University of Windsor

Scholarship at UWindor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

2006

Cleaning Web pages for effective Web content mining.

Jing Li

University of Windsor

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

Recommended Citation

Li, Jing, "Cleaning Web pages for effective Web content mining." (2006). *Electronic Theses and Dissertations*. 1443.

<https://scholar.uwindsor.ca/etd/1443>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

Cleaning Web Pages for Effective Web Content Mining

by

Jing Li

A Thesis

Submitted to the Faculty of Graduate Studies and Research through
the School of Computer Science in Partial Fulfillment of the Requirements
for the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2005

© 2005 Jing Li

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 978-0-494-17067-0

Our file Notre référence

ISBN: 978-0-494-17067-0

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Web pages usually contain many noisy blocks, such as advertisements, navigation bar, copyright notice and so on. These noisy blocks can seriously affect web content mining because contents contained in noise blocks are irrelevant to the main content of the web page. Eliminating noisy blocks before performing web content mining is very important for improving mining accuracy and efficiency. A few existing approaches detect noisy blocks with exact same contents, but are weak in detecting near-duplicate blocks, such as navigation bars.

In this thesis, given a collection of web pages in a web site, a new system, WebPageCleaner, which eliminates noisy blocks from these web pages so as to improve the accuracy and efficiency of web content mining, is proposed. WebPageCleaner detects both noisy blocks with exact same contents as well as those with near-duplicate contents. It is based on the observation that noisy blocks usually share common contents, and appear frequently on a given web site. WebPageCleaner consists of three modules: block extraction, block importance retrieval, and cleaned files generation. A vision-based technique is employed for extracting blocks from web pages. Blocks get their importance degree according to their block features such as block position, and level of similarity of block contents to each other. A collection of cleaned files with high importance degree are generated finally and used for web content mining. The proposed technique is evaluated using Naïve Bayes text classification. Experiments show that WebPageCleaner is able to lead to a more efficient and accurate web page classification results than existing approaches.

Keywords

Web page cleaning, Web content mining, Noise block, Text classification, Near-duplicate, Text similarity

Acknowledgement

I would like to give my sincere thanks to all who have contributed to this work in any way.

First, I am very grateful to my supervisor Dr. Christie Ezeife for her grand support and help throughout my Masters study. Dr. Ezeife has guided me to the research direction, got me interested in research, and carefully reviewed phased updates. I would like to express my admiration for her dedication to teaching and research, and my special thanks for her guidance, support, and encouragement on this thesis.

I would like to thank the external reader, Dr. Karen Fung, the internal reader, Dr. Luis Rueda, and the thesis committee chair, Dr. Dan Wu for their time to review my thesis, and for their valuable suggestions.

I am infinitely grateful to my family. Deep appreciations to my parents and husband for their unconditional love and care to me. I also express my thanks to all members in WODD lab for any help they provided. Thank you all.

Table of Contents

Abstract	iii
Acknowledgement.....	iv
List of Figures	vi
List of Tables.....	vii
Chapter 1 INTRODUCTION	1
1.1 Web Mining.....	1
1.2 Web Page Classification and Categorization	2
1.3 Web Page Cleaning	2
1.4 Data Cleaning for Data Warehousing	5
1.5 Why Web Page Cleaning	6
1.6 Thesis Problem and Contributions	6
1.7 Outline of Thesis	7
Chapter 2 RELATED WORKS.....	8
2.1 Web Page Partition and Cleaning.....	8
2.1.1 <TABLE> Tag-based.....	8
2.1.2 DOM-based	10
2.1.3 Hyperlink-based	12
2.1.4 Vision-based.....	13
2.2 Strings and Files Similarity Detection	16
2.2.1 The Basic Field Matching Algorithm	16
2.2.2 Edit Distance among the Q-grams	17
2.2.3 Cosine Similarity Metric with IDF Weighting.....	18
2.2.4 Document Similarity	19
2.3 Duplicate Elimination for Data Warehousing.....	21
2.3.1 Naïve Method.....	21
2.3.2 Sorting and Merging Based.....	22
2.4 Document Classification	27
2.4.1 Naïve Bayes Text Classification	27
2.4.2 Web Page Classification.....	29
Chapter 3 PROPOSED TECHNIQUE FOR WEB PAGE CLEANING	30
3.1 Block Extraction.....	31
3.2 Block Importance Retrieval	39
3.3 Cleaned Files Generation	44
3.4 The Overall Algorithm.....	47
Chapter 4 EXPERIMENTS AND EVALUATIONS.....	48
4.1 Implementation Environments	48
4.2 Performance Measure.....	48
4.3 Experiments on Web Page Cleaning.....	49
4.4 Experiments on Web Page Classification	52
4.5 Analysis of Classification Results.....	58
Chapter 5 CONCLUSIONS AND FUTURE WORK.....	60
References	62
Vita Auctoris	67

List of Figures

Figure 1.1: Blocks in CNN web page	3
Figure 1.2: Noise blocks in Future Shop web site.....	4
Figure 2.1: Building the style tree.....	11
Figure 2.2: Block extraction of an HTML page.....	15
Figure 3.1: Overall process of cleaning web pages.....	31
Figure 3.2: Algorithm for block extraction	35
Figure 3.3: Two web page fragments and their segmentation results using VIPS.....	36
Figure 3.4: The DOM tree structure of the HP product web page in Figure 3.3	36
Figure 3.5: The content structure of the HP product web page in Figure 3.3	37
Figure 3.6: Algorithm for removing blocks with same contents.....	40
Figure 3.7: Algorithm for getting similar level of contents	42
Figure 3.8: Algorithm for generating cleaned files	44
Figure 3.9: Output blocks in a web page from www.bestbuy.ca	46
Figure 3.10: The overall algorithm for web page cleaning	47
Figure 4.1: The whole process of experiments	50
Figure 4.2: Execution time for web page cleaning in chart	51
Figure 4.3 Average accuracy on 10 trials for each sub case	54
Figure 4.4: Classification accuracy on 4-fold cross validation	57
Figure 4.5: Classification efficiency on 4-fold cross validation	57

List of Tables

Table 2.1-1: Duplicate records in a database.	22
Table 2.1-2: Table with unclean data	25
Table 2.1-3: Table after merging.....	25
Table 3.1-1: Blocks table after doing the block extraction	38
Table 3.1-2: Table Blocks after removing records with the same contents	40
Table 3.1-3: Result table after retrieving block importance.....	43
Table 4.1-1: Distribution of web pages in their web site and categories	50
Table 4.1-2: Execution time for web page cleaning in table.....	51
Table 4.1-3: Average accuracy and standard error on 10 trails for each sub case.....	53
Table 4.1-4 Classification efficiency comparison for case 1	54
Table 4.1-5: Data distribution on each fold for 4-fold cross validation case	55
Table 4.1-6: Classification performance on each case of 4-fold cross validation.....	56
Table 4.1-7: Average accuracy and standard error on 4-fold cross validation	56

Chapter 1 INTRODUCTION

1.1 Web Mining

With the expansion of the World Wide Web, a lot of information on merchandise, education, daily news, and others are coming from the web. Data mining research on web information is a hot area of interest nowadays. According to [Etz96], web mining is the use of data mining techniques to automatically discover and extract information from web documents and services.

Due to properties of the web, such as enormous information size, dynamically changed content, and diversity, it is very easy to get lost when facing such an information forest. Web mining research provides automated assistance directing users through with what they want. Web mining has been widely researched, which covers a wide range of areas, including mining to the web data [Cha02] and mining to the web structure [JAKN03, KLC02]. In [KB00], the authors survey existing web mining research, and clarify some ambiguous terms in web mining. The authors of [MRNL99] claim that three types of data are involved on the web: data on the web (content), web structure data, and web log data (usage). Therefore, they compose the three categories of web mining as web content mining, web structure mining, and web usage mining respectively.

Web content mining is used for automatic search of information resource that is available online. Web contents include multi-types of data such as image, audio, video, text, metadata, hyperlinks and so on. Some of them are structured data like the data in tables or database generated data, others are semi-structured data such as HTML documents, or structured data like text data. The data multiplicity makes data content mining a more complicated approach than mining on pure text.

Web structure mining tries to discover the link structure of the hyperlinks of the web to generate structure model of the web page or web site. This model is useful for categorizing web pages and generating information such as the similarity and relationship between different web sites.

Web usage mining tries to discover the user navigation patterns while they are surfing on the web. Unlike web content mining and web structure mining, which use real

and primary data on the web, web usage mining uses secondary web data derived from the interactions between the users and the web, which includes the data from web server access logs, user profiles, user sessions, cookies, and so on.

Web mining has a close relationship with information retrieval, information extraction, machine learning, and databases in that they provide a sound foundation for web mining.

1.2 Web Page Classification and Categorization

To deal with enormous amounts of existing and emerging information on the web, automated organization of these contents is very necessary. Web page organization applications include key-word based search engine, such as MSN Search (<http://search.msn.com/>), and taxonomic web page categories, such as Yahoo directory (<http://dir.yahoo.com/>) and Google directory (<http://directory.google.com>). Key-word based search engines can return a ranked list of web pages including all relevant documents, as well as many non-relevant or uninterested contents. Users still have to go through a long list of search results to get their links of interest.

Web page classification is one of the essential approaches of web mining [Rib02, SCZ+04, SK04]. Web pages, as an important interface for retrieving information, have become one of the largest data sets that need automatic methods to handle. Taxonomic web page categories used in web search is one application of in web page classification. In many cases, categorizing web pages into different topics of classes is often the first step of mining the web. For example, to improve correlation of search results, one of the methods is to search upon pre-classified categories, or organize search results. One example of the latter can be seen in [CD00]. The authors claim that their category-based interface is superior to the typical ranked list interface of search results in both accuracy and search time. By organizing search results, users are able to focus on items in categories of interest rather than have to browse through all the results sequentially.

1.3 Web Page Cleaning

Observing a web page, we can say that it consists of many blocks. A block is a semantic part of the web page, which intends to have its own content, style and function.

For example, the CNN web page¹ shown in Figure 1.1 can be seen as a block aggregate involving site heading block, navigation bar blocks, main content block, advertisement blocks, and copyright notice blocks. In this page, only the main content block represents the essential part that users are mostly interested in. Although other blocks are either useful in enriching functionality and vision effect of the page such as the navigation bar, or necessary for site owners such as the advertisements and copyright notice, they seriously affect web mining such as web page clustering, classification, and web search. Therefore, they are regarded as noise blocks.

Figure 1.1: Blocks in CNN web page

Normally, it is easy for a person to differentiate noise blocks from main content blocks in a page manually, while it is not such an easy task for a machine to do

¹ <http://www.cnn.com>

automatically because syntactically, there is nothing to differentiate contents that belong to main blocks from contents that belong to noise blocks. However, noise blocks do have some features that can be identified. First, contents in noise blocks are irrelevant to the main content in a web page. Second, in a given web site, noise blocks appear frequently in a collection of the site web pages. Third, noise blocks share common contents and look and feel in a web site. Fourth, noise blocks are usually located at the edge of a web page. Fifth, noise blocks usually contain many links in their content. For example, two web pages shown in Figure 1.2 both belong to the Future Shop web site, and they present two different products. Noise blocks in these two pages have highly similar styles and contents, such as Future Shop head block, search block, delivery declaration block, company information block, and advertisement block. All of these noise blocks have the features stated above, and appear frequently in the Future Shop web site.

Figure 1.2: Noise blocks in Future Shop web site

In Figure 1.2, only the centre blocks which introduce products are the contents that most people are interested in, and they represent the topics of the whole page that differ from others. Noise blocks may incur topic drift, which is the case when user desired web pages get lower rank than pages that are pointed to by advertisement links in noise blocks. For instance, in web search area, the number of backlinks (i.e., citations) of a web page is an important factor for ranking pages. An example is the PageRank [PBMW98], which is a method for rating web pages when web search is done. It exploits link structure of the web to produce ranking of every page. Similar to citations in academic articles, highly linked pages are more important than fewer linked pages. Therefore, it might be the case that an advertisement web page always gets a high ranking value, because many web pages have links pointing to that advertisement page. It can be seen that links to these advertisement pages constitute noise blocks that will disturb web mining and search, and should be removed. Web page cleaning aims at removing noise blocks from web pages in order to improve the precision and efficiency of web mining and web search.

To remain only important information in documents, some approaches work on extracting data from documents [Ade98, BLP01, GKNG03], others address removing noisy data from documents [Jus99]. In this thesis, we discuss the problem of removing noisy blocks from web pages. The basic idea of web page cleaning is first to segment web pages into a set of blocks, then, calculate the block importance based on its frequency of appearance in these web pages. Then, finally comparing the block weight with a threshold, to decide if this block is noisy or not. Existing literatures [YLL03, BR02, LH02] addressing this problem are discussed in Section 2.1.

1.4 Data Cleaning for Data Warehousing

Data cleaning is an important process in data warehouse integration [CGGM03, Coh98, GFS+01, RD00], and the process of web page cleaning has similar situations with data cleaning at particular points. From the point of data cleaning, when data are to be integrated from heterogeneous databases, data cleaning is responsible for processing data inconsistency, and redundancy phenomena that arose during the integration. In particular, as an important issue in data cleaning, duplicate elimination [ACG02, SLS02] takes on

the task of detecting multiple tuples, which describe the same entity in the real world. From the point of web page cleaning, the task is to remove noise blocks from the page. Because of the frequent appearance feature of the noise blocks in a given web site, most of the noise blocks are duplicated or approximately duplicated. Thus, web page cleaning can be seen as the process of removing duplicate blocks from this point. Since data cleaning has been widely researched, some of the effective methods can be borrowed into web page cleaning, such as the most common sorting and merging [HS95] based approaches.

1.5 Why Web Page Cleaning

As stated in Section 1.2, web page classification is of great importance in practice. As a more general approach that has close relationship to web page classification, text classification has been well learned from the machine learning techniques. It is very natural to have the thought of using text classification techniques to solve the web page classification problem. However, due to a large variety of noisy information in web pages, web-page classification is much more difficult than pure-text classification. If a pure-text classification method is directly applied to web pages, it will incur focus loss and information bias on the main content. Thus, it is critical to pre-process the web pages, such as extracting main content or removing noise blocks, so that text classification techniques can be applied to web page classification with more accuracy.

