

# Keyword Spices: A New Method for Building Domain-Specific Web Search Engines

Satoshi OYAMA, Takashi KOKUBO\* and Toru ISHIDA

Department of Social Informatics  
Kyoto University, Kyoto 606-8501, Japan  
{oyama, t-kokubo, ishida}@kuis.kyoto-u.ac.jp

Teruhiro YAMADA†

Laboratories of Image Information  
Science and Technology  
yamateru@kuis.kyoto-u.ac.jp

Yasuhiko KITAMURA

Department of Information and  
Communication Engineering  
Osaka City University, Osaka 558-8585, Japan  
kitamura@info.eng.osaka-cu.ac.jp

## Abstract

This paper presents a new method for building domain-specific web search engines. Previous methods eliminate irrelevant documents from the pages accessed using heuristics based on human knowledge about the domain in question. Accordingly, they are hard to build and can not be applied to other domains. The keyword spice method, in contrast, improves search performance by adding domain-specific keywords, called keyword spices, to the user's input query; the modified query is then forwarded to a general-purpose search engine. Keyword spices can be effectively discovered automatically from web documents allowing us to build high quality domain-specific search engines in various domains without requiring the collection of heuristic knowledge. We describe a machine learning algorithm, which is a type of decision-tree learning algorithm, that can extract keyword spices. To demonstrate the value of the proposed approach, we conduct experiments in the domain of cooking. The results confirm the excellent performance of our method in terms of both precision and recall.

## 1 Introduction

The expansion of the Internet and the number of its users has raised many new problems in information retrieval and artificial intelligence. Gathering information from the web is a difficult task for a novice user even if he uses a search engine. The user must have experience and skill to find the relevant pages from the large number of documents returned, which often cover a wide variety of topics. One solution is to build a

domain-specific search engine [McCallum *et al.*, 1999]; an engine that returns only those web pages relevant to the topic in question.

This paper proposes a new method for building domain-specific search engines automatically that it is based on applying machine learning technologies to determine keyword occurrence in web documents.

When one of the authors used a popular Japanese search engine (Goo<sup>1</sup>) to find some beef recipes, he input the obvious keyword *gyuniku* (beef), but only 15 of the top 25 returned pages (60%) pertained to recipes. He hit on the idea of adding another keyword *shio* (salt) to the query, at which point all but one of the returned pages (96%) contained recipes. Surprised at this enhancement, he used the same approach for other ingredients such as pork and chicken... the same improvement in search performance was seen. This indicated the possibility of making a domain-specific search engine simply by adding a few keywords to the user's query and forwarding the modified query to a general-purpose search engine. Our *keyword spice* method is a generalization of this finding.

Several research papers have described domain-specific web search services. A straightforward approach to building a domain-specific web search engine is to make indices to domain documents by running web-crawling spiders that collect only relevant pages. Cora<sup>2</sup> [McCallum *et al.*, 1999] is a domain-specific search engine for computer science research papers. Its web-crawling spiders effectively explore the web by using reinforcement learning techniques. SPIRAL [Cohen, 1998] or WebKB [Craven *et al.*, 1998] 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, domains such as personal home-

\*Presently with NTT Docomo, Inc.

†Presently with SANYO Electric Co., Ltd.

<sup>1</sup><http://www.goo.ne.jp>

<sup>2</sup><http://cora.whizbang.com/>

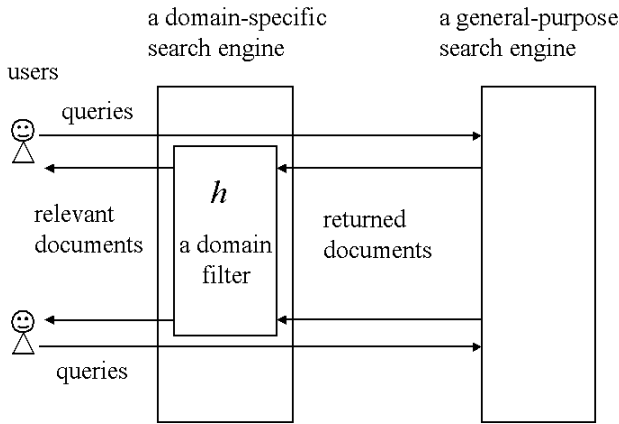


Figure 1: Filtering model for building domain-specific web search engines

pages or cooking pages, which are dispersed across many web sites, are not well handled by spiders since the time and network bandwidth consumed are excessive. Accordingly, such types of systems are suitable only for those domains that have few web sites.

