until it reaches a leaf and the document is asserted to belong to the class corresponding to the value of the leaf. This tree classifies Web documents into T (domain documents) and F (the others) and the Web document, for example, that does not have "tablespoon," has "recipe," does not have "home," and does not have "top" belongs to class T.

We make the initial decision tree using an information gain measure [14] for greedy search without using any pruning technique. The information gain ratio [14] and gini index [15] are also commonly used as measures for choosing attributes with which to create a decision tree. Information gain ratio was proposed to avoid the problem that the information gain measure unfairly favors attributes

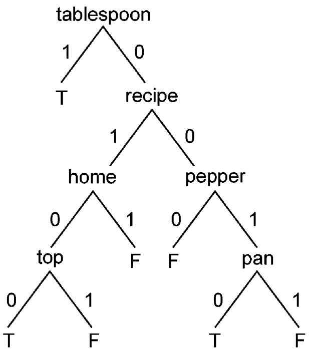

Fig. 4. An example of a decision tree that classifies documents.

$$
\begin{aligned}
& \text{tablespoon} \\
\vee\ & (\neg\text{tablespoon} \wedge \text{recipe} \wedge \neg\text{home} \wedge \neg\text{top}) \\
\vee\ & (\neg\text{tablespoon} \wedge \neg\text{recipe} \wedge \text{pepper} \wedge \neg\text{pan})
\end{aligned}
$$

Fig. 5. Boolean expression yielded by the tree in Fig. 4.

with many values. In our case, however, all attributes take one of two values, indicating whether a page contains the keyword or not. An empirical comparison shows that there are no significant differences between information gain and gini index with regard to the accuracy of a tree and its size [16]. Therefore, we arbitrary adopt information gain as the measure for selecting splitting attributes.

In our real case, the number of attributes (keywords) is large enough (several thousands) to make a tree that can correctly classify all examples in the training set $\mathcal{D}_{training}$. For each path in the induced tree that ends in a positive result, we make a Boolean expression that conjoins all keywords (a keyword is treated as a positive literal when its value is 1 and a negative literal otherwise) on the path. Our aim is to make a Boolean expression query that specifies the domain documents and that can be entered into search engines; accordingly, we consider only positive paths.

We make a Boolean expression h by making a disjunction of all these conjunctions (i.e., we make a disjunctive normal form of a Boolean expression). This is the initial form of keyword spices. Fig. 5 provides an example of a Boolean expression yielded by the tree in Fig. 4.

3.2 Simplifying Keyword Spices

Fig. 6 shows a decision tree induced from collected Web documents in the experiments described in the next section.[3] Decision trees are usually very large, which triggers the overfitting problem. Furthermore, too-complex queries cannot be accepted by commercial search engines, so we have to simplify the induced Boolean expression. We developed a two-stage simplification algorithm (described below) that is similar to rule postpruning [17].

1. For each conjunction c in h we remove keywords (Boolean literals) from c to simplify it.
2. We remove conjunctions from disjunctive normal form h to simplify it.

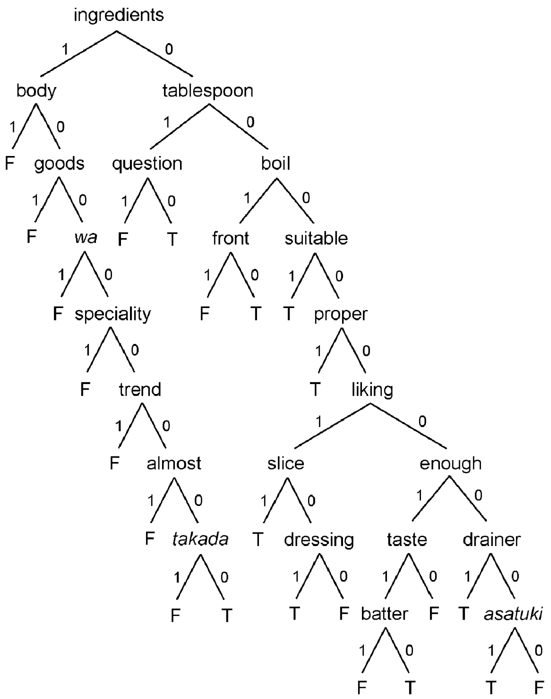

Fig. 6. A decision tree induced from Web documents. (The original keywords are Japanese.)

