University of Alberta

ANALYZING AND EXTRACTING LISTS ON THE WEB

by

Afsaneh Esteki

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Fall 2013
Edmonton, Alberta

Abstract

The amount of information available on the Web is rapidly growing, and the need for extracting more useful and relevant data from this tremendously large source has become an interesting research challenge. Among various types of useful information that can be extracted, lists in particular are highly valuable as they provide groupings of related items. Such groupings are often interpretable and may present data in a more structured and condensed format that can be fed to other applications.

In this thesis we explore some of the properties of lists embedded in web pages. Based on these properties, we propose a technique for classifying web pages into two categories: those containing lists, and the rest. Our results show that unlike some previous work, not all list-specific html tags are useful for identifying list-containing web pages. We also study the related problem of locating lists in a page. We cast the problem of detecting the boundaries of a list as a classification task and build a classifier using relevant page features. As the classifier produces a sequence of labels for each page, we examine some of the properties of this sequence and show how the accuracy of the detection can be further improved by rejecting some of the sequences that are less likely to indicate a list.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Davood Rafiei for his extreme support, patient and guidance through my research.
I am also thankful to my family, specially my parents for all their supports throughout my life. Finally I would like to thank my husband, Rouzbeh, for his continued support and encouragement during the past few years.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The amount of information available on the Web is growing very fast. Extracting more useful and relevant data from this tremendously large collection of pages has become a research challenge [2, 9, 30]. Such relevant data may be useful for meta-search engines, shopping agents, or even building a knowledge base. Among these valuable information, lists, or simply sets of similar or related items grouped together by content providers or writers, are of more interest because they are interpretable and present data in a more structured and condensed format. Examples of lists are lists of medicines, airlines, or restaurants. Since there are a variety of lists on the web, with different characteristics, there are many studies on extracting and analyzing embedded information in them [3, 13].

We adopt a definition similar to Gatterbauer et al. [14] and define a list as: *"A set of similar or related data items."*. The definition is general enough to include tables as well if an item is a record with multiple fields. However our focus in this thesis is on unary column lists.

With the large variation in content and the formatting of web pages on the web, finding similar items can be a challenge. For example, a web page that contains a list of cell phone providers is clearly a good source, but a news article or even a bibliography page may or may not be considered a good source. We treat the problem as a binary classification where given a page, we want to detect if the page has some 'useful' lists of items. To the best of our knowledge, this is the first work that studies the problem based on the properties of the page. Our method mostly relies on the underlying HTML markup, structural and content-based properties of

the web page. Instead of focusing just on structural attributes, we also consider context features based on the content and the title of pages. One of our findings is that not all list-specific html tags such as $<li>$ and $<Ol>$, used in some previous work, are useful for identifying a list-containing page.

Once we can detect relevant web pages and to some degree separate those containing useful lists, we further analyze the positive category: the category which likely contains one or more. We study some of the properties that can help us locate the place of possible lists in a page. Related work often uses heuristics for finding data regions and record boundaries in a page [5, 22, 24]. Sleiman and Corchuelo [29], define a region as a fragment which shows information about one or more related items. We treat the problem of detecting the boundaries of a list also as a classification problem. We start by shifting a fixed-size window over the page, dividing the page into smaller sub-pages. Then by applying the classifier on each sub-page separately, we are able to assign a label to it. The result is a sequence of labels for each page, which shows the location(s) of the list(s) in that page. For example the sequence "NYYYYYN" shows that the entire page is divided into 7 sub-pages and that there is no lists in the starting and the ending sub-pages, but the middle sub-page does have lists.

Locating lists in a page solely based on one such classifier does not provide a perfect result. As a result, we propose a model based on Markov chain, which takes into account the dependence between labels in a sequence, and tries to optimize the likelihood of the predicted sequences and therefore increases the accuracy.

1.1 Thesis Statements

Our thesis statement is that using solely HTML list related tags, employed mainly by web page providers for grouping a set of items, is a simple approach for identifying list-containing pages, but at the same time error-prone. We hypothesize that employing structural and context based features embedded in HTML web pages is more effective for detecting list-containing web pages and their enclosed lists.

1.2 Thesis Contributions

The following are the main contributions of this thesis:

- A study on some of the characteristics of lists and their embedding in web pages.
- A classifier for detecting lists containing pages.
- An effective algorithm for identifying the location of a list in a page.

1.3 Thesis Organization

The remain of this dissertation is organized as follows. In section 2 we review related works, briefly explain them and compare their similarities and differences to our work. In section 3 we present our problem definition, data set preparation and pre-processing steps. In section 4 we discuss the methodology, classifier and feature selection methods and results for the classification section. Section 5 presents the second part of the study, which is locating lists in each page which likely contains one or more lists. We then present a method based on the properties of Markov Model for increasing the accuracy of results obtained in previous section. Finally and in section 7 we discuss about results and present future works.

Chapter 2

Background and Related work

Information extraction, as the task of identifying and extracting relevant data from web documents, is studied from different aspects. Relevant to our work, is extracting data from unstructured or semi-structured data, which will provide a rich source of information for further processing. The work on information extraction can be classified according to the type of pages from which the information is being extracted. For the purpose of this thesis, pages on the Web may be categorized into three classes [6]:

- Unstructured or free-text pages that are written in natural text, such as news articles and journals.
- Semi-structured pages, such as pages containing relational tables.
- Structured pages, such as XML pages

In this section we review some of the related work and explain their relationships to our works.

2.1 NLP-Based Information Extraction

Free-text web documents usually require natural language processing (NLP) techniques, such as part-of-speech tagging or a lexicon, to build semantic and syntactic-based relations used for deriving extraction rules [20]. Examples of such systems are RAPIER [25], which uses a bottom-up supervised algorithm for learning extraction patterns from a filled template, and also SRV described by Freitag [12].

SRV learns extraction rules using token-oriented features obtained from annotated corpora. As mentioned, NLP-Based techniques are suitable for documents which contain free text, while we are considering semi-structured and HTML documents.

2.2 Wrappers and Wrapper Induction

A traditional approach for extracting target information from structured documents on the web is through a procedure, called wrapper. Wrappers are usually hand-coded functions for extracting tuples from a particular information resource. Writing and maintaining manual wrappers is a tedious and challenging task, hence wrapper induction methods are introduced. Wrapper induction techniques generate extraction rules usually from a set of training samples. They build delimiter-based wrappers and unlike NLP-Based systems are suitable for semi-structured and HTML documents. Sarawagi categorized HTML wrappers into three groups: record-level, page-level, and site-level. Record-level wrappers are targeted for extracting homogeneous data records in a web page by discovering regularities in the page. Page-level wrappers are capable of identifying multiple kinds of records in a page, e.g., name, courses and publications from a personal home page. Finally, site-level wrappers extract information from all pages of a web site (e.g., list of courses from a university web site) [28].

Kushmerick pioneered a wrapper induction technique for automatically generating wrappers by introducing a family of six wrapper classes: Left-Right (LR), Head-Left-Right-Tail (HLRT), Open-Close-Left-Right (OCLR), Head-Open-Close-Left-Right-Tail (HOCLRT), Nested-LR (N-LR), and Nested-HLRT (N-HLRT). The first four classes are suitable for semi-structured documents which have a tabular format but not nested, while the two others are used for nested documents. Based on the structure of a web site, one class of wrapper may be more appropriate than others. For example if the set of pages have Open-Close-Left-Right structure (e.g., web pages have an opening delimiter, left and right delimiters for extracting flat tuples, and a closing delimiter) an OCLR wrapper would be generated [19]. Kushmerick's work does not consider missing or reordered attributes in the input. It also

does not allow for disjunction, where there might exist more than one delimiter per attribute. In order to handle disjunction and missing attributes, some related work introduced finite-state transducers (FST), where each different attribute permutation is encoded as a path [17].

RoadRunner [8] is an unsupervised wrapper induction method that starts by comparing some sample documents of the same class and generates a grammar based on their similarities and differences. Since RoadRunner is designed for page-level extraction tasks, the generated wrapper is capable of extracting other instances in the same web site. A difficulty with wrapper induction, in general, including RoadRunner is that since each web site has its own template, it requires a separate and sometimes tedious wrapper induction process. In other words a wrapper cannot be trained to extract general lists from the Web.

As an alternative, Brin presented DIPRE as a system for extracting patterns and relations from the web, based on a small sample seed set [2]. The technique is based on exploiting pattern-relation duality, to expand the target set. The author uses this approach to extract a set of book-author pairs. However, this approach as well is not domain-independent and should be provided with a different set of starting samples for each domain, such as musics, restaurants, etc.

2.3 Table Extraction

The problem of information extraction from embedded tables on the web is also related to our work. Web tables are a common schema for representing grouped data or attribute-value relationships, such as airline schedules, book-author information, student enrollment statistics in a particular year, etc. Penn et al. classify HTML tables into: genuine and non-genuine tables [26]. They use the *genuineness* term to denote tables where a relational content exist, while non-genuine tables are considered those where the content is visually grouped using table tags but there is no logical relation among cell entities [26].