Reusing the large indices of general-purpose search engines to build domain-specific ones is a clever idea [Etzioni, 1996]. For example, Ahoy!<sup>3</sup> [Shakes *et al.*, 1997] is a search engine specialized for finding personal homepages. It forwards the user's query to general-purpose search engines and sifts out irrelevant documents from the returned ones to increase precision by domain-specific filters. We call this the *filtering model* for building domain-specific search engines (Figure 1). Ahoy! has a learning mechanism to assess the patterns of relevant URLs from previous successful searches, but overall accuracy basically depends on human 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 non-relevant ones, has been a major research topic in both information retrieval [Baeza-Yates and Ribeiro-Neto, 1999] and machine learning [Mitchell, 1997].

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. Thus previous methods of text classification cannot be directly applied to the problem of building domain-specific web search engines.

The keyword-spice method considers only those web pages

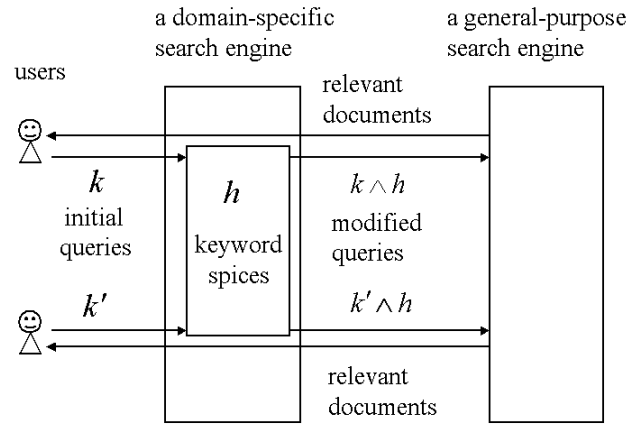


Figure 2: The keyword spice model of building domain-specific web search engines

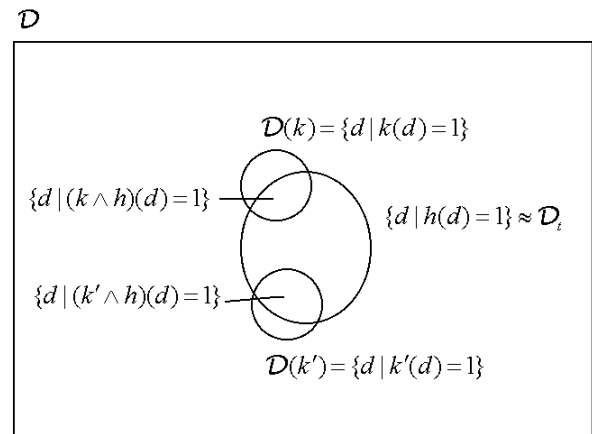


Figure 3: Sampling with input keywords to increase the ratio of positive examples

that contain the user's input query keyword, not all web pages. This eliminates the problem of finding positive examples and enables us to make domain-specific search engines at low cost.

The remainder of this paper is organized as follows: Section 2 presents the idea of building domain-specific search engines using keyword spices. Section 3 describes a machine learning algorithm for discovering keyword spices. Section 4 evaluates our method and our conclusions are given in Section 5.

## 2 The keyword spice model of building domain-specific web search engines

Here we introduce some notations to define the machine learning problem. We let  $\mathcal{D}$  denote the set of all web documents;

<sup>3</sup><http://ahoy.cs.washington.edu:6060/>

$\mathcal{D}_t$  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 equal to finding hypothesis  $h$  that minimizes the error rate

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

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

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), which better classifies the domain documents, and passes the extended query to a general-purpose search engine (Figure 2). This model is just the reverse of the filtering model.