In information retrieval research, we normally use precision and recall for query evaluation. Precision is the ratio of the number of relevant documents to the number of returned documents and recall is the ratio of the number of relevant documents returned to the number of relevant documents in existence. High precision means that the retrieved results contain few irrelevant pages and high recall means that few relevant pages are missed from the results. In this section, precision P and recall R are defined over validation set $\mathcal{D}_{validation}$ as follows:

$$
P = \frac{|\mathcal{D}_{domain} \cap \mathcal{D}_{Boolean}|}{|\mathcal{D}_{Boolean}|}
$$

$$
R = \frac{|\mathcal{D}_{domain} \cap \mathcal{D}_{Boolean}|}{|\mathcal{D}_{domain}|},
$$

where $\mathcal{D}_{domain}$ is the set of relevant documents classified by humans and $\mathcal{D}_{Boolean}$ is the set of documents that the Boolean expression identifies as being relevant in the validation set.

In our case, we use the harmonic mean of precision P and recall R [18]

$$
F = \frac{2}{\frac{1}{R} + \frac{1}{P}},
$$

as the criterion for removal. High values of F occur only when both precision P and recall R are high. The partial differentials of F with respect P and R are:

$$
\frac{\partial F}{\partial P} = \frac{2(\frac{1}{P})^2}{(\frac{1}{R} + \frac{1}{P})^2}
$$

[3] The original keywords are Japanese.

$$\frac{\partial F}{\partial R} = \frac{2(\frac{1}{R})^2}{(\frac{1}{R} + \frac{1}{P})^2}.$$

Therefore, when $P > R$, $\frac{\partial F}{\partial R} > \frac{\partial F}{\partial P}$ and the improvement of R has greater contribution to F than the improvement of P and vice versa when $R > P$. In other words, the harmonic mean weights low values more heavily than high values. This means that if we simplify keyword spices in the way that results in a high value of F, we can obtain the keyword spices that are well-balanced in terms of precision and recall.

We can also consider the weighted harmonic mean of recall and precision as follows:

$$F_\beta = \frac{1 + \beta^2}{\frac{\beta^2}{R} + \frac{1}{P}},$$

where β is a parameter to specify the relative importance of P or R. This is the complement of van Rijsbergen's E measure [19]: $E = 1 - F_\beta$. For $\beta > 1$, R is of more importance than P and, for $\beta < 1$, P is more important than R. For $\beta = 1$, F_β is equal to the normal harmonic mean F. By changing the value of β, we can control the trade off between recall and precision according to the characteristics of target domains.

In the first stage of simplification, we treat each conjunction as if it is an independent Boolean expression. We calculate the conjunction's harmonic mean of recall and precision over the validation set. For each conjunction, we remove the keyword (Boolean literal) if it results in the maximum improvement in this harmonic mean and repeat this process until there is no keyword that can be removed without decreasing the harmonic mean.

When we remove a keyword from a conjunction, the recall either increases or remains unchanged. Before the simplification, each conjunction usually yields high precision and low recall. Accordingly, we can remove the keywords that result in an improvement in recall in exchange for some decrease in precision because the harmonic mean weights lower recall values more heavily. The removal of the keywords from a conjunction by the harmonic mean may appear to cause some problems. If the initial conjunction contains only a few relevant documents, the algorithm makes conjunctions that contain very large numbers of irrelevant documents. However, we can remove such conjunctions from the keyword spices by the algorithm for simplifying a disjunction as is described below.

In the second stage of simplification, we try to remove conjunctions from the disjunctive normal form h to simplify the keyword spices. We remove the conjunctions so as to maximize the increase in harmonic mean F. We repeat this process until there is no conjunction that can be removed without decreasing the harmonic mean F.

After the first stage of simplification, each conjunction is generalized and changed to cover many examples. As a result, the recall of h becomes rather high, but some conjunctions may cover many irrelevant documents. We can remove the conjunctions that cause a large improvement in the precision with a slight reduction in recall. Those components that cover many irrelevant documents are removed in this stage because the other conjunctions cover most of the relevant documents and the removal of the defective conjunctions does not cause a large reduction in recall. This yields simple keyword spices composed of a few conjunctions.

Please note the pruning method proposed does not necessarily realize the global optimum of F. It is based on local search because finding the Boolean expression that achieves the global optimum is hard. If we prune a tree itself, the pruning process is easily trapped in local optima because nodes near the root are more difficult to remove than the nodes near the leaves. Converting the tree to rules democratizes the removal of any keyword in the rules and reduces the risk of being trapped in poor local optima.