1.6 Thesis Problem and Contributions

Given a collection of web pages in a web site, the thesis proposes a scheme, called WebPageCleaner, which deals with the problem of eliminating noise blocks from these web pages. It aims at improving the precision and efficiency of web content mining.

Thesis contributes to the web page cleaning problem as follows:

- (1) Employs a vision-based content structure [CYWM03a] for cleaning a large set of web pages in a given web site, which has not been used in previous approaches. This method overcomes the shortcoming of DOM tree approach by providing a semantic content structure.

(2) Unlike other approaches, WebPageCleaner uses both content of a block and other block features (e.g., block position, percentage of link texts) for computing block importance. Our approach detects both blocks with the same contents and near-duplicate contents as noisy blocks.

(3) Gives an enhanced implementation to the template detection method [BR02] by providing different page partitions and detecting near-duplicate blocks as well.

1.7 Outline of Thesis

The rest of the thesis is organized as follows. Related work including page segmentation, string similarity, and duplicate elimination are discussed in Chapter 2. In Chapter 3, thesis presents WebPageCleaner, the proposed technique for web page cleaning. Chapter 4 discusses experimental results and performance analysis. Conclusions and future work are presented in Chapter 5.

Chapter 2 RELATED WORKS

Apart from some existing web page cleaning methods [LH02, YLL03, BR02], the thesis work involves data cleaning for data warehousing [HS95, Her96, EO05], web page partitioning [CYWM03a], syntactic similarity of strings or files [ME96, GIJ+01, CL97, BGM97], and web (text) classifications [FGG97, MN98].

2.1 Web Page Partition and Cleaning

Web page cleaning has close relationships with web page partitioning in that the partition results directly determine the granularity of a block that should be kept or removed. Both large and small block granularity lead to unsatisfactory cleaning results, in which useful contents may be eliminated (over cleaning) or noise blocks are still kept (under cleaning). In this section, some existing web page cleaning methods are discussed. A common feature of these approaches is that cleaning work is done based on a predefined page partition scheme.

2.1.1 <TABLE> Tag-based

In [LH02], the authors proposed a method for discovering informative contents from a set of web pages of a web site. Their system, InfoDiscoverer, consists of the following five modules:

(1) Extracting Content Blocks from a Page. A web page is partitioned into several content blocks according to HTML tag <TABLE> in a page. The authors argue that many (almost 70%) web sites use <TABLE> tag as a layout to present their pages. Also, <TABLE> tags are easy and convenient when segmenting an HTML page into several content blocks.

(2) Extracting Features of Content Blocks. Features are meaningful keywords here. Extracting features is done by using the Porter stemming algorithm [Por80], which is a process for removing the common morphological and inflexional endings from words in English. For example, the word “listing” will become “list” after stemming, words like “a”, “an”, “the” will be removed, and so on and so forth. It is usually a normalization process to set up an Information Retrieval system.

(3) Calculating Entropy Values of Features. Entropy is a basic concept in information theory, which measures the randomness of an event. For example, predicting what the next character is in English text can be seen as a random event. However, there still exists clues for predicting its randomness, such as character 'e' is very common to appear in the English text compared with character 'w'. Therefore, character 'e' will get a higher entropy value than character 'w'. Shannon [Sha48] defines entropy in terms of a random event F , with possible n states, which is formulated as: $H(F) = -\sum_{i=1}^n p_i \log_2 p_i$,

where p_i is the probability of state i in event F . In the case of measuring the entropy value of the feature in the document, the probability of a feature appearing in documents is calculated. By normalizing the feature entropy to the range $[0, 1]$, the above formula is modified as: $0 \leq H(F_i) = -\sum_{j=1}^n w_{ij} \log_n w_{ij} \leq 1$, where w_{ij} is the probability of feature F_i in document D_j . For example, suppose we have a page list consisting of two pages (D_1 and D_2) with features F_1 to F_6 appearing in one or more pages. F_4, F_5 , and F_6 belong to one block, i.e., $D_1 = \{F_1, F_2, F_3, F_6\}$, $D_2 = \{F_1, F_2, (F_4, F_5, F_6)\}$. The entropy value of each feature is calculated as:

$$H(F_1) = H(F_2) = H(F_6) = -\sum_{j=1}^2 \frac{1}{2} \log_2 \frac{1}{2} = 1$$

$$H(F_3) = H(F_4) = H(F_5) = -1 \log_2 1 - 0 \log_2 0 = 0$$

(4) Estimating Entropy of Content Blocks. A content block may contain many features. Thus, the entropy value of a content block, $H(CB)$, is the average of all feature

entropies in the block: $H(CB_i) = \frac{\sum_{j=1}^k H(F_j)}{k}$, where k is the number of features in block CB_i , and F_j is the feature of CB_i . In the above example, suppose F_4, F_5, F_6 belong to one block CB_1 in document D_2 , then, the entropy of CB_1 is $H(CB_1) = \frac{0+0+1}{3} = \frac{1}{3}$.

(5) Classifying Content Blocks. A greedy approach is employed to dynamically select the entropy-threshold according to different sites. Based on $H(CB)$, blocks are partitioned into either informative or redundant: if $H(CB)$ is higher than threshold, or

close to 1, the content block is redundant because most of the features in this block appear in every page; if $H(CB)$ is less than a threshold, the block is informative. The greedy approach is processed by starting the threshold from 0 to 1.0 with an interval such as 0.1, the increase of the threshold will cause including more features since more blocks are included. If increasing threshold does not include any more features, the threshold is obtained at this point.

The InfoDiscoverer system can be used as the preprocessor of Information Retrieval and Information Extraction systems since it can successfully increase the precision of retrieval by reducing the size of index. Under a prior assumption that the page clusters have been learned, the greedy approach is feasible for finding the optimal threshold of block entropy for different web sites with different templates. However, their method is restricted to tabular (with `<TABLE>` tags) web pages, especially for the news web site in which most blocks are collections of words.

2.1.2 DOM-based

A Style Tree structure is proposed in [YLL03] to capture the layout and contents of the pages in a given web site. Based on the Style Tree, the entropy value for each element node in the tree is evaluated to get its importance degree, thus, the unimportant elements (noises) are eliminated.

The Style Tree (ST) is built on the basis of the DOM² (Document Object Model) tree, which provides an interface representing the structure of an HTML or XML document. Considering that the DOM tree can only represent a single HTML page, to get the overall presentation of a web site or a set of pages, the DOM trees are combined to build a Style Tree. For example, the Style Tree in Figure 2.1 (cited from [YLL03]) is the combination of the DOM tree d_1 and d_2 .

The basic idea of building a Style Tree is to combine the common tags, and count their numbers of occurrence, while leaving the distinguishing tags as they are. For example, the `<BODY>` and `<TABLE>`-`<IMG>`-`<TABLE>` tags both exist in d_1 and d_2 , so their count number is 2 in the Style Tree. The sequences of tags in the lowest layer are different in d_1 and d_2 , which are P-IMG-P-A and P-BR-P respectively, so they are set in

² <http://www.w3c.org/DOM/>

the Style Tree separately. With the Style Tree, the common and distinguishable parts for a set of pages are clear to be seen.

Figure 2.1: Building the style tree
(cited from [YLL03])

The node importance is determined by its entropy value, which is similar with the method introduced in Section 2.1.1. If the importance of the node itself and all its descendants is less than a specified threshold, then this node is noisy. The threshold is determined through experiments. For each web site, clean a small number of pages using a number of threshold values, then, select the threshold which leads to the best cleaned pages from observation as the final threshold for this web site.

Their web page cleaning technique greatly improves the results of two web mining tasks, web page clustering and classification. However, the process of building a Style Tree is complex for large number of web sites, because they need to analyze every single DOM node when building the Style Tree.

2.1.3 Hyperlink-based

Bar-Yossef et al. [BR02] proposed a template detection technique. Here, the template is defined as a pre-prepared master HTML shell page that is used as a basis for composing new web pages in a web site. Templates broadly exist in professionally designed web sites. Since pages sharing a template also share a large number of common links, it makes sense that the main function of the template is that of aiding navigation. Thus, templates are the harmful issues in web mining.

The other important keyword in this paper is pagelet. Pagelet is a self-contained logical region within a page that has a well defined topic or functionality, such as advertisement banners, and news headlines. The authors claim that pagelets are the more appropriate unit than pages for information retrieval since they are more structurally cohesive. A syntactic definition of pagelet is given as follows. An HTML element in the parse tree of a page p is a pagelet if (1) none of its children contains at least k hyperlinks; and (2) none of its ancestor elements is a pagelet. In their implementation, k is chosen as 3. So the basic idea is that when an HTML element contains at least 3 links (when $k=3$), it is likely to represent some independent topic; otherwise, it is likely to be topically integrated in its parent.

After partitioning HTML document into pagelets, they can be used to detect a template. Pagelets with the same shingle are considered as a template, and should be deleted. A shingle is a text fingerprint, but it is invariant under small updates to text. Shingling technique is proposed in [BGM97] that will be further introduced in Section 2.2. The partition results are finally stored in the database as relations:

PAGELETS (page_key, pagelet_serial, pagelet_shingle)

Thus, cleaning process involves simply sorting and grouping the PAGELETS according to pagelet_shingle. Each such group represents a template.

The link-based partition technique is simple and feasible for a set of pages from different web sites. However, the partition results totally depend on a pre-defined number of links regardless of the actual layout styles of each web site. This will make the pagelets non-content cohesive of themselves, and consequently result in under cleaning or over cleaning phenomenon. Furthermore, it can only detect exact duplicates because it will remove any pagelets with the same shingle value.

2.1.4 Vision-based

A Vision-based Page Segmentation (VIPS) algorithm is proposed in [CYWM03a, CYWM03b]. It simulates using human visual perceptions in understanding the web layout structure for the purpose of dividing an HTML page into several semantic blocks. People usually view a web page as several parts at a glance without even knowing its contents. The reason is due to human perception enhanced by spatial and visual cues. For example, users always unconsciously assume that the area with the same font and background represents a semantic part (visual cues), and the main contents appear at the centre of a page (spatial cues), etc.

A vision-based content structure will be generated by combining the DOM structure and the visual cues. The algorithm consists of three steps: block extraction, separator detection and content structure construction. Basically, a web page is processed through all these three steps firstly, and gets a coarse hierarchical structure consisting of several content blocks. Then, for each block, applying the same process recursively to get smaller blocks, until the Degree of Coherence (DoC) value for final blocks are greater than a pre-defined DoC (PDoC) value. The DoC value (ranging from 1 to 10) is defined to measure how coherent the content of a block is. The content within the block is more consistent with higher DoC value. The details of the block extraction step are described below.

In general, each node in DOM tree can be seen as a visual block. However, some nodes such as table tag <TABLE> or paragraph tag <P> are too general for representing a consistent block, and need to be further divided. The rules for dividing a DOM tree node or not are based on the properties of the DOM node itself and its children. The properties are obtained according to some important cues, including tag cue, color cue, text cue, and size cue. For example, tags such as <HR> (places a horizontal line across the screen) are often used to separate different topics, and are preferred to be divided. If the background of a DOM node is different from its children, it is divided. If most of the children of a DOM node are text nodes, it is not divided, and so on. Specifically, the heuristic rules are made based on these cues. According to the rules, we can judge if a DOM node should be divided. Some of the rules are listed below; we will use an example to explain it.

Rule 1: If the DOM node is not a text node and it has no valid children, then this node cannot be divided and will be cut.

Rule 2: If the DOM node has only one valid child and the child is not a text node, then divide this node.

Rule 3: If the DOM node is the root node of the sub-DOM tree (corresponding to the block), and there is only one sub DOM tree corresponding to this block, divide this node.

.....

Rule 8: If the background color of this node is different from one of its children's, divide this node, and the child node with different background color will not be divided in this round. Set the DoC value (6-8) for the child node based on its html tag.

.....

Rule 11: If previous sibling node has not been divided, do not divide this node.

.....

For example, Figure 2.2 (cited from [CYWM03a]) shows a fragment of HTML page with its corresponding DOM tree. The above rules are used to divide this DOM tree. When the <TABLE> node is met, since it only has one child node <TR>, according to rule 2, it can be traced into <TR> node. The <TR> node has five children and three of them are valid. For the first <TD> child, its background color is different from its parent's background color, so the <TR> node is divided according to rule 8. The first <TD> node is not divided in this round and it is put into the pool as a split block, thus creating the block VB2_1. The second and the fourth <TD> nodes are not valid, so they are cut. Since the first, the third and the fifth <TD> nodes are siblings, according to rule 11, they are not further divided in this round, and are put into the pool. Finally, three blocks VB2_1, VB2_2, and VB2_3 are obtained in this round.

Each extracted block will be checked to see whether it meets the granularity requirement. That is, if the DoC value of a block is greater than a pre-defined PDoC

value, then, this block is ready for output; otherwise, the above process will be recursively done until it satisfies the PDoC requirement.

Figure 2.2: Block extraction of an HTML page
(cited from [CYWM03a])

The second (separator detection) and the third (content structure construction) steps are used to generate a hierarchy structure of semantic blocks. After extracting blocks from the first step, horizontal or vertical lines are inserted between blocks to separate them, and each separator gets a weight based on some visual cues. The weight is used to identify the coherence between the neighboring blocks. Then, the content structure construction step works for merging blocks with light weight to generate higher level blocks. A semantic content tree will be generated finally. All contents in the leaf nodes consist of a full content of the whole web page. This thesis is more concerned with blocks extracted in the first step rather than a hierarchy structure.

The VIPS algorithm has been used in web information retrieval, WWW image search, and learning block importance on a single page [SLWM04]. It has not been used in web page cleaning for a collection of web pages. Obviously, the content structure it

generated is a more aggregate one than DOM, and it overcomes the disadvantage of DOM structure in that each of its nodes represents a semantic content. Also, it is efficient since it just traced down the DOM tree, and does not need to analyze each node in DOM tree. In web page cleaning, we expect to have blocks with appropriate size and that are content coherent. Therefore, the VIPS algorithm can provide an ideal page segmentation technique for web page cleaning.

