Automated Extraction of Hit Numbers From Search Result Pages

Yanyan Ling[1], Xiaofeng Meng[1], and Weiyi Meng[2]

[1] School of Information, Renmin University of China, China
{lingyy,xfmeng}@ruc.edu.cn
[2] Dept. of Computer Science, SUNY at Binghamton, Binghamton, NY 13902, USA
meng@cs.binghamton.edu

Abstract. When a query is submitted to a search engine, the search engine returns a dynamically generated result page that contains the number of hits (i.e., the number of matching results) for the query. Hit number is a very useful piece of information in many important applications such as obtaining document frequencies of terms, estimating the sizes of search engines and generating search engine summaries. In this paper, we propose a novel technique for automatically identifying the hit number for any search engine and any query. This technique consists of three steps: first segment each result page into a set of blocks, then identify the block(s) that contain the hit number using a machine learning approach, and finally extract the hit number from the identified block(s) by comparing the patterns in multiple blocks from the same search engine. Experimental results indicate that this technique is highly accurate.

1 Introduction

Today the Web has become the largest data repository and more and more information on the Web can be accessed from search engines and Web databases (WDBs). Ordinary users can retrieve information from search engines and WDBs by submitting queries to their search interfaces. As we observed, besides the wanted results, most web sites also return a number (we call it hit number) indicating how many results are found for this query. Fig.1shows an example obtained by submitting a query "game" to Ikotomi, which is a web site providing online technical documents. In the result page, besides the results, we also get knowledge from the hit number on how many matching records there are in Ikotomi's database (in our example, it is 1400). Furthermore, by comparing the hit numbers of different web sites providing similar information, such as documents on java, a rank can be assigned to each one showing their capabilities. And based on them, interesting applications of integrating or utilizing these web sites can be implemented, such as giving user suggestions according to his/her queries. For example, when a user wants to find technical documents using key word "java, the system can suggest to him/her the web site having the largest database about Java, which probably returns results he/she wants. Therefore, there is a need for the technique for automatic discovery of this hit number from any search result web pages.

Much useful information about a search engine can be derived from the hit numbers. First, for a single-term query t submitted to a search engine S, the hit number is in fact the document frequency of t in the document collection indexed by S.

As we know, document frequency information is critical in some applications, such as metasearch engines that utilize document frequency of each term to compute the importance/usefulness of terms in representing and differentiating different document databases. Second, the hit numbers can be used to estimate the size of search engines. Third, hit numbers can be used to classify search engines. For example, the method proposed in [4] uses the hit numbers of probe queries, which represent specific categories (e.g. sports), to compute the coverage and specificity of each search engine and then classify it into an appropriate category of a concept hierarchy. These applications suggest that, we need an automatic hit number extraction technique to be devised. Despite the importance to obtain the hit numbers automatically, this problem has not been seriously studied before to the best of our knowledge.

Intuitively, we may think that most hit numbers are placed in a special block in the result web pages (we call it the hit number block, HNB for short)."1400 results found, top 500 are sorted by relevance" is the HNB in Fig.1. Sometimes by using some simple patterns, HNBs may be identified and the hit numbers can be extracted. This naive solution does solve this problem in some cases, but, unfortunately, when we want to extend it and devise a general method, it broke. Our study shows that it is quite difficult to accurately and automatically extract hit numbers from general search result web pages because HNBs vary from one web site to another. In other words, there are numerous formats and patterns, which will be studied in Section 2 in detail. So, the main problem studied in this paper is to automatically find hit numbers in returned result web pages of search engines or web databases, and our contributions are: (1) we report a detailed study on various cases of HNBs; (2) we propose a novel method for solving the hit number extraction problem by applying machine learning techniques; and (3) we evaluate on our method experimentally and the experimental results show that our approach is highly effective.
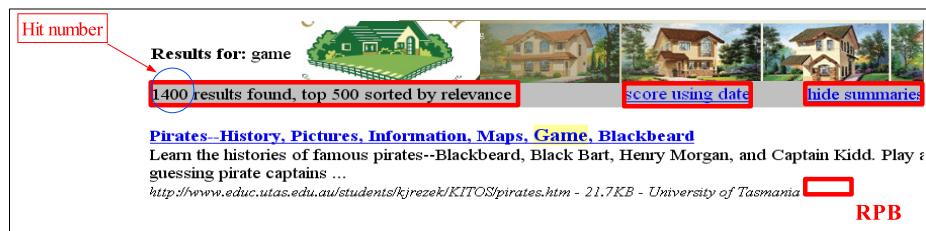
Fig. 1. A Result Page obtained from Ikotomi