Chen et al. tried to identify genuine tables using heuristic rules and cell content similarity [7]. Their method relies mostly on the HTML table tag. The WebTa-

bles system [4] extracts relational information from structured tables on the web again using the table tag. The system uses some human-marked samples along with a classifier built on features extracted from table layout and content, to filter out relational tables. The authors report that actually 1.1% of the tables on the Web represent relational information. Since this approach relies mainly on table tag, it is biased towards certain Web structures and does not cover a uniform sample of web pages.

Wang and Hu [32] detect genuine tables from non-genuine ones, using HTML table tag and a machine leaning classification approach which relies on both layout and content type features such as average and standard deviation of number of rows and columns in a table.

Gatterbauer et al. tried to identify tabular data without using the HTML table tag. Instead they employ visual presentation features of a page such as data placement on the screen [14]. This method might be used in other table extraction techniques which rely only on table tag, in order to enrich them with visual features.

Yin et al. developed a system, called FACTO, that extracts tables on the web to answer fact lookup queries in search engines, e.g., Barack Obama date of birth [35]. FACTO extracts entity-attribute-value triplets from tables and stores them in a repository. This is done by distinguishing attribute-value tables from other tables, extracting the main entity of each page and finally joining extracted entities with attribute-values. Given a search query, FACTO decides if it is a fact lookup query, finds out about the existence of the query or its equivalent in the database, and provides the most confident result as the answer of the query. Authors compare their system with popular search engines such as Google and Ask.com and report that it obtains higher precision and comparable coverage. However their approach considers only attribute-value tables in the web. There are many tables which are not in attribute-value format. For example a table may be an entity-attribute relation with the first column corresponding to the entity name and the second to an attribute of the entity. Thus FACTO is not capable of answering queries about an entity without any specific attribute.

2.4 Detecting Record Boundaries

Buttler et al. introduced OMINI to automatically learn rules for extracting objects and data records, e.g., list items from the largest data region in a web page [3]. The largest data region in the DOM tree of a page has the largest number of children, the largest contents and also the largest number of tags. Some heuristics are used to rank tags as candidate data separators. This work is close to ours as some of the heuristics are similar.

Mining Data Records (MDR) [22], extracts data records from Web pages by mining the DOM-structure of pages and string matching. MDR is built on the hypothesis that data records have repetitive regular HTML tags and patterns, and they are rooted in a single parent node [33]. Based on this hypothesis a subset of adjacent nodes with the same parent are considered as strings and then grouped together if the edit distance between them is less than a fixed threshold.

Tag Path Clustering is a similar approach to MDR which clusters tag paths based on DOM path similarity [24]. It tries to detect nested data records, while MDR does not handle nested data objects.

Vision-based Page Segmentation (VIPS) [5] is intended to locate all blocks of a document, using its visual cues. Examples of visual features are: fonts, horizontal or vertical lines, color, and background images. Unlike our work, this algorithm is tag-tree independent and as a consequent, it will return all blocks of data, even header, footer or other blocks that are irrelevant.

VSDR [21] is proposed for extracting data records which are similar and consist of several types of information such as images, text and lists of hyperlinks. Since it assumes that a data region is composed of different kinds of contents, it may fail if lists are very simple, e.g., composed of only text or images.

The majority of detecting record boundary methods are unsupervised and do not need training samples of the regions to be extracted. This is similar to our work, as we do not rely on a supervised method for locating lists in a web page, and we do this using a previously trained classifier.

Those proposals which use probabilistic methods are related to our work on

the key idea of combining some heuristics, usually based only on tag information. However we combine both probabilistic information from DOM with content and title features.

Also, in general, most of the related work which use tag dependent heuristics, such as OMINI, rely on the fact that some types of tags are more probable to be separating tags. One of the main contributions of our work is that we do not make any assumptions about some specific types of tags for indicating a web page as a list containing page. This means that we do not consider a group of specific list-related tags. For example we assume both $< li >$ and $< div >$ tags have the same probability of indicating a list.

Regarding noise detection and filtering, some heuristics are employed in a pre-processing step of our method to filter out and remove noisy sections, such as footer, header, and Drop-Down menus. This is applicable in our case since we use a DOM tree information, unlike some other approaches in which noisy blocks are removed after detecting all regions.

2.5 List Extraction

List extraction is widely useful in data mining tasks and has also attracted some commercial interest. Google Sets was a famous service (still available in Google Drive) for automatically generating lists given one or more examples. However the exact framework of Google Sets has not been published, but a simple explanation of how the system works is that it attempts to identify lists as it crawls web pages. It may first extract items from lists by considering specific HTML tags, which are mainly used for identifying lists on the web, such as *<OL>*, *<UL>*, *<DL>*, and *<H1>*-*<H6>* tags, and also by looking for tables, or lists separated by commas, semicolons or tabs. It then ranks all extracted items based on their co-occurrence in statistics on the web as a pool of data [30]. KnowItAll which is a named-entity extraction system, has a list extractor component which resembles Google Sets [10]. The SEAL (Set Expander for Any Language) project is also a set expander similar to Google Sets but its extraction method is based on a wrapper learner [31]. As

a possible extension of our work, set expansion may be applied to the set of pages identified by our method.

Fumarola et al. [13] developed a hybrid list extraction system which uses both visual alignment of list items, embedded in modern web browsers, and structural features. Their method extracts all items and all lists from a given web page without filtering out any noisy ones. In comparison to our work, we try to purge some explicit noisy data regions, such as footer and header of a page, in a pre-processing step.

Recently Wang et al. have studied the problem of extracting top-*k* lists from the web [36, 37]. Top-*k* lists describe top k instances of a particular topic or concept. Examples of these lists include: "Top 10 podcasts", and "20 Most Influential Scientists Alive Today". They use a binary classifier for classifying the title of a page to "top-*k* like" or not "top-*k* like", using a CRF model and extract features such as POS tag, lemma and concept of the title. However, their approach considers just extracting lists from web pages which have a "top-*k* like" title, and is missing other pages which include a list not necessarily containing top ones.

Chapter 3

Problem Definition and Dataset Preparation

3.1 Problem Definition

The purpose of our study is first to recognize web pages that contain lists and then locate the place of those lists inside the pages. Hence the problem has two parts and can be defined as follow:

1-Given a set of web pages, classify them into positive and negative classes so that pages in positive class are likely to contain lists, while pages in the negative class do not.

2-Given a positive web page, which contains a list or a set of lists, locate the place of lists.

3.2 System Overview

The input for the system is a set of HTML web pages and the output is pages possibly containing list or lists, as well as possible locations of those lists in the corresponding pages. In order to classify web pages, we initially need a manually labeled data set that can be used to train a classifier. Having the labeled training data, a trained classifier is used for the task of binary classification. Once the classifier is trained, it can be used to predict output for unseen data. Classifier is trained based on the features selected from the training web pages. Features are obtained by looking into characteristics of list containing pages, such as the effectiveness of

using HTML tags, tag density inside a page or in a piece of page, and also context features of the page.

After identifying list-containing pages, the next step is to analyze those pages and locate the start and end positions of each list inside the page. We consider this as a binary classification problem by dividing each page into sub-pages and applying the trained classifier on each sub-page. Results would be a sequence of labels for each page, which could be optimized by rejecting some of the sequences that are less likely to indicate a list.

3.3 Dataset Preparation

We used the following two approaches to collect our dataset, 500 annotated URLs, out of which 250 were in the positive class, meaning they contained list(s), and 250 were in the negative class.

- Two sets of keywords that are related to each class and are likely to occur in web pages from each class were used. We used these keywords in Google search and chose the first few top results returned by each keyword. Some of the keywords used in this approach include: {*list of manufacturers, list of brands, list of items*} for the first class which contains list, and {*article, story of, a letter to*}, for the second class, likely not containing any list. In total 400 URLs were selected using this approach, 200 of them in positive class and 200 in negative.

- In order to come up with more random samples which can be selected without using a keyword set, we used a list of related items to gather more URLs. Examples of these grouped items are name of some *"scientists"*, *"cars"* and *"mobile brands"*. Again we typed these search queries to Google and selected some of the first top results. We managed to collect 100 URLs using this method, 50 of them in positive class and 50 in negative.

