Computational Linguistics and Chinese Language Processing
Vol. 13, No. 4, December 2008, pp. 387-404

Feature Weighting Random Forest for Detection of Hidden Web Search Interfaces

Yunming Ye*, Hongbo Li*, Xiaobai Deng* and

Joshua Zhexue Huang+

Abstract

Search interface detection is an essential task for extracting information from the hidden Web. The challenge for this task is that search interface data is represented in high-dimensional and sparse features with many missing values. This paper presents a new multi-classifier ensemble approach to solving this problem. In this approach, we have extended the random forest algorithm with a weighted feature selection method to build the individual classifiers. With this improved random forest algorithm (*IRFA*), each classifier can be learned from a weighted subset of the feature space so that the ensemble of decision trees can fully exploit the useful features of search interface patterns. We have compared our ensemble approach with other well-known classification algorithms, such as SVM, C4.5, Naïve Bayes, and original random forest algorithm (*RFA*). The experimental results have shown that our method is more effective in detecting search interfaces of the hidden Web.

Keywords: Search Interface Detection, Random Forest, Hidden Web, Form Classification

1. Introduction

Hidden Web refers to the Web pages that are dynamically generated from searchable structured or unstructured databases. Different from the Publicly Indexable Web that is accessible through static hyperlinks, the pages in a hidden Web can only be obtained through queries submitted via the search interface to the databases containing data about the hidden Web. Search interfaces are usually encoded as HTML forms that need to be filled out and

* Shenzhen Graduate School, Harbin Institute of Technology, China
E-mail: yeyunming@hit.edu.cn; {wave118, dawndeng}@gmail.com
+ E-Business Technolgy Institute, The University of Hong Kong, Pokfulam Road, Hong Kong
E-mail: jhuang@eti.hku.hk

[Received August 31, 2008; Revised February 23, 2009; Accepted February 16, 2009]

submitted by users to obtain information. On the Web, there are many different HTML forms, and many of them are not search interfaces (He, Patel, Zhang, & Chang, 2007). To extract useful information from hidden Web pages, effectively detecting the search interfaces is an essential step since the interface is the only entrance to the hidden Web. Therefore, we first need to find the entrance to the hidden database. The entrance (*i.e.*, search forms) is mixed with lots of non-search forms in HTML pages. Thus, it is very important to distinguish the two types of forms in order to enable the Hidden Web crawler to locate the entrance and extract information further.

Information extraction from the hidden Web has been a hot research topic (BrightPlanet.com, 2000) since the term "Hidden Web" was first coined (Florescu, Levy, & Mendelzon, 1998). Most previous work, however, has been focused on the problems of automatic query generation (Ntoulas, Zerfos, & Cho, 2005), form filling (Cavelee, Liu, & Probe, 2004), and wrapper generation for extracting structured information (Wang & Lochovsky, 2003), where detecting search interface was performed manually or by some heuristic methods. Using heuristic rules to find search forms is the simplest and most effective method (Raghavan & Garcia-Molina, 2001; Lage, Silva, Golgher, & Laender, 2004). As hidden Web sites adopt different search forms, though, it is time-consuming to compose different rules for different search forms. Employing machine learning and information retrieval techniques to learn classification models from the content of search forms and using the models to classify different forms automatically is a more desirable approach with respect to scalability and robustness. This approach regards search interface detection as a two-class classification problem. One example of this is using decision trees to build form classification models to detect search interface (Cope, Craswell, & Hawking, 2003).

Automatic search interface detection was first explored by Raghavan and Garcia-Molina in their hidden Web crawler HiWE (Hidden Web Exposer) (Raghavan & Garcia-Molina, 2001). Their crawling system used heuristic rules to detect the search entrance to the hidden database. Juliano P. Lage (Lage, Silva, Golgher, & Laender, 2004) used two heuristic rules to perform detection tasks. The first heuristic was the same as HiWE. The second one was to check whether the form contains the "password" HTML element or not. The disadvantage of this method, however, lies in that it does not have auto-leaning capability. Moreover, it is not robust and scalable to diverse hidden Web databases because the rules are too simple to match different form structures.

Cope *et al*. (2003)used a decision tree classification algorithm to detect search interfaces. This method usually generates long rules due to the large size of the feature space in the training set (the number of training samples is too small compared to the number of features). Therefore, it is prone to overfitting, and the classification precision is not satisfying.

Zhang *et al*. (2004) presented a best-effort parsing framework to address the problem of

understanding Web search interfaces. The authors transformed search interfaces into a visual language under the hypothesis that automatic construction of search interfaces is guided by a hidden syntax. This hypothesis enables parsing as a principled framework to understand the semantic model of the visual language. The experimental results testified the effectiveness of their approach.