The rest of the paper is organized as follows. In Section 2, we report our survey of the diverse patterns of hit number presentations by different systems. From this survey, we can see that the hit number extraction problem is not trivial. In Section 3, a result page is represented as a set of blocks and we discuss how to identify the HNBs among all blocks based on its special features including layout features, visual features and semantic features. In Section 4, we propose an approach to extract the hit numbers from the HNBs. In Section 5, we report our experimental results based on 500 real result

pages from various search engines and WDBs. We review related works in Section 6. We conclude the paper in Section 7.

2 Diversity of Hit Number Patterns

In this Section we show our investigation on various hit number patterns based on observing numerous real life Web sites. Below, we summarize our investigation: first we introduce some intuitive patterns used by web page authors; second we show many diverse cases which cannot easily be covered by naive pattern recognition techniques. Finally, in support of our strategy, we will also discuss the reason why intuitive approaches actually cannot solve the hit number extraction problem.

2.1 Intuitive Patterns of Hit Numbers

Web page authors frequently report the hit numbers of specific queries using a literal structure containing the word "of". Table 1 shows some examples of this kind. Generally speaking, this pattern consists of more than one number, which is usually in the order "X V Y U Z", where "X" and "Y" are two numbers forming a range, "V" is preposition such as "-", "to" and "through", "U" contains a variation of the "of" structure such as "of about" and "out of", and "Z" is the hit number we want. Based on our investigation, 41.7% (167 out of 400) result pages contain hit numbers in this pattern. And as we also observed, designers of search engines prefer this kind of pattern, and there are also some variations (e.g. the fourth example in Table 1 contains four numbers, in which the fourth one denotes the query processing time).

(1) Jobs 1 to 50 of more than 1000
(2) Now displaying vehicles 1 - 4 of total 4
(3) Results 1 through 10 of about 19,000,000
(4) Hits 1-10 (out of about 1,259,425 total matching pages) in 0.51 seconds

Table 1. Most common pattern

(1) Your search resulted in 1 business that matched
(2) Your single word search for tax found 202 names - top 150 listed
(3) Your search for "life" returned 230 of 33793 records
(4) Your search produced 26 records

Table 2. Active voice pattern

(1) 32 documents match your query
(2) 24 objects returned
(3) 3 occurrence(s) found for "troy"
(4) 54 unique top-ten pages selected from 48,809,621 matching results

Table 3. Passive voice pattern

(1) Document count: apple (30)
(2) There are 6 institutions
(3) For: "WEB" Total Hits: 40
(4) 2 Total Resources Below:
(5) Page: 1 2 3 4 5 of 100 or more hits

Table 4. Diverse cases

Our survey also shows that the designers of search engines and web databases also like another type of pattern to show hit numbers. For example, Table 2 and Table 3 give out some examples retrieved from web databases and search engines. In general, there are

two kinds of patterns. One embeds hit numbers in active voice structures (Table 2 shows examples of this kind). As we can see, the hit numbers usually follow a verb, such as "return", "result in", and "produce". Among the 400 we surveyed, 122 or 30.5% are in this pattern, and some variations also exist, such as the 3rd example in Table 2, which contain extra numbers. The other pattern embeds hit numbers in passive voice structures (Table 3 shows some examples of this kind). It is similar to the active voice case with differences. The hit numbers appear before verbs, as we can see in the examples in Table 3. 13.3% (53 out of 400) web pages we surveyed are in this pattern and there are also variations.

Though most of the web pages (85.5% in all) are covered by intuitive patterns we have discussed. Naively developing programs based on them do not lead to an effective solution. Two reasons are as follows: (1) there are still 14.5% pages that are not covered by these patterns; (2) variations for each pattern make it difficult to use only basic patterns. Thus, we need a general, robust and automated solution.