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.

As described in Figure 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 possible to build a domain filter, which is not possible with random sampling.

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

It is rather easy to find good keyword spices for any input keyword  $k$  (for example “beef”). The problem is to find that the keyword spices that provide enough generalization to handle all future user keywords.

We let  $p(k)$  denote the probability of that a user will input keyword  $k$  to a domain-specific search engine. Then

$$\sum_{k \in \mathcal{K}} p(k) \sum_{d \in \mathcal{D}(k)} \frac{1}{|\mathcal{D}(k)|} \delta((k \wedge h)(d), f(d))$$

is the expectation of the error rate when users try to locate domain documents using this system.

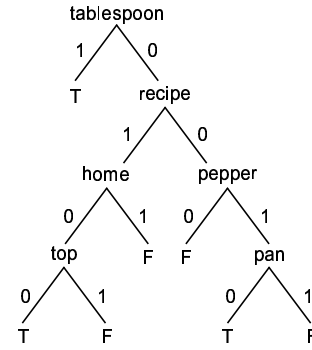


Figure 4: An example of decision tree that classifies documents

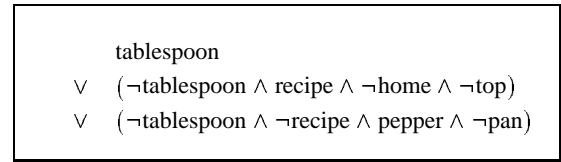


Figure 5: An example of Boolean expression converted from the tree in Figure 4

The Boolean expression that minimizes the above expectation value is the most effective keyword spice. It would be best to make training examples using  $p(k)$  but we do not know  $p(k)$  beforehand. Obviously, we have to start with some reasonable value of  $p(k)$ , and modify the value as statistics on input keywords are collected.

In this paper, we choose several input keyword candidates in the cooking domain. We assume that all candidates have the same probability of occurrence and collect the same number of documents for each keyword as described in Section 4. 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). Figure 4 shows an example of simple decision tree that classifies documents.

The node indicates attribute, the value of branch indicates the value of the attribute, and the leaf indicates the 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 include “tablespoon”, does “recipe”, does

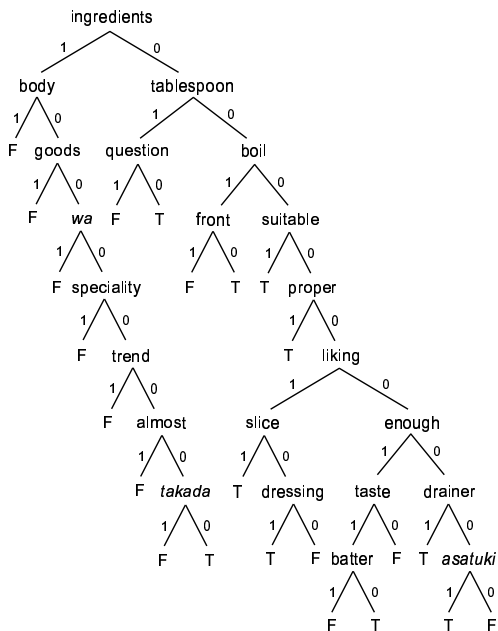


Figure 6: A decision tree induced from web documents

not “home”, and does not “top” belongs to class  $T$ .

We make the initial decision tree using an information gain measure [Quinlan, 1986] for greedy search without using any pruning technique. 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  $\mathcal{D}_{training}$ . Then 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. Figure 5 provides an example of a Boolean expression converted from the tree in Figure 4.

### 3.2 Simplifying Keyword Spices

Figure 6 shows a decision tree induced from collected web document in the experiments described in the next section<sup>4</sup>. Decision trees usually grow very large which triggers the over-fitting problem. Furthermore, too-complex queries cannot be accepted by commercial search engines and so we have to simplify the induced Boolean expression. We developed a two-stage simplification algorithm (described below) that is like rule post-pruning [Quinlan, 1993].

1. For each conjunction  $c$  in  $h$  we remove keywords (Boolean literals) from  $c$  to simplify it.

<sup>4</sup>The original keywords are Japanese.