2.2 Strings and Files Similarity Detection

Since we intend to consider that noise blocks share common contents, it is necessary to use string or file similarity techniques to detect this kind of similarity in content. In this section, some string matching techniques will be introduced first, then, a shingling technique for finding files similarity is described.

2.2.1 The Basic Field Matching Algorithm

The basic idea of field matching is described in [ME96]. “A simple definition of the degree to which two fields match is the number of their matching atomic strings divided by their average number of atomic strings”. In the basic algorithm, two atomic strings match if they are the same string or if one is a prefix of the other. The implementation of the algorithm is straightforward. First, extract atomic strings from fields. Second, sort atomic strings in each field. Third, search each atomic string of one field in the other field’s strings. Then, record the number of matched atomic strings. Last, substitute recorded values into the formula: $k / ((|A| + |B|) / 2)$, where k is the number of matched strings, A and B are fields to be compared. For example, consider the following fields:

$A = \text{“Comput. Sci. \& Eng. Dept., University of Windsor, ON”}$

$B = \text{“Department of Computer Science, Univ. Windsor, Ontario”}$

Before matching the two fields, stop words in the set {and in for the of on & - /} are first removed. After that, $k (=5)$ strings in the first field match some strings in the second field, they are: “Comput”, “Sci”, “Univ”, “Windsor”, “ON”. The number of atomic strings in field A and B is 7 and 6, respectively. Therefore, the overall matching score is:

$$k / ((|A| + |B|) / 2) = 5 / ((7 + 6) / 2) = 0.77$$

In an integrated cleaning process, this matching score (or edit distance) is supposed to be compared with a threshold, which is a value given by people according to some user observations or experiment results. Strings with greater distance scores than the threshold will be picked as matched fields. The basic field matching algorithm is very easy to understand and to use, but it does not take into account abbreviations which are not prefixes.

2.2.2 Edit Distance among the Q-grams

Edit distance is a basic notion used in comparisons of string similarity. The general definition of edit distance is given in [GIJ+01]: the *edit distance* between two strings is the minimum number of edit operations (i.e., *insertions*, *deletions*, and *substitutions*) of single characters needed to transform the first string into the second. The edit distance function described in [CGGM03] is that the edit distance between two strings s_1 and s_2 is the minimum number of character edit operations required to transform s_1 into s_2 , normalized by the maximum of the length of s_1 and s_2 . For example, the edit distance between two strings “company” and “corporation” is $7 / 11 = 0.64$. Because we need at least seven steps to transform “company” into “corporation” by substituting ‘r’ for ‘m’, inserting ‘o’, ‘r’, ‘t’, ‘i’, ‘o’ and deleting ‘y’, and the maximum length of two strings is the length of “corporation”, which is 11.

Q-grams, which are the short substrings of length q , can be used to make edit distance calculation more efficient. For example, suppose $q=3$, the 3-gram set of “address” is {add, ddr, dre, res, ess}. The core of the method that uses edit distance among q -grams relies on matching q -grams (short substrings of length q) of pair of strings, and applies them to possible edit distance functions to measure their similarity. In the case of comparing strings from attributes of two databases, cross product of two tables is required. Then, the computations of edit distance are applied to every record in the cross product. This is a very expensive operation. Therefore, to find a compromise between the computation time requirement and the correctness of the results is always a challenge in approximate string matching field. Many approaches are seeking a better solution to

improve the matching efficiency. The authors take into account token weights in [CGGM03] so that only tokens with high weights, instead of all q-grams, are involved in the computation of edit distance. We will illustrate token weights in the next section.

2.2.3 Cosine Similarity Metric with IDF Weighting

The cosine similarity metric was proposed by Cohen in [CL97] based on the consideration that the most expensive cost in string matching is spent on the computation of the dot product between a query vector and a large instance vectors in a database. Among all these dot products, we are interested mainly in instance vectors that have high dot products. So, the computation of all dot products is a waste. The authors present an algorithm based on a random sampling method, which avoids explicitly computing all dot products.

In [GIKS03], the authors perform textual matching via the cosine similarity metric, which is formalized as:

$$sim(v_{t_1}, v_{t_2}) = \sum_{j=1}^{|D|} v_{t_1}(j) v_{t_2}(j)$$

, where v_{t_1}, v_{t_2} are normalized weight vectors of two tuples t_1 in relation R_1 and t_2 in relation R_2 , and D is the set of all tokens in R_1 and R_2 .

Two terms of weight are commonly used: term frequency (tf) and document frequency. Term frequency indicates that words that repeat multiple times in a document are important. Document frequency means that words that appear in many documents are common and are not important for document. This is usually assigned by Inverse Document Frequency (IDF) weights, which has been successfully exploited in information retrieval community [Sin01]. It is intuitive that the more important tokens two vectors share, the more similar they are. For example, “CIBC Bank” and “TD Bank” will have lower similarity as they share a common token “Bank”, which has low weight as it frequently appears in many different tuples. While “CIBC” and “CIBC Bank” will have high similarity, since “CIBC” turns out to have high weight. To get tuple pairs with high similarity, we are concerned only with those that have high values of $v_{t_1}(j)v_{t_2}(j)$. Intuitively, either $v_{t_1}(j)$ or $v_{t_2}(j)$ should have high value. This means that we can effectively ignore tokens of low weights.

2.2.4 Document Similarity

Document similarity detection has become increasingly important with the expansion of the web. The applications include searching documents online, copy detection, duplicate mirror site detection, and so on. In many cases, documents are approximately duplicated, such as the plagiarism documents. In [BGM97], resemblance is used to measure roughly the same documents. To compute resemblance, each document is viewed as a sequence of words by removing all formatting, HTML tags, and capitalization. Then, each document is associated with a set of subsequence of words. A continuous subsequence contained in D is called a shingle. Given a document D , its w -shingling $S(D, w)$ is defined as a set of all unique shingles of size w contained in D . For example, the 2-shingling of “no pain, no gain” is $\{(no, pain), (pain, no), (no, gain)\}$. For a pre-defined shingle size, the resemblance of two documents A and B is defined as:

$$r(A, B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$

For instance, given $A = \text{“no pain, no gain”}$ and $B = \text{“no pain means no gain”}$, assume the single size is 2, the 2-shingling of B is $\{(no, pain), (pain, means), (means, no), (no, gain)\}$, thus, $r(A, B) = 2/5 = 0.4$. The resemblance is a number ranges from 0 to 1. The closer it is to 1, the more similar two documents are.

Each shingle is mapped to a number, called fingerprint, which is calculated by Rabin’s fingerprint technique [Rab81]. Fingerprints are short tags for larger objects [Bro93]. The relationship between an object and its fingerprints follows two rules: (1) If two objects are equal, their fingerprints are also equal. (2) If two objects are not equal, the probability that they get the same fingerprints is very small. Rabin fingerprint technique considers an m -bit string as degree- m polynomials over Z_2 . Suppose $A = a_1a_2 \dots a_m$ is a binary string, it can be associated with a polynomial $A(t)$ of degree $m-1$ with coefficients in Z_2 . Therefore, $A(t) = a_1t^{m-1} + a_2t^{m-2} + \dots + a_m$. For example, the binary string “1101” can be represented as $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$.

The fingerprints of A is defined as $f(A) = A(t) \bmod P(t)$, where $P(t)$ is an irreducible polynomial of degree k over Z_2 . The irreducible polynomial cannot be factored into nontrivial polynomials over the same field. For example, over the field of

rational numbers, $P(x) = x^2 - 4 = (x - 2)(x + 2)$ is reducible, but $P(x) = 1 + x$ is irreducible. It is helpful to compare irreducible polynomials to prime numbers, which are irreducible integers.

The implementation of Rabin's fingerprint is efficient, and can be computed in linear time. We can compute $f(a_1, \dots, a_{m+1})$ from $f(a_1, \dots, a_m)$. If

$$f(a_1, \dots, a_m) = (a_1 t^{m-1} + a_2 t^{m-2} + \dots + a_m) \bmod P(t) = r_1 t^{k-1} + r_2 t^{k-2} + \dots + r_k$$

then

$$f(a_1, \dots, a_{m+1}) = (f(a_1, \dots, a_m)t + a_{m+1}) \bmod P(t) = r_2 t^{k-1} + r_3 t^{k-2} + \dots + r_k t + a_{m+1} + (r_1 t^k) \bmod P(t)$$

This consists of one shift left operation to $f(a_1, \dots, a_m)$, where a_m is input as the least significant bit, and one xor (exclusive or) operation to P with the leading coefficient removed. The xor operation is conditional upon $r_1 = 1$.

For example, suppose A is a binary string "1001", and P is a prime number 0111 (number 7). If we know that the fingerprint of A , $f(A) = f(1001) = (1001) \bmod (0111) = 0010$, then, the fingerprint of B "10011" can be easily computed from $f(A)$. By a shift left operation to $f(A)$, and the least significant digit of B '1' as input, we got 0101, which is the final result of $f(B)$.

Fingerprints can be viewed as a hash value of the document for efficiently identifying different objects. Using fingerprints, we can make fast calculation, also it will bring low collision rate for the hash value.

To estimate the resemblance efficiently, each document D keeps a sketch, which consists of a set of shingles. Suppose the shingle size is w , we can use the "modulus" method for reducing the number of singles, while keeping an unbiased estimate of the resemblance of two documents. The method is to hash all shingles of size w to a set of numbers, let U be a set of numbers, $MOD_m(U)$ be the set of elements of U that are 0 mod m , and $f: U \rightarrow U$ be a permutation chosen uniformly and randomly from U , define $V(A) = MOD_m(f(S(A)))$. Then, the value $\frac{|V(A) \cap V(B)|}{|V(A) \cup V(B)|}$ is an unbiased estimate of the resemblance of document A and B .

In [BGM97], the authors set the shingle size w to 10. They first remove all HTML formatting and convert all words to lowercase. Getting shingle values based on

Rabin fingerprints [Rab81] function is the next step. Then, selecting the value of m to be 25, and using the modulus method to get the sketch of the document. After calculating the sketches for each document, it compares the sketches for each pair of documents to see if they exceed a threshold of resemblance. Finally, it combines the pairs of similar documents. However, comparisons between each pair of sketches is impractical since there may exist hundreds of thousands of documents for processing. Therefore, a “divide, compute, merge” method is used. The data are divided into pieces, computation performed on each piece, and the results are merged.

2.3 Duplicate Elimination for Data Warehousing

After extracting web page blocks, and storing them as a flat database format, we can borrow some ideas from duplicate elimination for data warehousing. Duplicate elimination is also known as merge/purge problem [HS95]. It is an important issue in data cleaning that takes on the task of detecting multiple tuples, which describe the same entity in the real world [BD83, HS95, ME97]. Approximate string matching methods discussed in previous section are employed in most duplicate elimination approaches as a necessary part of solutions. Besides, most duplicate elimination implementations emphasize on reducing the number of comparisons between pairwise records so that the expensive string matching computation could be avoided. The common method under this motivation is the sorting and merging based approaches. This was first proposed in [BD83], and was further developed in [HS95, Her96, ME97].

This section reviews several primary approaches in literature that are used to eliminate duplications in data warehouses. Start with the naïve method, an in-depth discussion of sorting and merging based methods are also given.

2.3.1 Naïve Method

Given one or more data sources, to get their integrated and consistent data warehouse requires eliminating data quality problems. One such data quality problem is duplicate elimination, which can be described with a plain mean: first, join records to get their Cartesian product which will take a quadratic time process [HS95], and thus produces a dirty database. Second, sort the database to bring all duplicated records

together. Third, compare every record with every other record to detect duplicates. This is a reliable but infeasible way when the number of records is large, which is always the truth for sure. Therefore, many approaches devote to improving the naïve method for decreasing the execution time, as well as increasing the number of correct duplicate results.

2.3.2 Sorting and Merging Based

Most duplicate elimination methods are actually an extension of the naïve method, which are based on sorting and merging. The early approach was brought by [BD83], in which the authors utilize the intuition that duplicated records will come together after sorting. In the rest of this section, we will introduce Sorted Neighborhood Method (SNM), and its two alternative algorithms [HS95]. Also, we will put emphasis on the Duplicate Elimination Sorted-Neighborhood Method (DE-SNM) [Her96], which is an improvement of SNM. A token-based technology [EO05] will also be illustrated.

2.3.2.1 Sorted Neighborhood Method (SNM)

[HS95] proposed a Sorted Neighborhood Method (SNM) to obtain efficient execution. The records are first sorted over the chosen important key, then, the comparison of records is brought to a close neighborhood. The SNM can therefore be summarized in three phases: create keys, sort data and merge. The merge is applied by limiting the pairwise comparisons in the fixed size window. The key chosen is a crucial step here, which determines the accuracy of the results. For example, record 1 and 2 in the Table 2.1-1 are duplicated, if we choose the Address field as the key to sort the database, then these two records will be very far apart after sorting. Because one starts with alphabet “Apt”, while the other starts with numeric string “3-1131”. If we choose PID field as the key, the chance of record 1 and 2 being brought closer will be much higher.

Record	Name	PID	Age	Address
1	Mary Hong Liu	11012908	23	Apt-3, 1131 Box Ave, Windsor, ON
2	Liu Mary Hong	11012908	23	3-1131 Box Ave. Windsor, ON

Table 2.1-1: Duplicate records in a database.

Two alternative algorithms are proposed based on SNM in [HS95]. One is based upon clustering. It first maps the records into n-dimensional clusters using an n-attribute key extracted from the record, then applies SNM independently on each cluster. This takes less time than using SNM alone since records are partitioned into small clusters. It brings negative effect of decreasing the accuracy of results in that two duplicated records may be mapped in two different clusters.

The other SNM-based method is the multi-pass approach. This strategy executes several independent runs of the SNM, each time using a different key and a relatively small window. The results are the union of all pairs of each running and inferred ones by applying the transitive closure (i.e., if A and B are duplicate pair, B and C are also duplicate, then A and C are duplicate). The multi-pass approach drastically increases the number of correct duplicated results. However, the side effect is that the number of false duplicated results is also increased because transitive closure will propagate the errors.