2.2 Diverse Cases of Hit Numbers

The remaining 14.5% web pages introduce diverse cases on embedding hit numbers. In our survey, we found out some formats that are beyond our imaginations. Table 4 displays some diverse cases. As one can see, some do not have any verbs (1st example), some do not have complete sentences (1st, 3rd, and 4th examples). The appearance location of hit numbers also varies, such as at the beginning of a sentence (4th example), the middle of a sentence (2nd example), and the end of a sentence (1st and 3rd examples). One of the worst cases is the 5th example, where it does not show the accurate hit number at all (it uses the word "more"). Instead it displays numbers like 1, 2, 3, 4, 5, and 100, which can easily cause most solutions to fail.

In summary, due to the diversity and unpredictability of the patterns used to present hit numbers by different search systems, traditional pattern recognition strategies are unlikely to work well for solving the hit number extraction problem. In this paper, we explore the special features in the web context such as visual features and layout features, to solve this problem.

2.3 The Problem of an Intuitive Solution

Before we introduce our solution, in this subsection we would like to discuss an intuitive solution by giving the reasons why it is not general enough, which also provides a clearer picture on how complex this problem is.

One may think that the hit numbers can be easily detected since they often appear at the beginning of the result pages and have distinguishing colors to attract users' attention. In fact, this is not true in general. Sometimes the hit number appears before the data records while other times after them. Due to the different sizes of data regions and advertising regions of different result pages, the absolute positions of hit numbers on result pages are uncertain. Also, only a small percentage of result pages use special colors to display hit numbers.

Therefore, based on all these analyses above, we take all the helpful cues into consideration because none of them is a decisive factor. In our method, all these helpful

information are utilized to achieve high performance. Generally speaking, the basic idea of our approach is that we first split web pages into blocks (we will explain it below) and select the ones that probably contain hit numbers, and then we check all these blocks in the hope of finding hit numbers. Thus our method consists of following two main steps - Hit Number Block discovery and Hit Number extraction.

3 Hit Number Block Discovery

As we have stated, the mission of this step is to split web pages into blocks and then select the ones that probably contain hit numbers. In our strategy, a block in web pages is just a group of adjacent words, in which no more than 3 consecutive white spaces are allowed. By ignoring tags in web pages, result web page can be split into many blocks according to this definition, and we call them result page blocks (RPB for short). In Fig.1, the contents in thick-lined boxes are examples of RPBs. After a result web page is split into a set of RPBs, the next step is to identify HNB(s), which probably contain hit numbers.

3.1 Splitting Web Page and Preprocessing RPBs

Splitting web page into RPBs is easy and straightforward. By viewing web pages as a token sequence, RPBs can be derived according to their definition, where in our work, a token can be a HTML tag, a single word, or consecutive white spaces. As the techniques used for splitting are fairly simple, we do not discuss them here.

Next we do basic preprocessing on obtained RPBs. Two kinds of RPBs will be removed from the set as they definitely do not contain hit numbers. One is RPBs that do not contain any digit, which are definitely not hit number block. The other is RPBs that do contain digit(s) but these digit(s) cannot be hit numbers. For example, Such RPBs contain only float numbers (they can be easily identified by their formats), or numbers for prices (they have a common prefix such as "$"), and numbers for date (they usually are concatenated by special characters, such as "/"). After preprocessing, for a given result web page, the RPBs that contain possible hit numbers are identified. The remaining steps only need to identify whether they contain hit numbers or not. In our solution, a machine learning method is employed to perform this.

3.2 HNB Identification by Decision Tree

Many features of web pages, which can be discovered by simply observing example web pages, could be utilized to identify hit number blocks. However, not all of these features are significant. Some of them are critical in identifying hit number block while some are not. Thus we need a mechanism to discover and select significant features. Thus in our work we apply the C4.5 learning technique to induce a decision tree, which is implemented based on Weka [7].Then we can do identifications based on induced decision tree. In overall picture, the steps of our strategy are as follows.