2. We remove conjunctions from disjunctive normal form  $h$  to simplify it.

In information retrieval research, we normally use precision and recall for query evaluation. Precision is the ratio of 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.

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  $R$  [Shaw Jr. et al., 1997]

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

as the criterion for removal. The harmonic mean weights low values more heavily than high values. High values of  $F$  occur only when both precision  $P$  and recall  $R$  are high. So if we simplify keyword spices in the way that results in high value of  $F$ , we can obtain the keyword spices that are well-balanced in terms of precision and recall.

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 conjunction recall either increases or remains unchanged. Before the simplification, each conjunction usually yields high precision and low recall. Accordingly, we can remove the keyword that results in 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 the 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 the conjunction 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  $F$ .

0. Generate input keywords according to some estimate of distribution  $p(k)$  and collect web pages that contain keyword  $k$  and classify them into positive and 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  $\mathcal{D}_{training}$  using an information gain measure without any pruning technique.
3. Convert the tree so learned into a set of positive conjunctions by creating one conjunction for each path from the root node to each leaf node: this classifies positive examples.
4. Make a disjunctive normal form of Boolean expression  $h$  by making a disjunction of all positive conjunctions.
5. **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**
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  $F_h$ .
- Return**  $h$

Figure 7: The keyword spice extraction algorithm

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 the 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.

After the above simplification processes  $h$  is returned as the keyword spices for this domain. Our algorithm for extracting keyword spices is summarized in Figure 7.

Table 1: Collected web documents in the cooking domain

Keyword	relevant	irrelevant	total
beef	47	153	200
chicken	88	112	200
paprika	79	121	200
potato	49	151	200
pumpkin	42	158	200
radish	64	136	200
salmon	15	185	200
tofu	45	155	200
tomato	33	167	200
whitefish	103	97	200
Total	565	1435	2000

Table 2: Pruning results

		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	onjunctions	2	2	2	2	2
	keywords	4	3	4	4	4

## 4 Evaluation in the Cooking Domain

### 4.1 Experimental Settings

As described in the previous section, we gathered two thousand sample pages of the cooking domain that contained human-entered keywords in Japanese: *gyuniku* (beef), *toriniku* (chicken), *piman* (paprika), *jagaimo* (potato), *kabocha* (pumpkin), *daikon* (radish), *sake* (salmon), *tofu* (tofu), *tomato* (tomato), and *shiromizakana* (whitefish). We used a Japanese general-purpose search engine Goo to find and download web pages containing the above input keywords. We collected two hundred sample pages for each initial keyword. We examined the pages collected and classified them as either relevant or irrelevant by hand (Table 1).

In splitting the collected documents into the training set and validation set, we paid no attention to which keywords were input. Thus each set was randomly composed of documents containing the input keywords. We performed 5 trials in which the sample pages were split randomly in this fashion.

Table 2 shows the pruning results after each step. 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 to 4. This number of keywords can be accepted by commercial search engines.

Different trials yielded different keyword spices. Figure 8

(ingredients  $\wedge$   $\neg$ speciality  $\wedge$   $\neg$ goods)  
 $\vee$  tablespoon

Figure 8: Extracted keyword spices

Table 3: Average precision of the queries over the index of a general-purpose search engine

Query	The input query	The query with keyword spices
pork	0.271	0.995
spinach	0.205	0.979
shrimp	0.063	0.986

Table 4: Estimated recall of the queries with keyword spices over the index of a general-purpose search engine

Query	$Reldoc_{index}$	$Reldoc_{spice}$	Estimated recall
pork	10728	10084	0.940
spinach	4744	4126	0.870
shrimp	5868	5728	0.976

shows, as an example, the keyword spices discovered in the first trial. We used these keyword spices in subsequent experiments.

To conduct realistic tests with external commercial search engines, we choose the keywords of *butaniku* (pork), *horenso* (spinach) and *ebi* (shrimp) which were not used to generate the keyword spices.