2.3.2.2 Duplicate Elimination Sorted-Neighborhood Method (DE-SNM)

Duplicate Elimination Sorted-Neighborhood Method (DE-SNM) proposed in [Her96] is an improvement of the SNM method in [HS95]. Given two or more databases, DE-SNM method concatenates them into one sequential list at first. Then it goes through the following steps:

1. Create keys: extracting fields or portions of fields as a key for each record.
2. Sort data and eliminate duplicates: sort and merge the data in the list using the key created in step 1, separating the sorted output into two lists: “duplicates list” and “no-duplicate list”. In the “duplicate list”, put all records for which duplicate keys are detected. In the “no-duplicate list”, put all other records that do not share any key with others, i.e. records with a unique key.
3. Sort the duplicate list: in step 2, the “duplicate list” is generated incrementally, it may not be done in order. Therefore, we need to sort it using the key to get a sequential list of duplicate records.
4. First window scan: to limit the comparisons of records, move a “small” window through the list of duplicate records, suppose the size of the “small” window is k, move downward the window over the duplicate list to let a new record enter the

window and to let the first record in the window slide out of the window each time. The new record entering the window is compared with the previous $k-1$ records in the window to find matching records. If the key of the new record is not matched with the keys of all the other records in the window, appending this new record to the “returned list” of records, then, move the window $k-1$ positions, making the new record the first one in the window. Also append to the “returned list” the record that was matched the most (at least once) with other records with the key. This kind of record is the “prime representative” of its key that will be used in later steps.

5. Merge: merge the “returned list” of records got in step 4 with the records in the “no-duplicate list”. Add an extra bit field to identify where a record comes from: “returned list” or “no-duplicate list”.
6. Second window scan: move a fixed sized (e.g. size w) window through the list produced in step 5. Similar to the first window scan, every new record that enters the window is compared with the previous $w-1$ records to find matching records. If the new record comes from the “returned list”, then it is compared only with records that come from “no-duplicated list”. The reason is that those records in the “returned list” were compared during step 4 in the first window scan, and were found to be “non-matching”. On the other hand, if the new record comes from the “no-duplicate list”, it should be compared with all the previous $w-1$ records in the window.

To understand the previous steps, let us walk through an example. Suppose a data table shown in Table 2.1-2 is to be cleaned. Suppose the “name” attribute is selected as the sort key. After the second step, the data table in Table 2.1-1 is divided into two tables. One contains duplicate records; the other contains all other records (i.e. no-duplicate records). In this case, tuples set {002, 003, 004, 005, 006, 007, 008, 009} should be in the first table (duplicate table), while tuples {001, 010} are in the second table (no-duplicate table). After the first window scan (step 4), suppose an ideal string matching algorithm is used, pairs {002, 003}, {004, 005}, {004, 006}, {007, 008}, {007, 009} will be found as matched and are supposed to be merged. The “returned list” will be generated at the same time, append representative of each key in the “duplicate list” to it. In this case, it would be tuples {002, 004, 007}. Next comes step 5, by merging the “returned list” and “no-duplicate table”, we can get table containing tuples {001, 002,

004, 007, 010} (Table 2.1-3). Here, in the field “ComesFrom”, ‘0’ represents “no-duplicate table”, and ‘1’ means “returned list”. Next, the second window scan (step 6) is applied to the table shown in Table 2.1-3. Based on rule of step 6, tuples 001, 010 are supposed to be compared with tuples 002, 004, 007 respectively. As the result, pairs {001, 004} and {002, 010} are matched and merged. The final table after cleaning would be the tuples {001, 002, 007}.

ID	Name
001	Hernandes, M
002	Hernandez, A
003	Hernandez, A
004	Hernandez, M
005	Hernandez, M
006	Hernandez, M
007	Herndat, F.
008	Herndat, F.
009	Herndat, F.
010	Hernsnde, A

Table 2.1-2: Table with unclean data

ID	Name	ComesFrom
001	Hernandes, M	0
002	Hernandez, A	1
004	Hernandez, M	1
007	Herndat, F.	1
010	Hernsnde, A	0

Table 2.1-3: Table after merging

2.3.2.3 A Token-Based Data Cleaning Technique (TB Cleaner)

A token-based data cleaning algorithm, TB Cleaner, was proposed in [OE03, EO05]. It pre-processes records before sorting and merging them. Pre-processing method refers to the operations such as data type checks, format standardization, data fields tokenization, and inconsistent abbreviations resolution etc. [LLK99]. It effectively increases the chances of finding duplicate records.

The algorithm consists of four steps listed below:

1. **Selection and ranking of fields:** selecting and ranking two or three fields that typically represent an entity. For example, in a banking application, we can pick “Birth”, “Name”, and “Address” as ordered selected fields.
2. **Extraction and formation of tokens:** a divisible element is decomposed into several tokens. These tokens are then recomposed in a desired order for the element. For example, a date format “15-Oct-2004” is decomposed into three tokens: “19”, “Oct”, and “2004”. The final token sorted in ascending order is “041015”. Detailed decomposition and recombination rules are given for numeric, alphabetic, and alphanumeric tokens.
3. **Sorting of tokens:** tokens obtained in step 2 are sorted on the selected and ranked fields given in step 1 separately. Each sorting result may obtain different neighbor records. These duplicate detection results are eventually combined to give the final results.
4. **Duplicate detection, elimination and generation of warehouse identification (WID):** the WID is generated from the first record in the duplicate set obtained in step 3 by concatenating the most important tokens used in sorting the table of tokens, which are obtained from steps 1 and 2. For example, suppose we select Date and Name as the two most important tokens in an application, and suppose there is one record with the date “15-Oct-1999” and the name “Tom Smith”, we can get token “101599” for the date and token “ST” for the name. Therefore, the final WID is “101599ST”. This new WID will replace the old WID as the warehouse identification in a duplicate-free and cleaned table.

The contribution of TB Cleaner is that it drastically lowers the dependency of data cleaning on match “threshold” choice. It first defines smart tokens composed from most

important fields of records, which are used for identifying duplicate records. The experimental results show that token-based technique achieves better result than record-based technique, and a high recall is achieved by using TB Cleaner.

2.4 Document Classification

The rapid growth of online document has made document classification a very important task in Information Retrieval and text mining [LG94]. Document classification provides an efficient manner for searching and browsing by organizing large bodies of text. For example, a user study conducted by Chen et al. [CD00] shows that searching in a topic-grouped class is more efficient than searching in documents that may contain all kinds of topics.

Given a predefined set of categories and a set of documents, the task of document classification is that of assigning the most likely category to each given document based on its contents. The state of the art text classification methods include Naïve Bayes (NB) [LIT92, FGG97, MN98], k-Nearest Neighbor (K-NN) [YLY03], Support Vector Machines (SVM) [CV95, Joa98], etc.

In this thesis, we will evaluate our web page cleaning results with Naïve Bayes text classification method, which is one of the most popular approaches for solving text classification tasks.

2.4.1 Naïve Bayes Text Classification

Naïve Bayes text classification applies Bayesian theorem on text classification task. Despite its simplicity, Naïve Bayes classifier has been proved to perform document classification very well by many approaches [FGG97, MN98].

The simplicity of Naïve Bayes Classifier comes from the independent assumption that all words in the documents are independent of each other, and the lengths of documents are independent of document class. The property of independent is that if event A and B are independent, then $P(A, B) = P(A)P(B)$. Suppose we have a document D, which consists of a set of unique words w_1 through w_n , then, the probability of a document D, given a class C, is $p(D | C) = p(w_1, w_2, \dots, w_n | C)$. Since we have assumed

that all words are independent of each other, so $p(w_1, w_2, \dots, w_n | C) = \prod_{i=1}^n p(w_i | C)$. That

$$\text{is, } p(D | C) = \prod_{i=1}^n p(w_i | C).$$

However, the problem of text classification is that of finding the probability of categories given a document, i.e., $p(C | D)$. According to Bayes Rule,

$$p(C | D) = \frac{p(C)p(D | C)}{p(D)} = \frac{p(C)}{p(D)} \prod_i p(w_i | C).$$

Assume there are two classes, C and $\neg C$ (complement of C), $p(C) + p(\neg C) = 1$.

Then,

$$p(C | D) = \frac{p(C)}{p(D)} \prod_i p(w_i | C), \quad (1)$$

$$p(\neg C | D) = \frac{p(\neg C)}{p(D)} \prod_i p(w_i | \neg C) \quad (2)$$

Dividing equation (1) by (2) to remove P(D) will give:

$$\frac{p(C | D)}{p(\neg C | D)} = \frac{p(C)}{p(\neg C)} \prod_i \frac{p(w_i | C)}{p(w_i | \neg C)} \quad (3)$$

By taking logarithm of equation (3) and the fact that $p(C | D) = 1 - p(\neg C | D)$, it gives that:

$$\ln \frac{p(C | D)}{1 - p(C | D)} = \ln \frac{p(C)}{1 - p(C)} + \sum_i \ln \frac{p(w_i | C)}{p(w_i | \neg C)} \quad (4)$$

After some arithmetic transitions, our probability of interest $p(C|D)$ can be calculated as:

$$p(C | D) = \frac{\exp(\ln \frac{p(C)}{1 - p(C)} + \sum_i \ln \frac{p(w_i | C)}{p(w_i | \neg C)})}{1 + \exp(\ln \frac{p(C)}{1 - p(C)} + \sum_i \ln \frac{p(w_i | C)}{p(w_i | \neg C)})} \quad (5)$$

, where $p(C)$ and $p(w_i|C)$ are learned from a set of training data.

Naïve Bayes classifier greatly simplifies the learning process because of the independence assumption. And this advantage is highlighted especially when there are large numbers of attributes in the model, i.e., in a document classification task where

attributes correspond to words, and the numbers of attributes (words) are usually very large.

2.4.2 Web Page Classification

Web page classification provides categorized information for effective web search and user navigation. From the first sight, web pages can be classified by applying document classification techniques directly. However, web pages often contain noisy contents such as advertisement blocks, navigation bars, etc, especially in those commercial web sites. When the above Naïve Bayes text classification method is directly applied to web page documents, words that typically belong to noisy contents will impose bias on the main content of the web page. Thus, a preprocess step that removes noisy contents from the web pages is beneficial to web page classification.

Although some work has been done on web page cleaning, as introduced in Section 2.1, few of them evaluate the effectiveness of their cleaning on web page classification. In this thesis, in addition to contribution to web page cleaning discussed in Section 1.6, where noisy blocks are removed from the web pages, thesis algorithm also directly forwards the cleaned results to a pure text classification method. This will give tangible evidence that the proposed web page cleaning method greatly improves the accuracy of web page classification.

Chapter 3 PROPOSED TECHNIQUE FOR WEB PAGE CLEANING

This chapter gives details of the proposed technique for web page cleaning, which is named WebPageCleaner. The problem of eliminating noise blocks (e.g. navigation bar, advertisements, etc) from the web pages of a given web site is addressed. The aim is to improve the web content mining (typically for web page classification in this thesis) results through WebPageCleaner. Thesis claim is that dealing with web pages from one web site is much more efficient than directly processing a mixed collection of web pages from many web sites, because the presentation style in one web site is strongly represented in its own collection and more easily captured. After cleaning pages in each web site, cleaned files of different web sites can simply be merged.

Given a set of web pages from a web site, the whole cleaning work consists of three modules:

- (1) Block Extraction. It is used to segment each web page to be cleaned, and to extract the contents and other features of blocks for storing them in the database for further analysis.
- (2) Block Importance Retrieval. In this step, blocks get their importance degrees according to the similarity of contents, position on the web page, and percentage of their link texts.
- (3) Cleaned Files Generation. The blocks with high importance degrees on each web page are grouped into files, and made ready to be used for web content mining. Each module is described below. The process can be depicted as shown in Figure 3.1.

Figure 3.1: Overall process of cleaning web pages

The proposed scheme is evaluated with web page classification. The experimental results show that the classification using cleaned web pages greatly improves classification efficiency by increasing accuracy and decreasing number of words in classification. Web cleaning approach proposed in thesis is also compared with the hyperlink-based template detection method proposed in [BR02].

3.1 Block Extraction

As introduced in Section 2.1, there are several web page cleaning techniques that partition the web pages first in their implementation. Generally, three types of page segmentation methods can be seen in the literature: tag-based, hyperlink-based, and vision-based. A <TABLE> tag-based method is used in [LH02]. Their system, InfoDiscoverer, first segments a page into content blocks based on <TABLE> tag, then, the entropy of blocks is calculated based on the entropy value of features in each block. However, this method is restricted to tabular (with <TABLE> tags) pages, and the system is based on the assumption that it knows which blocks are the same in different pages.

In [BR02], web pages are seen as templates, which is a collection of pagelets. Pagelets are defined as a region of a web page that has its own topic or functionality. Web pages are partitioned into several pagelets based on the number of links an element has. Then, the template detection is transformed to detect duplicate pagelets. Although the hyperlink-based segmentation method is simple, its results are not always content organized. This is because it only uses the numbers of links as partition criterion, while ignoring the layout styles that a web site typically has, which can be an important clue for semantic partition. Furthermore, their technique assumes that only pagelets with same single values are templates, keeping other pagelets as useful contents. However, in some situations, pagelets with different single values are also templates. For example, navigation links “Home->Software->SnagIt”, and “Home->Software->Outlook” will get different shingle values, and are kept when they should be regarded as templates, and removed.

Another noisy eliminating technique is proposed by Yi et al [YLL03]. A Style Tree structure is constructed for a collection of web pages from one web site. According to the numbers of appearance of each node in Style Tree, the importance of each node is calculated for deciding whether noisy or not. Their experiments show that a great improvement on accuracy is achieved for web clustering and web classification. However, the Style Tree is complex to be built when the number of web sites is large.