1. We select some example web pages, and manually identify HNBs from RPBs. The results will be recorded in a log file.

2. For each example web page, values of various web page features of each block are checked and calculated. The results will also be logged.
3. We apply C4.5 algorithm to induce a decision tree with selected examples. Each condition in decision tree is represented in the form whether a specific web page feature is fulfilled.
4. Finally we evaluate the induced decision tree with training set and possibly refine it.

In our work, for each RPB, almost 50 web features are checked and calculated. As we investigated, some features have discrete values, such as where specific word exists, but most of them have continuous values, such as various offsets. Therefore we prefer the C4.5 algorithm to ID3 for its ability to handle continuous values. The most important part, which is also our contribution, is to identify various web page features. Next we will discuss each of them in detail.

3.3 Web Page Features

This subsection describes various web page features used in inducing decision tree for identifying HNBs. You will see that for each RPB, various features can be utilized to do identification. In particular, human beings also rely on these features to separate HNB from other parts of the web page. They can be visual cues, text characteristics, and even frequent words. In the following paragraphs we will describe them one by one. Before doing that, we would like to introduce an illustrative example first, which is shown in Fig.2. As we can see, there is a hit number block "3577 stories and blog posts contain the term health" in the broken-line box, and let us mark it as B.
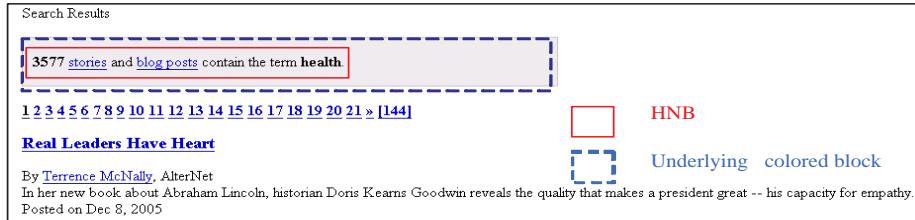

Fig. 2. A Result Page obtained from AlterNet

Layout related features Firstly, we observe that HNB is not randomly placed on a result page and the HNB is usually small in size. Thus the absolute position of each RPB in the result page is taken into consideration, such as *BlockOfsetX*, *BlockOfsetY*, *BlockWidth*, *BlockHeight*. For example, the distance between the left border of B and the left border of the page along the X-axis is 14 pixels (*B.BlockOfsetX* = 14 pixels).

Secondly, considering the result page structure, we find that HNB(s) usually occur either near the beginning or the end of the data region. Hence, the relative distance between HNB(s) and data regions should also be taken into consideration, such

as *DataRegionOfsetX*, *DataRegionOfsetY*, *DataRegionWidth*, *DataRegionHeight*, *RelativeY1*, *RelativeY1Ra*, *RelativeY2* and *RelativeY2Ra*. Many works deal with the problem of data region identification. [6] proposed a technique which is able to mine both contiguous and noncontiguous data records. [6] proposed a novel partial alignment technique based on tree matching and extract data region very accurately. In general, we cannot dismiss RPBs within the data region,because none of the current techniques on data region identification is perfect.So we will calculate the above features for each candidate RPB. For example, B is closer to the top border of the data region, and the distance between the bottom border of RPB and the top border of the data region along the Y-axis is 38 pixels (*B.RelativeY1* = 38 pixels).

Finally, sometimes the HNBs are emphasized with a background block with outstanding color to attract people. Then some layout features of the background block are taken into consideration, such as *BoxOfsetX*, *BoxOfsetY*, *BoxWidth* and *BoxHeight*. If no background block exists, the 4 features are set to zero. For example, the height of the background block of B is 42 pixels (*B. BoxHeight* = 42 pixels).

Color related features This class of features is based on our observation that HNBs on a result page usually have special appearances distinguishable from other RPBs. The designers of Web pages often make the HNB salient with outstanding background color and foreground color, in order to emphasize it and attract users' attention. Therefore, some color related features are taken into consideration, such as *BackgroundColor*, *ForeColor*, *BodyBackgroundColor*, *ColorCount* and *IsHighlight*. For example, the color behind the content of B in Fig.2 is gray (*B. BackgroundColor*=gray). In addition, 2 different font colors are used in B (*B.ColorCount*=2).

