DOMAIN PARKING RECOGNIZER: AN EXPERIMENTAL STUDY ON WEB CONTENT CATEGORIZATION

by

Heng Du

B.Sc., Simon Fraser University, 2009

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

© Heng Du 2013
SIMON FRASER UNIVERSITY
Summer 2013

All rights reserved.
However, in accordance with the *Copyright Act of Canada*, this work may be reproduced without authorization under the conditions for "Fair Dealing." Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.

APPROVAL

Name: Heng Du

Degree: Master of Science

Title of Thesis: Domain Parking Recognizer: An Experimental Study On Web Content Categorization

Examining Committee: Dr. Wo-Shun Luk
Chair

Dr. Qianping Gu,
Professor, Computing Science
Simon Fraser University
Senior Supervisor

Dr. Jiangchuan Liu,
Associate Professor, Computing Science
Simon Fraser University
Supervisor

Dr. Jian Pei,
Professor, Computing Science
Simon Fraser University
Examiner

Date Approved: July 23rd, 2013

Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the "Institutional Repository" link of the SFU Library website (www.lib.sfu.ca) at http://summit/sfu.ca and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, British Columbia, Canada

revised Fall 2011

Abstract

Text based web content categorization is an important area in web data mining. It may be time and bandwidth consuming to categorize a web site based on its text contents. On the other hand, 30% $\sim$ 40% of the daily new registered domain web sites are hosted for advertisement purpose, known as domain parking web sites. It is more resource efficient to exclude the domain parking web sites before a web content categorization algorithm is applied. However, our study shows that the existing web content categorization methods do not work well for recognizing the domain parking web sites. In this thesis, we propose a new domain parking recognizer (DPR) to find the domain parking web sites. Our DPR evolves from the text based web content categorization algorithms and has two key components: key features of domain parking web sites and a tailor-made algorithm. The experimental results show that our DPR has a much better performance than the well known web site categorization methods Naïve Bayes and Support Vector Machine for recognizing domain parking web sites. Our DPR is also time efficient.

Keywords. Domain parking recognizer, Machine learning, Web content categorization, Authority name server, MapReduce, MongoDB, Feature confidence index.

To my family

"Stay Hungry. Stay Foolish."

— *Stanford Commencement Speech,* Steve Jobs, *2005*

Acknowledgments

I would like to express my special thanks of gratitude to my mentor, Dr. Qianping Gu, who gave me the opportunity to do this thesis study. He not only brought me into the gate of the graduate school of computer science, but also showed me a very promising study area - Machine Learning. He also inspired me many great ideas in this thesis. Without his guidance, the study could not be completed.

I would also like to thank my teacher, Dr. Jiangchuan Liu, who gave me many suggestions for research topics, Dr. Jian Pei, who helped me to overcome difficulties that I encountered in this thesis study, and Dr. Wo-Shun Luk, chair of my thesis defense, for devoting his time to help me complete my thesis study.

In graduate school, I acquired abundant knowledge. Many thanks to Dr. Greg Mori, who taught me the fundamental knowledge of Machine Learning. Many thanks to Dr. Ke Wang, who led me to study in data mining area. Many thanks to Dr. Arthur Liestman, who introduced advanced knowledge of algorithms to me.

I would like to thank all professors, teachers and students in the department of computer science, who taught me, instructed me, corrected me, advised me, inspired me, helped me, and even almost failed me. All study experience with you are precious memories in my life.

Last, I would like to thank my family, especially my wife and my daughter. Without their supports, I could not complete my thesis study.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The World Wide Web (WWW) on the Internet provides an easy way for people to retrieve information from the web pages. There are hundreds of millions web sites that provide the web pages and millions of new web sites are added annually. Figure 1.1 shows that, there are more than 633 million hostnames hosting web sites on the Internet as of December 2012[19].

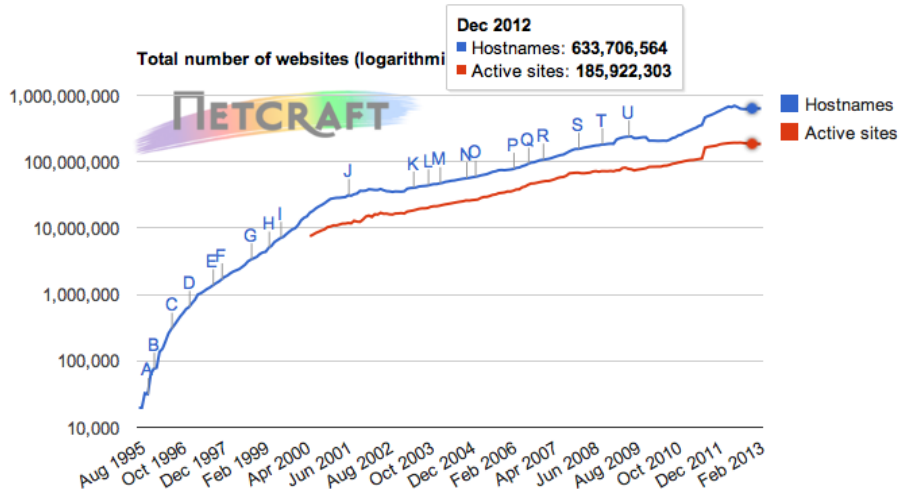


Figure 1.1: Total number of web sites on the Internet

Due to the huge number of web sites, it is a challenge to find the web sites that contain the useful information efficiently. An important research area in web information retrieval is web content categorization that assists people to find useful information from web sites efficiently. Several approaches have been developed for the web categorization. These approaches

categorize a web site based on the contents in the web pages at the web site. It is non-trivial and may be resource consuming to categorize a large number of web sites by examining the web contents. On the other hand, a huge number of web sites are created to seize the domain names for sale or advertisement without any meaningful information. Such a web site is known as a *domain parking web site*. It is more resource efficient to exclude the domain parking web sites when web contents based algorithms are applied to web content categorization. In this thesis, we propose efficient algorithms to identify the domain parking web sites.

In this chapter, we introduce the motivation of studying the domain parking web sites problem. We point out the limitation of the widely studied text categorization algorithms for the domain parking web sites problem. Then we summarize our contribution on developing new algorithms for the domain parking web sites problem. Finally, we give the structure of the thesis.

1.1 Motivation

1.1.1 Web Content Categorization

As one area of web data mining, web content categorization is widely studied in many areas, including academic research and commercial applications. The Internet is a globalized library in which people in the whole world are able to access and retrieve the information anywhere. However, unlike books in a library, web sites (web documents) are not archived by their categories when they are launched. We have to deliberately categorize web sites thereafter. For example, search engine vendors, such as Google, mainly focus on advertisement business. They provide more than 4000 categories for their billions of indexed web sites[12]. Another example is in web security area. In order to protect end users from going to malicious web sites containing malware and phishing purpose, or, to restrict end users to access illegal adult web sites, the web security providers may provide around 100 categories, which will be applied to ingress and egress network traffic policy[10].

In general, most of web contents refer to text content and image content. We ignore multimedia content in this thesis study because most of multimedia web sites are along with text content and image content. The techniques for web content categorization are mainly based on text-based categorization and image-based categorization.

Text-based Categorization

Text-based classification or categorization is to classify the web sites into pre-defined classes based on the text contents of the web sites. Text based classification dates back to early 1960s[24] and becomes a major tool for web document categorization and indexing with the popularity of the Internet in the middle of 1990s[24]. Several supervised machine learning models using training samples have been investigated for the text based classification[3]. There are two phases in these supervised machine learning models: learning and categorization. In the learning phase, a classification algorithm F is created by a set of training samples. Each training sample is a web site and the class of the web site decided by the features of the web site. The features of a web site include a set of pre-defined keywords or phrases. In the categorization phase, given an un-classified web site $W = \{w_1, w_2, \ldots, w_n\}$, where each w_i is a feature, F classifies the web site W based on the features $\{w_1, w_2, \ldots, w_n\}$ of the web site into a class in a pre-defined set $C = \{c_1, c_2, \ldots, c_k\}$ of classes.

There are a wide variety of application domains in text categorization, including (i) automatic keyword extraction from individual documents[3], (ii) document organization and retrieval, (iii) Email classification and spam filtering[2], and (iv) web data mining[15].

Along with pattern recognition and machine learning techniques widely studied, many machine learning algorithms and models can be used for text categorization. Some recent works on text categorization show that, several key classifiers are commonly adopted. They include:

(1) Naïve Bayes Classifier (NBC), a probabilistic classifier based on modeling the underlying text features in different known categories. It has an assumption that every text feature is independently presented on the web page[23].

(2) Support Vector Machine (SVM) Classifier, a quadratic approximation problem to solve optimal boundaries between the different categories. It has been considered the most promising algorithm in text categorization[3].

(3) Rule Based Classifier (RBC), a set of rules, in which the left-hand side corresponds to a word pattern, and the right-hand side corresponds to a category. The key part in RBC is to determine the word patterns which are most likely to be related to the different categories[2].

(4) Hidden Markov Model (HMM), a simple case of dynamic Bayesian network, where the

hidden states are forming a chain, and only some possible values for each state can be observed. In text categorization, HMM is to infer the hidden states according to the observed text and their dependency relationships[25].

Image Based Categorization

Instead of taking a set of keywords or phrases as features for web site categorization, images shown on the web site have their own feature set, such as color, texture and scale-invariant feature transform descriptor[16]. In the academic research, we can apply image/picture recognition techniques to classify web sites for certain purpose. However, in reality, compared with text categorization, it takes more resources to retrieve the information and process the classification. For example, probably the most useful case is to recognize pornography web sites by analyzing certain number of pictures downloaded from the web sites. We conducted a very simple experiment to recognize only one pornography web site by using text classification and image classification. Both approaches show the positive result. However, the execution time for image classification takes three times[1] longer than that of text classification. The execution includes retrieving pictures and skin-detecting for pictures[29].

Since many researchers and organizations are working on image recognition and image indexing, we believe the efficient machine learning algorithms are emerging to enhance web content categorization. In fact, some web search engine vendors start to provide image search service[11].

Web Content Categorization Summary

Since this thesis study is mainly for recognizing domain parking web sites, which have less images in the web page than that of other web sites, we will not consider image based categorization techniques.

1.1.2 Domain Parking Web Sites

There are two prerequisites to launch a web site service on the Internet: (i) register a `domain name`, and (ii) a web hosting account for `name servers` mapping to the `domain name`. One class of web sites are known as domain parking web sites defined below:

[1]The ratio will be different in terms of the size of pictures.