A vision-based page segmentation (VIPS) method is proposed in [CYWM03a]. It represents a web page as a semantic content tree by combining the DOM tree and the visual cues. The partition results effectively keep related content together, and the partition granularity is adjustable using different Pre-defined Degree of Coherent (PDoC) value. Their paper is only concerned with the web partitioning method and is not a web page cleaning technique. Their technique can be used in many web applications such as information retrieval, and information extraction.

In [SLWM04], a model for learning block importance is proposed. It first partitions the web pages into different segments with VIPS algorithms, then, extracts spatial and content features for each block. Learning algorithms are used to train the model to assign important values to each segment in web pages. However, their method tries to detect important regions in a single web page, it is different from this thesis

problem, which tries to remove noise blocks from a collection of web pages. Furthermore, they do not take block contents similarity into consideration.

This thesis introduces WebPageCleaner, which employs the VIPS algorithm to implement web page segmentation. We observe that when PDoC value is set to a value greater than 5, the contents in each block start getting clearly aggregated. Small PDoC value results in big block, which is too coarse to use. Large PDoC value results in finer granularity, but it also greatly increases the number of records for duplicate detection. In the experiments, we set the PDoC value to 6 to see the cleaning effects on performance and accuracy.

After getting the semantic content tree using VIPS, we extract each block in the leaf nodes, and extract several features including ids, contents, positions, and percentage of linkages. A set of features for every block is stored in database relationship as follows:

Blocks(PageID, BlockID, BlockText, Fingerprint, PosLevel, LinkPer,
SimilarLevel, ImLevel)

, in which PageID denotes the page being processed. BlockID represents the blocks extracted from the page. BlockText contains all contents in a block. We do some further job to process the contents. First, extract the content from the block. Blocks with contents less than 10 words are removed since small blocks do not contribute to the main contents of the page. Then, remove the punctuations and transform all letters into lowercase. By doing this, the contents consist of a sequence of words, which are separated by white space. For example, after removing punctuations and ignoring case, the text “Watch an exclusive clip from tonight’s ‘Apprentice’” will be transferred to “watch an exclusive clip from tonights apprentice”. Furthermore, the title of web page is extracted as a special BlockText, with BlockID 0 for each of page because we observe that page title in some web sites provide important information on categories.

Fingerprint can be seen as a hash value of the BlockText attribute, which map string value in BlockText into a number. For implementation in this thesis, we produce a 64-bit ($m=64$) hash value for the content of the block based on Rabin’s fingerprints, which has been introduced in Section 2.2.4. A brief example is that the fingerprint of string “as” is the number 24947 (0x6173). This is obtained by shift left ‘a’ (0x61) 8 bits, and xor with ‘b’ (0x73). If the string is long enough to get a number out the range, the

modular is obtained by mod operation with a prime number. We expect to get better performance when sorting and grouping based on fingerprints for identifying duplicate blocks.

For PosLevel, we first extract four features from the content tree, they are PageRectWidth, PageRectHeight, ObjectRectLeft, and ObjectRectTop. PageRectWidth and PageRectHeight denote the width and the height of the web page respectively. Given that the top left corner of the web page is the grid origin, ObjectRectLeft denotes the distance from block's left edge to page's left edge, ObjectRectTop is the distance between block's top edge and page's top edge. Based on these features, we first calculate block's horizontal (P_x) and vertical (P_y) position to the whole page, which is defined as $P_x = \frac{ObjectRectLeft}{PageRectWidth}$, and $P_y = \frac{ObjectRectTop}{PageRectHeight}$. Thus, P_x and P_y are values between 0 and 1. Since the noise blocks are more likely to be set at the edge of a web page, the closer P_x or P_y is to either 0 or 1, the closer the block is to the edge of the page, then the more likely it is a noise block. To represent the position importance using a single parameter instead of two, we introduce PosLevel, which is defined as

$$PosLevel = 1 - \frac{PosLevel_x + PosLevel_y}{2} \quad (0 \leq PosLevel \leq 1) \quad (1)$$

, where $PosLevel_x$ is a value between 0 and 1, which indicates the horizontal position of a block on the web page, and $PosLevel_y$ indicates the vertical position of a block on the web page. The lower the value of PosLevel, the closer the block is to the edge of web page. $PosLevel_x$ is formulated as:

$$PosLevel_x = \begin{cases} 2P_x & \text{if } 0 \leq P_x \leq 0.5 \\ 2(1-P_x) & \text{if } 0.5 < P_x \leq 1 \end{cases} \quad (2)$$

; $PosLevel_y$ is defined in a similar way by substituting P_y for P_x . Substituting formula (2) into (1), we can get formular (3) as below, in which the value of PosLevel is calculated directly from P_x and P_y .

$$PosLevel = \begin{cases} 1 - P_x - P_y & \text{if } 0 \leq P_x \leq 0.5 \text{ and } 0 \leq P_y \leq 0.5 \\ P_y - P_x & \text{if } 0 \leq P_x \leq 0.5 \text{ and } 0.5 < P_y \leq 1 \\ P_x - P_y & \text{if } 0.5 < P_x \leq 1 \text{ and } 0 \leq P_y \leq 0.5 \\ P_x + P_y - 1 & \text{if } 0.5 < P_x \leq 1 \text{ and } 0.5 < P_y \leq 1 \end{cases} \quad (3)$$

In this way, we get a single position measurement PosLevel, which the closer it is to 1, the less important a block is.

LinkPer denotes the percentage of link texts in a block. To get the value of LinkPer, we extract two features, LinkTextLen (length of hyperlinked texts) and TextLen (length of all texts) from the content tree. LinkPer is formulated as

$$LinkPer = \frac{LinkTextLen}{TextLen}$$

For example, suppose the italic words denote text with hyperlink. Then, in the text “*click here* for full terms”, TextLen is equal to 25, LinkTextLen is 10, so its LinkPer value is 0.4. Since a noise block usually has large percentage of link texts, if the LinkPer is close to 1, there are many links in this block, then it is more like a noise block.

SimilarLevel denotes how similar two block contents are. It will be assigned values in the second module, the initial value is set to 0. ImLevel denotes the overall importance of a block. We assume each block is important at the very beginning, so the initial value of ImLevel is set to 1, and will be assigned new value in the second module of the WebPageCleaner system. The BlockExtraction algorithm is shown in Figure 3.2.

```

Algorithm BlockExtraction()
Input: n Web pages
Output: table Blocks
Begin
    for each page  $W_i$  in n pages
        content_tree = VIPS( $W_i$ , PDoC)
        ExtractFeatures(PageRectWidth $_i$ , PageRectHeight $_i$ )
        for each block j in the leaf nodes
            ExtractFeatures(BlockID $_j$ , BlockText $_j$ , ObjectRectLeft $_j$ , ObjectRectTop $_j$ ,
                TextLen $_j$ , LinkTextLen $_j$ )
            Fingerprint $_j$  = GetFingerprints(BlockText $_j$ )
            Px $_j$  = ObjectRectLeft $_j$  / PageRectWidth $_i$ 
            Py $_j$  = ObjectRectTop $_j$  / PageRectHeight $_i$ 
            PosLevel $_j$  = GetPosLevel(Px $_j$ , Py $_j$ )
            LinkPer $_j$  = LinkTextLen $_j$  / TextLen $_j$ 
            Write2DbBlocks(i, BlockID $_j$ , BlockText $_j$ , Fingerprint $_j$ , PosLevel $_j$ , LinkPer $_j$ ,
                0, 1)
        end for
    end for
End

```

Figure 3.2: Algorithm for block extraction

For example, suppose we have two web pages for processing. Figure 3.3 shows the fragments of these two pages grabbed from Future Shop web site, and the corresponding DOM tree structure for the HP product is shown in Figure 3.4.

Figure 3.3: Two web page fragments and their segmentation results using VIPS.

Figure 3.4: The DOM tree structure of the HP product web page in Figure 3.3

Now we can apply VIPS algorithm on this DOM tree. Suppose PDoC is set to 6, when <table> tag is met, since <table> node is the root node of the sub-DOM tree, according to rule 3 (details of the rules can be seen in section 2.1.4), <table> node is divided, and traced into its first <td> node. Since the background color of <td> node is

different from its child's background color, according to rule 8, the `<td>` node is divided and its child node `<table>` gets a DoC value of 8, since it is greater than PDoC value 6, it is put into the pool as a block VB1. According to rule 11, the second `<td>` node is the sibling of the first `<td>` node, it is not be further divided in this round, but put into the pool as block VB2. In the second round, we just need to process VB2, since VB1 has satisfied PDoC value. For VB2, `<td>` node will be divided according to rule 3, and the tree branch along the first `<table>` tag will be divided until `<td>` node is met, according to rule 4, all of the child nodes of the `<td>` node are text nodes, and their font size and font weight are the same, so the DoC value of `<td>` can be set to 10. The `<td>` node and all its children are put into the pool as block VB2_1. Similarly, the second `<table>` node will be put into the pool as block VB2_2. Therefore, we have got three blocks. The separator between the first block and the other two blocks will get a higher weight than the separator between the second and the third block because of the different background color. Therefore, the second and the third blocks will be merged to form a hierarchical content structure, which is shown in Figure 3.5.

Figure 3.5: The content structure of the HP product web page in Figure 3.3

For experiments in this thesis, we use the PDoC value equal to 6. This is based on the observation that contents in a block are getting clearly integrated when PDoC is set to 6. It makes the block not too big and not too small, which suits the purpose of web page cleaning.

We will extract all leaf nodes in the content tree, which are blocks VB1, VB2_1, and VB2_2. Suppose the two pages are represented as P_1 and P_2 , respectively. After segmenting pages using VIPS algorithm, for page P_1 , we can get page features from the content tree, where PageRectWidth = 780, PageRectHeight = 2001. For block VB1 in P_1 ,

we have ObjectRectLeft = 6, ObjectRectTop = 96, TextLen = 27, LinkTextLen = 15, ID = "1", and BlockText = "Entire Site Advanced Search". So $P_x = \text{ObjectRectLeft} / \text{PageRectWidth} = 6 / 780 \approx 0.008$, $P_y = 96 / 2001 \approx 0.048$, $\text{PosLevel} = 1 - P_x - P_y = 1 - 0.008 - 0.048 = 0.944$. $\text{LinkPer} = 15 / 27 \approx 0.556$. The BlockText is then being processed, such as transforming to lower case and removing punctuations. This step is called stemming. The BlockText after stemming will be "entire site advance search". Fingerprints is then calculated for this text, assume the value is 12178. Similarly, get all of these features for every block. We can get a Blocks table shown in Table 3.1-1.

Page ID	BlockID	BlockText	Finger print	PosLevel	LinkPer	Similar Level	ImLevel
1	1	entire site advance search	12178	0.944	0.556	0	1
1	2-1	home tv video 40 and projection television more 40 and projection television	34098	0.724	0.903	0	1
1	2-2	toshiba 52hm84 52 digital widescreen dlp tv	18902	0.706	0	0	1
2	1	entire site advance search	12178	0.93	0.556	0	1
2	2-1	home computer desktop computer more desktop computer	23412	0.71	0.859	0	1
2	2-2	hp pavilion a810 athlon 64 3300 2.4ghz computer	27681	0.696	0	0	1

Table 3.1-1: Blocks table after doing the block extraction

From the above table, we can see that besides the distinguishable blocks for product title (i.e., block 2-2 in page 1 and 2), other blocks are noise blocks of the web pages, such as the navigation bars in block 2-1 of pages 1 and 2, and the search bar in block 1 of pages 1 and 2. These noise blocks share same or similar contents most of the time, have relatively large LinkPer and PosLevel value. We do not say that how large is large enough to be a noise block, but the LinkPer and PosLevel values of noise blocks are

greater than those of informative blocks usually. Thus, we are able to get the most important blocks among all blocks in a web page. We will show how to get content similarity (SimilarLevel) in next section. According to the content similarity, position and linkages percent, we are able to get the importance degree for each block.

3.2 Block Importance Retrieval

The data sets got from the block extraction step are quite noisy. Three phenomena for the data exist: exactly same, approximately same, and different. Examples of exactly same contents include the heading of the web site, copyright notice, etc. They are noise blocks that should be removed first. The challenge is on how to recognize blocks with approximately same contents. Once we can figure this out, the rest of the blocks with different contents are the distinguishable parts of a page that will be viewed as outputs in our problem.

The most common example with approximately same contents is the navigation bar. For example, in a commercial web site, we can always see the navigation bar for directing users to different products. Typically, the navigation bar for two different products may be like A="Home | Computers | Desktop Computers | Cicero SP4185", and B="Home | Computers | Desktop Computers | eMachines T3624". In this case, their similarity depends on the resemblance value to be calculated. Based on the definition of resemblance (see Section 2.2.4), $r(A, B) = 4/8 = 0.5$.

To differentiate distinct blocks from others, we implement two steps: removing exact same blocks and retrieving block importance for the rest of non-exact same blocks. The first step can be done simply with an SQL statement. The basic idea for the second step is to get the content similarity level (SimilarLevel) for each pair content of blocks first. Then, the block importance level (ImLevel) will be calculated according to the values of PosLevel, LinkPer, and SimilarLevel. Blocks with ImLevel greater than a pre-defined threshold are regarded as important blocks. Our solution is implemented over standard database systems. Therefore, we can make full use of the database engine to remove blocks with exact same contents, as well as borrow ideas from duplicate detection for data warehousing. Details for the two steps are shown below.

(1) Removing exact same blocks. In this step, the blocks with exactly same fingerprints (contents) will be removed. We will simply use an SQL statement to remove exactly content matched records, which is depicted in Figure 3.6.

```

Algorithm RemoveSameBlocks(TB)
Input: table Blocks
Output: table BlockEx
Begin
    DELETE FROM Blocks WHERE Fingerprint IN (
        SELECT Fingerprint FROM Blocks
        GROUP BY Fingerprint
        WHERE COUNT(*) > 1);
End

```

Figure 3.6: Algorithm for removing blocks with same contents

This will remove any record that has at least one duplicate content with it. For instance, the processing results for the Table 3.1-1 are shown in Table 3.1-2. Blocks 1 in page 1 and 2 are deleted because they have exact same contents.