Characters related features Firstly, we observed that numbers occurred in HNB(s) may have their own features distinguishable from the numbers in other RPBs. Some features such as *DigitalCount*, *CharacterCount*, *DigitPercent*, *NumberCount*, *WordCount* and *NumberPercent* are taken into considerations. For example, the amount of numbers appeared in the RPB (*NumberCount*) is usually within a range. In Fig.2 *B.NumberCount*=1 ("3577") and *B.NumberPercent*=11% while *NumberPercent* denotes the ratio of the *NumberCount* and the total amount of words in the RPB.

Secondly, we observed that seldom words were hyperlinked within a HNB and some words were overstriked in order to emphasize them and attract the users' attention. Therefore, *LinkCount* and *BoldCount* may also help in identifying HNB. For example, *B.BoldCount*=2 while 2 words are overstriked ("3577" and "health").

Semantic related features This class of features is based on our observation that some frequent words play an important role in HNB identification. These words will be called clue words. A survey is conducted to identify the frequent terms in a set of more than 400 HNBs from different result pages. Top-n words were selected (here n is set to 20). A fraction of the results is listed in Table 5. Though such terms would often appear in HNBs, their appearances alone are not enough to accurately identify HNBs. First, such words may also appear in other RPBs. For example, "of" appears in "Page 1 of 20".

Second, some HNBs do not have any of these frequent words. For example, none of the frequent words appear in "There are 6 institutions" even though this block is a real HNB.

$$Frequency = \frac{occurrences \quad of \quad HNBs}{total \quad number \quad of \quad HNBs}$$

Terms	Frequency	Terms	Frequency
-	47%	search	17%
result	38%	return	16%
of	34%	document	11%
for	19%	you	11%
match	18%	found	9%
show	17%		

Table 5. Frequent Terms

For each frequent word w, we use *B.Count (w)* to denote the number of times the word occurs in block B. For 20 frequent words, 20 such Counts will be collected as 20 features for each block. In addition, we use *CountTotal* to denote the summation of these Counts for each block. Overall, there are 21 (20+1) features involved in this class. For example, *B.Count(Term)* denotes how many times the word "term" occur in B and we have *B.Count(Term)* = 1.

4 Hit Number Extraction from HNB

After the HNBs are found, the next step is to extract the hit numbers from them. Intuitively, we may think that the largest number in a HNB is the hit number. However this simple heuristic rule is not valid in general. For example, there is an example "Found 72 of 1,434 searched" we find when doing investigations. It is clear that the hit number is "72", which is not the largest number ("1,434").

Fig. 3. Example obtained from Salon(www.dir.salon.com)

On the other hand, one may discover that for a given web site, different result pages returned by submitting different queries share similar template. For example, consider

the HNBs, where one is "1-10 of 4,199 results for java" and the other is "1-10 of 7,509 results for database". By comparing them, we find out that only hit numbers (4,199 vs. 7,590) and query keywords (java vs. database) are different. It seems a solution can be devised based on comparing changed numbers. Unfortunately, it is also not general enough. Let us check a complex example in Fig.3. By comparing "Race, articles 1- 13 of 120 << 2 3 4 5 6 7 8 9 10 >> Next" with "Craig Seligman, articles 1- 13 of 16 << 1 2>> Next", we can see that more than one number changed and the idea of simply comparing changed numbers does not work.