To summarize, little previous work has addressed the special characteristics of the search interface detection domain, for instance, large and diverse features, small size of training samples with many missing values, *etc*. The high dimension and sparse data of search interfaces present a tricky problem for the traditional single classifier approach. As collecting training samples (*i.e.*, HTML forms) is costly, the training data set is usually small, while the number of features in the learning space is relatively large, due to multi-type features existing in forms. It's difficult for a single classifier to fully exploit the rich feature space (very sparse in the data matrix). For many classification methods, the single classifier tends to be overfitting. To attack this problem, we propose a multi-classifier ensemble approach in this paper.

Our method is based on the random forest algorithm. A random forest model consists of a set of decision trees that are constructed by bootstrapping the training data. In our approach, we develop a weighted feature selection method to select a subset of features for each decision tree in the tree induction process. Classification is made by aggregating predictions of individual decision trees in the forest. Since each classifier is learned from a subset of the feature space, the ensemble approach can fully exploit the useful features in search forms. We have conducted experiments on several real data sets. The experimental results have shown that our random forest approach improves the classification accuracy in search interface detection.

The contributions of this paper can be summarized as follows:

1. We explored the random forest approach to attacking the problem of detecting search interfaces from the sparse feature space of hidden Webs where specific feature extraction and representation techniques were used.
2. We extended the random forest algorithm with a weighted feature selection method to select a subset of features for each decision tree. The new algorithm can automatically remove the noisy features in search forms so that decision tree classifiers can be learned from more representative subsets of the feature space.
3. We conducted experiments on real data sets to compare the improved random forest algorithm with other well-known classification algorithms, such as SVM and C4.5. The experimental results have shown that the new method is more effective in detecting search interfaces of the hidden Web.

The rest of this paper is organized as follows. In Section 2, we formalize the detecting search interface problem as a form classification problem and present the feature extraction techniques. Section 3 describes the improved random forest algorithm for form classification. Experimental results and analysis are presented in Section 4. Section 5 concludes this paper and presents our future work.

2. Feature Extraction for Form Classification

Search interface detection is a process of distinguishing the search forms of the hidden Web from non-search forms. It is a two-class classification problem in machine learning. This section describes how to extract form features from HTML pages and discusses the characteristics of the data matrix for form classification.

2.1 Feature Extraction Rules

An HTML form usually begins with the tag $< FORM >$ and ends with the tag $< /FORM >$. According to this rule, HTML forms can be extracted by searching the $< FORM >$ tag in HTML pages. Each extracted form is a sample in the training set. The features of each form are generated by parsing the corresponding $< FORM >$ HTML block.

HTML forms contain two kinds of features: one is the attributes of forms and elements, and the other is the statistics of those attributes. A form mainly contains four kinds of elements, that is, “INPUT”, “SELECT”, “LABEL” and “TEXTAREA”, which are the children elements of “FORM” element. Element “INPUT” contains several types, such as “text”, “hidden”, *etc*. The hierarchy of a form is shown in Figure 1. All of these elements contain a set of attributes, such as “name”, “value”, “size”, *etc*. The attributes of “form” elements are “method”, “action”, and “name”. Attribute “method” indicates the method for the form to submit query data, such as “POST” or “GET”. Attribute “action” indicates the address of the corresponding server of the form, and attribute “name” indicates the name of the form. Some elements and attributes can be removed because they are not useful for form classification, for instance “option”, “size”, “width”, *etc*. Besides, the statistics about the number of elements or attributes in each element can also be computed as important features.

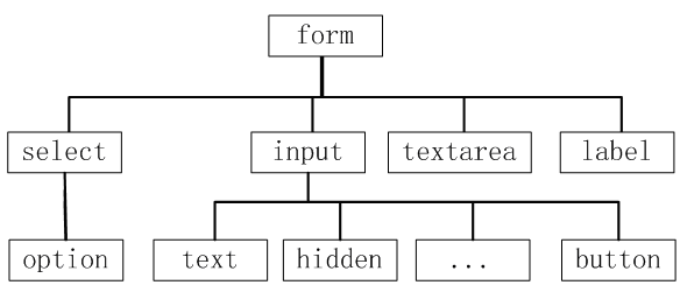

Figure 1. The hierarchy of form elements

Figure 2 shows an example of a search form. The form contains three elements: one "SELECT" element and two "INPUT" elements. There are three attributes in the "SELECT" element : "name" with value "and", "size" with value "1", and "width" with value "50". It also contains several "OPTION" elements. The "size" and "width" attributes, along with the "OPTION" elements are not useful for form classification, so they can be removed. The corresponding HTML codes are shown in Figure 3.