The above simplification processes yield h as the keyword spices for this domain. Our algorithm for extracting keyword spices is summarized in Fig. 7.

4 EXPERIMENTS

4.1 Extracting Keyword Spices

In this section, we present some experimental results of our keyword spice method in the domains of recipes, restaurants, and used cars. As mentioned in Section 2, we gathered sample pages of the recipe domain that contained the names of ingredients in Japanese. For the restaurant domain, we used the names of menu items such as "steak," "pizza," "sushi," etc. to collect sample pages. We used the names of cars for the used car domain. For each domain, we selected 10 initial keywords for sampling. The keywords used to collect sample Web pages are listed in Table 1. We used a general-purpose Japanese search engine, Goo,[4] to find and download Web pages containing the above input keywords. We collected 200 sample pages for each initial keyword. Thus, there was a total of 2,000 sample pages for each domain. We examined the pages collected and classified them as either relevant or irrelevant by hand. We randomly divided these pages into two disjoint sets of the same size, the training set with 1,000 pages for generating the initial decision tree and the validation set with 1,000 pages for simplifying the tree. In splitting the collected documents into the training set and validation set, we paid no attention as to which keywords were input. Thus, each set was randomly composed of documents containing the input keywords.

For the recipe domain, we performed five trials in which the sample pages were split randomly in this fashion. Table 2 shows the pruning results after each step in the recipe domain. In the early steps, induced trees are very large and, after translating trees to conjunctions, we have more than 10 conjunctions; the number of keywords in these conjunctions exceeded 62. This number is too large to permit entry into commercial search engines. After Step 5, the number of keywords was reduced to one third. Step 6 removed redundant conjunctions and keyword number was reduced again to 3 or 4. This number of keywords can be accepted by commercial search engines.

Table 3 shows the keyword spices discovered for a recipe search engine. Different trials yielded different keyword spices, but they are composed of similar keywords. We used the keyword spice of the first trial in subsequent experiments.

Table 4 shows how the value of β affects the extraction results when we use weighted harmonic mean of recall and

[4] http://www.goo.ne.jp

```
0. Generate sample keywords according to some estimate of the user's input and collect
   web pages that contain these keywords and classify them as either positive or negative
   examples by hand.
1. Split the examples into two disjoint subsets, the training set, $\mathcal{D}_{training}$ (for generating
   the initial decision tree) and the validation set, $\mathcal{D}_{validation}$ (for simplifying the tree).
2. Make the initial decision tree from $\mathcal{D}_{training}$ without any pruning technique.
3. Convert the tree so learned into a set of positive conjunctions by creating one con-
   junction for each path from the root node to each leaf node: this classifies positive
   examples.
4. For each conjunction $c$ in $h$ do
       Repeat
           • Remove the keyword (Boolean literal) from the conjunction $c$ that results
             in the maximum increase in the harmonic mean
                 $F_c = \frac{2}{\frac{1}{R_c} + \frac{1}{P_c}}$
             of precision measure $P_c$ and recall measure $R_c$ of $c$ over the validation set.
       Until there is no keyword that can be removed without decreasing $F_c$.
   End
5. Make a disjunctive normal form of Boolean expression $h$ by making a disjunction of all
   positive conjunctions.
6. Repeat
       • Remove the conjunctive component from the disjunctive normal form $h$ that
         results in the maximum increase in the harmonic mean
             $F_h = \frac{2}{\frac{1}{R_h} + \frac{1}{P_h}}$
         of precision measure $P_h$ and recall measure $R_h$ of $h$ over the validation set.
   Until there is no conjunction that can be removed without decreasing $F_h$.
7. Return $h$
```

Fig. 7. The keyword spice extraction algorithm.

precision as a criterion for simplification. Precision and recall here are evaluated over the validation sets. We can control the trade off between precision and recall by changing the value of β. When we give weight to precision, the number of keywords in the keyword spice grows large. On the other hand, when we attach importance to recall,

TABLE 1
Keywords Used to Collect Sample Web Pages
(The Original Keywords Are Japanese)

Domain	Keywords
Cooking Recipe	beef chicken paprika potato pumpkin radish salmon tofu tomato whitefish
Restaurant	udon curry pizza hamburger noodle sushi cutlet steak grill sukiyaki
Used Car	Celsior Civic Golf Legacy Odyssey Silvia Skyline Stepwgn Wagon-R