Definition 1. ***Domain Parking Web Sites:*** *A domain parking web site is a web site created for the purpose to own a domain but not yet to provide web service. The domain name will usually resolve to a single web page containing advertising listings and links[28].*

Majority of domain parking web sites are monetized, meaning that advertisements are shown to visitors, while the registrant gains revenue by clicking on those links. Domain parking web sites are easily recognized by human being. Figure 1.2 shows some examples of domain parking web sites.

As we can see, they are always single-page web sites with no more than 10 advertisement links. Because of monetization, domain parking service providers (DPSPs) run the business to allow people "park" their domains, generate profits from advertisement, or even trade attractive domain names. Based on the Google returned result ranking on searching "domain parking web sites service provider" on March 2013, some famous DPSPs include Sedo, Bodis and Godaddy. All web sites that belong to one DPSPs may have same web page layout and styles. Only a few of domain parking web sites are non-monetized. An "Under Construction" or a "Coming Soon" message may be put in the domain web site by the registrar. Recently, the existence of domain parking web sites on the Internet is more easily observed. The domain parking web sites may have negative effects on the Internet, for instance: (i) it is not joyful experience for the Internet users to come across domain parking web sites either accidentally or on purpose; and (ii) it is treated as a spam web site when a search engine performs the web site indexing.

Web Categorization for Domain Parking Web Sites

Can we categorize domain parking web sites by text based web categorization techniques? In order to have an answer, we first define a generic web content categorization engine. Generic Web Content Categorization Engine (GWCCE) is an engine to actively learn features (text) from labeled web sites and eventually evaluate and classify an unlabeled web site, then decide the category for this web site. If we claim the answer is yes, then we could apply GWCCE to categorize all monthly-new-added domain web sites. However, one of properties of domain parking web sites is monetization. It provides advertisement link with anchor text. The anchor text is only for attracting end users to click on it, which may not represent the category of the web site.

In order to see if GWCCE can be applied to domain parking web sites, we will use the

(a) Example 1

(b) Example 2

Figure 1.2: Domain parking web sites examples

model of communication along a noisy channel. We view the channel to be the GWCCE. The transmitted signal is web page text content. The received signal is statistical result of the web page category by giving the text content. The channel, which is the GWCCE engine, can maximum the mutual information between the transmitter and receiver.

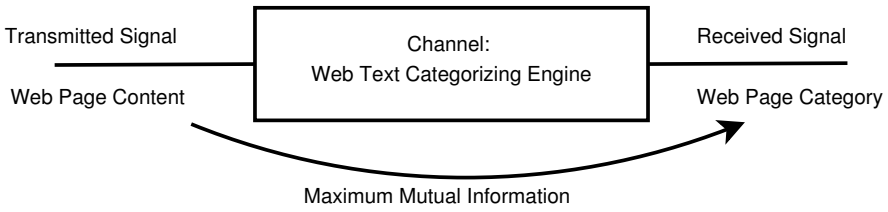


Figure 1.3: Communication along a noisy channel

We assume this model can achieve maximum mutual information for domain parking web sites as well. If a domain parking web site has advertisement links about shopping, then the web page category is defined as shopping. If a domain parking web site has advertisement links about education, then the web page category is defined as education. However, the information on transmitter side is domain parking. The channel doesn't maximize the mutual information, which contradict with our assumption. We use false negative rate to evaluate missing information in the channel. False negative rate is the fraction of missing information divided by original information from sender. Therefore, we can safely draw a conclusion that, GWCCE will generate higher false negative rate for identifying domain parking web site, consequently, GWCCE conveyed less mutual information for domain parking web sites than that of other web sites.

1.2 Domain Parking Recognizer Problem Statement

In order to find valuable information from new domain web sites, it is bandwidth and hardware resource consuming to do web site data mining, especially web content categorization. However, we observed that, certain percentage of new registered domain web sites are just placeholders, either under construction or providing advertisement links by domain parking service providers. Based on the analysis we illustrated in previous section, when we try to categorize all domain web sites by GWCCE, we may need to recognize those domain parking web sites and eliminate them at the first hand. Therefore, we propose Domain Parking Recognizer (DPR) as a machine learning engine to recognize those web sites.

Identifying domain parking web sites is a two-class prediction problem. In this problem, given a set of web sites represented by a collection of feature sets $DP = \{W_1, W_2, \ldots, W_n\}$, where W_i is the feature set of the i^{th} web site, our task is to find an algorithm which, given a feature set W_t, identifies whether the web site represented by W_t is a domain parking web site or not.

The feature set W selection is a critical part for DPR. We will introduce a case study in Chapter 2 using text features to directly identify web site categories. However, web text features may not work well for DPR. The reason is that, they may not reflect the nature of domain parking web sites. If we only use text features from the web site, DPR may fail to recognize domain parking web sites. The machine learning classifier selection is another critical part for DPR. We will introduce the evolution of DPR in Chapter 3 that, directly text-based classifiers may not be an appropriate choice. The reasons include: (i) the features selected may be correlated, and (ii) some features among the feature set may have more "weight" than the others. This thesis concentrates on these two critical parts for developing efficient DPR.

1.3 Contribution

The goal of our research is to develop a machine learning algorithm that accurately distinguishes domain parking web sites from other web sites. In this thesis, we propose a DPR, a new engine to recognize domain parking web sites. To our best knowledge, this is a first work concentrating on domain parking web sites problem.

For feature set selection, we introduce some features based on our observations. For example, we observed that, authority name server records resolved by Domain Name System query always appear in several certain groups, which belong to name servers of DPSPs. We also noticed that, for domain parking web sites, JavaScript source attribute in retrieved HTML file always link to hosts of DPSPs, or hosts of advertisement providers. Additionally, some phrase "domain for sale" is displayed on domain parking web sites. Therefore, in order to construct a feature set for domain parking web site, we claim features as follows: (i) Name Server Match, (ii) Text Content Match, (iii) Tracking Cookies, (iv) Internal Links, and (v) JavaScript Source. Due to relations among those features, we re-defined a feature set by adding five association rules as new features. Eventually, we adopt only three features: (i) Name Server Match, (ii) Text Content Match, and (iii) Association Rules Match, which are

more suitable for the classifier we proposed.

For learning model selection, we first selected Naïve Bayes Classifier because of its popularity. Due to a limited training data set, the accuracy of experiment results were not as good as we expected. The detailed analysis is shown in Chapter 3. We then tried to find the relations among those features. We generated association rules by MapReduce based Apriori algorithm. We found some association rules have higher confidence values. The Rules and their confidence values will be illustrated in Chapter 3. We replaced some features by using five new generated rules, plus Name Server Match and Text Content Match two features. There are total seven features: (i) Name Server Match, (ii) Text Content Match, (iii) Rule 1, (iv) Rule 2, (v) Rule 3, (vi) Rule 4, and (vii) Rule 5. Based on new defined features, we propose a DPR called Feature Confidence Index (FCI) classifier.

Based on the new defined feature set, we compared Support Vector Machine (SVM) classifier with the FCI classifier. The experimental results shows that, even though SVM classifier has a good performance for text features, the average F_1 score (It is a commonly used criterion for evaluating classifiers with a higher percentage for a better result. A formal definition is shown on Chapter 2) for domain parking web sites was around 77%, which still has a space to improve. The proposed FCI classifier has a better performance. Its average F_1 score can achieve to 91%. DPR is more resource efficient than content categorization.

The results show that the proposed FCI classifier is an efficient DPR for recognizing domain parking web sites.

1.4 Thesis Structure

The rest of this thesis is structured as follows. In Chapter 2, we introduce a web content categorization case study, which contains background knowledge and previous research on text-based web site categorization. This case study provides a base for studying the domain parking web sites problem. We formulate our problem and propose our approach for solving it in Chapter 3. Experimental results of our approach are provided and analyzed in Chapter 4. Finally, Chapter 5 contains the conclusion of this thesis.

Chapter 2

Preliminaries

The Domain Parking Recognizer (DPR) proposed in this thesis builds on Generic Web Content Categorization (GWCC). In both DPR and GWCC, one of key issues is the data source selection. In this chapter, we first introduce the data source for DPR. We choose the zone files as the data source for DPR. Then we briefly review the major techniques used in GWCC and give a case study as an example to explain GWCC so that we have a base for explaining DPR later.

2.1 Data Source

Data source selection is a key factor for web content categorization. For example, if we want to distinguish web sites between sports news and other news, we need to retrieve data from the news web sites such as CNN or BBC. Like wise, because majority of domain parking web sites can be easily retrieved from zone files, if we want to identify domain parking web sites, we need to narrow down data scope to focus on zone files. For better understanding of zone files, we first briefly introduce Domain Name System.

2.1.1 Domain Name System

The structure of Domain Name System (DNS) is a distributed database, which is indexed by domain names. For example, Simon Fraser University (SFU) computer science host name, `www.cs.sfu.ca`, has a country code top-level domain (ccTLD) "ca", the first-level domain named "sfu" and the second-level domain name "cs". In terms of DNS documentation, "cs"

is also a sub-domain of domain "sfu". The terms "domain" and "sub-domain" are often used interchangeably.

In order to decentralize administration for Domain Name System, an organization that administrates domains can divide them into smaller, more manageable units by delegation to other organizations. These units are called zones. A zone contains all Resource Records (RR) for the domain with the same domain name contains, except for domain names in delegated sub-domains. For instance, the top-level domain "ca" has sub-domains "sfu" and "uvic". Authority for the `sfu.ca` and `uvic.ca` may be delegated to name servers in each University. The zone `sfu.ca` and `uvic.ca` are separated zones from "ca" zone. Figure 2.1 shows the relationship of Domains and Zones.

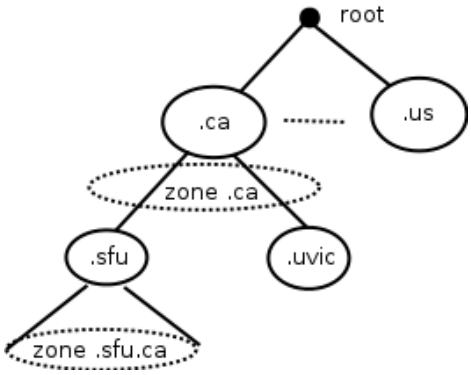


Figure 2.1: Domains and zones example

2.1.2 Zone and Zone Files

The term zone is coexisted with domain in Domain Name System. The difference between zone and domain is subtle. However, they have different perspectives. Domain namespace is the way to define unique host name on the Internet, while zone is the way to manage domain data in name servers.