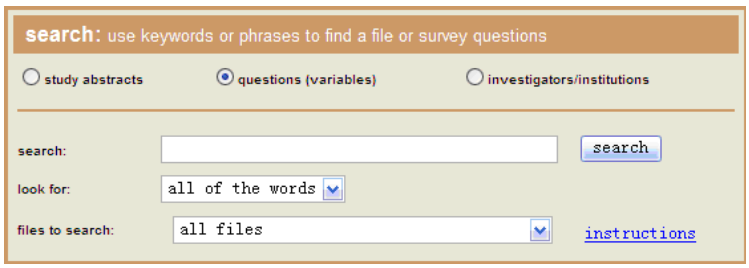

Figure 2. A search form

```
<form action=http://www.thearda.com/cgi-bin/fastsearch.exe>
    . .
  <input size=41 name=searchterms>  
  <input type=submit value=search>
  <select size=1 name=and width="50'>
  <option value=1 selected>all of the words
  <option value=0>any of the words
  <option value=2>exact phrase
  </option></select>
  ...
</form>
```

Figure 3. The HTML codes of the form in Figure 2

Figure 4 shows an example of a non-search form. The corresponding HTML codes are shown in Figure 5. According the feature extraction rules, the useful form elements in this form include "INPUT", "LABEL", and "FORM", which can be used to compose the feature space. Elements "TABEL", "FONT", and "TR" can be removed because they are not useful.

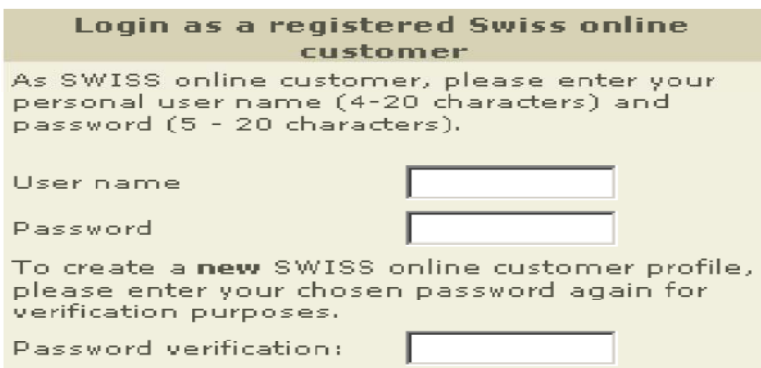

Figure 4. A non-search form

```
<form name="DATES"
action="/servlet/crossair_w_login" method=post>
<table>
<tr>
  <td align=center class="tabletitle" >
  Login as a registered Swiss online customer
</td></tr>
…
<tr>
  <td >User name</td>
  <td><input type=text name="LOGIN"
   value="" size=10 ></td></tr>
<tr>
   <td>Password</td>
   <td><input type=password
    name="PASSWORD"value=""
    size="10"></td> </tr>
…
</table>
</form>
```

Figure 5. The HTML codes of the form in Figure 4

There are some important differences between the features of search form and non-search form. First, the number of "INPUT", "SELECT", and "TEXTAREA" elements in search forms is larger than that in non-search forms. Second, the value of the "method" attribute in "FORM" elements is always set as "POST" in search forms, while it is always set as "GET" in non-search forms. Moreover, the elements' values of search forms often contain some keywords such as "search", "find", or other words that have the same meaning as "search". These differences, however, are not the only decisive factors. There are other features that can be explored by classification algorithms.

According to the major differences, six kinds of rules are used in the feature extraction process as follows:

1. Extract the "name" attribute values from "input", "select", "textarea", and "label" elements;
2. Extract the "value" attribute value from "input", "textarea", and "label" elements;
3. Extract the "name" and "method" attribute values from "form" elements;
4. Extract the words that appear between slashes(/) in the "action" attributes of the "form" elements;
5. Extract the words that appear between slashes(/) in the "src" and "alt" attributes of the "input-image" element;
6. Calculate the number of "input", "select", "label", and "textarea" elements in each form.

The next step is to standardize the value of the features that are extracted from the forms.

First, all strings are transformed into lowercases; then the string type values are aggregated and mapped to specific enumerating values. For instance, the values of "search", "find", or "srch" are mapped to "search".

2.2 The Sparse Data Matrix

The extracted features and the labels of forms are used to compose the data matrix for the classification algorithm. The formalized data matrix is shown in Table 1.

Table 1. The data matrix for form classification

class	t_1	t_2	$\cdots$	t_i	$\cdots$	t_m
c_1	a_{11}	a_{12}	$\cdots$	a_{1i}	$\cdots$	a_{1m}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
c_j	a_{j1}	a_{j2}	$\cdots$	a_{ji}	$\cdots$	a_{jm}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
c_n	a_{n1}	a_{n2}	$\cdots$	a_{ni}	$\cdots$	a_{nm}

The set $T=\{t_1,t_2\cdots,t_m\}$ in Table 1 represents the names of the form features. For form classification, the label of a form is represented as an element in the set $C=\{yes,no\}$, while "yes" indicates that the form is a search interface of hidden Web and "no" indicates a non-search interface. Each row is a sample form. a_{ji} represents the value of feature t_i in the j th form, and c_j indicates the class of the j th form. Table 2 illustrates two examples of a search form and a non-search form as shown in Figure 2 and Figure 4.