With the above observations in mind, we learn that simple pattern-matching algorithm is not general enough to identify hit numbers. A hybrid algorithm combining characteristics observed so far should be invented. Thus in our work, we propose a novel algorithm which identifies the hit number step by step.
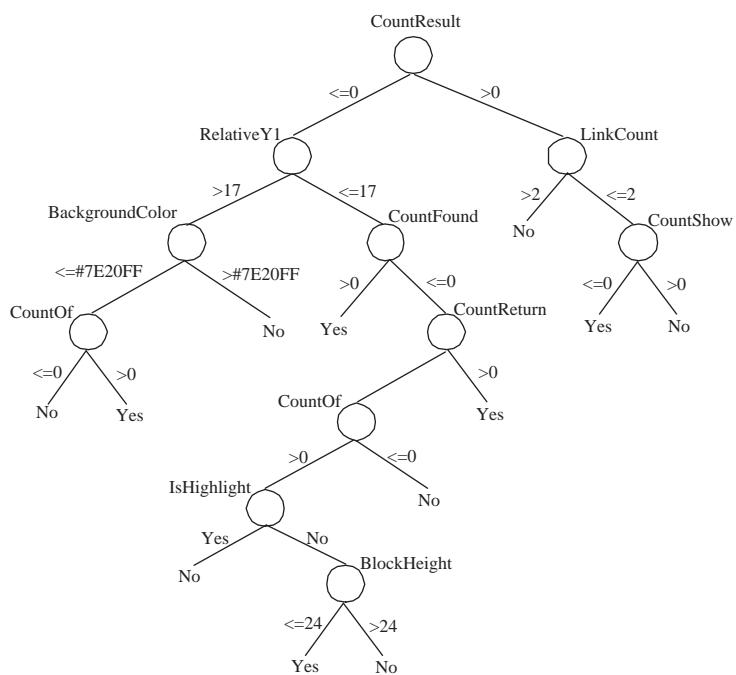

Fig. 4. Induced Decision Tree, which is induced with 50 features

The input of our algorithm consists of two or more HNBs retrieved by submitting different query keywords to the same web site. And the output is the hit numbers (if there is one). Our algorithm consists of the following steps:

1. Identify number sequence and remove it from HNBs: A number sequence is a sequence of integer numbers, such as "2 3 4 5 6 7 8 9 10" in the left portion and "1 2" in the right portion in Fig.3. Number sequences could not be hit numbers.

2. Identify range pattern and remove it from HNBs: A range pattern consists of two integers with a connector in between, where the connector can be "-", "to", and "through". For example, "1-13" is a range pattern. Range patterns do not contain hit numbers.
3. Identify all non-numbers and remove them from HNBs: The words or characters strings that are all non-numbers in HNBs will be removed. For the two HNBs in Fig.3, the static contents include "articles", "of", "<<", ">>", and "Next".
4. Identify query keyword(s) that is a number and remove them from HNBs: They are the key words which may be numbers submitted to retrieve the result pages. The reason for doing this is to remove possible numbers used in queries.
5. Identify hit number by comparing HNBs: At this time, each number in HNB could be the hit number and we simply choose the largest one.

The basic idea behind our algorithm is to gradually identify impossible cases. It is easy to implement as well as effective. Next we will show experiments to support it.

5 Empirical Evaluation

5.1 Data Set

In this section we will show the empirical evaluations of our prototype system for hit number extraction. First of all we introduce the data set of web pages obtained from real-life Web pages.

These web pages are manually retrieved by submitting queries to Web sites listed in www.completeplanet.com. To be general enough, web sites belonging to a broad range (news, media, business, society, science etc..) are selected. In all, there are 500 web sites in our data set, and for each of them, one result web page was retrieved. Half of them are used as the training set and the other half as the testing set.

Each result web page in the entire data set is broken into RPBs, and we manually mark out HNBs. Remember that we remove RPBs that cannot be HNBs (Section 3.1), however, the number of RPBs (including Non-HNBs and HNBs) is lager than the number of web pages. In general, some web pages may contain more than one HNB for different purposes. Therefore, we generated a training set containing 272 positive examples (HNB) and 352 negative examples (non-HNB). Similar work is also done on the testing set where 269 positive instances (HNB) and 376 negative instances (non-HNB) are identified. The training set will be used for inducing decision tree and the testing set will be used in empirical evaluation.

5.2 Evaluatation of the HNB Discovery Algorithm

Fig.4 shows the decision tree induced from web page features described in Section 3.3. 9 out of 50 features are selected and associated with nodes in the decision tree. Feature selection is automatically implemented by this classification algorithm. All 50 features were weighted and ranked individually and the top 9 highest weighted features were selected, which work very well on random selected HNB discovery in our experiments.

Then we do classification by using the reduced decision tree on our testing set. In order to measure the performance, recall and precision are defined. They are defined as

$$Recall = \frac{|PredictedHNB \cap ActualHNB|}{|ActualHNB|} \quad Precision = \frac{|PredictedHNB \cap ActualHNB|}{|PredictedHNB|}$$

where ActualHNB is the set of real HNBs in the testing set (it is manually obtained), and PredicatedHNB is the set of HNBs discovered by our method.