Zone data file, as known as zone file, is a text file that describes a DNS zone. The programs that store information about the domain namespace are called name servers. There are two types of name servers: primary masters and secondary masters. The primary master name servers load their zone data files. The secondary master name servers for a zone get the zone data from another name servers authoritative for the zone. Zone data file contains a sequence entry of resource records. Every entry is a text description delimited

by spaces or tabulations[18].

In this thesis study, the data source is coming from six major top-level zone files: (i) *.com* zone file, (ii) *.org* zone file, (iii) *.info* zone file, (iv) *.biz* zone file, (v) *.net* zone file, and (vi) *.us* zone file. Every month, there are more than three millions new registered domains among six major top-level domains. According to one sampling statistics on December 2012, there are around 30%∼40% new registered domain web sites belong to domain parking web sites. Figure 2.2 shows the statistics of new incremental domains in these zone files.

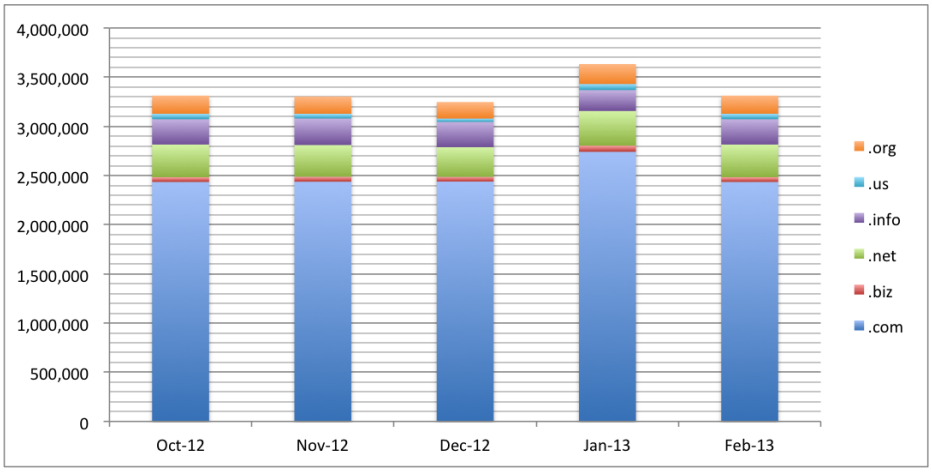


Figure 2.2: Total number of monthly new registered domains

2.2 Generic Web Content Categorization Case Study

GWCC is a base of this thesis study. Before we propose our DPR, we will demonstrate a case study of GWCC that cover all aspects, from fetching web text content, learning labeled web sites, all the way up to finally predicting the web site category. On the first part, we show two models of web content data information retrieval. On the second part, we briefly introduce two popular classifiers for text categorization. We also introduce MapReduce, first parallel computing model adopted in large scale computer clusters, and NoSQL database, alternative data storage compared with relational database. On the last part, we show the experimental results for this case study.

2.2.1 Case Study Problem Statement

The case study of GWCC is to classify web sites based on their contents to two categories (two-class prediction) by a machine learning model. Given web site categories $C = \{c_s, c_{ns}\}$, where c_s stands for sports news and c_{ns} stands for non-sports news, design a generic web content categorization engine, based on a certain number of sports news and non-sports news web sites, the engine is able to recognize sports news web sites by given any web sites.

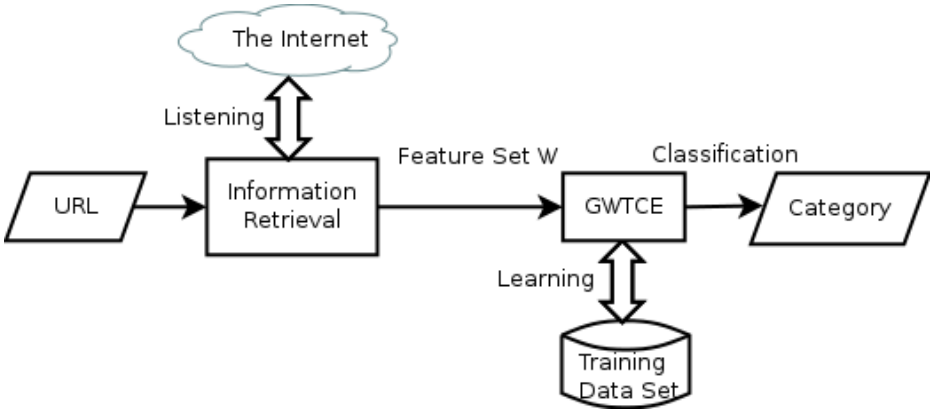


Figure 2.3: A generic web content categorization engine workflow

We denote by W a feature set of training data and by W_t a feature set of a web site to be classified. The Figure 2.3 shows a very high-level workflow for how the engine works. The input is one valid URL (Uniform Resource Locator), which can be used to visit the web site on the Internet. The workflow goes to listening phase first. This phase is to fetch information from web sites, retrieve useful data by using defined retrieval models, and generate a feature set W. The workflow then goes to learning phase. A learning algorithm F_{learn}, taking a training data set W as input, generate a classifier model M, as illustrated in function (2.1). For a given web site, the engine will start from listening phase, retrieving web information and constructing the feature set W_t. A prediction algorithm $F_{predict}$, taking a feature set W_t and classifier model M as input, predicts the category of the web site, as illustrated in function (2.2).

$$F_{learn} : W \Rightarrow M \tag{2.1}$$

$$F_{predict} : (M, W_t) \Rightarrow c, while\ c \in C \tag{2.2}$$

2.2.2 Listening: Web Content Information Retrieval

Web data contain rich information, including web content data, web link data and web traffic data. Here, we only consider to category web sites by using web content data. Web content data are majority of text shown on the web page. They also contain other information such as meta keywords, hyperlinks and JavaScript in HTML. In this case study, we are going to use each word of text as one feature.

Text Preprocessing

Before we pass all text data to information retrieval models, we need to preprocess text. For web text documents, the tasks are: (i) stop word removal, (ii) stemming, (iii) handling of digits, hyphens, punctuation, or cases of letters, and (iv) HTML tags removal.

First, we need to remove all stop words. Stop words are defined as insignificant words in a language, such as a, the and of. Those stop words convey less information than other words do. Secondly, we need to stemming all words. For example, *computer*, *computing* and *compute* can be reduced to *comput*. Thirdly, we need to handle punctuation and make all letters as lower case. For example, the text on the web page shows "Yankees beat Diamondbacks 4-2", we need to remove "4-2" because it contains digits and hyphen. Also, we convert "Yankees" to "yankees" and "Diamondbacks" to "diamondbacks". Lastly, by using HTML parser or other tools, all HTML tags shouldn't be counted as web text content.

For text processing, we import a NLTK (Natural Language Tool Kits) package. We use the following three modules:

(1) Word tokenize module. The module will divide a set of keywords or phrases into words by treating any sequence of white space characters as a separator.

(2) Frequent distribution module. The module will record the number of times each word occurred in a web site.

(3) Porter stemmer module. The module follows the Porter stemming algorithm[21].

Text Content Indexing

Information retrieval is different with data retrieval from relational database by SQL query. All data in database are structured and stored in relational tables. Text contents in web sites are sparse, unstructured and even independent data. In order to find the information

from web content, we need to have a method to organize and store retrieved web content data. The one of methods is to index text content.

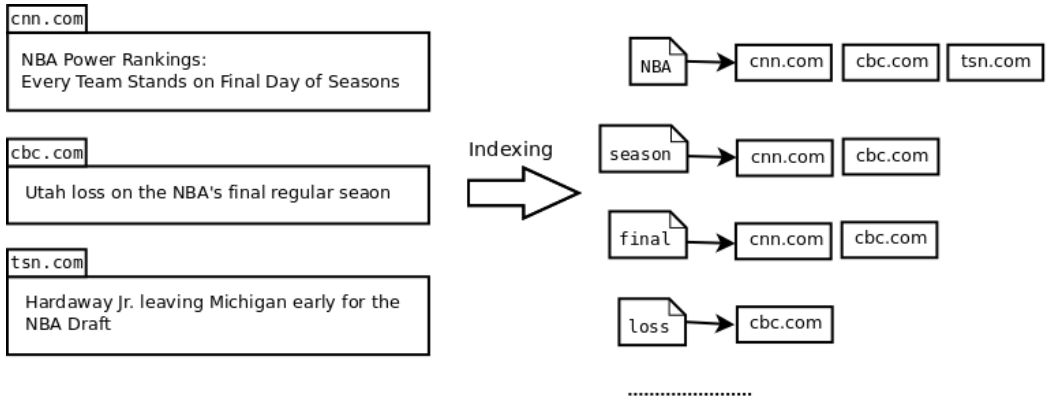


Figure 2.4: A simple example illustrates web content indexing

Figure 2.4 illustrates a basic idea how to index web text content. We take a snippet of text shown on each web site, including `cnn.com`, `cbc.com` and `tsn.com`. After indexing, each word becomes the index of those web sites. As we can see, the word "NBA" associates with three web sites while the word "loss" only associates with one web site.

Vector Space Model and Statistical Model

Information retrieval model is the representation of relevant data retrieved from web sites. The representation is normally adopted by corresponding machine learning models. There are varieties of information retrieval models, but we only adopt two of them in this study: (i) vector space model, and (ii) statistical language model[15].

For vector space model, one web page is represented as a weighted vector. We take each word on the web page as one term. Each dimension corresponds to a separated term. There is a weight assigned to each term, which is computed based on TF-IDF scheme. Term Frequency (TF) and Inverse Document Frequency (IDF) are numerical statistics which reflect how important a term is to a web page in a collection of web sites. The web page vector space is defined as:

$$W_j = \{w_{1j}, w_{2j}, \ldots, w_{nj}\}$$

In vector W_j, each dimension corresponds to a separated word shown on the web page. If a word occurs in the web page, its value w in the vector is non-zero. We are going to use

TF-IDF weighting scheme to compute this value.

First, we calculate TF for each term. Let n be the size of the vector. Let f_{ij} be the raw frequency count of term w_{ij} in document W_j. Then, the normalized term frequency tf_{ij} of w_{ij} in W_j is given by

$$tf_{ij} = \frac{f_{ij}}{max(f_{1j}, f_{2j}, \ldots, f_{nj})}$$

Then, we calculate IDF for each term. Let N be the total number of web sites in the data set and df_i be the number of web sites in which term w_i appears at least once. The inverse document frequency idf_i of term w_i is given by:

$$idf_i = log\frac{N}{df_i}$$

Finally, we calculate term weight TF-IDF for term w_{ij} in web site W_j.

$$w_{ij} = tf_{ij} \times idf_{ij}$$

For statistical language model, it is based on probability and statistical theory - Bayesian Theorem. The goal of this model is, based on words and categories of labeled web sites (training data), to predict conditional probability $Pr(c_j|W_t)$ of category c_j by given a web site W_t. By using labeled web sites $\{W_1, W_2, \ldots, W_n\}$, we have prior probability $Pr(c_j)$ of c_j. We can calculate the likelihood $Pr(W_t|c_j)$ of web site W_t by given its category c_j. We also have probability $Pr(W_t)$ of web site W_t. According to Bayes rule, we have:

$$Pr(c_j|W_t) = \frac{Pr(W_t|c_j) \times Pr(c_j)}{Pr(W_t)}$$

Vector space model is applied to Support Vector Machine classifier and statistical language model is applied to Naïve Bayes classifier. We have an introduction for these two classifiers in the following section.