The expression of t_i is a four-tuple of "element name"-"type"-"attribute name"-"sequence number". The "element name" contains six values: "FORM", "SELECT", "INPUT", "TEXT AREA", "LABEL", and their statistics. For element, "INPUT", the value of "type" can be "text", "hidden", and so on. "attribute name" has six options: "name", "value", "src", "alt", "method", and "action". Sequence number represents the sequence of the features in the form.

As illustrated in Table 2, the combination of "element name","type","attribute name", and "sequence number" has many unique alternatives. This will result in a high-dimensional feature space for form classification. Furthermore, since each form has just a few features, the data matrix for classification is very sparse and there are many missing values and noisy features. This problem presents a big challenge for search form detection.

Table 2. Two examples of form vectors

class	form-action-1	form-action-2	form-action-3	$\cdots$	input-text-number	input-submit-number
yes	www.thearda.com	cgi-bin	search	$\cdots$	1	1
no	servlet	login	?	$\cdots$	3	0

3. Feature Weighting Random Forest Algorithm

This section presents an improved random forest algorithm, which extends the classical random forest method with a feature weighting technique. We describe the basic random forest classification approach in Subsection 3.1 and our new algorithm in Subsection 3.2.

3.1 Random Forest Algorithm

Random Forest (*RFA*) (Ho, 1998; Breiman, 2001) is an ensemble of unpruned classification or regression trees, which is induced from bootstrapping samples of the training set, using random feature selection in the tree induction process. Prediction is made by aggregating the predictions of the ensemble. Random Forest grows many classification trees. To classify a new object from an input vector, it passes the sample vector to each of the trees in the forest. Each tree gives a classification decision. All the classification results of individual trees are combined to choose the classification having the most votes over all the classification trees in the forest.

Random forest generally exhibits a substantial performance improvement over single tree classifiers, such as CART (Breiman, Friedman, & Olshen, 1984) and C4.5 (Quinlan, 1993). It presents a good solution for classification of sparse data sets. Since basic *RFA* selects features randomly, it's easy to select unimportant or noisy features, especially when there are many noisy features in the training data. This may lead to bad classification results. As discussed in previous sections, the data matrix for form classification contains many missing values. It's necessary to enhance basic *RFA* so that the performance can be improved in search form classification.

3.2 Improved Random Forest with Weighted Feature Selection

Due to the sparse feature space, there are a lot of missing values in the training data set. The features with too many missing values become less important and can be treated as noisy features. Random selection of features often obtains many unimportant or noisy features, which leads to bad trees in the forest. To avoid this, we extend basic *RFA*, using a weighting scheme in feature selection to replace random selection. We use χ^2 statistic to measure the importance of features (Larson, 1982). The χ^2 statistic of a feature A against the class feature is computed as follows.

$$\chi^2 = \sum_{i=1}^{m} \sum_{j=1}^{2} \frac{(o_{ij} - e_{ij})^2}{e_{ij}} \tag{1}$$

where

- m is the number of values in feature A
- o_{ij} is the count of joint event (A_i, C_j), defined as:

$$o_{ij} = count(A = a_i \cap C = c_j) \tag{2}$$

- e_{ij} is the expected value of joint event (A_i, C_j), defined as:

$$e_{ij} = \frac{count(A = a_i) \times count(C = c_j)}{N} \tag{3}$$

where N is the number of the samples in the training data, $count(A = a_i)$ is the number of samples whose value of feature A is a_i, and $count(C = c_j)$ is the number of samples whose value of the class feature is c_j.

An χ^2 statistic weight is calculated for each feature. From the weights, we select only different subsets of features with high weights to build individual decision trees.

Given a set of decision trees built from different subsets of features, we use a probability estimation technique to combine the results of individual classifiers. Assume x is a test instance and is given to each classifier h_j $(j = 1...k)$ for deciding a possible class c_i. The output of an individual classifier can be computed as $P(I(x) = c_i \mid h_j)$. The final classification result is achieved by combining the probability values as:

$$P(I(x) = c_i) = \frac{1}{k} \sum_{j=1}^{k} P(I(x) = c_i \mid h_j) \tag{4}$$

If class c_i has the highest probability, c_i is the class of x. Kittler has provided a more profound explanation of this method (Kittler, Hatef, Duin, & Matas, 1998). The pseudo-code of the new algorithm (*IRFA*) is given in *Algorithm* 1.