Page ID	BlockID	BlockText	Finger print	PosLevel	LinkPer	Similar Level	ImLevel
1	2-1	home tv video 40 and projection television more 40 and projection television	34098	0.724	0.903	0	1
1	2-2	toshiba 52hm84 52 digital widescreen dlp tv	18902	0.706	0	0	1
2	2-1	home computer desktop computer more desktop computer	23412	0.71	0.859	0	1
2	2-2	hp pavilion a810 athlon 64 3300 2.4ghz computer	27681	0.696	0	0	1

Table 3.1-2: Table Blocks after removing records with the same contents

(2) Retrieving block importance. In the above step, we have removed exact same blocks. For the rest of blocks in table Blocks, we will learn their importance from three

measures: content similarity between each other (SimilarLevel), block positions in a web page (PosLevel), and percentage of links (LinkPer). This is based on the consideration that 1) the more similar block contents, the less important the block; 2) the closer a block is to the edge, the less important the block; 3) the higher the percentage of links, the less important the block. At first, all blocks get the importance level 1, the new block importance level is calculated after getting all values of the three measures. As introduced in the first module, two measures which are PosLevel and LinkPer have been obtained. Therefore, the task in this module is to get value of the block's SimilarLevel so that ImLevel can be calculated finally.

To get the block similarity level, we need to compare contents of each pair of blocks. The similarity is measured by SimilarLevel, which is defined as the number of common tokens over the number of all tokens in two blocks. Tokens for each block are distinct here; same tokens in one block will be merged first. For example, the contents for blocks 2-1 of page 1 and 2 in Table 3.1-2 are C_1 = "home tv video 40 and projection television more 40 and projection television" and C_2 = "home computer desktop computer more desktop computer" respectively. To calculate the similarity between these two blocks, we first remove duplicate words in each block, so the contents are transformed as C_1 = "home tv video 40 and projection television more" and C_2 = "home computer desktop more". Thus, $\text{SimilarLevel}(C_1, C_2) = 2/10 = 0.2$. And the SimilarLevel value for blocks 2-2 of page 1 and 2 is 0, since they do not have any common words.

Theoretically, we need to compare every pair of contents to get their similarity; however, this is not necessary and is infeasible in practice. Since the size of the data sets involved may be large, we need to restrict the comparison times between pairs of records. We make the data sets sort ascending according to the BlockText field to bring similar contents closely. Then, apply a sliding window technique throughout the data sets with the window size 2. That is, compare two neighboring blocks to get their value of SimilarLevel and slide down the window to get one new record in the window each time, the first record slides out of the window. SimilarLevel is a value ranging from 0 to 1, the closer it is to 1, the less important a block is.

The SimilarLevel value is set for both records being compared. When sliding down the window, a new SimilarLevel value is obtained for newly compared blocks. Since the aim is to record the most similar pair of tuples among neighboring tuples, if the new SimilarLevel is greater than the old SimilarLevel, then update the SimilarLevel value with the new one; otherwise, keep the old one. For example, suppose the first two records get SimilarLevel value as of 0.7. When comparing the second and the third records, assume their SimilarLevel is 0.8, which means that the second record is more similar to the third record than the first record. So, SimilarLevel of the second record is updated to 0.8 from original value of 0.7.

Now, we can calculate block importance level, since we have got all three values of PosLevel, LinkPer, and SimilarLevel for each block. We define a formula to get the importance level (ImLevel) as follows:

$$\text{ImLevel} = 1 - \left(\frac{1}{2} \text{SimilarLevel} + \frac{1}{3} \text{LinkPer} + \frac{1}{6} \text{PosLevel} \right) \quad (0 \leq \text{ImLevel} \leq 1)$$

In this formula, we take the SimilarLevel as the most important measurement, then, LinkPer and PosLevel. The reason is that the content is the most distinct feature that differentiates one block from the other. The percentage of links and block positions can be used as auxiliary measurements for deciding the block importance. The closer ImLevel is to 1, the more important a block is. The algorithm for retrieving block importance is described in Figure 3.7.

```

Algorithm BlockImportanceRetrieval( $T_B$ )
Input: table Blocks  $T_B$ 
Output: updated Blocks with SimilarLevel and ImLevel being set
Begin
    SortBlockText( $T_B$ )
    for each record  $R_i$  in table Blocks
        newSimilarLevel = GetSimilarity( $R_i$ .BlockText,  $R_{i+1}$ .BlockText)
        if(newSimilarLevel > SimilarLevel $i$ )
            SimilarLevel $i$  = newSimilarLevel
            ImLevel $i$  =  $1 - \left( \frac{1}{2} \text{SimilarLevel}_i + \frac{1}{3} \text{LinkPer}_i + \frac{1}{6} \text{PosLevel}_i \right)$ 
        end for
    End

```

Figure 3.7: Algorithm for getting similar level of contents

For example, the table in Table 3.1-2 will get the results shown in Table 3.1-3. First, sort records according to BlockText. The first two BlockText is compared and gets the SimilarLevel 0.2 for both of them. Next, the 2nd and the 3rd record are compared, and get the SimilarLevel 0. Because the new SimilarLevel (0) is less than the SimilarLevel of the second record, we will keep the SimilarLevel (0.2) for the 2nd record. And the next comparison goes to the 3rd and the 4th record; they still got SimilarLevel 0. Suppose we record the three importance measurements as sequence “SimilarLevel-LinkPer-PosLevel”, blocks 2-1 in page 1 and 2 in Table 3.1-3 have the importance factors “0.2-0.903-0.724” and “0.2-0.859-0.71”, respectively. So the importance level for block 2-1 in page 1 is computed as:

$$\text{ImLevel} = 1 - (0.2 / 2 + 0.093 / 3 + 0.724 / 6) = 0.478$$

Similarly, the ImLevel value for block 2-1 in page 2 is 0.481. Let us see other two blocks 2-2 in page 1 and 2, which have the importance factors “0-0-0.706” and “0-0-0.696”, respectively. The ImLevel values for them are 0.882, and 0.884. ImLevel is used as the final measurement of whether a block is noisy or not. Blocks with the highest ImLevel values in each page are seen as important blocks, such as blocks 2-1 in page 1 and 2. Details for determining important blocks are given in Section 3.3.

Page ID	BlockID	BlockText	Finger print	PosLevel	LinkPer	Similar Level	ImLevel
2	2-1	home computer desktop computer more desktop computer	23412	0.71	0.859	0.2	0.481
1	2-1	home tv video 40 and projection television more 40 and projection television	34098	0.724	0.903	0.2	0.478
2	2-2	hp pavilion a810 athlon 64 3300 2.4ghz computer	27681	0.696	0	0	0.884
1	2-2	toshiba 52hm84 52 digital widescreen dlp tv	18902	0.706	0	0	0.882

Table 3.1-3: Result table after retrieving block importance

3.3 Cleaned Files Generation

The basic idea of this module is that we want to output blocks with the N highest ImLevel value for each page. This module contains two steps. First, decide the number of blocks (N) in each web page for output. Second, export final cleaned files for classification. We select one or two pages from a web site, and run VIPS algorithm on them to observe how many blocks represent the main content of the whole web page. For example, when Best Buy web site is being cleaned, we set PDoC value to 6, and observe that there are usually two blocks which contain the main content of the page. Therefore, the contents in these two blocks are output of this web page, i.e., $N = 2$. The N value is set for each web site using the same way.

After getting the N value, we sort the database according to ImLevel for each page, the first two records in each page, which have the highest ImLevel values, will be exported. Algorithm for this step is shown below.

```
Algorithm CleanFilesGen( $N$ ,  $T_B$ )
Input: number of important blocks, table Blocks
Output: a collection of files with noise contents removed
Begin
    SELECT PageID, BlockText FROM Blocks
        ORDER BY PageID ASC, ImLevel DESC
    pageid = 1;
    for each record  $i$  in query result  $R$ 
        if (pageid  $\neq$   $R_i$ .PageID)
            count = 0;
        end if
        if (pageid ==  $R_i$ .PageID && count <  $N$ );
            count = count+1;
            WriteContent2File(BlockText $_i$ , File $_{pageid}$ )
        end if
        pageid =  $R_i$ .PageID;
    end for
End
```

Figure 3.8: Algorithm for generating cleaned files

At the end of this module, each input web page will be associated with an output file in which noise blocks have been removed. Besides removing noise blocks, the file is

quite ready to use for web content mining, since it has features of ignoring capitalization, punctuations, and morphology.

For the example in Table 3.1-3, if N is set to 1, then, only blocks 2-2 in page 1 and 2 are exported as clean contents. That is, we will get two files with contents “toshiba 52hm84 52 digital widescreen dlp tv” and “hp pavilion a810 athlon 64 3300 2.4ghz computer”, respectively. Web content mining will then be applied on these files.

Figure 3.9 shows a web page from Best Buy web site. This page is partitioned using PDoC value equals to 6, and we set N to 2. As the cleaning result, the two blocks with red rectangle will be exported. We can see that our algorithm can successfully get the most important blocks in a web page.

[Weekly Ad](#)
[Store Locator](#)
[Help](#)
[Research Centre](#)
[Contact Us](#)
[Best Buy](#)

[Entire Site](#)
[Keyword Search](#)

[Home](#) > [Software](#) > [Hobbies & Self Improvement](#) > [Lifestyle](#)

[ValueSoft MasterCook Deluxe Low Carb V. 7.0](#)
 You're trying to control your weight and re-energize your body by watching your carbohydrate intake. Now you can eat great without the hassle! ValueSoft MasterCook Deluxe Low Carb features delicious, satisfying appetizers, entrees and desserts - all with 10 grams of carbs or less!
 Quantity Remaining: 4
[Add to your wish list](#) [Compare](#)

[Buy](#)
[Online Availability](#)
[In Stock](#)
[Usually ships next business day.](#)
[Check Store Stock](#)

Product Features

FEATURES:

- Satisfying Recipes: Enjoy 1300+ delicious low carb recipes.
- Simplified Meal & Menu Planning Stay On Track! Plan a single low carb meal or entire menus in advance.
- Expert Nutritional Information: Get all the information you need to maintain a low carb diet, and even track sodium, sugars and more.
- Professional Tips & Techniques: No matter your skill level in the kitchen, helpful tips and techniques will have you cooking low carb recipes like a pro.
- Gather Your Favourite Low Carb Recipes: Import your favourite low carb recipes and even share them with friends.
- Shopping Made Easy: Print menus and shopping lists or download them to your PDA.
- Party Planning Made Easy: Plan for 5 or 50 guests with festive entertaining ideas to dazzle your guests.
- Cooking Discussion And Recipe Exchange: Visit our message boards for delicious, user-submitted recipes, plus the answers to all of your questions from the kitchen.

SYSTEM REQUIREMENTS:

- CD ROM: 2X
- CPU Speed: 90MHz
- CPU Type: Intel Pentium
- Disk Space: 75MB
- Display: SVGA 800 x 600
- Memory (RAM): 24MB
- Operating System Compatibility: Win98/ME/XP; Not Mac Compatible

Product Details

Web Code: 10043804
 Mfr. Part Number: VLS10352

While we strive for accuracy, Best Buy reserves the right to make changes at any time, without notice, to prices, descriptions, images and policies. Web store prices, products and promotions may differ from our retail stores. [Click here](#) for full terms and conditions.

About Best Buy Canada	Weekly Ad	Gift Cards	Rebates	Car
BestBuy.com	Store Stock Availability	Best Buy Credit Card	Service Plans	Home Entertainment
Community Relations	Store Locator	Card	PC Upgrades	
Returns & Exchanges	Web Code Search	Price Guarantee	Safe Shopping Guarantee	

YOUR CART

[Your Account](#)
[Order Status](#)
[Wish List](#)
[Gift Cards](#)

Your email address

[Research Centre](#)
[What is Windows Xp](#)
[Media Center](#)
[Edition About?](#)
[Extend Your Digital Entertainment Possibilities](#)
[Digital Entertainment Just Got Cooler](#)
[PC Technology And Digital Entertainment Combined](#)
[Spam...](#)
[What It Is and How to Fight It!](#)
[More topics](#)

[Order Status](#)
[Shipping Rates](#)
[Returns](#)
[Rebates](#)
[Help](#)

© 2000-2004 Best Buy Canada Ltd. All rights reserved. For personal, noncommercial use only.

[Version](#)
[Logon](#)

Figure 3.9: Output blocks in a web page from www.bestbuy.ca

3.4 The Overall Algorithm

All the steps of our web page cleaning algorithm are shown in Figure 3.10. Given a number of web pages in one web site, we segment each page to extract its content blocks. All contents and other features in a block will be stored in a Blocks table. By calculating the similarity level between close blocks after sorting on contents, and combining other factors, including position level and link percentage, we can get the final importance level of each block. Blocks with the highest importance level will be exported as cleaned blocks, and used for web content mining.

```
Retrieve n web pages from a given web site randomly
Algorithm WebPageCleaning(W)
Input: a set W containing n web pages
Output: a collection of files with noise contents removed
Begin
     $T_B = \text{BlockExtraction}(W)$     //  $T_B$  – Blocks table
    RemoveSameBlocks( $T_B$ )
    BlockImportanceRetrieval( $T_B$ )
    CleanFilesGen(N,  $T_B$ )
End
```

Figure 3.10: The overall algorithm for web page cleaning

Chapter 4 EXPERIMENTS AND EVALUATIONS

This chapter evaluates the proposed WebPageCleaner system. Experiments contain two parts: web page cleaning and classification on cleaning results. Since the goal of removing noisy blocks from web pages is to improve web content mining, such as classification and clustering, in this thesis, we use Naïve Bayes text classification method [MN98] to test WebPageCleaner system. For experiments on web page cleaning, the execution speed between template detection method [BR02] (TPL) and WebPageCleaner (WPC) is compared. For experiments on classification, Naïve Bayes text classification is performed on web pages with HTML tags removed and without cleaning (NC), web pages cleaned by template detection method [BR02] (TPL), and web pages cleaned by WebPageCleaner (WPC), respectively. The classification accuracy and speed is compared for these three methods.