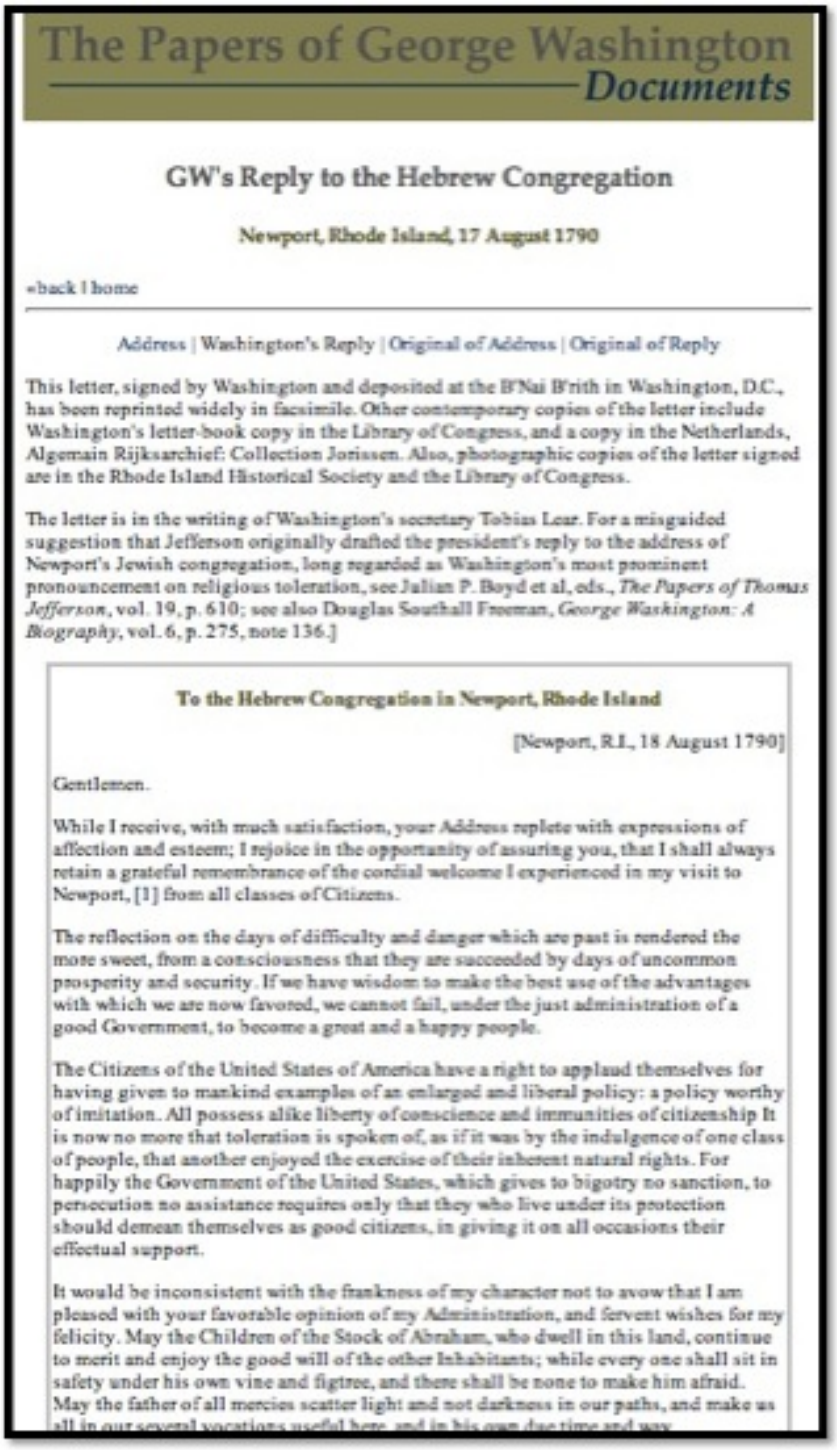
Having these candidate URLs, we fetched their web pages and kept a local copy to be able to obtain consistent results over time. We asked 2 annotators to

label URLs into positive or negative labels. In order to decide about the class of the URL, we considered the following criteria:

- *Menus* are not considered as lists. This includes *Dropdown* menus, which might even contain a meaningful list. The reason for considering this criterion is that, lists should be visible and not hidden in some parts of the page.
- Some regions in the page such as: ”read more”, ”related articles” and ”advertisements” are not considered as lists.
- Pages that contain elements of a visible list would be considered as positive pages.
- Pages that contain one or more text but not a list of similar items, would be considered as negative pages. Also pages with some peices of data and a small number of grouped items (less than 5), are considered as negative pages. As well, navigation bars are not considered as lists even if they present a set of links.

Figure 3.1 shows an example of a negative page, while Figure 3.2 shows an example of a positive page.

Since distinguishing between the two categories is sometimes confusing, annotators had the option to label a page into one of the $\{Y, N, Unknown\}$ categories. Here Y means that the page contain list, N means that it does not contain any list, and *Unknown* means that annotator was not sure about the label. After annotating all pages, we selected for our experiments only those ones in which both annotators agreed upon and also the label is Y or N. Annotators agreed on 70% of URLs obtained from the first method, and on 73% of URLs obtained from the second method.



The Papers of George Washington
Documents

GW's Reply to the Hebrew Congregation

Newport, Rhode Island, 17 August 1790

«back | home

Address | Washington's Reply | Original of Address | Original of Reply

This letter, signed by Washington and deposited at the B'Nai B'rith in Washington, D.C., has been reprinted widely in facsimile. Other contemporary copies of the letter include Washington's letter-book copy in the Library of Congress, and a copy in the Netherlands, Algemain Rijksarchief: Collection Jorissen. Also, photographic copies of the letter signed are in the Rhode Island Historical Society and the Library of Congress.

The letter is in the writing of Washington's secretary Tobias Lear. For a misguided suggestion that Jefferson originally drafted the president's reply to the address of Newport's Jewish congregation, long regarded as Washington's most prominent pronouncement on religious toleration, see Julian P. Boyd et al, eds., *The Papers of Thomas Jefferson*, vol. 19, p. 610; see also Douglas Southall Freeman, *George Washington: A Biography*, vol. 6, p. 275, note 136.]

To the Hebrew Congregation in Newport, Rhode Island

[Newport, R.I., 18 August 1790]

Gentlemen.

While I receive, with much satisfaction, your Address replete with expressions of affection and esteem; I rejoice in the opportunity of assuring you, that I shall always retain a grateful remembrance of the cordial welcome I experienced in my visit to Newport, [1] from all classes of Citizens.

The reflection on the days of difficulty and danger which are past is rendered the more sweet, from a consciousness that they are succeeded by days of uncommon prosperity and security. If we have wisdom to make the best use of the advantages with which we are now favored, we cannot fail, under the just administration of a good Government, to become a great and a happy people.

The Citizens of the United States of America have a right to applaud themselves for having given to mankind examples of an enlarged and liberal policy: a policy worthy of imitation. All possess alike liberty of conscience and immunities of citizenship It is now no more that toleration is spoken of, as if it was by the indulgence of one class of people, that another enjoyed the exercise of their inherent natural rights. For happily the Government of the United States, which gives to bigotry no sanction, to persecution no assistance requires only that they who live under its protection should demean themselves as good citizens, in giving it on all occasions their effectual support.

It would be inconsistent with the frankness of my character not to avow that I am pleased with your favorable opinion of my Administration, and fervent wishes for my felicity. May the Children of the Stock of Abraham, who dwell in this land, continue to merit and enjoy the good will of the other Inhabitants; while every one shall sit in safety under his own vine and figtree, and there shall be none to make him afraid. May the father of all mercies scatter light and not darkness in our paths, and make us all in our several vocations useful here, and in his own due time and way

Figure 3.1: Example of a negative page

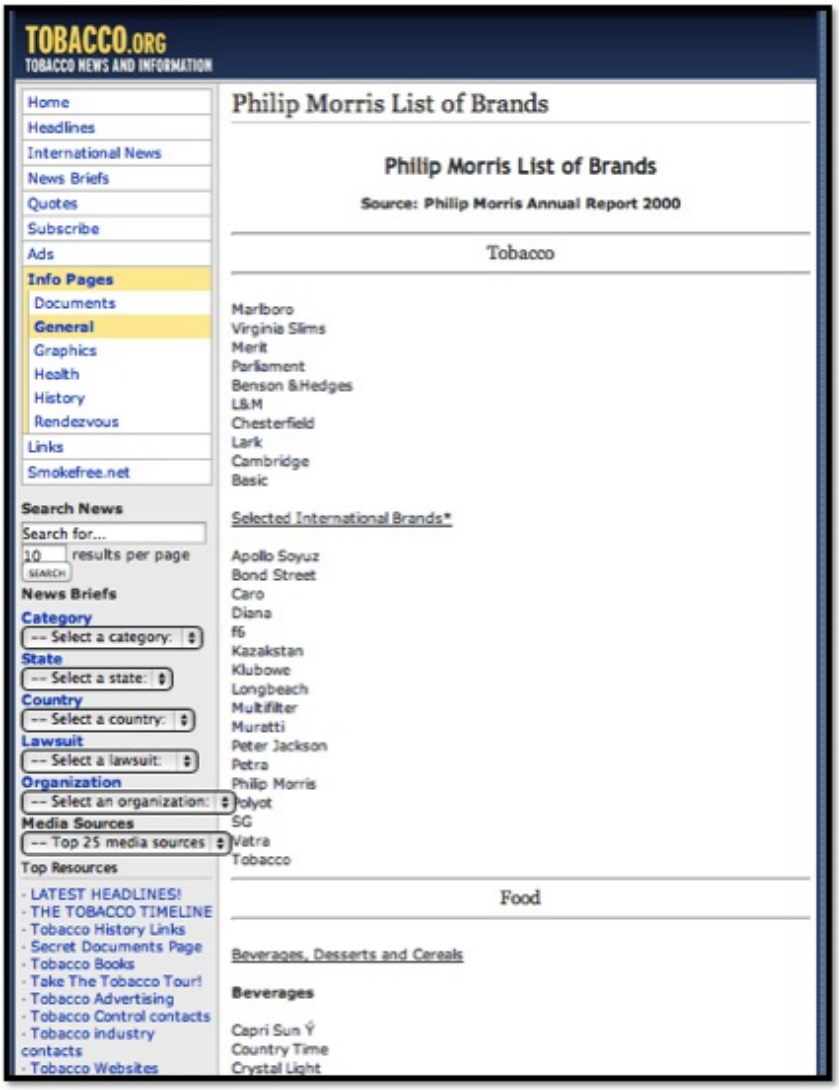


Figure 3.2: Example of a positive page

Chapter 4

Classification Methodology

In this chapter we discuss our pre-processing techniques, learning algorithms, and features selection methods used throughout our classification task.