Step 1 is to compute a weight for each feature according to (1). Step 2 sorts the features in descending order of feature weights. Step 3 selects n' features from the entire feature set according to a given feature selection rate β. Step 4 learns individual classifiers from the selected training samples (and selected features). Selection of training samples employs the bootstrapping method. The method of sampling without replacement is used to select t features from n' features, where $t = \lfloor \log_2 n + 1 \rfloor$. After each iteration, the learned decision tree classifier will be added to forest M^*. After forest M^* is grown, Step 5 classifies the unlabeled instances based on (4).

Algorithm 1 The pseudo-code of the feature weighting random forest algorithm

Input:

- D : the training database (its number is d),
- N : the features of the forms (its number is n),
- C : the target class attribute $C = yes, no$,
- k : the number of decision trees,
- β : the selection rate of features.

Output: the decision forest M^* .

Process:

1. Compute the weight W based on Formula (1);
2. Sort N on the descending order of weight W ;
3. Let $n' = \lfloor \beta \cdot n \rfloor$, and select $\lfloor \beta \cdot n \rfloor$ features with larger weights as the training samples;
4. *for* $i = 1$ *to* k *do*
   (a) Select d' samples from the training samples by bootstrapping;
   (b) Randomly select t features; where $t = \lfloor \log_2 n' + 1 \rfloor$ and the selection is biased towards the features with larger weights;
   (c) Build a decision tree from the d' samples with selected features;
   (d) Add the learned decision tree to M^* ;
   endfor
5. Using M^* to do classification based on Formula (4).

3.3 The computational complexity

The computational complexity of *RFA* (Breiman, 2001) is $O(ktd \log d)$, where k is the number of decision trees, t is the number of attributes, d is the number of training samples. In *IRFA*, the enumerating number of the feature attribute is constant (Formula (1)). The computational complexity of all feature weights is $O(n)$. Using the bucket sorting method, the weights can be sorted in linear time. Therefore, the computational complexity of the *IRFA* is $O(ktd \log d + \alpha n)$, where $t = \lfloor \log_2 n + 1 \rfloor$. Therefore, the computational complexity of *IRFA* is very close to the complexity of *RFA*.

The computational cost depends on three factors: the number of decision trees k , the number of features n , and the number of training samples d . We will discuss how to select the number of decision trees k and the number of features n to balance between classification accuracy and computational cost in Section 4.

4. Experiments

4.1 Data Sets

We used two Web page collections in our experiments. One was taken from project Metaquerier[1], and the other was created by crawling the website Search Engine Guide[2] with a Web crawler implemented in Java. The two collections represent a pseudo-random crawling of the Web. A HTML parser was developed to extract the HTML forms and their context features from these two collections. The extracted forms were used to compose the final data sets for experiments.

Table 3. The three data sets used in the experiments

	search	non-search	features	content of data sets
Data set 1	46	43	198	Extracted from the website collection crawled from Search Engine Guide by crawler
Data Set 2	65	116	208	Extracted from artificial website collection of Metaquerier project
Data Set 3	51	96	202	The forms are selected from data set 1 and data set 2.

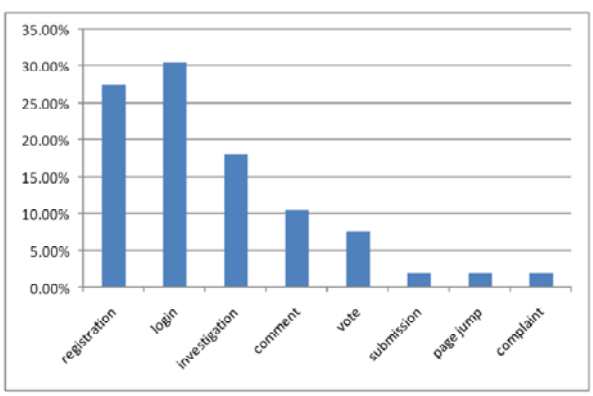

Figure 6. The domain distribution of non-search forms

We manually classified the extracted forms into search forms (*i.e.*, real search interface of hidden Web) and non-search forms. Three classification data sets were constructed from the classified forms, as shown in Table 3. The three data sets have a variety of sample distributions. Data set 1 and Data set 2 cover different domains, while Data set 3 is a mixture

[1]http://metaquerier.cs.uiuc.edu/repository/ training data set

[2]http://www.searchengineguide.com

of the two domains. The three data sets also have different feature types and feature numbers. The average number of features in the data sets is over 200, while the average number of samples is less than 140. The matrices of the three data sets were quite sparse, and the number of features was quite large.

To further test the robustness of our method, the non-search forms in the data sets were made of a variety of forms, including registration forms, login forms, network investigation forms, *etc*. Figure 6 shows the distribution of different non-search forms.