2.2.3 Learning: Web Content Learning Models

Based on two information retrieval models, we are going to select machine learning models. Since our case study is a classification problem, there are many machine learning models that can be adopted, such as: (i) Tree Induction, (ii) K-Nearest Neighbor (K-NN), (iii) Naïve Bayes, (iv) Rule Induction, (v) Logistic Regression, (vi) Support Vector Machine, and (vii) Linear Discriminant[4].

In this case study and thesis study, we have a relatively small size of training data set. They have a property of high bias and low variance. K-NN classifier has an advantage to process low bias and high variance data, otherwise, overfitting problem occurs. Tree induction, such as decision trees classifier, does not support online learning. It has to rebuild tree structure if new training data are added. It is also prone to be overfitting for a small training data set. Logistic regression classifier is a discriminative model, which needs more training data compared with Naïve Bayes[4].

Naïve Bayes classifier is a simple model but delivers relatively good results in practice. Support Vector Machine has been shown to be efficient and effective for text-based classification. In this case study and thesis study, we select Naïve Bayes and Support Vector Machine two classifiers.

Naïve Bayes Classifier

Naïve Bayes Classifier (NBC) is a learning and classification method based on probability theory. Given a web site $W_t = \{w_1, w_2, \ldots, w_n\}$, NBC uses three probabilities to compute posterior probability $Pr(c|W_t)$:

(1) the prior probability $Pr(c)$ that is the fraction of the web sites in training data set belonging to a category c.

(2) the conditional probability $Pr(W_t|c)$ (likelihood) that is the product of probabilities of each feature w_i shown in the web site W_t by given the category c.

(3) the probability $Pr(W_t)$ (evidence) that is the product of probabilities of each feature w_i shown in the web site W_t.

By Bayes' theorem and the assumption that each feature w_i is conditionally independent of every other feature w_j for $i \neq j$,

$$Pr(c|W_t) = \frac{Pr(c) \times Pr(W_t|c)}{Pr(W_t)} = \frac{Pr(c) \times \prod_{i=1}^{n} Pr(w_i|c)}{\prod_{i=1}^{n} Pr(w_i)}$$

where $Pr(w_i|c)$ is the conditional probability (likelihood) that given category c, a web site with feature w_i belongs to c and $Pr(w_i)$ is the probability (evidence) that feature w_i appears in the training data set.

For this case study, there are two categories c_s and c_{ns}. Given a web site with a feature set $W_t = \{w_1, w_2, \ldots, w_n\}$, the probability that the web site belongs to c_s is calculated by:

$$Pr(c_s|W_t) = \frac{\prod_{i=1}^{n} Pr(w_i|c_s) \times Pr(c_s)}{\prod_{i=1}^{n} Pr(w_i)}$$

and the probability that the web site belongs to c_{ns} is calculated by:

$$Pr(c_{ns}|W_t) = \frac{\prod_{i=1}^{n} Pr(w_i|c_{ns}) \times Pr(c_{ns})}{\prod_{i=1}^{n} Pr(w_i)}$$

The web site is classified to the category c given by:

$$c = \arg \max_{c \in \{c_s, c_{ns}\}} Pr(c|W_t)$$

Due to labor intensive to collect training web data, we only take small portions of sport news from hundreds of web sites. We observed that, the more training data involved, the more accurate result achieved. However, the unbiased learning of NBC is impractical. For example, if W_t is a web site containing 30 words, each word as one feature, then we will need to estimate more than 3 billion web sites to be a training set[17].

Support Vector Machine

Support Vector Machine (SVM) is a non-probabilistic machine learning binary classifier. Given a set of training data (W_i, y_i), $i = 1, ..., m$, where $W_i = \{w_1, w_2, \ldots, w_n\}$ is a set of features and $y_i \in \{-1, 1\}$ indicating the category that W_i belongs to. SVM finds a hyperplane that separates the W_i with $y_i = 1$ from those with $y_i = -1$. This hyperplane is then used to predict the category of a new web site.

In order to explain SVM in a formal way, we use x to represent W. Also, we take w as a normal vector. Then, the hyperplane can be calculated by $y(x) = w^T\phi(x) + b$, where $\phi(x)$ is a vector obtained from a set of features, w^T is a normal vector to $\phi(x)$ and b is a constant vector. When the training data are linearly separable, there are many solutions (hyperplanes). To provide a best separation of two categories, SVM finds the hyperplane that has the largest distance to the nearest data point to two classes. This hyperplane can be found by solving the following optimization problem:

$$\min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i$$

$$\text{subject to: } y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i$$

The training vectors x are mapped into a high-dimensional space via some transformation $\Phi : x \rightarrow \phi(x)$. Parameter C can be viewed as a way to control overfitting. It trades off the relative importance of maximizing the margin and fitting the training data. ξ is called slack variables. If $0 < \xi_i < 1$, the classification result is still inside margin which is still correctly classified. If $\xi_i > 1$, the classification result falls in another side of decision boundary, which means mis-classified.

Figure 2.5 gives an example on training data with two features $x_i = \{x_1, x_2\}$. Two classes c_s and c_{ns} could be classified by multiple $y(x)$. The signed distance to decision boundary $y(x) = 0$ is $\frac{y(x)}{||w||}$. The points with this minimum value are known as support vectors. There are three support vectors shown in the diagram. Therefore, SVM is the optimization problem which choose decision boundary by maximum margin, since in general, the larger the margin the lower the generalization error of the classifier.

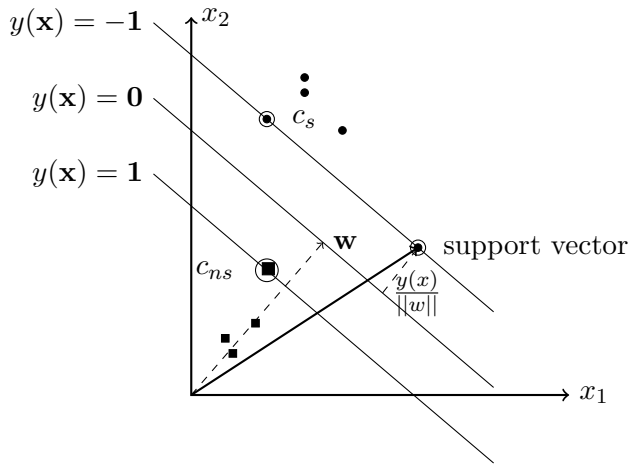


Figure 2.5: Support Vector Machine marginal geometry.

2.2.4 Classification: Experiments and Results of Case Study

In this section, we introduce data set selection for this case study. We also introduce two programming paradigms, MapReduce and NoSQL, which are considered in both this case study and the Domain Parking Recognizer developed in this thesis. Case study final result is shown at last.

Data Set Selection

We use Alexa top tennis web sites as part of training set, labeled as *sports*. We also use Alexa top business web sites as another part of training set, labeled as *non-sports*. In order to avoid bias when calculating category probability, each part has 1000 web sites.

We conduct experiments during 50 days. For each day, we choose 200 Alexa top sports and 200 Alexa top business to do the test. The assumption is, the web content of Alex top web sites should be updated every day.

Experiments Environment

We use one standalone computer to include all modules in this case study, including fetching the web sites, database and categorization engine. The computer is running Mac OS Mountain Lion. The CPU is Intel(R) Core(TM)2 Duo CPU T8300 at 2.40GHz. The computer has 6G memory space. All implementation are done by Python. We design and implement a Naïve Bayes classifier tool. For SVM classifier, we import third party library SVM^{light}. This tool is designed and maintained by Thorsten Joachims in Cornell University[14]. It is an implementation of Support Vector Machines in C, which can handle more large scale data.

MapReduce

Although this case study could be done by one personal computer, due to a large scale data on the Internet and variety of knowledge discovering behind those data, we need to plan and adopt a new technology to handle a large quantity of data. MapReduce is a new computation paradigm which is well-adopted in web data mining.

Before we dig into MapReduce paradigm, let us review sequential paradigm, which is traditional programming paradigm dominated for decades. For example, given a list of words with size n, we want to get a count of each distinguish word happened in the list. We have a hash map function that, iterates each word in the list, then puts the word in the hash map and increases count by 1.

$$F_{hashmap} : ([w_0, w_1, \ldots, w_n]) \rightarrow (HashMap\{w_{distinguish}, count\})$$

When the number n is small so that the word list can be contained in the memory of a computer, the hash map function is an appropriate choice for the programming. However, if

n is large so that the word list can not be held in the memory of a computer, then we have to chunk the list to many parts and to execute the hash function for each part. Assume that $F_{hashmap}$ can take at most l words once from the word list, due to the hardware limitation and takes $O(l)$ time to process the input words. Then a word list of n words is partitioned to $m = \frac{n}{l}$ parts and the total running time of $F_{hashmap}$ on a single computer is $O(n) = O(ml)$, which could be time consuming when n is large.

The intuitive solution for the problem above is: we could set up m computers as m workers. We have a master computer that is running main program. It divides the word list to m parts, and each part has l words. After assigning each part to each one of workers, those workers can process l words parallel. Thus, the running time of $F_{hashmap}$ can be improved to $O(l)$ by parallel computing with m computers. MapReduce paradigm builds on this purpose.

MapReduce is parallel paradigm with message passing. It was inspired by the functional language such as LISP. It is a programming model for expressing distributed computations on massive amounts of data and an execution framework for large-scale data processing on clusters of commodity servers. The idea is to break one object, which could be a list of words, a document or even a data set, into many small simple manageable key-value objects, then to process those objects parallel.