4.1 Web Page Pre-Processing and Filtering

In this section and before focusing on algorithms, we explain about web page representation, pre-processing, and filtering obvious non-list containing items, recognized as noisy sections.

4.1.1 Web Page Representation

The dataset consists of a set of URLs and the corresponding HTML page fetched from the web. HTML TIDY [27], was used for finding and correcting errors in HTML source files. HTML Tidy is especially useful for checking and cleaning up deeply nested HTML files. It detects and fixes missing or mismatched end tags, and corrects mixed up tags.

In this work we use HTML tags to find tag-related characteristics of a web page and also for filtering out noisy sections. Most of the tags are in pair and consist of an opening and a closing tag. There are also other tags inside such tag pairs, called nested tags. Thus an HTML page can be represented in the form of a tag tree, where each node in the tree represent one pair of tags in the corresponding web page. Tag trees are often implemented as DOM trees. A DOM parser can parse HTML source stored in a string into a DOM tree. Jsoup DOM parser [16] is used in this work to

represent DOM-based tag trees.

4.1.2 Web Page Cleaning and Filtering

An HTML page usually consists of multiple data blocks. While one or more of these blocks contain the main contents of the page and are considered as the main blocks, others may consist of simply noisy. Such noises can be grouped into two categories according to their visibility in the browser:

- Invisible noises: These are noisy sections in the HTML page which are not visible to the user. Examples include HTML comments and items with invisible or hidden attribute.

- Visible noises: These are noisy items which cannot clearly represent any useful data, even a list of items. Such noises include page header, copyright notices, or items appearing in almost every web page.

These noisy items within a web page can harm the accuracy of a data mining task and web page classification. Some related work have addressed the problem of eliminating noisy sections in web pages [1, 34], by filtering all sections of the page except the main content block. For example Yi et al. [34] try to eliminate all noisy sections including navigation bars and banner advertisement. However we believe that apart from the main content block, such so-called noisy sections may also contain a useful set of lists. For example in Figure 4.1, while the horizontal navigation bar in segment 1 is a noisy section and should be eliminated, the vertical dotted area in segment 2 is a navigation bar which clearly provides a set of related items and we do not want to remove it.

As a result and in order to improve the performance of the classification task, several pre-filtering steps are needed to remove obvious non-list containing noisy sections of a page. We decompose the page DOM tree structure of a page into subtrees and remove some of the subtrees, including visible or invisible nodes. The types of sections eliminated from pages include:

- Subtrees with empty leaf nodes - A piece of data by definition must contain non-empty parsed data. All subtrees in the tag tree which do not contain non-

Figure 4.1: Different sections in a Web page

empty leaf nodes are removed from the DOM. This includes *Input* elements, producing specific user interfaces, where the user can enter data.

- *Comments* subtrees - Comments and their tags are not displayed in browsers and are removed from the source code of the pages. A *comment* tag is usually used by programmers to explain the code or to insert comments.

- *Head* section - All subtrees under the head section, except *Title* of the page, are excluded from further consideration. <*head*> tag is a container for all the head elements.

- *Footer* subtrees- The footer sections under a footer tag are eliminated from the source page. A <*footer*> tag defines a footer for a document. Apart from a footer tag, there are also other tags which wrap footer elements. For example the footer part in a page might be identified by <*div id="footer"*>. As a result, we eliminate all sub-trees in which the attribute of the HTML element contains term "footer".

- *Javascript* objects - While HTML tags generate objects, JavaScript lets users manipulate those objects, adds functionality to the page or communicate with the server, etc. Since the <*script*> tag is used to include JavaScript code directly into an HTML file, we remove internal Javascript sections from the DOM tree of a page.

- *STYLE* nodes - A <*style*> tag specifies how HTML elements should render in a browser and also defines the style information for an HTML document. Since we do not need rendering information, we remove the <*style*> tag and all the content inside it.

- *Dropdown* menus - Subtrees containing Dropdown menus, which are usually defined by <*select*> HTML tags, are excluded from a DOM tree. However these menus might contain a list of similar items, they usually contain a set of simple items for user to choose from; such as dates for date of birth, or country of residence. As mentioned, these menus are often general, hence excluded from the DOM tree.

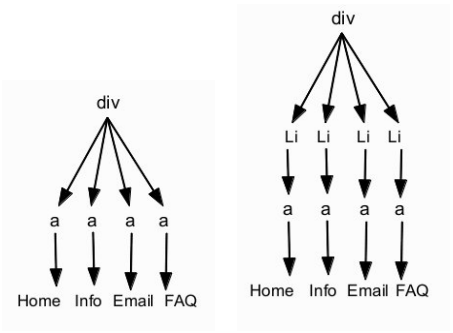
- Hidden subtrees - Usually there are some hidden sections in a page, with their contents invisible or not displayed in the browser. These parts should be discarded as much as possible. Useful heuristics for eliminating these subtrees, are searching for HTML tags in which their *display* attribute is set to *hidden* or *none*.

- Not all the footer or header elements in an HTMl page are wrapped by a special tag like <*footer*>. For example the footer of a page might be built using a table tag, and this make the task of finding such noisy sections difficult.

Table 4.1: Set of some of the keywords used for eliminating noisy sections

home, FAQ, related articles
about us, contact us
help, careers, copyright
downloads, terms of use
log in, read more, news

As an alternative, we are interested in detecting items which appear in almost every web page, eg. navigation bars at the top of the page containing *Home* and *News* buttons, or footer sections containing *Contact Us* information or links to *Terms of use*. For this purpose we use a predefined set of keywords and try to find those blocks of the page which at least x% of their total number of items (leaf nodes) belong to this predefined list. Table 4.1 shows some of the keywords used in repeating noisy sections. In order to detect such blocks, we first look for these keywords in the leaf nodes and keep their corresponding tags as well as the tags of their siblings, if any. If a corresponding tag does not have any sibling, we keep its parent's siblings as long as all of the parents siblings are the same. Then for each detected leaf, we keep searching its siblings and the siblings of its parent and also consider their corresponding leaf data items. Finally we eliminate those sections in which the total number of predefined keywords are bigger or equal than x% of all their items. We set this threshold to 50%, which means that at least 50% of leaf nodes in that block must belong to our predefined keywords. Figure 4.2(a) shows an example of a block containing a noisy section identified by a group of siblings tags. Also in Figure 4.2(b), a noisy section can be identified using identical parent's siblings.



(a) Sibling tags used to detect a noisy section

(b) Identical parent's sibling tags used to detect noisy sections

Figure 4.2: Keywords and tags used in detecting noisy sections

After all these pre-processing, we normalize all the HTML tags. Normalizing HTML tags means removing attributes and properties inside them. The reason is that in our method, we need to work only with normalized HTML tags, not their attributes. Figure 4.3 shows an example of an HTML normalized tag. Note that when removing some noisy sections using predefined keywords, we also use normalized tags, since we need to find identical sibling tags.

Figure 4.3: Example of a normalized HTML tag

4.2 Supervised Classification

Supervised classification, also called class prediction, involves assigning instances to predefined categories. Algorithms used in a supervised classification are usually developed on a set of training data and would be tested on an independent test data set to evaluate the accuracy of the algorithm. There are numerous classifier paradigms to choose from. Support Vector Machines (SVM) are a group of supervised learning algorithms based on statistic learning theory. Another group of supervised methods are decision trees. Each node in the tree represents a feature

and the tree branches out based on the values in the node. The leaf nodes represent classes. Unlike SVM classifiers, decision trees are directly interpretable.

4.3 Feature selection for classification: finding the optimal features

In this section we study the task of classifying web pages into one of the list-containing and non-list-containing categories. In order to perform a supervised learning, we need to choose a set of appropriate features. We looked into various features including structural and content based features inside web pages, carried out experiments based on them, and analyzed the performance of the classifier with each set of features.