TABLE 2
Pruning Results in the Cooking Recipe Domain

		Trials				
		1	2	3	4	5
Initial	Conjunctions	10	15	13	15	10
	Keywords	65	89	76	87	62
Step 5	Conjunctions	10	15	13	15	10
	Keywords	17	32	26	34	19
Step 6	Conjunctions	2	2	2	2	2
	Keywords	4	3	4	4	4

TABLE 3
Extracted Keyword Spices for the Cooking Recipe Domain (The Original Keywords Are Japanese)

Trials	Keyword Spices
1	(ingredients $\wedge$ ¬speciality $\wedge$ ¬goods) $\vee$ tablespoon
2	(ingredients $\wedge$ ¬Tokyo) $\vee$ tablespoon
3	(ingredients $\wedge$ ¬goods $\wedge$ ¬results) $\vee$ tablespoon
4	(ingredients $\wedge$ ¬outbreak $\wedge$ ¬goods) $\vee$ flavor
5	(ingredients $\wedge$ ¬season $\wedge$ ¬description) $\vee$ tablespoon

simpler keyword spices are extracted. The keyword spices extracted for the domain of restaurants and used cars when $\beta = 1$ are listed in Table 5. These keyword spices are used in the experiments.

We can notice some interesting characteristics by observing the keyword spices for various domains. Words that directly specify the domain, such as "recipe," "restaurant," or "car," do not appear in the keyword spices. Instead, they contain words that are used to describe the contents of the domain pages.

Words that are used as a negative literal have the role of excluding the pages of other domains. It may seem strange that a word "Kanto," which is a region in Japan, appears in the keyword spice for finding restaurants. In fact, the pages containing names of menus include many pages for online shopping and the word "Kanto" is used to describe delivery charges for each region in these pages. Therefore, using "Kanto" with negation is useful for removing these pages, which are irrelevant to restaurants.

4.2 Evaluation Using a General-Purpose Search Engine

Here, we present realistic tests conducted using an external commercial search engine. To confirm the effectiveness of the keyword spice, we compared the precision values of the results of queries containing only keywords to those of queries with keyword spices. Here again, we used Goo for the evaluation. The evaluation used keywords that were not used to generate the keyword spices.

Table 6 presents the precision of test queries in each domain. The precision is domain dependent, but the queries with keyword spice have much higher precision than those without keyword spices.

Fig. 8 compares the precision values of the queries containing only keywords to those of the queries with keyword spices as ranked by the search engine Goo. According to the ranking algorithm of the general-purpose search engine, the precision fluctuates as the number of pages viewed increases, but the precision of queries with keyword spices is always higher than that of queries without keyword spices.

You might raise the question of what will happen if we enter the query "beef recipe" to a general-purpose search engine. Of course, users are likely enter the query simply conjoined with the name of the domain. Table 7 shows the precision values of the sample queries conjoined with "recipe." Precision values of these queries fall behind that of queries with keyword spices because Web pages containing the keyword "recipe" do not always describe recipes. We also compared the coverage of these two types of queries. Most search engines show the total number of pages that matched the query. We can calculate the estimated number of all relevant documents that matched the query by multiplying the number of matched documents by the average precision of the query. The ratio of relevant pages searchable with the name of the domain and relevant pages searchable with keyword spices is also presented in Table 7. The query with the keyword "recipe" finds fewer relevant Web pages than the query with keyword spice. When we extract keyword spices, we consider recall as well as precision and make a disjunction of several conjunctions and this results in the broader coverage of keyword spices. Of course, there may be some bias in missed pages. For example, pages with neither "tablespoon" nor "ingredients" cannot be retrieved with our method. This is a limitation of our method; using small keyword spices that can be accepted by search engines.

Please note that the keyword spice itself does not always classify domain pages. For example, when we enter the keyword spice for restaurants into a general-purpose search engine, the results contains only 21 restaurant pages in the top 100; the other pages are on various shops or public facilities. This is natural because these pages also contain keywords that are used to describe their business hours. This phenomenon shows another characteristic of keyword spices; they are truly effective only if used together with the user's input keyword.