MapReduce is also pipelined procedure where there are (i) map phase and (ii) reduce phase corresponding to two functions: F_{map} and F_{reduce}. Let us take word counting as an example again. Map function takes each document name W_k and document contents $[w_0, w_1, \ldots, w_n]$ as input, then maps them to word and count key-value objects (w_i, c_i). c_i is always 1 because one word is only counted once. Reduce function collects those word and counts key-value object $(w_i, [c_i, c_j, c_k, \ldots])$, then sums up the counts based on the same word.

$$F_{map} : (W_k, [w_0, w_1, \ldots, w_n]) \rightarrow [(w_i, c_i)]$$
$$F_{reduce} : (w_i, [c_i, c_j, c_k, \ldots]) \rightarrow (w_i, \sum_i c_i)$$

From MapReduce architecture point of view, there are several mappers and reducers which are running F_{map} and F_{reduce} respectively on each commodity computer server. It also could be running on the same computer server. For this case study, we calculated word counts by MapReduce, which is only running on one computer.

From implementation point of view, MapReduce is a higher level abstraction so that users (developers) only need to specify mapper function and reduce function. All jobs assignment, fault-tolerance, data distribution and message passing are handled by the hiding library. Many complex web data mining algorithms for machine learning and processing big data are easy to express using the MapReduce paradigm, which make large scale parallel computing much easier than implementing on shared memory or low level message passing.

Data Storage Selection

When we store all retrieved web content data to database, we first select relational database system, such as MySQL. The relational database has been developed around 30 to 40 years. It was applied to many areas, especially bank service, due to its well designed transaction paradigm.

In order to get words count from a set of retrieved web sites, as well as web sites associated with those words and categories, we create two tables: (i) web table, and (ii) content table. The web table contains an ID as an index, web site name (URL) and its category. The content table contains an ID associated with the web site in web table and the word. Figure 2.6 shows the relationship between these two tables. In order to get all word counts for sports category, we use the following SQL query:

```
> SELECT word, count(word) FROM web, content WHERE web.id=content.id AND category='sports' GROUP
  BY word
```

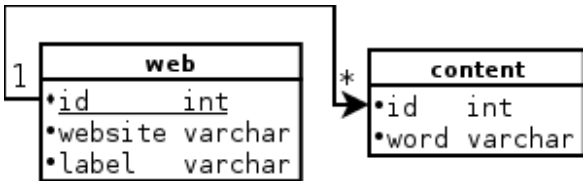


Figure 2.6: Web content data storage by relational database.

However, when we prepare training data set, we retrieved 2000 web sites. Each web site contains more than 100 words. The content table quickly grows to more than 200,000 records. Although the scale of data are still small since we only select 2000 web sites, we probably need more hard drive space and fast processor if the number of web sites reaches to 1 million. Along with the emerging of web sites searching technology development, there is a new data storage technology - NoSQL, which is flexible and scalable for Big Data.

Big data is defined not only because of large size of data set but also because of flexible data structure and demanding of high performance computing platform. "Big data are high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization"[8]. There are three properties of big data:

(1) The size of data set is primary consideration. Big data size is continuously growing, especially on the Internet domain. As of 2012, big data size is ranging from a few dozen terabytes to many petabytes of data in a single data set[6].

(2) The structure of data set is flexible. Compared with tightly coupled relational database, big data have unstructured but complex interrelationship data[1].

(3) The capability to process and analyze is demanded. In order to handle big data, high performance computing (HPC) platform became major role in data center nowadays[13].

NoSQL stands for "not only SQL". Although it has no special advantages over the relational database model, it does address certain limitations for current relational database. NoSQL is schema free and horizontally scalable. It gives us an alternative way to store web content data. For example, we do not need to consider schema normalization which is very important in relational database. We also do not need to joint multiple tables, because we can put all relative information in one collection. There are several reasons to adopt a NoSQL database system.

(1) Minimize table join in terms of relational database. For web data mining, There are a lot of features have many-to-many properties. By using relational database, multiple table joints are unavoidable.

(2) Decentralize data to the document-oriented database system.

(3) Parallel computing by using commodity computers.

In this case study, also for DPR, we decide to use MongoDB as a database system. MongoDB (Mongo is from humongous) is an open-source document database that is written in C++. It provides document-oriented storage. In terms of MongoDB, the training data set is one collection. Each web site and its properties, such as text, are stored as one document in the collection. Let us transfer data schema from the example above to MongoDB. We

only have one collection (table in relational database) that contains all web content data. Each document of the collection is similar to one record of one table in relational database. Here, each document is each web site. We can put text of web site in the same document with web site name and its category. Figure 2.7 shows the structure of one document in MongoDB.

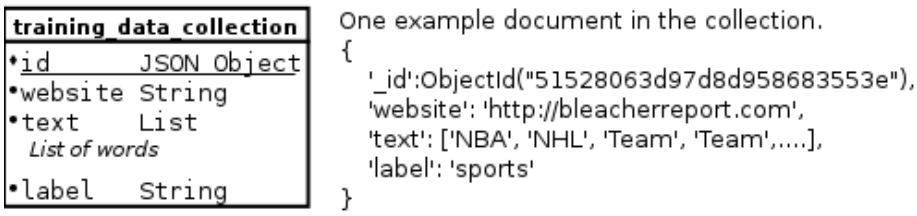


Figure 2.7: Web content data storage by MongoDB

MongoDB also provides MapReduce operations for many simple aggregation task. Thus, we can easily get word count among a collection of documents. The following JavaScript code snippet shows the map function executed in MongoDB. The variable text is a list of words. The map function basically iterates all words in text, then emits the word as a key and the count 1 as a value.

```
1 function () {
2   this.text.forEach(function(z) {
3     emit(z, 1);
4   });
5 }
```

The following JavaScript code snippet shows the reduce function running in MongoDB. For each word as a key and its list of counts as values, the function sums up counts and returns the final result.

```
1 function (key, values) {
2   var total = 0;
3   for (var i = 0; i < values.length; i++) {
4     total += values[i];
5   }
6   return total;
7 }
```

Results Evaluation

There are two metrics, precision and recall, that could be the measures of results evaluation. Precision is the fraction of retrieved instances that are relevant, while recall is the fraction of relevant instances that are retrieved[27]. Let tp be true positive that stands for the number of domain parking web sites that DPR correctly recognized. Let fp be false positive which stands for the number of non-domain-parking web sites that DPR wrongly recognized. Let fn be false negative which stands for the number of domain parking web sites that DPR missed. Then, we have precision definition:

$$precision = \frac{tp}{tp + fp}$$

and recall definition:

$$recall = \frac{tp}{tp + fn}$$

The goal of GWCC case study and Domain Parking Recognizer is to achieve higher recall value while keeping false positive, consequently precision value, in an acceptable level. Thus, we adopted F_1 score as a metric to evaluate accuracy results because F_1 score is harmonic mean of precision and recall.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

We conducted the experiments 50 times in 50 days. For each test, we calculated F_1 score for Naïve Bayes classifier and SVM classifier. As we can see form Figure 2.8, the best score $F_1 \approx 96\%$ was achieved by SVM classifier. We also noticed that, overall, SVM classifier has better performance than Naïve Bayes classifier, even though they almost have the same scores for some tests.

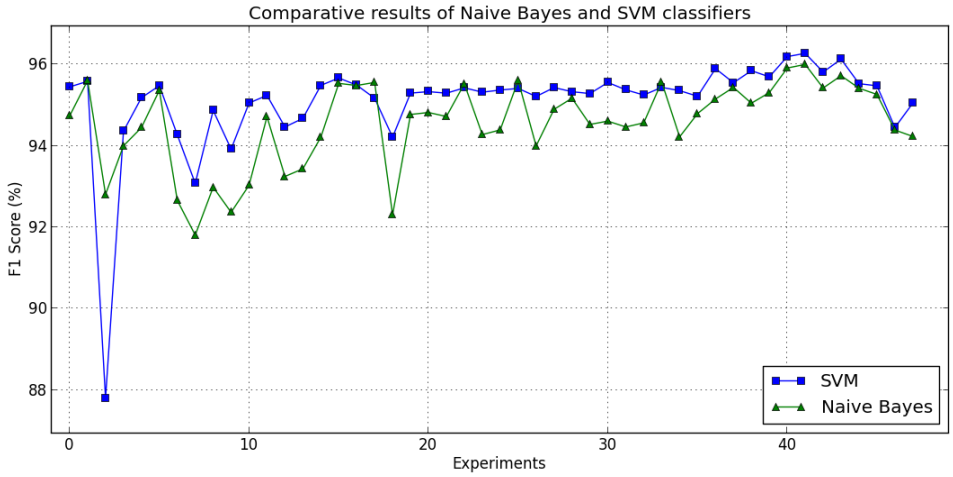


Figure 2.8: Comparative results in GWCC case study

Chapter 3

Domain Parking Recognizer

In this chapter, we will discuss the evolution of Domain Parking Recognizer (DPR). DPR is based on a supervised learning algorithm. First, we will present all possible features that we observed. Then, we choose Naïve Bayes as a classifier. We observe that, the limitation of the classifier may cause a lower F_1 score. Association rules are applied for finding the relations among those features. New features are defined by using association rules. We select SVM as a classifier for the new features. We notice that, some features may have more confidence in the classification than the others. We finally come up with the Feature Confidence Index (FCI) classifier for DPR.

3.1 Features for Identifying Domain Parking Web Sites

Feature selection is a critical task for DPR, since in general, right features lead a machine learning classifier to predict a correct categorization. Besides Text Content Match feature, we also select Name Server Match feature, JavaScript Source feature, Tracking Cookies feature and Internal Links feature.

3.1.1 Text Content Match Feature

We have demonstrated a case study in Chapter 2 that, text contents play a very important role for GWCC. A set of keywords or phrases in a web site will be features for web content categorization. We also showed in Chapter 1 that, domain parking web sites convey less or even wrong information in terms of text content shown on those web pages. However, a

part of text content shown on a domain parking web site still could be a feature for DPR.

Domain sales is a part of monetization for domain parking service. For example, `knockouts.com` was listed as price of 275,000 dollars in a DPSP `sedo.com`. From what we have viewed hundreds domain parking web sites, most of them contain the phrases such as "this domain for sale" or "inquire this domain". We claim that, if one web site contains such phrases, the web site is most likely domain parking web site. Therefore, these phrases shown on the domain parking web sites are still features that we can recognize them easily. Based on our observation, we define text content match features.

Definition 2. ***Text Content Match Feature:*** *The feature is presented if and only if one phrase of web text content matches the phrases defined in reference list (Appendix A Section A.3).*

In order to get each phrase, e.g. a plain text nested inside anchor tags, from a web page, we need a HTML parser to extract data from the web page. Due to various statements shown on domain parking web sites but only for one purpose - domain sales, we need to use collected phrases to match those statement. The following two sections show HTML parser and String Matching in details.