4.3.1 Structural based Features

Structural based features depend mainly on the characteristics of the HTML tags and their nesting structure. We would call each piece of non-empty text between two tags, as *PCDATA*. Below is the list of structural based features used in our method:

- **Frequency of tag-sequences:** We define a sequence of tags as follows:
  All tags appearing between two pieces of data, are considered as the tag-sequence for the second piece of data. The idea is to find out patterns based on the sequences of tags and a repeating tag-sequence is more likely to identify a list. For each piece of data, we obtain its tag-sequence, and also all the sub-sequences of that sequence. For example for the the following sequence:
  $\{
<h2>PCDATA\}$
  All the tag-sequences are:
  - $\{
<h2>\}$
  - $\{
<h2>\}$
  - $\{<h2>\}$

We then count the frequency of each tag-sequence in the whole page. Since all pages should be considered as equally important, independent of their size, a normalization factor should be incorporated with the aim of normalizing the frequency values. The tag-sequence frequency measure is extended to incorporate the normalization factor, which is defined by Equation 4.1. The normalization factor is based on the size of the corresponding web page and an average page size, chosen to be the same for all pages.

$$Normalized\text{-}Tagseq\text{-}Frequency = \frac{Tagseq\text{-}Freq \cdot Avg\text{-}Page\text{-}Size}{Page\text{-}Size} \tag{4.1}$$

A tag-sequence should be repeated enough times to be considered a good candidate for identifying a set of items. Though, we can discard tag-sequences which are not repeated more than a predefined number of times in the whole page. By changing this value, we found out that the best number in our experiment was 5. Finally we selected the first two most frequent normalized tag-sequences as two separate features. The intuition is that the more a sequence of tags that have no data in between are repeated, the more likely they would identify a set of related items. Also unlike a single tag, a pattern of a sequence of two or more tags, such as the given example, is more likely to identify related items. For the given example, if it identifies items of a list in a page, probably the first two most frequent tag-sequences in the page would be {<*li*><*h2*>} *and* {<h2>} , with a high value for both. However for a negative page even if the first frequent tag-sequence is repeated many times (some tags such as <*a*> might be repeated in many parts of a page), it is less probable that the second frequent tag-sequence is repeated for a large number of time. As a result we chose *two* most frequent tag-sequences as two separate features.

- **Standard deviation (SD) of distances between all occurrences of a tag-sequence:** For each tag-sequence, first the distances between each pair of consecutive occurrences of the sequence is calculated and then the standard deviation

(SD) of them is obtained. The intuition behind this heuristic is that tags are more likely to identify a list, if the standard deviation of the size of the data between all occurrences of them is low. Note that we assign a high value of SD to a pattern which is repeated less than two times, since its SD would be zero and we do not want to consider it as a candidate.

We then chose the corresponding standard deviations for the two frequent tag-sequences, selected in the previous section, as two more separate features.

- **Ratio of frequency of tag-sequences to SD:** Best candidate tag-sequences, which identify a list of items, have a higher frequency and a lower standard deviation between their occurrences. The ratio of *frequency of tag-sequences* and its *standard deviation* for the two top selected tag-sequences are used as two more features.

- **Highest ratio of frequency of tag-sequences to SD:** A weight is assigned to each tag-sequence based on the ratio of its frequency and its standard deviation. We then rank all these weights and select the highest one as another feature. Usually a higher weight identifies a denser section in the page.

- **Normalized-Number-of-PCData:** We count number of PCData in a web page and normalize it based on the size of the page and the average page size. The idea is that pages with a larger number PCData usually are more likely to contain a list, since a list carry a group of items. On the other hand we observed that pages which do not contain any list usually had few pieces of data. An example is a page which contains a *story* or a *letter*.

- **Average Length of PCData:** For each web page the average size of PCdata is calculated (in terms of the number of characters) and used as another feature. The intuition behind this heuristic is that average length of data in web pages which do not contain any kind of lists, is usually higher than pages containing list. For example in a *forum* page, we usually see some chunks of long text.

4.3.2 Page Content and Title based Features

Terms from title and content usually contain useful information about the category of a particular web page. For example, analyzing web pages manually revealed that list-containing pages often contained words such as *list*, *item*, and *table*. We followed the Bag-of-words (BOW) approach and extracted unigrams from the text of the HTML pages. The reason for using Bag-of-words method for extracting words from title and context is that, these parts are usually in the form of normal texts with well-separated words. Description of Metadata tags are also added to the BOW, and a set of keywords are selected based on the mutual information. We now discus how we extract features from title and context:

- **Titles:** After using separators and tokenizing each title , the Bag-of-words method is applied and unigramsin the are extracted. A title may contain multiple segments, which are separated by separators like ”—”, or ”-”. Among all these segments, we only use the first segment, since it is the main segment showing the topic of the page, while other segments provide additional information about the page.

- **Context:** A web page text without any HTML tags is considered as its context. In order to extract this context, we first remove all HTML tags and then the contexts is tokenized. Finally we applied the Bag-of-words approach to extract unigrams from the context.

- **Description from Metadata Tags:** We gather words used in the content attribute of the *<META name=”keywords”>* and *<META name=”description”>* tags. The *keywords* mata element, identifies itself as containing the keywords relevant to the document. While the meta *description* often appears in Google search results to describe the link. We combine terms obtained from context and also metadatatags together.

- **Grammar of the headword in title:** By applying *Stanford CoreNLP* on title phrases, dependencies between terms and then the headword of the phrase is obtained. The headword of a phrase is its most important word, which all of

the other words modify that. Having the headword, we use its grammar and plurality as other features. Usually the head word in the title of a positive page, is a *Noun phrase* and is plural (eg. *List of all cellphones*).

4.3.3 Dimensionality Reduction

Applying Bag-of-words on the data set, generates a high number of distinct words and we need to reduce the number of dimensions. Dimensionality reduction is a necessary process to avoid the over-fitting problem. Over-fitting generally occurs when the model fits the training data in the sense that it will have poor predictive performance for new unseen data.

We first use the most intuitive and simplest heuristics for dimensionality reduction by removing the following terms from the BOW:

- Numbers; Numbers are special terms not considered relevant to the task of classification.
- Stop words; A standard stop word list is considered for removing general terms such as prepositions, articles, conjunctions, pronouns and some adverbs.
- Short terms; Terms with less than 3 characters in length are eliminated.
- Rare terms that occur two or fewer time; Based on Zipf's law few terms occurs frequently while many terms occurs rarely [11]. Therefore removing rare terms would lead to a great saving in feature space.
- Terms used as queries in the Google search; Without removing search query words, collection of top selected features might be biased towards those words.

Context Feature Weighting

Even after removing terms such as stop words, and rare terms, still number the of dimensions is too high to be appropriate for the task of classification and we need to perform more dimensionality reduction to elect the most informative features.

One of the main approaches for dimensionality reduction is feature selection. Generally feature selection methods aim at selecting some of features that have higher importance to the classification process. Among several feature scoring methods, we adapt expected *Mutual Information* (MI), as the utility function for selecting useful terms. In probability theory, the mutual information is a quantity that measures the mutual dependence of two random variables. In this context, MI measures how much information a term contains about a class compared to another class. If a term distribution is the same in the class as it is in all pages, then MI=0. On the other hand an ideal word which happens in all of the pages in one class, but not in the other class, is a perfect indicator for that class. In such case MI reaches its maximum value.

Formally, the Mutual Information of a term and class pair is calculated as shown in Equation 4.3 [23]:

$$MI(T;C) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} P(T = e_t, C = e_c) Log_2 \frac{P(T = e_t, C = e_c)}{P(T = e_t)P(C = e_c)} \quad (4.2)$$

where T is the discrete random variable "feature" that takes the value $e_t = \{1, 0\}$ (feature T occurs in page or not), and C is the discrete random variable "class' that takes the values $e_c = \{1, 0\}$ (page belongs to class or not).

The probabilities can be estimated by using various page counts from the training data. Therefor Equation 4.3 is equal to Equation 4.2 [23]:

$$\begin{aligned} MI(t;c) = & \frac{N11}{N} log_2 \frac{NN_{11}}{N_{1.}N_{.1}} + \frac{N01}{N} log_2 \frac{NN_{01}}{N_{0.}N_{.1}} \\ & + \frac{N10}{N} log_2 \frac{NN_{10}}{N_{1.}N_{.0}} + \frac{N00}{N} log_2 \frac{NN_{00}}{N_{0.}N_{.0}} \end{aligned} \quad (4.3)$$

Where N is the total number of samples, and N_s are counts of pages that have two subscripts, showing values of 0/1 for term and class. For example N_{11} is the number of pages that contain the term and are in the class. $N_{1.} = N_{10}+N_{11}$ shows number of pages that contain the term without considering the class.

MI measure the contribution of presence or absence of a term in the classification task. This means that while the presence of a term like "list" in a positive

page is a good indicator for classifying that page in positive class, the absence of the same term is also a good indicator for classifying the page in negative class.

Finally to select k terms for positive class, we rank all terms based on their scores and select those with a score greater than a predefined threshold. The smaller the threshold, the more features would be selected. For example, Table 4.2 shows the top 10 terms for the list-containing web pages, obtained from the context.

Table 4.2: Top 10 terms obtained from the page context for list-containing Web pages

list	0.1289
top	0.1128
products	0.1005
international	0.0911
cart	0.0817
corp	0.0705
company	0.0699
usa	0.0697
sale	0.0668
supplies	0.0666

4.3.4 Examining List-Related Tags

In order to investigate the efficiency of list-related HTML tags in distinguishing a list-containing page, we tried to collect the set of all tags which are usually used for defining a list or table in HTML pages. As far as we know, these tags are as below:

- HTML tags used in defining a Table: {*table, th, tr, td, caption, colgroup, col, thead, tbody, tfoot*}
- HTML tags used in defining a list: {*ol, ul, li, dl, dt, dd*}

We defined a feature based on the normalized total number of these tags for each data point. Further investigation and results showed that this is not a good feature, since it causes less accuracy in classifying results. This shows that not all the web pages which contain lists or tables, always contain these types of HTML tags. On the other hand, there are many web pages containing these types of tags, but at

the same time they do not contain any list or table of similar items. Thus we can point out that not all list-specific HTML tags such as <*li*> and <*lL*>, are useful for identifying a list-containing page.