4.3 Comparison to the Filtering Model

Boolean expressions output from the algorithm of Fig. 7 also can be used as a domain filter which filters out irrelevant pages from the results of a general-purpose search engines, as described in Fig. 1. Table 8 shows the precision values of the sample queries in the filtering model. They are

TABLE 4
Extraction Results with Different Values of β

β	Cooking Recipe			Restaurant			Used Car		
	Precision	Recall	Keywords	Precision	Recall	Keywords	Precision	Recall	Keywords
0.5	0.941	0.928	5	0.840	0.655	9	0.887	0.811	8
1.0	0.919	0.945	4	0.785	0.659	3	0.806	0.783	3
2.0	0.907	0.938	3	0.295	0.988	3	0.601	0.840	4

TABLE 5
Extracted Keyword Spices for the Other Domains (The Original Keywords Are Japanese)

Domains	Keyword Spices
Restaurant	(open ∧ ¬Kanto) ∨ closed
Used Car	(inspection ∧ mileage) ∨ inspected

comparable to those in the keyword spice model because both models use the same Boolean expression.

One advantage of the keyword spice model over the filtering model is that the former can find more relevant pages than the latter when we use a real search engine. Most commercial search engines limit the number of Web pages returned to the user. For example, Google, Altavista, and Goo return only 1,000 pages even if more pages match the query. The upper limit of the number of relevant pages returned by the filtering model is the number of relevant pages among these 1,000 pages. Therefore, if the precision of an initial query input by the user is low, the number of relevant pages that can be retrieved by the filtering model becomes small.

Table 9 presents the numbers of relevant pages returned by the keyword spice model and by the filtering model. If the precision of the initial query is low, such as the case of "shrimp," the filtering model can return only a small number of relevant pages. On the other hand, the keyword spice model can return many more relevant pages. Since the query is modified so as to increase the ratio of relevant pages before it is entered to the search engine in the keyword spice model, the number of relevant pages returned approaches the limit of the search engine.

Another advantage of the keyword spice model is its efficiency when searching. As we mentioned in Section 1, the filtering model needs to download all pages returned by the general-purpose search engine and examine the contents of these pages to check whether they match with the Boolean expression. On the other hand, the keyword spice model dispenses with these processes. This is a significant benefit since downloading Web pages is a time consuming task. When the rate of pages that pass the filter among the results of the initial query is r, the system needs to download $\frac{1}{r}$ pages to present one final result to the user. Usually, the value of r gets smaller as the precision of the initial query falls. Table 10 shows the number of downloads required to present one result to the user in the filtering model. For example, in the case of "shrimp," the filtering model has to download five pages to obtain one result and so is quite inefficient.

TABLE 6
Precision of Test Queries Submitted to a General-Purpose Web Search Engine

Domain	Query	Top 20		Top 100	
		without spice	with spice	without spice	with spice
Cooking Recipe	pork	0.60	1.00	0.54	1.00
	spinach	0.40	1.00	0.27	1.00
	shrimp	0.00	1.00	0.03	0.98
Restaurant	eel	0.20	0.95	0.37	0.90
	spaghetti	0.50	0.95	0.24	0.91
	chaotzu	0.30	0.75	0.29	0.87
Used Car	Accord	0.15	0.75	0.07	0.65
	Corolla	0.00	1.00	0.02	0.50
	Crown	0.00	0.95	0.02	0.55