## 4.2 Precision

Figure 9 compares the precision values for the queries containing only keywords and the queries with keyword spices for the three input keywords. We checked up to the top 1000 pages as ranked by the search engine Goo. In general, as the number of pages viewed increases, the precision with query-only input decreases, while the precision of queries with keyword spices stays high. Table 3 lists the average precision of the top 1000 returned results. Precision is higher than 97% for all queries.

## 4.3 Estimated Recall

It is easy to achieve high precision if we do not address recall, but keeping both high is rather difficult. The recall of a query is much harder to calculate than the precision because  $\mathcal{D}_t$ , the set of all relevant documents in the web, is unknown. We estimated  $\mathcal{D}_t$  from the results returned from a general-purpose search engine. Most search engines show the total number of documents that matched the query. We can calculate the estimated number of relevant documents in the search engine’s index ( $Reldoc_{index}$ ) by using the average precision of the query for the top 1000 returned documents.

$$Reldoc_{index} \simeq$$

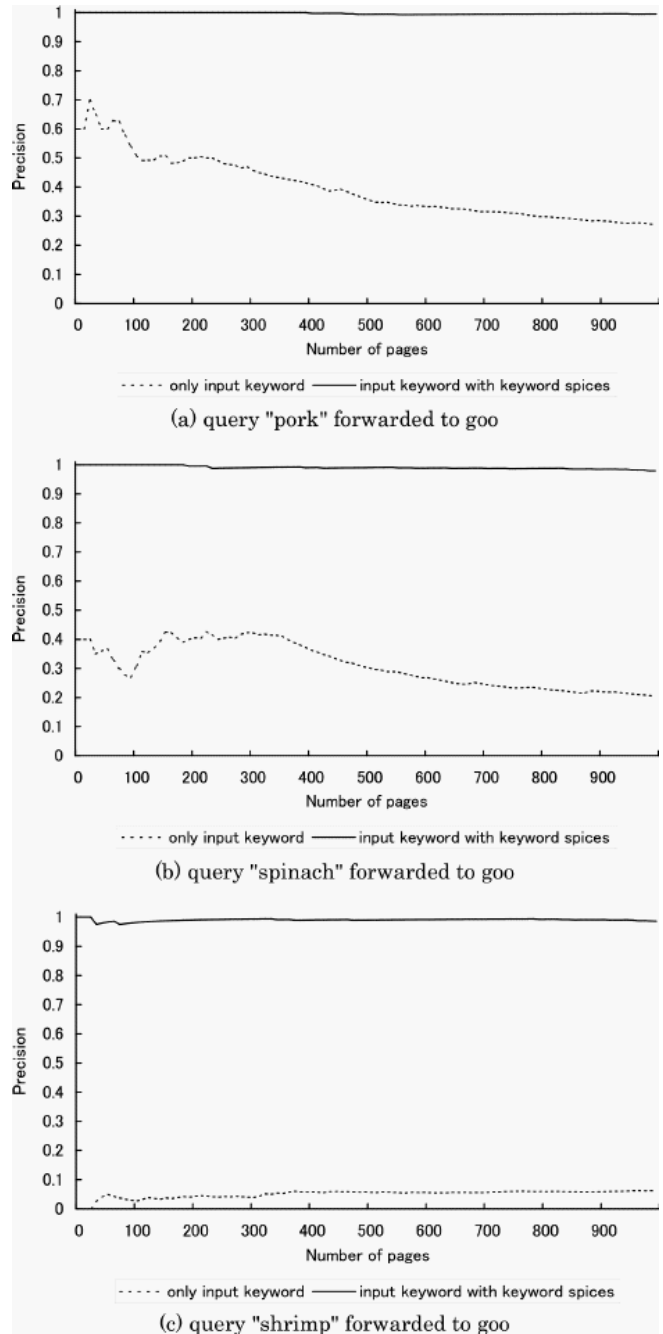


Figure 9: Precision of queries forwarded to a general-purpose search engine

$$\begin{aligned} & \text{(The number of document found with the input query)} \\ & \times \text{(Average precision of the input query)} \end{aligned}$$

The number of relevant documents found with the spice-extended query can be calculated in the same way.