Possible HNB Found	271
Predicted HNB	264
Actual HNB	269
Precision	$\frac{264}{271} = 97.4\%$
Recall	$\frac{264}{269} = 98.1\%$

Table 6. Evaluation Result on Test Set

Table 6 shows the evaluation result on the testing set. In fact, there are 269 actual HNBs, which is known when we preparing this set. Our algorithm found 271 HNBs, in which 264 of them are real HNBs (we manually checked them). Therefore, the precision is 98.1 % and the recall is 97.4%. As we can see, our method is highly effective.

5.3   Evaluation of the Hit Number Extraction Algorithm

After HNBs are identified, we proceed to evaluate our hit number extraction algorithm on them. In the previous step we obtained 264 HNBs, and, remember that in the training step, we also obtained 277 HNBs. So in all there are 541 HNBs. After the execution, our algorithm identified 536 right hit numbers (we also manually checked them). Therefore the accuracy is 99.1%, which clearly shows that our algorithm is highly effective.

6   Related Work

Discovering hit numbers from search result pages of search engines and WDBs can be considered as a special case of automated data extraction for specific information from web pages. There are lots of works focusing on extracting specific information from text documents and web pages of different domains. For example, there are works which automatically identify human names [1], or extracting individual product information, such as price [3]. The closest work to ours is [5], which tried to detect and extract postal addresses from web pages. But we are not aware of any prior work on automatic hit number extraction. In terms of techniques used, we adopted and adapted two kinds of techniques proposed in literature in our solution. One is classification by inducing a decision tree. The work reported in [2] induced a decision tree for automatically discovering search interfaces from a set of HTML forms. Similar classification techniques are also used in our work. The other is utilizing visual cues in web pages. As reported in [8], by utilizing specific visual cues, such as shape of HTML block, layout position, data records can be automatically extracted from result web pages returned

by search engines. However, in our work, by inducing a decision tree from many visual features of various representations of hit numbers, the above two techniques are combined together to form a novel and effective method to identify and extract specific pieces of data, which in our case are hit numbers.

7 Conclusion and Future Work

In this paper, we proposed an automatic approach to extract hit numbers from the result pages returned from search engines and WDBs. As mentioned in the introduction, automatic hit number extraction is important in several important applications. This approach consists of three steps. The first step segments a result page into a set of RPBs. The second step applies a machine learning technique to discover HNB(s) from the RPBs based on an extensive list of features. The third step identifies hit numbers from HNBs based on comparing the patterns among multiple HNBs from the same site. Experiments show our approach is highly effective with its accuracy close to perfection.

In the future, we plan to consider hit number extraction from result pages containing multiple data regions. Each region may have its own hit number and the presence of multiple data regions may pose new complications to this problem.

Acknowledgments

References

1. Zheng Chen, Liu Wenyin, and Feng Zhang. A new statistical approach to personal name extraction. In *ICML*, pages 67–74, 2002.
2. Jared Cope, Nick Craswell, and David Hawking. Automated discovery of search interfaces on the web. In *ADC*, pages 181–189, 2003.
3. Robert B. Doorenbos, Oren Etzioni, and Daniel S. Weld. A scalable comparison-shopping agent for the world-wide web. In *Agents*, pages 39–48, 1997.
4. Panagiotis G. Ipeirotis, Luis Gravano, and Mehran Sahami. Probe, count, and classify: Categorizing hidden web databases. In *SIGMOD Conference*, 2001.
5. Can Lin, Zhang Qian, Xiaofeng Meng, and Wenyin Liu. Postal address detection from web documents. In *WIRI*, pages 40–45, 2005.
6. Bing Liu, Robert L. Grossman, and Yanhong Zhai. Mining web pages for data records. *IEEE Intelligent Systems*, 19(6):49–55, 2004.
7. Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques (2nd edition)*. Morgan Kaufmann, San Francisco, 2005.
8. Hongkun Zhao, Weiyi Meng, Zonghuan Wu, Vijay Raghavan, and Clement T. Yu. Fully automatic wrapper generation for search engines. In *WWW*, pages 66–75, 2005.