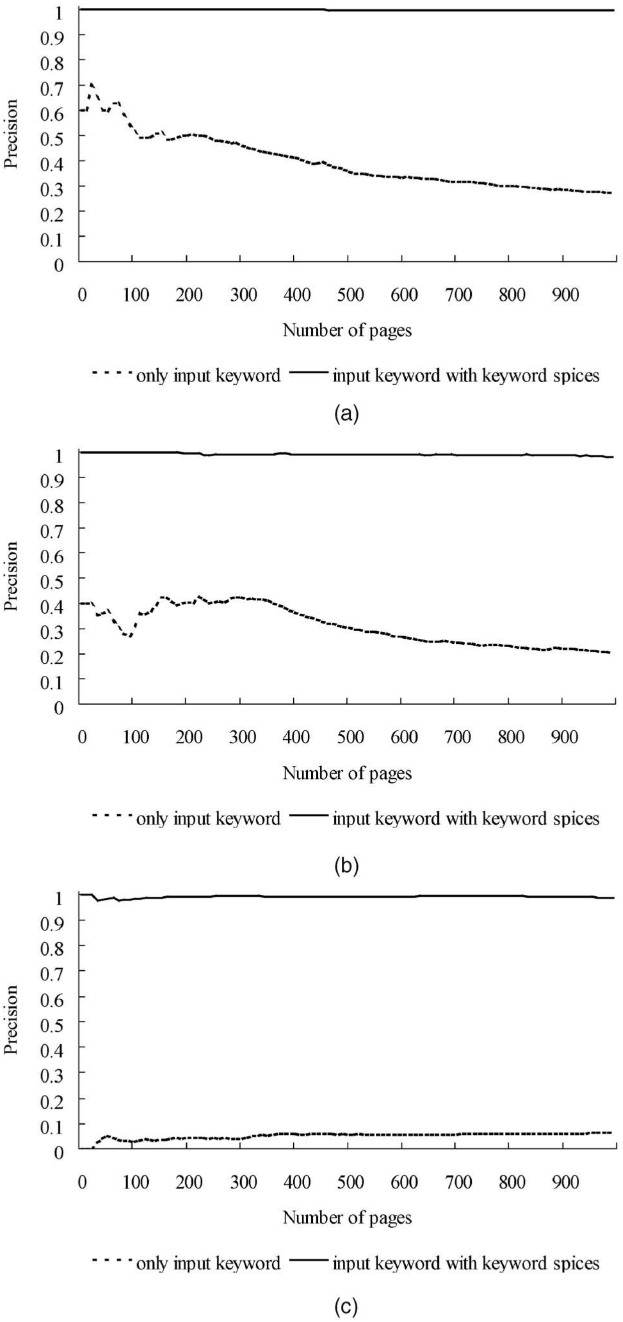

Fig. 8. Precision of queries in the recipe domain. (a) Query "pork" forwarded to goo, (b) Query "spinach" forwarded to goo, (c) Query "shrimp" forwarded to goo.

TABLE 7
Performances of the Queries Conjoined with the Name of the Domain

Query	Precision		Ratio of searchable
	Top 20	Top 100	relevant pages
pork	0.75	0.67	0.49
spinach	0.85	0.81	0.60
shrimp	1.00	0.72	0.54

TABLE 8
Precision of Test Queries in the Filtering Model

Query	Top 20	Top 100
pork	1.00	0.97
spinach	0.90	0.96
shrimp	0.80	0.95

TABLE 9
Number of Relevant Pages Returned

Query	Keyword Spice Model	Filtering Model
pork	947	499
spinach	930	397
shrimp	938	170

TABLE 10
Number of Downloads Required to Present One Result in the Filtering Model

Query	Downloads required for one result
pork	1.90
spinach	2.33
shrimp	5.56

5 Related Work

The keyword spice model, in which keywords are added to the user's original query by the system, can be regarded as a kind of query modification or query refinement. In *relevance feedback* [20], the system presents the results for the initial query and the user judges each document relevant or irrelevant. The system then modifies the query based on the user's feedback. By repeating this process, the query is incrementally refined so as to obtain more relevant results. This method modifies a query so as to meet a specific information need of a specific user. In contrast, our method finds query extensions that are effective for various queries in a specific domain in advance and instantly returns relevant documents to the user without any interaction.

Improving domain-specific (or category specific) Web search by query modification is also described in [21] and they extracted query modification rules for finding personal homepages and calls for papers. They formed the training set by combining the positive examples collected by humans from many resources and negative examples from logs of a search engine. To solve the problem of the mismatch between the training examples and the target search engines, they prepared a classifier based on support vector machines (SVMs) and a set of query modifications (combinations of one or two features) separately and then reranked the query modifications according to the classification results of test queries. Our approach differs from theirs in its selection of the learning method and the strategy of collecting training examples. We adopted decision tree learning and decision trees, which makes it easier to convert the classifier into query modifications. We collected sample pages from the general-purpose search engine that we also use when we build a domain-specific one according to the assumption of user's input. Thus, the extracted query modifications can be directly entered into the search engine without any change.

6 Future Work

We need training examples classified by humans to discover keyword spices and this is the major cost in our method. In this section, we present several future research directions, including an initial trial, for reducing this cost.

6.1 Using a Web Directory as a Source for Training Examples

One way to omit the cost of labeling training examples is using pages already classified. In the Web, there are Web directories such as Yahoo![5] or Open Directory[6] in which many pages are already classified hierarchically according to their topics. We can use Web pages classified in a certain category in a directory as positive examples and pages in other categories as negative examples. We have conducted experiments to use pages from Yahoo! as training examples. However, the performance of the extracted keyword spices changes significantly among domains. The main problems in using Web directories are as follows.