4.4 Classification Experiments and Evaluation

For the classification task, we experimented with several classifiers provided by Weka [15], namely decision tree based J48, SVM based SMO, and Logistic Regression (LR). We set parameters of classifiers to their default values. Also we report 10-fold cross validation results for these classifiers.

For comparison purposes, we considered two baselines. The first baseline (List/Table HTML Tag baseline), treats pages that contain at least one list/table HTML tag as positive class and others as negative. The most common HTML lists are ordered and un-ordered lists. An un-ordered list starts with the <*ul*> tag, while an ordered list is identified with the <*ol*> tag. We also consider description list, which is a list of terms, with a description of each term. This baseline is based on the fact that the most intuitive ways for creating multiple, similar items on a page, are using list and tables tags. Second baseline (*"List/Table"* term baseline) treats pages that contain *list/table* terms as positive class. It is different from the first baseline, as it only considers terms in a page, not tags. It considers page terms after tokenization.

We evaluate the quality of classification in terms of F-measure, which is the harmonic mean of precision and recall. Table 4.3 shows the confusion matrix for a binary problem.

Table 4.3: Confusion matrix for a binary classification problem

		True Class	
		Positive	Negative
Test Predictions	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

- **Precision (P):** Precision is defined as $\frac{TP}{TP+FP}$. In binary problems, precision describes the proportion of actual positive examples that are correctly identified.

- **Recall (R):** Recall is defined as $\frac{TP}{TP+FN}$. In binary problems, recall measures the fraction of positive examples that are correctly retrieved.

- **F-measure:** F-measure is a harmonic measurement defined as:

$$2.\frac{Precision \times Recall}{Precision + Recall}$$

  There is a trade-off between precision and recall: an increase in precision can lower recall, while an increase in recall lowers precision.

4.5 Classification Results and Discussion

Table 4.4 shows the results of the classification for our data set. Since having a large number of dimensions causes over fitting, while a small number cannot represent enough, we have to carefully choose a reasonable number of dimensions for context and title features. We tried different numbers and found out that a number between 10-20 is a reasonable number. Results are based on the 15 features obtained from context and title in total.

Table 4.4: Classification results using all features and 10-fold cross validation vs baselines

	Precision	Recall	F-measure	Std deviation of F-measure
SMO	86.4	85.8	85.8	0.05
J48	83.6	83.6	83.6	0.06
LR	85	84.8	84.8	0.05
List/Table Tag	51.3	86.4	64.3	NA
"List/Table" Term	54	82.8	65.4	NA

As Table 4.5 shows, both baselines have a high recall compared to other methods. The *List/Table Tag* baseline has a higher recall than the *"List/Table" Term* baseline but it also has a lower precision. We see that by combining context and title based features with structural features higher accuracy is obtained. For each classifier the standard deviation of F-measure results is also displayed.

In order to check if there is a significant improvement in our method or not, we applied t-Test on results obtained from each classifier and each baseline method.

The null hypothesis is:

Our method has no significant improvement.

According to t-Test results, we reject the null hypothesis at α=0.01 for all our classifiers compared to the list/table baseline, meaning that the difference is significant.

Table 4.5 shows results based on different sets of features. SMO is performing better for context and title features, while J48 is better for structural features.

Table 4.5: Classification results for combination of feature sets

		Titles	Context	Structural
SMO	Precision	74.8	85.2	77.2
	Recall	71.9	84.8	77.1
	F-measure	71	84.8	77.1
J48	Precision	70.3	82.6	79.4
	Recall	65.9	82.3	79.4
	F-measure	64	82.3	79.4
LR	Precision	74.8	83.1	72.3
	Recall	71.9	82.8	71.6
	F-measure	71	82.8	71.4

4.6 Ranking Structural features

In order to analyze features correlation we run feature selection from the *Select attributes* tab in Weka Explorer to see which features are more important. Table 4.6 illustrates the results after ranking structural features. It shows that the highest ratio of frequency of tags to SD is the most impressive feature, while the least impressive one is second SD of distance between tag-sequences.

Table 4.6: Ranking individual structural features

Feature's Rank
1-Highest Freq-TagSeq/SD
2-First (Freq-TagSeq)/(SD)
3-AVG-PCdata-Length
4-First Freq-Tagseq
5-Second (Freq-TagSeq)/(SD)
6-Normalized-Number-of-PCData
7-First SD-Dist-TagSeq
8-Second Freq-Tagseq
9-Second SD-Dist-TagSeq

Chapter 5

Locating Lists in a Page

5.1 Locating lists in a page

Having web pages which contain one or more list of items, we next need to find and extract lists inside the page. We would consider the problem of detecting lists in the web page as a classification problem, unlike many other approaches which try to identify record boundaries that allow determining which groups of data belong to a same object/record. For this purpose we randomly choose 50 positive pages and manually label start and end positions of lists in these pages (a page might contain more than 1 list). Here are the statistics of lists and their length distribution. In our sample set, %25 of lists included 10 items or less, about %60 of them contained 10-50 items, and only %15 of lists contained more than 50 items. Taking into account the number of terms inside each list, Table 5.1 gives the number of terms per each category (after tokenization and not including extra terms such as opening and closing tags). It shows that the majority of lists contain between 100 and 500 terms. Only 5% of lists contain more than 10000 terms.

Table 5.1: Distribution of lists size based on the number of tokens for a random sample of 50 pages

#Terms in lists	Percentage of lists
Less than 100	26%
100-500	41%
500-1000	16%
1000-10000	12%
More than 10000	5%

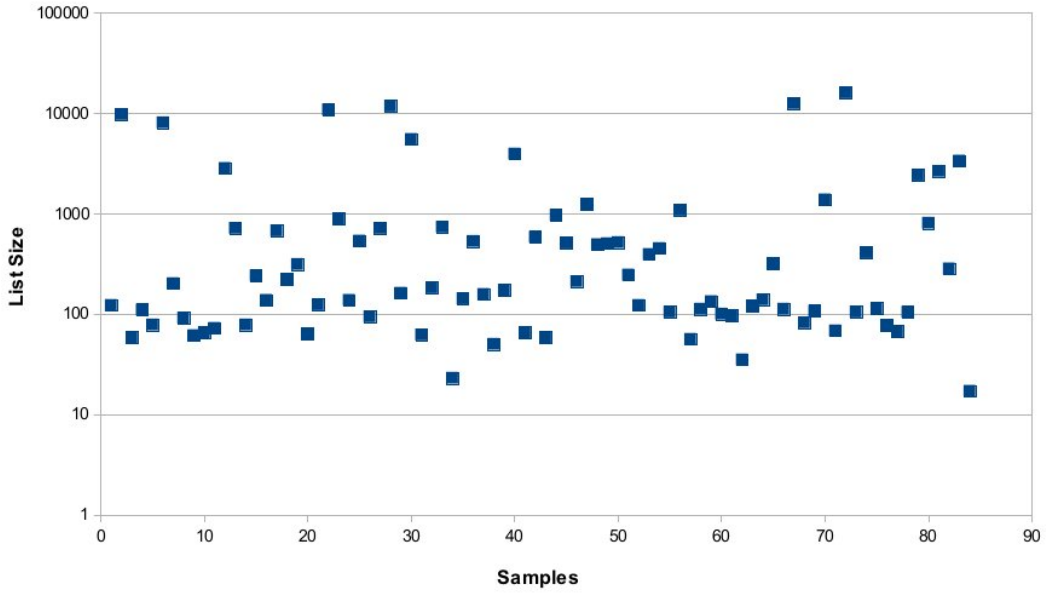


Figure 5.1: Logarithmic distribution of lists size

We then apply our classifier on consecutive pieces of data inside each page to locate the place of lists in that page. We start by breaking down each page into some sub-pages and making a new set of training data, which consists of smaller parts of pages. Having all these small pieces of pages, we then label them into positive and negative classes. By choosing a *window-size* and a *shift-size* we divide pages into these smaller sub-files. Window-size is the size of a section for which we apply our classifier. In order to avoid having too many sub-files, a shift-size is introduced and each window is shifted by the shift-size. Indeed shift-size is used to shift the start position of the window. These two parameters can be set based on the number of terms or characters. For example if window-size is set to 1000 characters, it means that every time that we shift the window, it would scan next 1000 characters. Each time we fix the window-size or the shift-size and build a set of test sample of sub-files. By applying our classifier on these sets of sub-files, we are able to calculate precision, recall and F-measure. As an example Figure 5.2 shows the correlation coefficient of classifier when the shift-size is fixed at 200 terms and for different values of window-sizes.