HTML Parser

HTML stands for Hyper Text Markup Language. It contains both tags and plain text. HTML has a nested structure, which can be modeled as a tree. Document Object Model (DOM) tree is commonly adopted.

Figure 3.1 shows a simple Apache HTTP server HTML default page. If we want to get phrase "It Works!", one round of processing by HTML parser is: (i) handle the start tag that is "h1"; (ii) extract data that is a plain text; and (iii) handle the end tag that is "h1".

String Matching

After we get a phrase from a web page, we want to check if the phrase matches what we collected in the list.

We first take known phrases in the list as regular expression patterns. Regular expressions are often implemented to represent a exactly matching phrase. For example, Let A be regular expression "domain for sale". The phrase on the web site B is "this domain for sale". Then A matches B. However, if the phrase C is "inquire this domain", the pattern

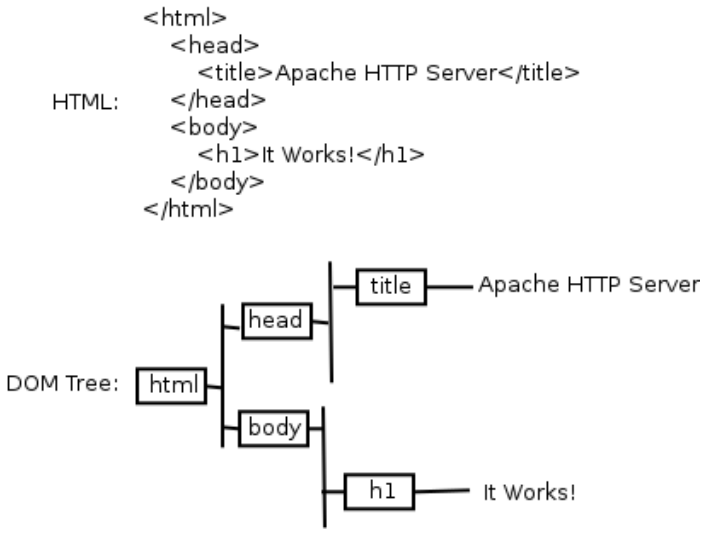


Figure 3.1: Illustration of the DOM tree built from a simple HTML page

A fails to match C. However, we can see that the meaning of phrase C is similar with that of pattern A.

We then adopt the most widely used string matching technique: string edit distance (also known as Levenshtein distance). We can treat one phrase as a string. The edit distance of two string s_1 and s_2 is defined as the minimum number of character changes so that s_1 can be transferred to s_2. The character changes include: (i) change a character, (ii) insert a character, and (iii) delete a character.

For example, the edit distance $d(s_1, s_2)$ between $s_1 = \{this\ domain\ for\ sale\}$ and $s_2 = \{domain\ for\ sale\}$ is 4. However, the edit distance $d(s_1, s_3) = 17$ while $s_3 = \{inquire\ this\ domain\}$. In order to make a decision that, if a phrase matches the phrase we collected, we need to normalize edit distance to a ratio. The edit distance ratio is defined as:

$$ratio(s_1, s_2) = \frac{d(s_1, s_2)}{(|s_1| + |s_2|)/2}$$

We define the edit distance ratio by calculating all combination of phrases in collected list. Let n be the size of collected list. s_i is the i^{th} phrase in the list. Then, we pick up the lowest ratio score as a threshold as:

$$ratio\ threshold = \arg \min_{i,j \in \{1,2,\ldots,n\}} ratio(s_i, s_j)$$

We have an algorithm to decide whether or not a given string is a matching.

Data: phrase, collected list, ratio threshold
Result: string matching is True or False
Randomly pick up $s_1 \in$ collected list;
$s_2 \leftarrow$ phrase;
if *ratio*(s_1, s_2) > *ratio threshold* **then**
    return True;
else
    return False;
end

Algorithm 1: Determining text content string matching

3.1.2 Name Server Match Feature

For domain parking web sites, they are normally hosted by domain parking service providers (DPSPs) due to unified web page style or advertisement link management. We observed that, for majority of domain parking web sites, their resolved Domain Name System (DNS) name servers are always coming from some DPSPs. It gives us an intuitive thinking that, if resolved name server of one web site belongs to one DPSP, then it has a higher probability to be domain parking web site.

In order to get a name server for a web site, we need to know how DNS query works. We also illustrate difference between NS Record and SOA Record. Finally we have a definition of Name Server Match feature.

Domain Name System Query

The Internet has two principal namespaces: (i) domain names, and (ii) IP systems. DNS is a network service for providing mapping between IP addresses and hosts that belong to domains. DNS name server is a computer server for providing response to queries against directory service. It plays an important role when DNS client, e.g. HTTP client, try to resolve IP address by given a web site URL.

There are two types of DNS queries: recursive and iterative. Recursive query is happened between DNS client and DNS server, which is normally configured in the DNS client. Iterative query is processed like “walking the DNS tree”. It is happened between DNS server

and all related DNS name servers. For example, to resolve `docs.google.com`, DNS server first queries DNS root name servers. Globally there are 13 DNS root name servers. The one of root name servers will return lower level name server, `google.com`, to DNS server. DNS server then queries `google.com` name server to get records of `docs.google.com`. Finally, DNS returns the IP address of `docs.google.com` to DNS client. The whole process of DNS query is finished.

NS Record vs. SOA Record

DNS name server stores the DNS records. There are many types of records, such as (i) IPv4 address record A, (ii) IPv6 address record AAAA, (iii) name server record NS, (iv) start of authority record SOA, (v) mail exchange record MX, and (vi) canonical name record CNAME.

We first select NS record as a feature for DPR. However, we notice that, DNS query for NS record may return empty result for some web sites. Let `docs.oracle.com` as an example. DNS iterative query first goes to root name server `a.root-servers.net.` and get top-level domain `.com.`, which name server is `a.gtld-servers.net.`. DNS server then gets domain `.oracle.com.`, which name server is `ns1.oracle.com.`. On name server `ns1.oracle.com`, there is no NS record for `docs.oracle.com`. It only maintains a CNAME record to point to `docs.oracle.com.edgesuite.net.` canonical name server.

Due to the reason above, we start to consider SOA record. It is the most essential part of a zone file. SOA record contains many attributes for a domain, such as: (i) updating frequency, (ii) timestamp of last update, (iii) time to check back for updating cache, (iv) administrator's Email address, and the most important, (v) authority name server. For the example above, when DNS queries name server `ns1.oracle.com`, the SOA record is returned with authority name server `n0d.akamai.net.`.

Therefore, SOA authority name server is the better feature choice for DPR. We have a definition as follows:

Definition 3. ***Name Server Match Feature:*** *The feature is presented if and only if the authority name server of the domain web site belongs to one of DPSPs.*

Sampling Learning

We collect 24 major DPSPs (Appendix A Section A.1). From zone files extraction, we have statistics that, there are totally 375,797 domains located on name server `ns1.bodis.com` and 2,562,742 domains located on name server `ns1.sedoparking.com`, as of December 2012. The question is, whether or not all domains in the name server are domain parking web sites? Or, if we learn from random samples, what is the probability approximate to correct learning. Here, we are going to use Probably Approximately Correct Learning, known as PAC-Learning[26].

We define a learning process A as, if the authority name server in DNS query falls in the name server list which belongs to DPSPs, then the web site is domain parking web site. We also define a concept class C, which is a set of concepts over $X = \{0,1\}^n$ instance space, while n is total number domains in one name server and 0/1 means true/false for domain parking web sites.

Let D be a given set of domains and $C \subseteq D$ be a subset of D such that each member of C is randomly independently selected from D without replacement based on the uniform distribution. Let p be the probability that a random sampling is not a domain parking web site. Let r be the size of c. Let b be a parameter to decide how approximately correct of the learning, while $0 < b < \frac{1}{2}$. A larger b means less approximately correct. Let $LE(p,r,t)$ be the probability of having at most t non-domain-parking web sites in r independent random samples. Then we have:

$$LE(p,r,t) \leq e^{-b^2 p \times r/2}$$
$$while\ t = (1-b) \times p \times r$$

The experiments show that, by using random samples in learning process A, we can achieve at most 95.12% approximately correct recognition rate with average probability $p = 20\%$ based on the PAC-learning theory.

3.1.3 JavaScript Source Feature

JavaScript is light weight program running on a HTTP client side, e.g. a web browser. JavaScript is able to realize dynamic function that static HTML can not make it. For example, JavaScript could easily add or delete web widget without get new HTML from HTTP

server side. Even more, JavaScript could construct a new HTML web page dynamically and direct web page visitor to this new web page.

JavaScript Redirection

Before we illustrate how JavaScript could be a feature of DPR, let us review the ways that can make URL redirection. URL redirection is a technique to redirect a web visitor to a specific URL. A web site owner can use this technique to redirect web visitors to a specific web site. There are three types of redirection: (i) HTTP header level redirection, (ii) HTML meta parameter redirection, and (iii) JavaScript level redirection.

(1) HTTP header level redirection is used by status code starting with 3 in header field. The redirected destination is defined in header as location field. For example, when a HTTP client receives HTTP header with 302, meaning moved permanently, the client will initialize a new HTTP session with the new URL defined in Location field.

(2) HTML meta parameter redirection uses HTML meta element. When *http-equiv* parameter is set the value to "Refresh" and the *content* parameter is given a time interval in seconds, the current web page will be redirected to a new web page that the *url* parameter is designated.

(3) JavaScript redirection uses *window.location.href* attribute. For example, if the attribute value is `http://www.new-website.com` in JavaScript, the web page is redirected to this new web site.

JavaScript Source

There are three ways to retrieve JavaScripts down to a HTTP client: (i) embedded with HTML, (ii) loading *.js* file from local HTTP server directory, and (iii) loading *.js* from third-party HTTP server directory.

Majority of domain parking web sites are created by DPSPs. They provide JavaScripts running on visitor's web browser to dynamically generate advertisement links. We observed that, for most of domain parking web sites, JavaScript source attributes do not point to local HTTP server directory. Rather, they point to their DPSP servers or third-party advertisement link providers, e.g. AdSense on `http://www.googlesyndication.com`.

Therefore, JavaScript source attribute is the feature that we could recognize domain parking web sites if JavaScript source is not from local HTTP server directory. Then we have:

Definition 4. ***JavaScript Source Feature:*** *The feature is presented if and only if the number of non-local JavaScript source is at least one.*

The following algorithm shows the way to determine the count of non-local JavaScript source.

```
Data: HTML source code, local hostname
Result: The count of non-local JavaScript source
HTML parser initialization;
source_cnt ← 0;
read first JavaScript tag;
while JavaScript tag attribute src exists do
    if src does not contain local host name then
        source_cnt ← 1 + source_cnt ;
    end
    go to next JavaScript tag;
end
```

Algorithm 2: Determine the count of non-local JavaScript source

3.1.4 Tracking Cookies Feature

HTTP cookies are used to store states by HTTP server, so that, stateless protocol HTTP can keep tracking stageful HTTP session.