- *Noise in the training set:* Web pages directly linked from the directory do not necessarily contain information useful for classification. For example, Web pages linked from the category of recipes in Yahoo! are usually the top pages of "portal" sites of cooking and do not always contain recipes themselves. We need to traverse several links from these top pages to find the recipe pages. However, the collected pages still contain large numbers of irrelevant pages because a Web site is usually composed of pages with various topics.
- *Bias in the training set:* Pages in a directory are a very small, biased sample of the Web. In our original method, we collected Web pages by submitting keywords to general-purpose search engines. As a result, pages were gathered from diverse Web sites. However, when we collected training examples from a relatively small number of Web sites linked from Yahoo!, the bias in the training set degraded the performance of keyword spices in real settings.

[5] http://www.yahoo.com/
[6] http://dmoz.org/

Using Web pages in a Web directory as training examples for learning query modification for category-specific (domain-specific) Web search was also described in [22]. They used Web pages collected from a Web directory in training without any selection. They reported that the performance for broad categories (domains) is better then that for narrow categories because more specific categories are more difficult to classify. We think the problems mentioned above must be solved if we are to achieve more precise classification of narrower domains.

6.2 Learning Classifiers from Partially Labeled Data

Another way to ease the cost of labeling training examples is employing algorithms that can learn classifiers from partially labeled data.

Nigam et al. proposed an algorithm that improves the accuracy of classifiers by augmenting a small number of labeled documents with a large number of unlabeled documents [23]. By applying this algorithm to partially labeled examples, a small number of pages are labeled and the others are left unlabeled; this can reduce the cost of preparing training examples.

Liu et al. proposed another algorithm that can learn from labeled positive examples and unlabeled examples [24]. No labeled negative examples are required in their method. Applying this algorithm to positive examples found in a Web directory or a link list and unlabeled examples collected randomly from the Web will cut the cost of labeling.

Both methods above are based on naive Bayesian classifiers, thus some method for converting classifiers into Boolean expressions is needed when we apply these techniques to our keyword spice model.

7 Conclusion

Domain-specific Web search engines are effective tools for reducing the difficulty in acquiring information from the Web. We have proposed a novel method for domain-specific Web searches that is based on the idea of keyword spices: Boolean expressions that are added to the user's input query to improve the search performance of commercial search engines. The keyword spice method enables us to build domain-specific search engines at low cost without human expertise or specific facilities. We described a practical learning algorithm to extract powerful but comprehensive keyword spices. This algorithm turns complicated initial decision trees into small Boolean expressions that can be accepted by search engines. We have extracted keyword spices for the domains of recipes, restaurants, and used cars. Experiments showed the effectiveness of keyword spices in these domains. The only human intervention needed in our method is classifying the training examples. We have also presented some future research directions for reducing the cost of preparing training examples.

Acknowledgments

The authors would thank Takayuki Yoshizumi of IBM Tokyo Research Laboratory, Teruhiro Yamada of SANYO Electric, and Yasuhiko Kitamura of Osaka City University. This research is being conducted in cooperation with these people. This research was partially supported by the Laboratories of Image Information Science and Technology and by the Ministry of Education, Science, Sports, and Culture, Grant-in-Aid for Scientific Research(A), 11358004, 1999-2001.

References

[1] D. Butler, "Souped-Up Search Engines," *Nature,* vol. 405, pp. 112-115, 2000.

[2] A. McCallum, K. Nigam, J. Rennie, and K. Seymore, "A Machine Learning Approach to Building Domain-Specific Search Engines," *Proc. 16th Int'l Joint Conf. Artificial Intelligence (IJCAI-99),* pp. 662-667, 1999.

[3] W.W. Cohen, "A Web-Based Information System that Reasons with Structured Collections of Text," *Proc. Second Int'l Conf. Autonomous Agents (Agents '98),* pp. 116-123, 1998.

[4] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery, "Learning to Extract Symbolic Knowledge from the World Wide Web," *Proc. 15th Nat'l Conf. Artificial Intelligence (AAAI-98),* pp. 509-516, 1998

[5] O. Etzioni, "Moving Up the Information Food Chain: Deploying Softbots on the World Wide Web," *Proc. 13th Nat'l Conf. Artificial Intelligence (AAAI-96),* pp. 1322-1326, 1996.

[6] E. Selberg and O. Etzioni, "The MetaCrawler Architecture for Resource Aggregation on the Web," *IEEE Expert,* vol. 12, no. 1, pp. 11-14, 1997.