4.1 Implementation Environments

Experiments consist of two parts: web page cleaning and classification. Web page cleaning is performed on a PC with 2.39 GHz AMD CPU, 1.00 GB of RAM, running on Windows XP Professional Operating System. Programs are coded in C++ language, and are implemented with Visual C++.NET development tool. The datasets are stored as tables in MS Access database.

We use the rainbow toolkit [McC98], which contains program designed for Naïve Bayes text classification, to classify documents. This program is run on UNIX systems.

4.2 Performance Measure

The performance is measured from two aspects, which are the speed of web page cleaning process, and the performance of classification on datasets before and after cleaning. For web page cleaning process, we record the running time in second between template detection algorithm (TPL) and WebPageCleaner (WPC) when we run the algorithm on each of the web site.

For classification, accuracy and efficiency are measured. Accuracy is measured by the percentage of the number of correctly labeled documents divided by the total number of testing documents.

$$Accuracy = \frac{\text{total number of correct classifications}}{\text{total number of classifications}} \times 100\%$$

Standard Error (SE) is used to measure the variance that occurs between the sample means when a number of different samples are drawn from the same population to build the classifier for several trials. It is defined as the square root of the sum of variance ∂^2 divided by the sample size n , which is shown below. The smaller the standard error is, the more stable the classification is done for each trial.

$$SE = \frac{\sqrt{\partial_1^2 + \partial_2^2 + \dots + \partial_n^2}}{n}$$

Efficiency is measured by the number of unique words in Naïve Bayes classifier model and the running time of classification. The common process of Naïve Bayes classifier is to first read the documents, get unique words in documents (which removes the most and less frequent words), then, based on unique words, the classifier builds a document-word model containing statistics information, such as the number of word occurrences in a document with a specific label. Since word is the basic unit for achieving a model and making classification, using documents that have less number of words can get better efficiency for classification.

4.3 Experiments on Web Page Cleaning

We download 2500 web pages from 4 commercial product web sites, including Future Shop³, Best Buy⁴, CNet⁵, and Amazon⁶. These web pages contain products from 5 categories, including computer, MP3, phone, software, and TV. Detailed distributions of pages are shown in Table 4.1-1.

³ <http://www.futureshop.ca>

⁴ <http://www.bestbuy.com>

⁵ <http://www.cnet.com>

⁶ <http://www.amazon.com>

#	Site Name	Computer	MP3	Phone	Software	TV	Total in site
S1	Future Shop	173	62	140	243	134	752
S2	Best Buy	134	87	157	111	96	585
S3	CNet	161	107	197	74	299	838
S4	Amazon	70	98	47	18	92	325

Table 4.1-1: Distribution of web pages in their web site and categories

These 2500 documents (in .htm format) are inputs to WebPageCleaner. They are organized by web sites. We run WebPageCleaner system (WPC) on each web site separately, that is, on 752 pages in Future Shop, 585 pages in Best Buy, 838 pages in CNet, and 325 pages in Amazon. The cleaned outputs for each web site are combined together as a big dataset (contains 2500 txt files), and are fed into Naïve Bayes classification algorithm. The process is shown in Figure 4.1. Template detection algorithm (TPL) is also performed on each individual web site, and has the same process as shown in Figure 4.1.

Figure 4.1: The whole process of experiments

As described in Chapter 3, WebPageCleaner has three modules in cleaning web pages: blocks extraction, block importance retrieval, and cleaned files generation. Table 4.1-2 and Figure 4.2 show the execution time of the whole process when cleaning different web sites in template detection method (TPL) and WebPageCleaner (WPC).

For WebPageCleaner, we set PDoC value to 6 to apply VIPS algorithm. N value is set to 3, which outputs three blocks in each web page in cleaned files generation module.

Template detection method [BR02] first partitions pages into several pagelets (blocks) according to the number of hyperlinks (k) an HTML element has. It then calculates shingle value (text fingerprint) for each pagelet. Pagelets with the same shingles are detected as templates (noisy blocks), and are deleted. The remaining pagelets are collected as cleaned files of these web pages. In this experiment, we set k to 3, which is the same as the authors used in [BR02]. It means that all children of HTML element in partitioned pagelets contain no more than 3 links (see Section 2.1.3 for details).

Web Sites	Number of Pages	Execution Time (Second)	
		TPL	WPC
Best Buy	585	107	61
CNet	838	230	150
Future Shop	752	143	92
Amazon	325	100	54

Table 4.1-2: Execution time for web page cleaning in table

Figure 4.2: Execution time for web page cleaning in chart

For both template detection method and WebPageCleaner, most of the execution time is spent on web page partitioning (blocks extraction module for WebPageCleaner). By comparing the execution time, we can see that the template detection method (TPL) always takes more time than WebPageCleaner (WPC) does. This is due to the different page partition methods they use. Compared to vision-based partition method (in WPC), which just traces down the DOM tree, and stops whenever all blocks meet the predefined granularity, hyperlink-based method (in TPL) traverses each node in DOM tree to obtain the number of links for each child of this node.

In block importance retrieval module, WebPageCleaner need to remove duplicate blocks, then retrieve block importance degree; while template detection algorithm only need to remove duplicate blocks. Thus, WebPageCleaner usually takes one more seconds than template detection method in this step, which is spent in calculating block importance. In this experiment, block importance retrieval module takes 2 seconds for Best Buy, 5 seconds for CNet, 4 seconds for Future Shop, and 3 seconds from Amazon.

The last module, cleaned files generation, always takes less than one second for both template detection method and WebPageCleaner.

4.4 Experiments on Web Page Classification

To run Naïve Bayes algorithm on the dataset, all experimental documents are divided into training data and testing data. Documents in training set have their class labels known, and are used for training classifiers. Testing documents are used to test the accuracy of trained classifiers. To investigate the way that noisy contents affect classification results, we set two cases for experiments based on whether training data are selected equably from each class of each web site.

Case 1: training data are selected automatically and evenly. Evenly means that training data contains web pages from each 5 categories, and documents in each category come for each 4 web sites. Since our datasets are collected from a small number of web sites, when training data are selected evenly from each site and each class, noisy elements are easy to be removed by Naïve Bayes classifier. Also, we could get reasonable classification accuracy using a very small number of training data (less than 20%) because of the equably distribution property in this experimental case. Six sub cases are

set which use 25, 50, 75, 100, 250, 500 documents automatically selected for training, and use the remaining documents for testing, respectively. For each sub case, each category has the same number of documents for training. For example, in sub case 1-1, five documents are selected from each five class to obtain 25 documents for training. Each of the case is tested 10 times using different training documents automatically selected from the dataset. The average percentage accuracy and standard error are shown in Table 4.1-3. Figure 4.3 gives chart representation of the average percentage accuracy for 10 times execution on each of the sub case.

Case	Train	Test	Methods	Average Accuracy (%)	Standard Error
1-1	25 (5 per class)	2475	NC	79.41	2.13
			TPL	88.63	1.41
			WPC	91.10	0.69
1-2	50 (10 per class)	2450	NC	90.52	1.01
			TPL	92.42	0.96
			WPC	95.44	0.40
1-3	75 (15 per class)	2425	NC	95.44	0.42
			TPL	94.19	0.70
			WPC	97.05	0.22
1-4	100 (20 per class)	2400	NC	94.89	0.43
			TPL	94.45	0.33
			WPC	97.11	0.20
1-5	250 (50 per class)	2250	NC	97.40	0.37
			TPL	97.33	0.21
			WPC	98.64	0.12
1-6	500 (100 per class)	2000	NC	97.97	0.27
			TPL	98.09	0.09
			WPC	99.00	0.06

Table 4.1-3: Average accuracy and standard error on 10 trails for each sub case

Figure 4.3 Average accuracy on 10 trials for each sub case

We compare the classification efficiency by applying Naïve Bayes method on different datasets, which are web pages without cleaning (NC), web pages cleaned by template detection method (TPL), and web pages cleaned by WebPageCleaner (WPC). The number of unique words in Naïve Bayes model and the average running time for 10 times classification on each dataset are compared, which is shown in Table 4.1-4. Naïve Bayes classifier generates the model for the whole dataset, which includes both training and testing documents. The number of unique words is counted for classification on each dataset.

Methods	Number of Unique Words in Model	Average Running Time (Second)
NC	58994	10.06
TPL	53417	4.56
WPC	32434	3.35

Table 4.1-4 Classification efficiency comparison for case 1

Case 2: four-fold cross validation. We divide the dataset with 2500 documents into four folders, which are 625 documents in each folder. To investigate the effect of

eliminating noisy element on classification performance, we test on category from one site by training that category from other sites. This is implemented by making each folder contains documents from all five categories, but some categories contain documents all from one web site. Therefore, the class distribution in the training set is uneven. Some classes have more samples than others. Detailed data distribution on each folder is shown in Table 4.1-5. For simplicity, we use BB to denote Best Buy, and FS to denote Future Shop below. The cell with “sitename_all” means that all class documents belong to that site is contained. Besides the cells with “sitename_all” flag, other documents are distributed evenly to obtain 625 documents on each folder.

Folder \ Class	1	2	3	4
Computer	72 Amazon_all	134 BB_all	99	233
MP3	99	95 BB_all	98 Amazon_all	62 FS_all
Phone	157 BB_all	137 Amazon_all	107	140 CNet_all
Software	200	60 Amazon_all	137 BB_all, FS_all	49
TV	97 BB_all	199	184 FS_all	141
Total	625	625	625	625

Table 4.1-5: Data distribution on each fold for 4-fold cross validation case

To balance the effect by unevenly selected training sets, a four-fold cross validation is implemented on this experimental case. Four sub cases are set by using three folders for training, and one folder for testing each time. So, sub case 2-1 test on folder 1, and using folders 2, 3, and 4 for training, and sub case 2-2 test on folder 2 using folder 1, 3, and 4 for training, and so on. We can see that a large size of training set (75%) is used in this experiment. The aim is to see that when important information (i.e., class information on a certain site) are missing from training set, the noisy items will harm

classification accuracy seriously even using a large amount of data for training. The classification accuracy and efficiency on each case of four-fold cross validation are shown in Table 4.1-6.

Case	Methods	Accuracy	Number of Unique words	Time (Second)
2-1	NC	81.28	49974	8
	TPL	93.43	45753	4.94
	WPC	97.92	29540	4.23
2-2	NC	89.44	52138	6.69
	TPL	98.88	49537	3.24
	WPC	99.04	24289	2.23
2-3	NC	81.28	46005	6.48
	TPL	94.39	37939	2.98
	WPC	97.44	29502	2.29
2-4	NC	66.72	55362	6.62
	TPL	98.24	49550	3.2
	WPC	98.08	30040	2.36

Table 4.1-6: Classification performance on each case of 4-fold cross validation

Table 4.1-7 gives the average classification accuracy and standard error on 4-fold cross validation. The chart representation for accuracy and efficiency can be seen in Figure 4.4 and Figure 4.5.

	Average Accuracy	Standard Error
NC	79.68	4.07
TPL	96.23	1.18
WPC	98.12	0.29

Table 4.1-7: Average accuracy and standard error on 4-fold cross validation

Figure 4.4: Classification accuracy on 4-fold cross validation

Figure 4.5: Classification efficiency on 4-fold cross validation

4.5 Analysis of Classification Results

We set two cases for classification experiments. The first case use a small amount of documents (less than 20%) for training, and the training data is automatically and evenly selected from each five class on each four web sites in our dataset. In this case, Naïve Bayes learns well from training set, including information on both web site and category. So even without web page cleaning, it can get reasonable classification results. However, classification on web pages cleaned by WebPageCleaner still performs better than web pages without cleaning and cleaned by template detection method. From Table 4.1-3 and Figure 4.3, we can see that accuracy is better with the training set gets larger for each method. In each of the case, WebPageCleaner gets the best accuracy with the smallest standard error using the less running time. The small standard error shows that classification result is stable on each of sub cases.

The second case use large amount documents (75%) for training, but the training data is selected unevenly from classes of web sites. Training data still involves a mixture dataset from all categories, but may not contain category documents from a certain web site. In this case, Naïve Bayes classifier is confused by noisy elements in multiple web sites because they are totally different. This makes the training model not accurate enough for testing. Affected by noisy contents, classification on web pages without cleaning always gets lower accuracy. When noisy elements are removed from web pages, Naïve Bayes classifier collects useful information on categories, while does not affected by noises, so classification results are much better. A four-fold cross validation is implemented to reduce the effect by unevenly selected training sets. Table 4.1-6 shows that WebPageCleaner obviously improves the accuracy compared to classification on web page without cleaning. It also performs better than template detection method. From Table 4.1-7, we can see that WebPageCleaner still gets the best classification accuracy and the less standard error on this case.

Although template detection method gets good accuracy, its cleaning results are still not as good as that of WebPageCleaner. On one hand, its cleaned files contain more noisy items, which decrease both classification accuracy and efficiency. On the other hand, its execution time for cleaning is much longer than WebPageCleaner.

For both of cases, compared to NC and TPL implementations, WebPageCleaner reduces many quantities of unique words for building training model, and gets better accuracy in the mean time.

As conclusion, the advantages of WebPageCleaner are that it decreases the number of words to be processed by text classification. Despite decreasing the volume of words to be processed, the accuracy of classification is still better than that of other methods. Thus, both accuracy and efficiency of text classification are improved by the WebPageCleaner system designed in this thesis.

Chapter 5 CONCLUSIONS AND FUTURE WORK

This thesis discusses the problem of eliminating noise blocks in web pages belonging to one web site first before aggregating all cleaned web pages for such data mining task as web page classification. Related literatures include techniques for web page segmentation, data cleaning in data warehousing, duplicate document detection, and document classification.