$$Reldoc_{spice} \simeq$$

(The number of document found with the query with keyword spices)

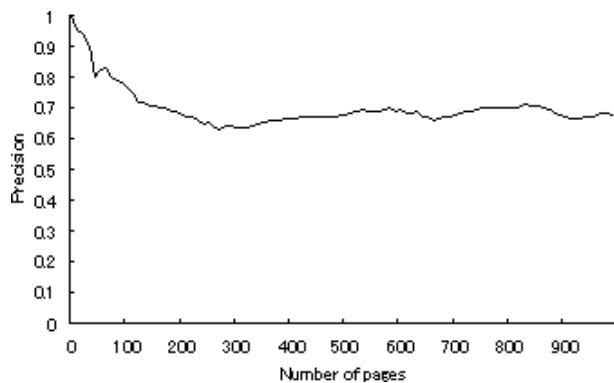


Figure 10: Precision of the query “pork AND salt” forwarded to Goo

× (Average precision of the query with keyword spices)

It is reasonable to use  $Reldoc_{index}$  because we have no consistent way of finding web pages that are not linked to any general-purpose search engine. We estimate the recall of a spice-extended query as follows

$$R \simeq \frac{Reldoc_{spice}}{Reldoc_{index}}$$

Table 4 shows the estimated recall values of different spice-extended queries over the index of Goo. The high value of recall (higher than 87%) indicates that our method filters out only non-relevant documents and does not drop any useful information in the search process.

To compare these results with the example in the Introduction, Figure 10 shows the results of submitting the query “pork AND salt” to Goo. The average precision and estimated recall for the top 1000 returned documents are 0.674 and 0.871, respectively. This shows that our systematic method yields a great improvement in search performance.

## 5 Conclusion

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. This method allows us to build domain-specific search engines without any domain heuristics. We described a practical learning algorithm to extract powerful but comprehensive keyword spices. This algorithm turns complicated initial decision trees to small Boolean expressions that can be accepted by search engines. Our experiments with an external general-purpose search engine yielded good results. For two different keywords in the field of cooking, precision was higher than 97%. High estimated recall (higher than 87%) over the search engine’s index was also confirmed.

We used the domain of cooking as an example, and we are now developing search services for other domains such as restaurant pages and personal homepages.

In this paper, we used input keywords selected by humans to make training examples. To be more comprehensive, we

need some criteria with which input keywords can be selected. As discussed in Section 3, it is sufficient to make examples based on the distribution of user’s input query  $p(k)$ . We are planning to open our recipe search system to the public through the web and we will obtain the value of  $p(k)$  afterwards. In future work we will study how the input keywords used to form the training examples affect the performance of the system.

## Acknowledgments

This research was partially supported by 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.

## References

- [Baeza-Yates and Ribeiro-Neto, 1999] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.
- [Cohen, 1998] William W. Cohen. A web-based information system that reasons with structured collections of text. In *Agents’98*, pages 116–123, 1998.
- [Craven *et al.*, 1998] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew McCallum, Tom Mitchell, Kamal Nigam, and Seán Slattery. Learning to extract symbolic knowledge from the world wide web. In *AAAI-98*, pages 509–516, 1998.
- [Etzioni, 1996] Oren Etzioni. Moving up the information food chain: Deploying softbots on the world wide web. In *AAAI-96*, pages 1322–1326, 1996.
- [McCallum *et al.*, 1999] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. A machine learning approach to building domain-specific search engines. In *IJCAI-99*, pages 662–667, 1999.
- [Mitchell, 1997] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [Quinlan, 1993] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [Shakes *et al.*, 1997] Jonathan Shakes, Marc Langheinrich, and Oren Etzioni. Dynamic reference sifting: a case study in the homepage domain. In *Proceedings of the 6th International World Wide Web Conference (WWW6)*, pages 189–200, 1997.
- [Shaw Jr. *et al.*, 1997] W. M. Shaw Jr., Robert Burgin, and Patrick Howell. Performance standards and evaluations in ir test collections: Cluster-based retrieval models. *Information Processing & Management*, 33(1):1–14, 1997.