[7] J. Shakes, M. Langheinrich, and O. Etzioni, "Dynamic Reference Sifting: A Case Study in the Homepage Domain," *Proc. Sixth Int'l World Wide Web Conf. (WWW6),* pp. 189-200 1997.

[8] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval.* Addison-Wesley, 1999.

[9] T.M. Mitchell, *Machine Learning.* McGraw-Hill, 1997.

[10] D.D. Lewis and M. Ringuette, "A Comparison of Two Learning Algorithms for Text Categorization," *Proc. Third Ann. Symp. Document Analysis and Information Retrieval (SDAIR-94),* pp. 81-93, 1994.

[11] T. Joachims, "A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization," *Proc. 14th Int'l Conf. Machine Learning (ICML '97),* pp. 143-151, 1997.

[12] T. Joachims, "Text Categorization with Support Vector Machines: Learning with Many Relevant Features," *Proc. 10th European Conf. Machine Learning (ECML-98),* pp. 137-142, 1998.

[13] S. Oyama, T. Kokubo, T. Ishida, T. Yamada, and Y. Kitamura, "Keyword Spices: A New Method for Building Domain-Specific Web Search Engines," *Proc. 17th Int'l Joint Conf. Artificial Intelligence (IJCAI-01),* pp. 1457-1463, 2001.

[14] J.R. Quinlan, "Induction of Decision Trees," *Machine Learning,* vol. 1, pp. 81-106, 1986.

[15] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, *Classification and Regression Trees.* Wadsworth, 1984.

[16] J. Mingers, "An Empirical Comparison of Selection Measures for Decision-Tree Induction," *Machine Learning,* vol. 3, pp. 319-342, 1989.

[17] J.R. Quinlan, *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[18] W.M. Shaw Jr., R. Burgin, and P. Howell, "Performance Standards and Evaluations in IR Test Collections: Cluster-Based Retrieval Models," *Information Processing & Management,* vol. 33, no. 1, pp. 1-14, 1997.

[19] C.J. van Rijsbergen, *Information Retrieval.* Butterworths, 1979.

[20] G. Salton and C. Buckley, "Improving Retrieval Performance by Relevance Feedback," *J. Am. Soc. Information Science,* vol. 41, no. 4, pp. 288-297, 1990.

[21] E. Glover, G. Flake, S. Lawrence, W.P. Birmingham, A. Kruger, C.L. Giles, and D. Pennock, "Improving Category Specific Web Search by Learning Query Modifications," *Proc. 2001 Symp. Applications and the Internet (SAINT 2001)* pp. 23-31, 2001.

[22] S.M. Pahlevi and H. Kitagawa, "Taxonomy-Based Adaptive Web Search Method," *Proc. Third IEEE Int'l Conf. Information Technology: Coding and Computing (ITCC 2002)* pp. 320-325, 2002.

[23] K. Nigam, A.K. Mccallum, S. Thrun, and T. Mitchell, "Text Classification from Labeled and Unlabeled Documents Using EM," *Machine Learning,* vol. 39, no. 2/3, pp. 103-134 2000.

[24] B. Liu, W.S. Lee, P.S. Yu, and X. Li, "Partially Supervised Classification of Text Documents," *Machine Learning: Proc. 19th Int'l Conf. (ICML 2002),* pp. 387-394, 2002.

Satoshi Oyama received the BEng, MEng, and PhD degrees from Kyoto University, Kyoto, Japan, in 1994, 1996, and 2002, respectively. He is currently a faculty member in the Department of Social Informatics, Graduate School of Informatics, Kyoto University. He was affiliated with NTT from 1996 to 1998. He was a research fellow of the Japan Society for the Promotion of Science from 2001 to 2002. His research interests include machine learning, data mining, and information retrieval.

Takashi Kokubo received the BEng degree from the Department of Information Science, Faculty of Engineering, Kyoto University, Kyoto Japan, in 1999. He received the MEng degree from the Department of Social Informatics, Graduate School of Informatics, Kyoto University in 2001. He is currently affiliated with NTT DoCoMo, Inc. His research interests include agent technologies and the Semantic Web.

Toru Ishida received the BEng, MEng, and DEng degrees from Kyoto University, Kyoto, Japan, in 1976, 1978, and 1989, respectively. He is a professor in the Department of Social Informatics, Graduate School of Informatics, Kyoto University. He has been working on parallel, distributed, and multiagent production systems and real-time search for learning autonomous agents. He is currently leading digital cities and intercultural collaboration projects in Kyoto. He is a fellow of the IEEE.