4.2 Comparison Experiments

We first carried out experiments to compare our random forest method with four well-known classification algorithms, *i.e.*, Support Vector Machine (SVM), C4.5, Naïve Bayes,and Random Forest Algorithm (*RFA*) implemented in Weka[3]. We also implemented our algorithm (*IRFA*) as a plug-in of Weka and conducted all experiments in this environment to make a fair comparison. We conducted the standard 10-fold classification experiments on the three data sets. The evaluation metrics used were precision and computation time. The number of trees was set to 100 for *IRFA* and parameter β set to 0.5. The final experimental results are shown in Table 4.

Table 4. The results of comparison experiments

		Bayes	C4.5 Decision Tree	SVM	RFA	*IRFA*
Data Set 1	Precision Time Cost(s)	82.02% <0.005	80.90% 0.05	79.78% 0.52	88.76% 3.16	91.01% 7.08
Data Set 2	Precision Time Cost(s)	83.43% <0.005	88.40% 0.03	82.87% 0.88	91.71% 5.22	92.27% 17.86
Data Set 3	Precision Time Cost(s)	84.35% <0.005	89.79% 0.02	85.71% 0.88	91.84% 3.58	93.88% 15.13

We can see that *IRFA* showed significant improvement over the other four algorithms. The result of C4.5 was better than SVM and Naïve Bayes. This was due to the fact that there were a lot of missing values in the data sets and SVM and Naïve Bayes did not perform well in this kind of sparse data. The high dimensionality in the training sets, however, causes an overfitting problem to C4.5 because the single decision tree could become very complex. *RFA* and *IRFA* can avoid this problem by selecting different subsets of features to build individual decision trees. Compared with *RFA*, *IRFA* uses features that are more correlated to the class label feature, so the accuracy of each individual tree is improved. Therefore, our method got better performance than *RFA*. Since *IRFA* needed to compute the χ^2 values for features, it

[3]http://www.cs.waikato.ac.nz/ml/weka/

took more computational time. This extra overhead, however, is worthwhile and acceptable in real applications because the training process is offline and not executed frequently.

4.3 Selection of the Number of Features

When building each decision tree, *IRFA* only selects a subset of the original features. The number of selected features is controlled by the selection rate β. Different selection rates result in different classification precisions and computational costs. We carried out experiments on the three data sets with $\beta = 0.1,\ 0.2,\ldots,\ 1.0$. Figure 7 plots selection rate β against precision, while Figure 8 is β against computational cost.

Figure 7 shows that when $\beta < 0.5$, the precision increases greatly as the selection rate increases. The reason is that a larger selection rate increases the number of features to be selected. When $0.5 \le \beta \le 0.8$, the classification performance becomes relatively stable. This means that the forest has selected enough discriminative features. When $\beta > 0.8$, the precision will decrease as the selection rate increases. This can be explained by the idea that having too large a selection rate will increase the possibility of selecting noisy features. Most experiments have shown that $\beta = 0.5$ was a good setting.

Figure 8 shows that the computational time of *IRFA* increases linearly as the feature selection rate increases. This property indicates that *IRFA* is scalable to large high-dimensional data.

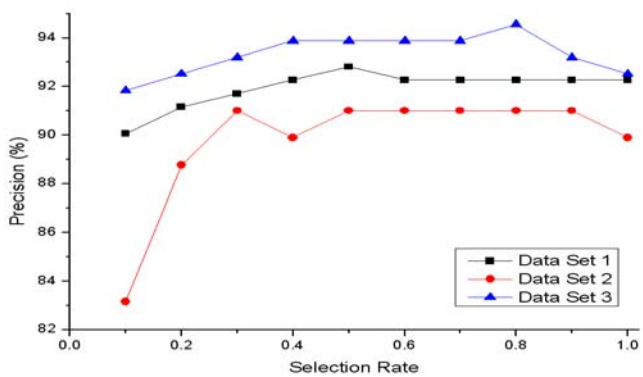

Figure 7. Influence of the number of features on precision

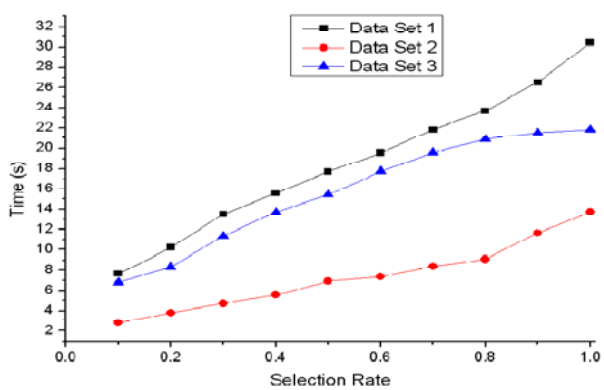

Figure 8. Influence of the number of features on time cost

4.4 Selection of the Number of Trees

Another interesting issue is whether the performance of *IRFA* highly depends on the size of a forest (number of decision trees in the forest). Since a large number of trees lead to a considerable computational cost, we need to find a good tradeoff between classification precision and computational cost. We performed experiments on the three data sets with different tree numbers $(k = 10, 25, 50, 75, 100, 125, 150, 200)$. The feature selection rate β was set to 0.5 in all experiments. The experimental results are shown in Figure 9 and Figure 10.