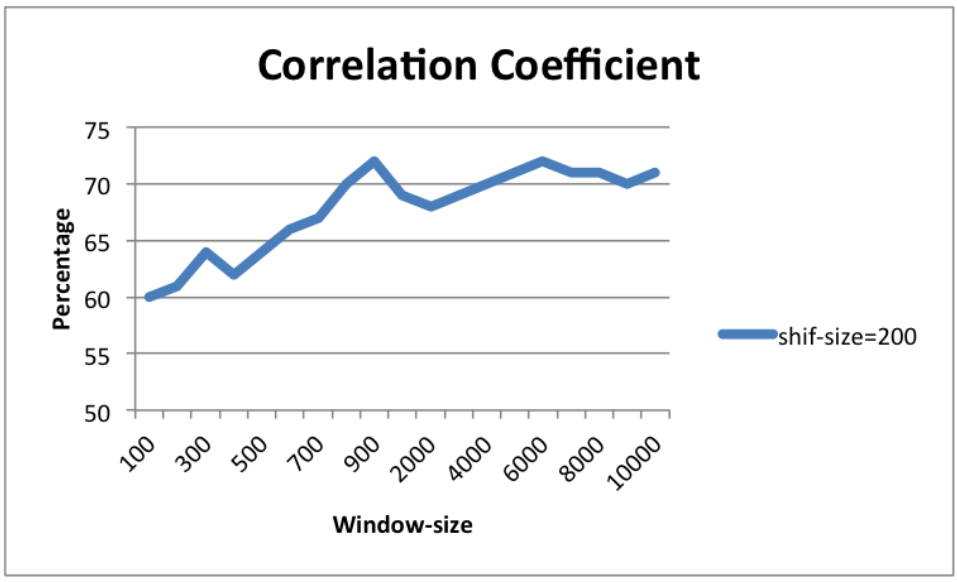


Figure 5.2: Correlation coefficient of the classifier for a fix shift-size and different window-sizes

5.1.1 Experiments

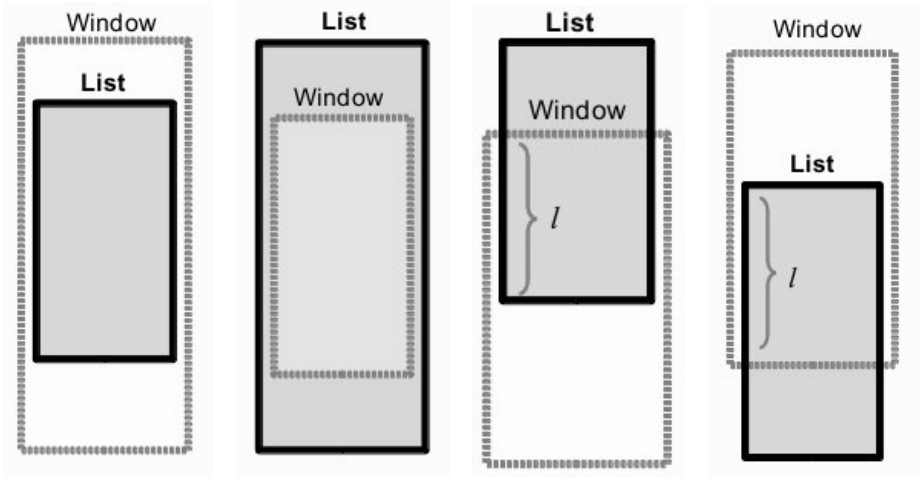
We divided each of the 50 positive random pages to three sub pages and categorized them into one of the 11 possible combinations of the labels given in Table 5.2. Note that we use the same title for all the sub-files of a page. Here *M* refers to two lists or more in that part of the page. *N* means that the sub-page doesn't have a list in that section, and *Y* means that sub-page includes a list in that section. Note that we do not consider N N N , since there is no positive page which does not contain any list. Results show that the classifier makes less mistakes in the beginning and at the end of the page, while most of the mistakes are happening in the middle of the page.

In order to analyze the effect of sliding different windows on the same set of test files, each time we need to label the generated sub-files with their real label. Even though we know the exact location of lists for each file in the ground truth, building the ground truth for sub-files needs more effort and explanations. Because of the overlapping relationship between a window and a list, after shifting a window over a page, the window can be placed in different positions according to the start and the end positions of the list. Figure 5.3 shows four possible situations regarding the positions of both a window and a list. Figure 5.3(a) shows a case where the

Table 5.2: 11 Possible combinations of the labels for 3 Sub-pages

set	%Pages	Accuracy
N N Y	4	70
N Y N	22	84
N M N	10	72
N Y Y	11	68
N M Y	7	72
Y N N	3	70
Y Y N	2	75
Y M N	9	65
Y N Y	2	60
Y Y Y	23	91
Y M Y	7	87

window covers the list, which means window size is larger than the list size. In Figure 5.3(b), window size is smaller than the list size and the window sits inside the list. In both cases we can easily determine the real label for the sub-file. Figure 5.3(c) shows the case in which the window starts after the starting point of the list and ends after the ending point of that list. In Figure 5.3(d) the window starts before the starting point of the list and also ends before its ending point. However in the last two situations, labeling the sub-file depends on the length of the list that sits in the window (l) ; if the window covers a very small portion of the list, we really cannot assign Y label to the sub-file. As a result we carefully choose l and set l to 50 tokens if the size of the list is greater than 500 (based on the number of tokens); for smaller lists at least %10 of the list should sit inside the window for receiving Y label.



(a) Window covers list (b) Window falls inside List (c) Window starts and ends after list (d) Window starts and ends before list

Figure 5.3: Four possible situations of a window and a list

Table 5.3 shows the average of top three window sizes for different list sizes, while windows ranges are set to 50. Here we do not mention lists greater than 1000, since more than %80 of lists are smaller than 1000. These results show that for each set of lists, top window-sizes are almost in the same range as the size of the list. Also our experiments show that in general, the best shift-size is the smallest one.

Table 5.3: Top 3 window-sizes for different lists sizes, with a fix shift-size

Lists size	Average of Top 3 window-sizes
Less than 100	50, 100, 200
100-500	150, 200 , 350
500-1000	700, 750 ,850

In order to compare results with a baseline, we consider a baseline based on the presence of HTML list related tags in a page. As in the previous section we consider $<OL>$, $<UL>$, and $<table>$ as our list tags. Since these tags have both opening and closing ones, they could easily identify the start and the end positions of lists. Results show that this approach have a high recall, however comparing to our method the precision is very low. Table 5.4 shows the results of locating lists using our method and the baseline.

Table 5.4: Total results for locating lists vs baseline

	Precision	Recall	F-measure
Our Method	75.1	74.3	74.7
List/Table Tag Baseline	50.1	85.3	63.17

Chapter 6

Increasing the Accuracy of Detection

In this chapter we study some of the properties of the sequences generated by the classifier for each page. Since detecting lists in a page based on the proposed classifier is not always accurate, we would like to improve the accuracy of detection by rejecting some of the sequences that are less likely to indicate a list. We model the problem as a Markov Chain and try to optimize the likelihood of the predicted sequences, therefore increasing the accuracy.

6.1 Markov Model

The assumption that the probability of a state depends only on the previous state, is called Markov assumption. Given the Markov assumption for the probability of an individual symbol, we can compute the probability of a complete symbol sequence using Eq. 6.1

$$P(s_1^n) \approx \prod_{k=1}^{n} P(s_k|s_{k-1}) \tag{6.1}$$

Maximum likelihood estimation (MLE) is used for estimating the bi-gram or N-gram probabilities. The MLE estimate for the parameters of an N-gram model is obtained by normalizing counts from the training data, and normalize them so that they fall between 0 and 1 [18]. For example the bi-gram probability of a symbol *y* given a previous symbol *n* is obtained by computing the count of the bi-gram (ny) and normalizing by the uni-gram count for symbol n:

$$P(y|n) = \frac{C(ny)}{C(n)} \tag{6.2}$$

We first augment each sequence of symbols with a special symbol $<s>$ at the beginning of the sequence, to give us the bi-gram context of the first symbol. We also use the special ending symbol $<e>$, for the bi-gram context of the ending symbol.

6.2 Rejecting Less Probable Sequences

Our target is predicting better sequences of symbols based on the best probability suggested by the Markov model. According to a Markov Model of order one, the next state depends only on the current state of the sequence of events that preceded it. We first build a Markov Model, then pass the results predicted by our classifier to the Markov Model and this model would suggest to change some symbols in each sequence in order to get a better accuracy. The idea is that based on the ground truth and the Markov Model which is built on the ground truth, some sequences are more probable than others. For example the probability of seeing 'YYYYYN' is more than 'YYNYYN'. So the model would suggests to change the first sequence to the second one.

We should note that the average length of sequences in our data set was 74 symbols (for the sequences obtained by the best fixed window-size and shift size). Most mistakes in our predicted results (before applying the Markov Model) are located in the middle of the sequence, but not in the beginning and at the end of the sequence. The reason is that for most of the pages lists are mainly located in the middle sub-file of the page. The MM could fix problems which are located in the middle of the sequence easier than those located exactly in the beginning and at the end of the sequence.