Cookies Neccessariness for DPR

In order to see the neccessariness of HTTP cookies, we choose two university web sites: University of British Columbia (UBC) and University of Washington (UoW). We also select two domain parking web sites to get their cookie set. We notice that, cookie set from both UBC and UoW web sites are empty. However, for two domain parking web sites, the cookies sets are shown as below.

(1) [a44cf7c6fbe58ae1d47f4b8c258c64b7=fe925477c0340d53c112d9f882638372, path=/]

(2) [uid=hyperinflation50b45c55b57494.37888289, expires=Thu, 27-Dec-2012 06:23:17 GMT, path=/]

Intuitively, we observe that, not all web sites need to be set cookies for tracking user behavior but most of domain parking web sites do.

Tracking Cookies

Tracking cookie is the way to collect web visitors' surfing hobbies so that domain parking web sites can provide advertisement links purposely. For DPSPs, in order to track visiting behavior to maximum advertisement profit, they are using cookies to record each visitor's hobbies and favorites. DPSP defines a cookie by adding Set-Cookie field to HTTP header. When one visitor uses web browser to visit this web site, the cookie stores either in the local memory or in the local file. Next time when the visitor visits the same web site, cookie can provide personalization and tracking information.

We also notice that, the size of cookie set may not be the feature of identifying domain parking web sites because some web sites, such as searching, online shopping, use more complicated cookies than that of parking web sites. Additionally, for some user-interactive web sites, such as `www.google.com` and `www.facebook.com`, the size of cookie set is large and there are more cookie attributes. For temporarily build-up web sites such as domain parking web sites, the size of cookie set is comparative small and there are less cookie attributes.

Therefore, we consider tracking cookies feature as, whether or not tracking cookie set is available. Then we have:

Definition 5. ***Tracking Cookies Feature:*** *The feature is presented if and only if the cookie set is not empty.*

3.1.5 Internal Links Feature

Internal links are HTML hyper-links that go from one web page on a domain web site to a different web page on the same site. They are commonly used in main web page navigation. Internal links are useful for three reasons: (i) navigate a web site inside the same domain, (ii) establish information hierarchy for the web site, and (iii) help to increase ranking for the web site.

Determine Internal Links

The HTML hyper-links are defined as one attribute "href" of anchor tag "a". If the value of "href" attribute contains schema such as "http" or "https", then it is treated as an URL of web site. If the value is a file with or without path, then web server treats it as a local directory and a file. Therefore, we have an algorithm to determine internal links.

```
Data: HTML source code
Result: The number of internal links: inlinkcnt
HTML parser initialization;
inlinkcnt ← 0;
read first anchor;
while anchor attribute href exists do
    if href contains "http" or "https" then
        if href contains host name is the same web site then
            inlinkcnt ← inlinkcnt + 1;
        end
    else
        inlinkcnt ← inlinkcnt + 1;
    end
    go to next anchor;
end
```

Algorithm 3: Determine the count of internal links

Internal Links for Domain Parking Web Sites

For most of generic purpose web sites, they have multiple levels of web pages. Consequently, they have many internal links in the main web page. Visitors can navigate the web site by those links. However, as we demonstrated in Chapter 1, domain parking web site always has only one web page. All hyper-links shown on the page point to advertisement links to generate network traffic. There is a few domain parking web sites that provide internal links. Therefore, we have definition:

Definition 6. ***Internal Links Feature:*** *The feature is presented if and only if the number*

of internal link is zero.

3.2 Classification Model Selection

In order to select a suitable classification model for DPR, we first select the Naïve Bayes classifier due to its popularity on text-based classification. Since (i) the training data set is small, and (ii) some features are not presence independently, the F_1 score by NBC is lower than we expected. Then we try to find relations among those features by using association rules approach. We define five rules that have higher confidence. We then redefine features by using five new defined rules. We consider Support Vector Machine as a classifier for new defined features because of its reputation on text-based classification. We notice that, by using provided kernel function, such as Radial Basis function or Polynomial function, the F_1 score still can not meet our expectation. Based on our intuitive thinking, we finally proposed our own algorithm based on Feature Confidence Index.

3.2.1 Naïve Bayes

We have briefly demonstrated Naïve Bayes in Chapter 2. It is a probabilistic supervised learning model. The experiment result in the case study showed that, it is a suitable classifiers for GWCC. For DPR, we have five features to contribute to the model: (i) JavaScript Source, (ii) Tracking Cookies, (iii) Internal Links, (iv) Name Server Match, and (v) Text Content Match. We have an assumption that, all the features in a domain parking web site, such as Internal Links and Tracking Cookies, are all equally likely to occur independently with what is happened on other domain parking web sites.

Let $Pr(J)$ be the probability of JavaScript Source feature that is present among all training data set. Let $Pr(J|D)$ be the likelihood of JavaScript Source feature that is true among domain parking training data set. Let $Pr(D)$ be the prior probability of domain parking web sites in training data set. Then, given a new web site, if JavaScript Source feature is presence, the posterior probability $Pr(D|J)$ can be calculated by:

$$Pr(D|J) = \frac{Pr(J|D) \times Pr(D)}{Pr(J)} \tag{3.1}$$

Based on formula (3.1), we add one more feature. Let $Pr(T)$ be the probability of Tracking Cookies feature that appears by picking up one sample from the training data set. Let $Pr(T|D)$ be the likelihood of the feature that appears in a randomly selected sample from

domain parking training data set. Then, by given a new web site which has JavaScript Source feature and Tracking Cookies feature, the posterior probability $Pr(D|J,T)$ is:

$$Pr(D|J,T) = \frac{Pr(J|D) \times Pr(J|T) \times Pr(D)}{Pr(J) \times Pr(T)}$$

Accordingly, let (i) $Pr(I)$ be the probability of Internal Links feature and $Pr(I|D)$ be the likelihood; (ii) Let $Pr(N)$ be the probability of Name Server Match feature and $Pr(N|D)$ be the likelihood; (iii) Let $Pr(S)$ be the probability of Text Content Match feature and $Pr(S|D)$ be the likelihood. Given a new web site which has all five features, the posterior probability $Pr(D|J,T,I,N,S)$ is defined as:

$$Pr(D|J,T,I,N,S) = \frac{Pr(J|D) \times Pr(T|D) \times Pr(I|D) \times Pr(N|D) \times Pr(S|D) \times Pr(D)}{Pr(J) \times Pr(T) \times Pr(I) \times Pr(N) \times Pr(S)}$$

Let $Pr(G)$ be the prior probability of non-domain-parking web sites in training data set. Accordingly, we have $Pr(J|G)$ for JavaScript Source, $Pr(T|G)$ for Tracking Cookies, $Pr(I|G)$ for Internal Links, $Pr(N|G)$ for Name Server Match, $Pr(S|G)$ for Text Content Match. Then, we have:

$$Pr(G|J,T,I,N,S) = \frac{Pr(J|G) \times Pr(T|G) \times Pr(I|G) \times Pr(N|G) \times Pr(S|G) \times Pr(G)}{Pr(J) \times Pr(T) \times Pr(I) \times Pr(N) \times Pr(S)}$$

Finally, one web site can be classified by Naïve Bayes to the category c given by:

$$c = \arg \max_{c \in \{D,G\}} Pr(c|J,T,I,N,S)$$

We found two issues when we evaluate experiment results. First, domain parking web sites were created by DPSPs, the features we defined are not independent. We observed some features are highly correlated. We will illustrate it in details in the following section. Second, in order to have a higher F_1 score, we need to have a large volume of training data set. However, in this thesis study, the domain parking training data are only couple of thousands records.

3.2.2 Association Rules Match

In this section, we show that, there are relations between each feature that we defined. First, we introduce association rules mining techniques. We present the algorithm to compute *support* value. Then, we apply the improved Apriori algorithm by using MapReduce. Finally, we show five association rules that we are going to take them as five features.

Association Rules Learning

Association rules learning is a fundamental data mining task. Its objective is to discover interesting relationship between variables in a large data set. The classic application of association rules learning is the super market basket data analysis. The interesting question is that, for example, how many people who bought milk also bought cheese?

Among five features we defined for domain parking web sites, we want to find interesting relationship between them in the training data set. The five features of association rules can be stated as follows: Let $W = \{w_1, w_2, \ldots, w_n\}$ be a domain parking training data set, where w_i is the i^{th} domain parking web site feature set $w_i = \{J, T, I, N, S\}$. $\{J, T, I, N, S\}$ stands for (i) JavaScript Source, (ii) Tracking Cookies, (iii) Internal Links, (iv) Name Server Match, and (v) Text Content Match respectively. The value of each feature could be either $True$ or $False$. An association rule is defined as, for example, $J \Rightarrow T$, where $J \subset w_i$, $T \subset w_i$, and T, J are both true in w_i.

The strength of a rule is measured by its *support* value and $confidence$ value. Let X and Y be two features. The confidence of a rule is defined as $conf(X \Rightarrow Y) = supp(X \cup Y)/supp(X)$, where $supp(X)$ is the support of X, which is defined as the proportion of all domain parking training data set that contains X. For example, Let X be "the number of internal link is zero". Let Y be "The third-party JavaScript source is larger than one". The rule $X \Rightarrow Y$ has a confidence $c\%$, which is the total count of X and Y shown in the same feature set divided by the count of X in the whole feature sets.

The *support* value of a rule, $supp(J \Rightarrow T)$ as an example, is the percentage of domain parking web sites that, for each feature set, both J and T are presence. The rule *support* value determines how frequent the rule is applicable in domain parking web sites training data. If the *support* value is too low for certain features, the relationship between them is very weak. Let n be the total number of domain parking web sites in training data. The *support* value of two features X and Y, where $X, Y \in \{J, T, I, N, S\}$, is computed as follows:

$$support = \frac{count(X \cup Y)}{n}$$

The $confidence$ of a rule, e.g. $conf(J \Rightarrow T)$, is the fraction of domain parking web sites that, for each feature set, both J and T are presence, divided by the number of J shown in the whole data set. The rule $confidence$ value determines the predictability of the rule. If the $confidence$ of $J \Rightarrow T$ is too low, J can not infer T confidently. The $confidence$ of two

features X and Y, where $X, Y \in \{J, T, I, N, S\}$, can be computed as follows:

$$confidence = \frac{support(X \cup Y)}{support(X)}$$

Finally, we define two thresholds *minimum support* and *minimum confidence* to determine association rules among five features in terms of domain parking web sites training data set.