A new scheme, called WebPageCleaner, is proposed in the thesis. It aims at improving the accuracy and efficiency of web page classification by eliminating noisy parts of web pages, such as navigation bars, advertisement blocks, etc. WebPageCleaner consists of three modules: block extraction, block importance retrieval, and cleaned files generation. Block extraction module aims at getting the semantic content blocks from web pages. The thesis employs the Vision-based Page Segmentation (VIPS) algorithm [CYWM03a] as the partition method. VIPS outperforms other page partitioning techniques in generating content coherent blocks with adjustable granularity, as well as efficient partitioning process. Features of blocks, including block position, linkage percentage of contents, are extracted in this module. In the block importance retrieval module, thesis uses the fingerprints as the hash value of block contents. This provides fast calculations based on databases, such as searching and grouping records. After sorting the datasets according to block contents which brings similar blocks together, the level of similarity is obtained by computing resemblance of contents between each pair of neighboring blocks. Using similar level of block contents as the most important factor, and linkage percentage and block position as auxiliary measurements, the importance level for each block is calculated. Finally, a set of cleaned files consisting of blocks with the highest N importance level in each page, are generated. These files are noise blocks removed, as well as capitalization, and punctuation ignored. They can be directly fed as inputs to web content mining. Thus, the scheme proposed here can also be viewed as a preprocessing step for web content mining.

WebPageCleaner contributes an enhancement to the template detection method [BR02] in that it employs a different web page partitioning method, which is more

efficient and uses both contents of blocks as well as other block features for measuring block importance. Blocks with near-duplicate contents are also detected as possible noisy blocks. These enhancements enable more noisy blocks to be eliminated so as to keep only the most informative contents in results.

The thesis implements full experiments to stress the harm of noisy blocks to web page classification. Thesis work is evaluated using Naïve Bayes text classification method. Classifications are implemented on web page documents without cleaning (with html tags removed), web page documents cleaned by template detection algorithm, and web page documents cleaned by WebPageCleaner. Comparative experiments show that WebPageCleaner outperforms other techniques in providing more accurate classification results using the fewer amounts of words in documents.

There are a number of work to be addressed in the future:

- (1) Exploring a scheme that utilizes the Blocks table as a template datasets for different web sites is an interesting future work. Thus will allow incoming web pages to be mapped to their templates in order to clean new pages online. This template datasets need to involve web pages from a large amount of web sites, and each web site keeps only a few typical web pages as its templates. Thus, the technique can be used to organize search results in real time.
- (2) Applying the proposed technique to improve the results in other Information Retrieval areas, such as clustering and search engine.
- (3) Making the technique applicable in portable devices, such as cell phones and PDA is another future work since these devices have small browsers and only the most informative contents appropriate are displayed.

Reference

- [ACG02] R. Ananthakrishna, S. Chaudhuri, V. Ganti, "Eliminating fuzzy duplicates in data warehouses", In *Proceedings of the 28th International Conference on Very Large Databases (VLDB) 2002, Hong Kong, Pages 586-597*.
- [Ade98] B. Adelberg, "NoDoSE: A tool for semiautomatically extracting structured and semistructured data from text documents", In *Proceedings of ACM SIGMOD Conference on Management of Data, 1998, Pages 283-294*.
- [BD83] D. Bitton, D. J. Dewitt, "Duplicate record elimination in large data files", *ACM Transactions on Database Systems, Vol. 8, No. 2, June 1983, Pages 255-265*.
- [BGM97] A. Z. Broder, S. C. Glassman, M. S. Manasse, "Syntactic clustering of the Web", In *Proceedings of the 8th International World Wide Web Conference (WWW6), 1997, Pages 1157-1166*.
- [BLP01] D. Buttler, L. Liu, C. Pu, "A fully automated object extraction system for the World Wide Web", In *Proceedings of the 21st International Conference on Distributed Computing Systems, 2001, Pages 361-370*.
- [Bro93] A. Z. Broder, "Some applications of Rabin's fingerprinting method", *Sequences II: Methods in Communications, Security, and Computer Science, Springer-Verlag, 1993, Pages 143-152*.
- [BR02] Z. Bar-Yossef, S. Rajagopalan, "Template detection via data mining and its applications", In *Proceedings of the International World Wide Web Conference (WWW) 2002, May 7-11, 2002, Pages 580-591*.
- [CD00] H. Chen and S. T. Dumais, "Bringing order to the Web: Automatically categorizing search results", In *Proceedings of CHI2000, 2000, Pages 145-152*.
- [CGGM03] S. Chaudhuri, K. Ganjam, V. Ganti, R. Motwani, "Robust and efficient fuzzy match for online data cleaning", In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, CA, 2003. ACM Press, Pages 313-324*.
- [Cha02] S. Chakrabarti, "Mining the web: Discovering knowledge from hypertext data", *Morgan Kaufmann Publishers, ISBN 1-55860-754-4, 2002*.

- [Coh98] W. Cohen. "Integration of heterogeneous databases without common domains using queries based in textual similarity", In *Proceedings of ACM SIGMOD, Seattle, WA, June 1998, Pages 201--212*.
- [CV95] C. Cortes, V. Vapnik, "Support vector networks", *Machine Learning*, 20:1-25, 1995.
- [CYWM03a] D. Cai, S. Yu, J.-R. Wen, W.-Y. Ma, "VIPS: a Vision-based page segmentation algorithm", *Microsoft Technical Report, MSR-TR-2003-79*, 2003.
- [CYWM03b] D. Cai, S. Yu, J.-R. Wen, W.-Y. Ma, "Extracting content structure for web pages based on visual representation", In *Proceedings of the 5th Asia Pacific Web Conference, Xi'an China, 2003, Pages 406-417*.
- [EO05] C.I. Ezeife, Timothy E. Ohanekwu, "Use of smart tokens in cleaning integrated warehouse data", *the International Journal of Data Warehousing and Mining (IJDW)*, Ideas Group Publishers, April-June 2005, Vol. 1, No. 2, Pages 1-22.
- [Etz96] O. Etzioni, "The World Wide Web: Quagmire or gold mine", *Communications of the ACM*, 39(11):65-68, 1996.
- [FGG97] N. Friedman, D. Geiger, M. Goldszmidt, "Bayesian network classifiers", *Machine Learning*, 29: 131-163, 1997.
- [GFS+01] H. Galhardas, D. Florescu, D. Shasha, E. Simon, C. A. Saita, "Declarative data cleaning: language, model, and algorithms", In *Proceedings of the 27th International Conference on Very Large Databases (VLDB 2001)*, 2001, Pages 371-380.
- [GFSS00] H. Galhardas, D. Florescu, D. Shasha, E. Simon, "An Extensible Framework for Data Cleaning", *ICDE 2000 poster paper, San Diego, Feb-March 2000, Page 312*.
- [GIJ+01] L. Gravano, P. Ipeirotis, H.V. Jagadish, N. Koudas, S. Muthukrishnan, D. Srivastava. "Approximate string joins in a database (almost) for free", In *Proceedings of the 27th International Conference on Very Large Databases (VLDB) 2001, Roma, Italy, Pages 491-500*.
- [GKNG03] S. Gupta, G. Kaiser, D. Neistadt, P. Grimm, "DOM-based content extraction of HTML documents", In *Proceedings of International WWW Conference, 2003, Pages 207-214*.

- [GKS05] S. Gupta, G. Kaiser, S. Stolfo, "Extracting context to improve accuracy for HTML content extraction", In *Proceedings of International WWW Conference, May 10-14, 2005, Chiba, Japan*.
- [Her96] M. Hernandez, "A generalization of band joins and the merge/purge problem", *Ph.D. thesis, Columbia University, 1996*.
- [HS95] M. Hernandez, S. Stolfo, "The merge/purge problem for large databases", In *Proceedings of the ACM SIGMOD, San Jose, CA, May 1995, Pages 127-138*.
- [JAKN03] S. Joshi, N. Agrawal, R. Krishnapuram, S. Negi, "A bag of paths model for measuring structural similarity in web documents", In *Proceedings of SIGKDD-03, 2003, Pages 577-582*.
- [Joa98] T. Joachims, "Text categorization with support vector machines: learning with many relevant features", In *Proceedings of ECML-98, 10th European Conference on Machine Learning, 1998, 137-142*.
- [Jus99] N. Kushmerick, "Learning to remove Internet advertisements", In *Proceedings of the 3rd International Conference on Autonomous Agents (AGENT '99), 1999*.
- [KLC02] H.-Y. Kao, S.-H. Lin, J.-M. Ho, M.-S. Chen, "Entropy-based link analysis for mining web informative structures", In *Proceedings of Conference on Information and Knowledge Management, CIKM 2002, Pages 574-581*.
- [LA99] B. Larsen, C. Aone, "Fast and effective text mining using linear-time document clustering", In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data mining, 1999, Pages 16-22*.
- [LG94] D. Lewis, W. Gale, "A sequential algorithm for training text classifiers", In *Proceedings of SIGIR, 1994, Pages 13-19*.
- [LH02] S.-H. Lin, J.-M. Ho, "Discovering informative content blocks from web documents", *KDD-02, 2002, Pages 588-593*.
- [LIT92] P. Langley, W. Iba, K. Thompson, "An analysis of Bayesian classifiers", In *Proceedings of AAAI-92, 1992, Pages 223-228*.
- [LLL00] M. L. Lee, W. Ling, W. L. Low, "Intelliclean: A knowledge-based intelligent data cleaner", *KDD-2000, 2000, Pages 290-294*.

- [LLK99] M. L. Lee, H. Lu, T. W. Ling, Y. T. Ko, "Cleansing data for mining and warehousing", In *Proceedings of the 10th International Conference on Database and Expert Systems Applications, Florence, Italy, August 1999*.
- [LYL03] B. Li, S. Yu, Q. Lu, "An Improved k-Nearest neighbor algorithm for text categorization", In *Proceedings of the 20th International Conference on Computer Processing of Oriental languages, Shenyang, China, August 2003*.
- [KB00] R. Kosala, H. Blockeel, "Web mining research: A survey", *ACM SIGKDD Explorations*, 2(1), pp. 1-15, 2000.
- [McC98] A. McCallum, "Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering", <http://www-2.cs.cmu.edu/~mccallum/bow/rainbow/>.
- [ME96] A. E. Monge, C. P. Elkan, "The field matching problem: Algorithms and applications", In *Proceedings of the Second International conference on Knowledge Discovery and Data Mining, AAAI Press, August 1996, Pages 267-270*.
- [ME97] A. E. Monge, C.P. Elkan, "An efficient domain-independent algorithm for detecting approximately duplicate database records", In *Proceedings of the SIGMOD 1997 workshop on data mining and knowledge discovery, 1997*.
- [MGCC03] L. Ma, N. Goharian, A. Chowdhury, M. Chung, "Extracting unstructured data from template generated web documents", In *Proceedings of Conference on Information and Knowledge Management, CIKM-03, 2003, Pages 512-515*.
- [MN98] A. McCallum, K. Nigam, "A comparison of event models for naïve Bayes text classification", In *Proceedings of AAAI-98 Workshop on Learning for Text Categorization. AAAIPress, 1998, Pages 41-48*.
- [MRNL99] S. K. Madria, S. S. Rhowmich, W. K. Ng, and F. P. Lim, "Research issues in Web data mining", In *Proceedings of Data Warehousing and Knowledge Discovery, First International Conference. DaWaK'99, 1999, Pages 303-312*.
- [OE03] T. E. Ohanekwu and C.I. Ezeife, "A Token-Based Data Cleaning Technique for Data Warehouse Systems", In *Proceedings of the International Workshop on Data Quality in Cooperative Information Systems, held in conjunction with the 9th International Conference on Database Theory (ICDT 2003), Siena, Italy, January 10-11, 2003, Pages 21-26*.

- [PBMW98] L. Page, S. Brin, R. Motwani, T. Winograd. "The PageRank Citation Ranking: Bringing Order to the Web", *Technical Report, Compute Science Department, Stanford University, 1998*.
- [Por80] M. F. Porter, "An algorithm for suffix stripping", *Program*, 14(3) :130-137, 1980, <http://www.tartarus.org/~martin/PorterStemmer/>.
- [Rab81] M. O. Rabin, "Fingerprinting by random polynomials", *Center for Research in Computing Technology, Harvard University, Report TR-15-81, 1981*.
- [RD00] E. Rahm, H. Do. "Data cleaning: Problems and current approaches", *Bulletin of the Technical Committee on Data Engineering*, 23, 4, 2000.
- [Rib02] D. Riboni, "Feature Selection for Web Page Classification", *In EURASIA-ICT 2000 Proceedings of the Workshop*.
- [SCZ+04] D. Shen, Z. Chen, H.-J. Zeng, B. Zhang, Q. Yang, W.-Y. Ma, Y. Lu, "Web-page classification through summarization", *In Proceedings of the 27th Annual International ACM SIGIR Conference (SIGIR'2004), July 2004, Pages 242-249*.
- [Sha48] C. Shannon, "A mathematical theory of communication", *Bell System Technical Journal*, Vol. 27, pp. 379-423 and 623-656, July and October, 1948.
- [SGM00] A. Strehl, J. Ghosh, R. Mooney, "Impact of similarity measures on web-page clustering", *In Proceedings of American Association for Artificial Intelligence (AAAI) Workshop on AI for Web Search, 2000, Austin, Pages 58-64*.
- [SK04] L. K. Shih, D. R. Karger, "Using URLs and table layout for web classification tasks", *In Proceedings of the WWW 2004, Pages 193-202*.
- [Sin01] A. Singhal, "Modern Information Retrieval: A Brief Overview", *IEEE Data Engineering Bulletin*, vol. 24(4), pp. 35-43, 2001.
- [SLS02] S. Y. Sung, Z. Li, P. Sun, "A fast filtering scheme for large database cleansing", *In Proceedings of 11th international conference on information and knowledge management, Pages 76-83*.
- [SLWM04] R. Song, H. Liu, J.-R. Wen, W.-Y., Ma, "Learning Block Importance Models for Web Pages", *In Proceedings of the WWW 2004, May 17-22, New York, NY USA, Pages 203-211*.
- [YLL03] L. Yi, B. Liu, X. Li, "Eliminating noisy information in Web pages for data mining", *SIGKDD 2003, August 24-27, 2003, Pages 296-305*.

Vita Auctoris

NAME	Jing Li
PLACE OF BIRTH	Heilongjiang Province, P. R. China
YEAR OF BIRTH	1977
EDUCATION	Department of Information Science Beijing Union University, Beijing, P. R. China 1994-1998 B.Sc. School of Computer Science University of Windsor, Windsor, Ontario, Canada 2003-2005 M.Sc.