Figure 6.1 shows the average $\Delta Accuracy$ for all sequences, based on the percentage of symbol changes of sequences. This figure illustrates that on average based on the suggested Markov model, best increment of accuracy is obtained when %6 of symbols are changed in each sequence. In order to compare this result with the ground truth we calculated the percentage of changes needed to convert the predicted sequence, which was obtained by our classifier, to the sequence built based

on the ground truth. We observed that based on all the test sequences an average of %21 symbol changes was needed. This provides an upper bound threshold for our experiments.

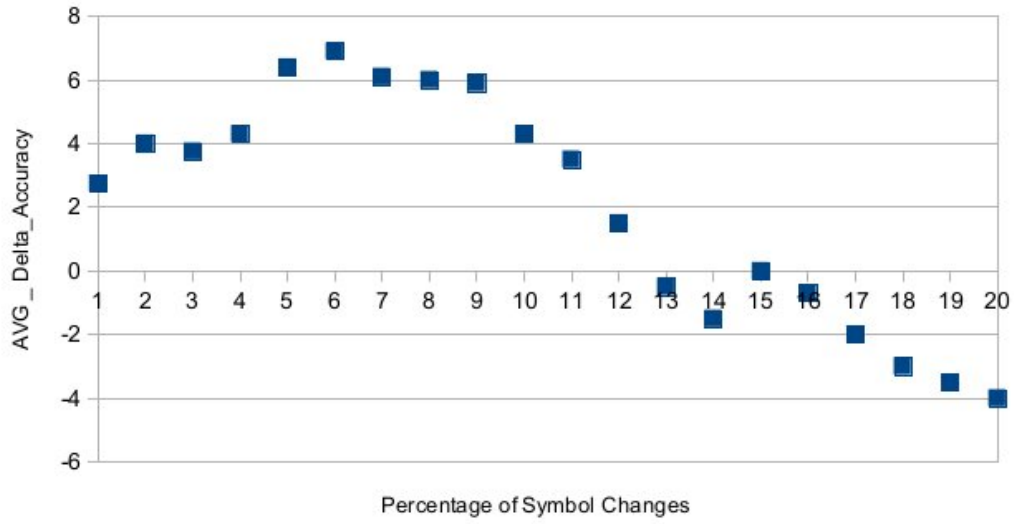


Figure 6.1: Average of $\Delta Accuracy$ for all sequences

Table 6.1 shows overall precision, recall, and F-measure obtained by changing a specific percentage of symbols in the total sequences of symbols. It shows that the overall accuracy are increasing due to using Markov predictions.

Table 6.1: Results for locating lists after applying Markov Model vs baseline

	Precision	Recall	F-measure
Our Method + MM	80.4	82.8	81.58
List/Table Tag Baseline	50.1	85.3	63.17

Chapter 7

Conclusions and Future work

In this dissertation we mainly considered list-containing classification of web pages using machine learning algorithms. We addressed that not all the web pages contain a set of similar items and explored multiple feature sets that can correctly predict the label of a web page and detect the location of lists in the page. We showed that while previous methods handling record boundary detection only use heuristics for locating groups of data records, structural features based on the HTML tags and their structure provide strong clues about detection of a list-containing web page. We also showed ways to combine context based features with structural ones to achieve higher accuracy. We showed that when searching for the location of the lists inside a Web page, dividing the page in to smaller sub-files and searching for a set of similar items in each sub-file is effective. We fulfilled this goal by applying a classifier on each sub-file and assigning a label to it. Finally we gained better performance by changing some predicted labels using a probabilistic model based on Markov Chain. Regarding our data set, one of the advantages of our method is that it is domain independent.

7.1 Future work

Further work is required for increasing the accuracy of detecting lists in a Web page. This might be achieved by investigating a Markov Model of higher order.

However our method mainly supports for semi-structured data, since the most important features rely on the HTML source, it also can be a baseline for semi-

structured text. Semi-structured pages contain free text from which data items can be inferred. The reason is that in our method page context attributes are used, which exist and can be found in free text pages. Of course by using more semantic and syntactic information, supporting for non-HTML pages would be also possible.

Some of the future work for distinguishing ambiguous list boundaries and identifying unwanted lists could be done by using semantic information. Further work can also be done on filtering out irrelevant list items. As an alternative identifying more clear and understandable lists could be taken in to account. Examples of these types of enriched information are top-k lists, which are more interesting for human.

Another direction for future research could be identifying the *Title* of each list. In this way each set of list items as well as its category, would be used as an important source for further data mining tasks.

As an further application of this research, extracted lists might be used to annotate and extract relationships between entities in the world. Entities like people, which are usually found together in a list, are more similar than those which are not frequently found together.

Bibliography

[1] Z. Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. In *Proceedings of the 11th international conference on World Wide Web*, pages 580–591. ACM, 2002.

[2] S. Brin. Extracting patterns and relations from the world wide web. In *The World Wide Web and Databases*, pages 172–183. Springer, 1999.

[3] D. Buttler, L. Liu, and C. Pu. A fully automated object extraction system for the world wide web. In *Distributed Computing Systems, 2001. 21st International Conference on.*, pages 361–370. IEEE, 2001.

[4] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *Proc. VLDB Endow.*, 1(1):538–549, Aug. 2008.

[5] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. Extracting content structure for web pages based on visual representation. In *Web Technologies and Applications*, pages 406–417. Springer, 2003.

[6] C.-H. Chang, M. Kayed, M. Girgis, and K. Shaalan. A survey of web information extraction systems. *IEEE Trans. Knowl. Data Eng.*, 18(10):1411–1428, 2006.

[7] H.-H. Chen, S.-C. Tsai, and J.-H. Tsai. Mining tables from large scale html texts. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, pages 166–172. Association for Computational Linguistics, 2000.

[8] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: automatic data extraction from data-intensive web sites. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pages 624–624, New York, NY, USA, 2002. ACM.

[9] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment*, 2(1):1078–1089, 2009.

[10] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134, 2005.

[11] G. Forman. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*, 3:1289–1305, Mar. 2003.

[12] D. Freitag. Information extraction from html: Application of a general machine learning approach. In *AAAI/IAAI*, pages 517–523, 1998.

[13] F. Fumarola, T. Weninger, R. Barber, D. Malerba, and J. Han. Extracting general lists from web documents: A hybrid approach. In *Modern Approaches in Applied Intelligence*, pages 285–294. Springer, 2011.

[14] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak. Towards domain-independent information extraction from web tables. In *Proceedings of the 16th international conference on World Wide Web*, pages 71–80. ACM, 2007.

[15] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. The weka data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[16] J. Hedley. Jsoup. http://jsoup.org/, 2009.

[17] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information systems*, 23(8):521–538, 1998.

[18] D. Jurafsky and J. H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

[19] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1):15–68, 2000.

[20] A. H. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *ACM Sigmod Record*, 31(2):84–93, 2002.

[21] L. Li, Y. Liu, and A. Obregon. Visual segmentation-based data record extraction from web documents. In *Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on*, pages 502–507. IEEE, 2007.

[22] B. Liu, R. Grossman, and Y. Zhai. Mining web pages for data records. *Intelligent Systems, IEEE*, 19(6):49–55, 2004.

[23] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[24] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, and L. E. Moser. Extracting data records from the web using tag path clustering. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 981–990, New York, NY, USA, 2009. ACM.

[25] R. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 328–334, 1999.

[26] G. Penn, J. Hu, H. Luo, and R. McDonald. Flexible web document analysis for delivery to narrow-bandwidth devices. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 1074–1078. IEEE, 2001.

[27] D. Raggett. HTML tidy. http://tidy.sourceforge.net/, 2004.

[28] S. Sarawagi. Automation in information extraction and data integration. Tutorial at the Intl' conf on Very Large DataBases 2002,Hongkong., 2002.

[29] H. Sleiman and R. Corchuelo. A survey on region extractors from web documents. 2012.

[30] S. Tong and J. Dean. System and methods for automatically creating lists, Mar. 25 2008. US Patent 7,350,187.

[31] R. C. Wang and W. W. Cohen. Language-independent set expansion of named entities using the web. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 342–350. IEEE, 2007.

[32] Y. Wang and J. Hu. A machine learning based approach for table detection on the web. In *Proceedings of the 11th international conference on World Wide Web*, pages 242–250. ACM, 2002.

[33] T. Weninger, F. Fumarola, R. Barber, J. Han, and D. Malerba. Unexpected results in automatic list extraction on the web. *ACM SIGKDD Explorations Newsletter*, 12(2):26–30, 2011.

[34] L. Yi, B. Liu, and X. Li. Eliminating noisy information in web pages for data mining. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 296–305. ACM, 2003.

[35] X. Yin, W. Tan, and C. Liu. Facto: a fact lookup engine based on web tables. In *Proceedings of the 20th international conference on World wide web*, pages 507–516. ACM, 2011.

[36] Z. Zhang, K. Q. Zhu, and H. Wang. A system for extracting top-k lists from the web. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1560–1563. ACM, 2012.

[37] Z. Zhang, K. Q. Zhu, H. Wang, and H. Li. Automatic extraction of top-k lists from the web.