Figure 9 plots the precision against the number of trees k. The results show that if the number of trees is too small, the classification performance of the forest will be unstable. If the number of trees is too large, however, the computational cost for generating a forest will be very high. The classification precision becomes stable when $75 < k < 150$. The near-optimal precision can be obtained when k is set to 100. Moreover, as shown in Figure 10, with the increase of the number of trees, the computational time increases linearly as well. Therefore, in most situations, $k = 100$ is a good balance between classification accuracy and computational cost.

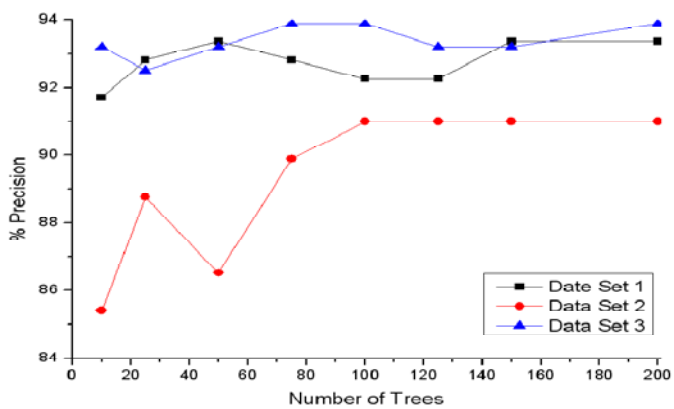

Figure 9. Influence of the number of trees on precision

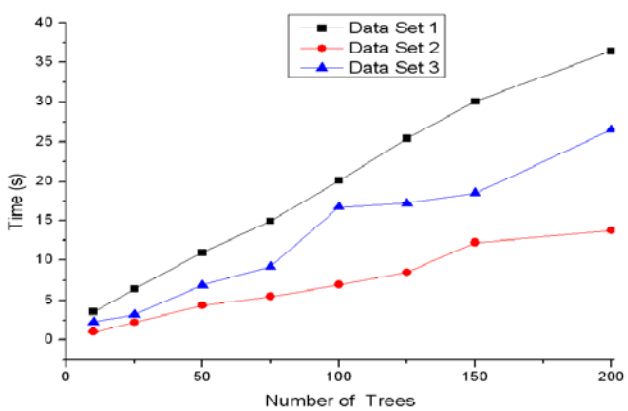

Figure 10. Influence of the number of trees on time cost

4.5 Comparison in Different Domains

The representation of search forms in different domains varies, so it is necessary to investigate if the performance of *IRFA* can be consistent in different domains. An experiment was designed to classify search interfaces from four different domains. We used the Web collection provided by Metaquerier[4], which contains the information about Books, Movies, Airfares, and Jobs. The number of trees for *IRFA* was set to 100, and parameter β was set to 0.5.

The experimental results are shown in Table 5. We can see the obvious variance of accuracy with regard to different domains. The detection accuracies for Books and Jobs domains are higher than those of Movies and Airfares. This was due to the different HTML structures of the search interfaces, as the search interfaces of Movies and Airfares domains are more complex and some of them require more than one step to get the content. The results imply that more formalized and simpler interfaces are more easily recognized. From the results, we also observe that the classification performance of *IRFA* was very stable in various domains. Even though the precision of SVM on the Movies data set was a little better than IRFA, it became the worst in other domains. Compared with other algorithms, *IRFA* was the most stable.

Table 5. Results of comparison experiments on different domains

	Dataset Size	Bayes	C4.5	SVM	RFA	IRFA
Books	251	92.83%	96.41%	89.24%	92.03%	**96.81%**
Movies	126	91.27%	91.27%	**92.06%**	87.30%	91.27%
Airfares	265	89.06%	90.94%	87.92%	91.69%	**91.70%**
Jobs	104	88.46%	94.23%	78.85%	**96.15%**	**96.15%**

4.6 Comparison with Different Feature Selection Schemes

We conducted experiments to demonstrate the performance of *IRFA* with different feature selection schemes. In this section, we used four commonly used feature selection functions mentioned in (Dash & Liu, 1997).

1. Random selection, which randomly selects the features using sampling without replacement, is the simplest feature selection method. Random feature selection is the method used in the original random forest (Breiman, 2001).
2. Information gain is defined as the difference between the original information requirement and the new requirement (Han & Kamber, 2007). The information gain value of a feature X

[4]http://metaquerier.cs.uiuc.edu/repository/training data set

is attained by computing the difference between the prior uncertainty and expected posterior uncertainty using X. Feature X is preferred to feature Y if the information gain from feature X is greater than that from feature Y (Dash & Liu, 1997). In this experiment, the information gain from feature X is set to feature X as its weight instead of Chi-square value in *IRFA*.

3. Gain Ratio is an extension of information gain. It attempts to overcome the problem that the information gain measure is biased toward tests with many outcomes (*i.e.*, it prefers to select attributes having a large number of values) (Han & Kamber, 2007). The process of embedding gain ratio into *IRFA* is similar to information gain in our experiment.
4. Chi-square χ^2 that has been explained in Section 3.2. The experiments in the prior sections were all based on this feature selection method.

Table 6. Comparison with different feature selection schemes

	Random	Information Gain	Gain Ratio	Chi-square χ^2
Data Set 1	88.76%	88.76%	89.89%	**91.01%**
Data Set 2	91.71%	91.16%	92.27%	**92.27%**
Data Set 3	91.84%	93.20%	93.20%	**93.88%**

We implemented the four feature selection methods in Weka's random forest package, and carried out experiments on the data sets described in Section 4.1. The results are shown in Table 6. From the results, we can see that the Chi-square method is better than the others. The reason that Information Gain and Gain Ratio scheme did not attain better performance can be explained as follows: since both of them use the same feature evaluation criterion in feature sampling and tree construction (for node splitting), the information of features cannot be fully exploited.

5. Conclusions

This paper has proposed *IRFA*, an improved random forest algorithm, for detecting search interfaces of the hidden Web. We extend the original random forest algorithm with a weighted feature selection method to automatically select a more representative subset of features for building each decision tree. The new method can overcome the problem of classifying high-dimensional and sparse search interface data through the ensemble of decision trees, each learned from a different subset of the original feature space. We have implemented the new algorithm and compared it with SVM, C4.5, Naïve Bayes, and original random forest algorithm (*RFA*). The experimental results have shown that our method is more accurate and robust. We have also observed that the new method is scalable to high dimensional data.

In the future, we plan to investigate more feature weighting methods for construction of random forests. Currently, we just use the features in the search forms. It is expected that using contextual information near the search forms may improve detection performance.

Acknowledgements

This research is supported in part by NSFC under grant No.60603066 and China National High-tech Program under grants No.2007AA01Z436 and No.2006AA01A124. Part of Joshua Huang's research was supported by the 863 project matching fund from The University of Hong Kong.

References

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32.

Breiman, L., Friedman, J. H., & Olshen, R. A. (1984). *Classification and regression trees*. New York: Chapman & Hall.

BrightPlanet.com (2000). The deep web: Surfacing hidden value. URL http://www.brightplanet.com

Caverlee, J., Liu, L., & Probe, D. B. (2004). Cluster and discover: focused extraction of qa-pagelets from the deep web. *Proceeding of the 20th International Conference of Data Engineering.*

Cope, J., Craswell, N., & Hawking, D. (2003). Automated discovery of search nterfaces on the web. *Fourteenth Australasian Database Conference*. Adelaide, Australia: Fourteenth Australasian Database Conference.

Dash, M., & Liu, H. (1997). Feature selection for classification. *Intelligent Data nalysis*, 1(3), 131-156.

Florescu, D., Levy, A. Y., & Mendelzon, A. O. (1998). Database techniques for the world wide web: A survey. *SIGMOD Record*, 27(3), 59-74.

Han, J., & Kamber, M. (2007). *Data Mining: Concepts and Techniques(2nd version)*. China Machine Press.

He, B., Patel, M., Zhang, Z., & Chang, K. C. (2007). Accessing the deep web. *Communications Of The ACM*, 50(5).

Ho, T. K. (1998). The random subspace method of constructing decision forests. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(8), 832-844.

Kittler, J., Hatef, M., Duin, R., & Matas, J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20, 226-239.

Lage, J. P., Silva, D., Golgher, P. B., & Laender, A. H. F. (2004). Automatic generation of agents for collecting hidden web pages for data extraction. *Data and Knowledge Engineering*, 49, 177-196.

Larson, H. J. (1982). *Introduction to probability theory and statistical inference*. New York: Wiley, 3 ed.

Ntoulas, A., Zerfos, P., & Cho, J. (2005). Downloading textual hidden web content through keyword queries.

Quinlan, J. R. (1993). *C4.5: Programs for Machine LearningMachine Learning*. Morgan Kaufmann.

Raghavan, S., & Garcia-Molina, H. (2001). Crawling the hidden web. *Proceedings of the 27th International Conference on Very Large Data Bases*. Roma, Italy.

Wang, J., & Lochovsky, F. (2003). Data extraction and label assignment for web databases. *Proceedings of the 12th International World Wide Web Conference*. Budapest, Hungary.

Zhang, Z., He, B., & Chang, K. C. (2004). Understanding web query interfaces: best-effort parsing with hidden syntax. *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. Paris, France.