Apriori Algorithm

There are many association rule mining algorithms, which have been widely studied. The results of those algorithms are all the same based on the definition of association rules. However, according to computational efficiency and memory requirements, the Apriori algorithm is widely selected[15].

In this thesis study, Apriori algorithm has two steps: (i) generate all frequent feature set, and (ii) generate all confident association rules from the feature set.

The following Python code snippet is revised version of Apriori algorithm for generating support values because of the following two reasons:

(1) We only have maximum five features in one feature set. We can generate all candidate list for each level $k \in \{1, 2, 3, 4, 5\}$ at the once. Function gen_candidates() has two loops to iterate each feature in the given feature set. Another loop on line 8 control each level's candidate list. The variable tuple_list is a temporary container to store each level candidate list.

(2) We do not use minimum support and minimum confidence to filter out some combination in the candidate list. Since we have small manageable quantity of features in the feature set, we want to evaluate all combinations among those features.

```
1 def gen_candidates(feature_set):
2   candidate_dict = dict()
3   for i in range(0, len(feature_set)):
4     tuple_list = []
5     tuple_list.append(feature_set[i])
6     for j in range(i+1, len(feature_set)):
7       tuple_list.append(feature_set[j])
8       for k in range(1, len(feature_set)+1):
```

```
9           if len(tuple_list) == k:
10            clist = list(tuple_list)
11            candidate_dict.setdefault(''.join(clist), [])
12            candidate_dict[''.join(clist)] = clist
13    return candidate_dict
14
15  def gen_support(retrieved_set):
16    count_dict = dict()
17    for item_set in retrieved_set:
18      candidate_dict = gen_candidates(sorted(item_set))
19      for fid, clist in candidate_dict.items():
20        if fid in count_dict.keys():
21          count_dict[fid] += 1
22        else:
23          count_dict.setdefault(fid, 1)
24    support = dict()
25    for fid, count in count.items():
26      support[fid] = count[fid] / length(retrieved_set)
27    return support
```

After analysis, we decide to select feature combinations which support larger than 60%. We have (i) Internal Links, (ii) JavaScript Source, and (iii) Tracking Cookies three feature and their combinations. Having support values for each level feature combinations, we are able to calculate confidence by Apriori algorithm.

For DPR, the size of the feature set is small. Thus, unlike the demonstrated case study in Chapter 2, MapReduce is not suitable for solving this association rules problem. The reason is shown in the following section.

MapReduce Efficiency

Can MapReduce apply to any application on any situation? In this thesis study, we don't have enough computer nodes to launch parallel computing. In order to get a proof-of-concept, we have one personal computer to run a program *octo*, which is a fast-and-easy MapReduce implementation by using Python[20].

The experiment results show that, in order to process 6.5M data, MapReduce took more execution time than one single-process-program does. The reason is, during the map phase in MapReduce, a mapper will put all key-value objects to the local hard drive. As we can imagine, for one computer, writing key-value objects takes too much disk I/O, which is

dramatically slower than writing to the memory. However, if the experiment is running on the cluster with 2000 computer nodes, the situation will be different. The 6.5M data can be processed quickly in parallel.

We have an intuitively analysis here by using word-counting as an example. Given n web sites, m words which occurring f times per web site on average, the total data size $D = n \times m \times f$. Let us assume that, the data which is emitted by all the mappers is D. Let $P = M + R$ is the number of processors while M is the number of mappers and R is the number of reducers. Let σ be the disk/network I/O time. Then, we have overhead of intermediate data written by each mapper defined by OH.

$$OH = \frac{\sigma D}{P}$$

According to overhead definition, OH will be maximized if there is only one mapper and one reducer. The question is, if increasing the number of processor will be minimize overheads OH? Let us consider wD is the useful work need to be done. w is the fraction parameter which indicates the percentage of useful data. We define MapReduce efficiency as the fraction of overheads for useful data over all overheads. Then we have the following equation:

$$\epsilon_{MR} = \frac{\frac{wD}{P}}{\frac{wD}{P} + 2\frac{\sigma D}{P}}$$

Then we have:

$$\epsilon_{MR} = \frac{1}{1 + \frac{2\sigma}{w}}$$

As we can see, P is independent of efficiency ϵ. We should try to decrease disk/network I/O σ and to increase percentage of useful data w, so that MapReduce has a better efficiency.

Therefore, if the size of data D can be processed in one machine's memory with minimum disk swap, the MapReduce may not be a good solution. However, when processing terabytes or even larger data, MapReduce parallel computing should be considered[22].

MapReduce Apriori Algorithm

We have demonstrated MapReduce efficiency above. In the case when the size of item set is large, MapReduce still improves the the performance of generating association rules.

MapReduce is a computational paradigm that can count item set parallel. We revised the Apriori algorithm by using MapReduce paradigm.

The implementation is based on program *octo*, which is introduced in previous section. The variable *source* is a dictionary that contains each web site name as a key and the corresponding feature set as a value. The function $mapfn()$ will take each key-value from *source*. It is similar with gen_candidates() function defined in classic Apriori algorithm section. The difference is, in stead of putting each feature combination (tuple_list) into local data structure in memory, the function simply put the feature combination as a key and count one as a value to disk/network I/O. The function $reducefn()$, running as another thread, collect key-value pairs yield by function $mapfn()$. Finally, we have support value for every feature combination.

```
source = dict((web_site, features)

def mapfn(key, value):
  for i in range(0, len(features)):
    tuple_list = list()
    tuple_list.append(features[i])
    for j in range(i+1, len(features)):
      tuple_list.append(features[j])
      for k in range(1, len(features) + 1)
        if len(tuple_list) == k:
          yield tuple_list, 1

def reducefn(key, value):
    return key, len(value)
```

New Features: Association Rules Match

We set $minimum\ support = 60\%$ and $minimum\ confidence = 80\%$. We keep the same symbols as previously defined for features. Let J, T and I be JavaScript Source, Tracking Cookies and Internal Links respectively. Then, we have the following association rules:

Rule 1 $J \Rightarrow I$. (support=66%, confidence=100%)

Rule 2 $J \Rightarrow T$. (support=66%, confidence=100%)

Rule 3 $T \Rightarrow I$. (support=100%, confidence=100%)

Rule 4 $T \Rightarrow J$. (support=100%, confidence=100%)

Rule 5 $\{J, T\} \Rightarrow I$. (support=66%, confidence=85%)

For each rule, we can treat it as one feature of domain parking web sites. For example, to recognize an unknown web site, we construct a feature set for the web site. The feature set contains: (i) The count of non-local JavaScript source is large than zero; and (ii) The count of internal link is zero. Then, the Rule 1 is matched. The Rule 1 is a feature to identify if this unknown web site belongs to domain parking web sites. In the following section, we will apply new generated rules as new features to SVM classifier.

3.2.3 Support Vector Machine

We have introduced SVM as one classifier of GWCC case study in Chapter 2. For DPR, since we discovered five association rules among the features we defined previously, we are going to take each rule as one feature. Therefore, the feature set for SVM classifier contains: (i) Name Server Match, (ii) Rule 1, (iii) Rule 2, (iv) Rule 3, (v) Rule 4, (vi) Rule 5, and (vii) Text Content Match.

Because we do not know whether two classes, domain parking web sites and non-domain-parking web sites, are linear separated or not, we want to experiment on both linear and non-linear SVM classifiers. We introduce basic four kernel functions in the following section.

SVM Kernel Function

We will keep the symbols as defined in Section 2.2.3. We use $X = \{x_1, x_2, \ldots, x_n\}$ as feature space in this section.

The linear SVM classifier relies on inner product between vectors $K(x_i, x_j) = x_i^T x_j$. For non-linear SVM classifier, it uses transferred high-dimensional feature space $\phi(x)$. Then the inner product becomes: $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$. A kernel function is a function that is equivalent to an inner product in some feature space. For example, given a input point x_i, we want to find point x_j with smallest distance. We have:

$$\begin{aligned} ||x_i - x_j||^2 &= (x_i - x_j)^T (x_i - x_j) \\ &= x_i^T x_i - 2x_i^T x_j + x_j^T x_j \end{aligned}$$

If we use a non-linear feature space $\phi(x)$, then we have:

$$||\phi(x_i) - \phi(x_j)||^2 = \phi(x_i)^T\phi(x_j) - 2\phi(x_i)^T\phi(x_j) + \phi(x_j)^T\phi(x_j)$$
$$= K(x_i, x_i) - 2K(x_i, x_j) + K(x_j, x_j)$$

Thus, a kernel function implicitly maps data to a high-dimensional space without the need to compute each $\phi(x)$ explicitly. There are four basic kernels that we can choose.

(1) Linear: $K(x_i, x_j) = x_i^T x_j$

(2) Polynomial of power d: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$

(3) Radial Basis Function (RBF): $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2), \gamma > 0$

(4) Sigmoid: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$

SVM Classifier Tool: SVM^{light}

SVM^{light} is a command-line program that runs SVM classifier. It is originally written in C but it also provides a Python module.

The following code snippet shows SVM learning process. First, we need to build up training data set. We construct feature sets for each retrieved web site. Then, we append the feature set to train_data list. After all training data are prepared, we have function *svmlight.learn*() to generate learning model. The learning model is an optimized hyperplane that can be used to classify a given web site. The parameters passed to the function are: (i) training data feature sets, (ii) type of learning - classification, (iii) kernel function - RBF and (iv) RBF γ value, and C parameter value. During the experiment, we also tried other kernel functions.

```
1 train_data = list()
2 feature_set = [
3   (1, name_server),
4   (2, rule_1),
5   (3, rule_2),
6   (4, rule_3),
7   (5, rule_4),
8   (6, rule_5),
9   (7, text_match)]
10 train_data.append((label, feature_set))
```

```
11 model = svmlight.learn(train_data, type='classification',
12         kernel='rbf', rbf_gamma=2, C=0.5, verbosity=1)
```

The following code snippet shows SVM predicting process. First, we need to build up test data set. We construct a feature set for a given web site. Then, we append the feature set to test_data list. By using generated learning model above, we have function *svmlight.classify*() to predict whether or not the given web site belongs to domain parking web site.

```
1 test_data = list()
2 feature_set = [
3   (1, name_server),
4   (2, rule_1),
5   (3, rule_2),
6   (4, rule_3),
7   (5, rule_4),
8   (6, rule_5),
9   (7, text_match)]
10 test_data = [(0, feature_set)]
11 predictions = svmlight.classify(model, test_data)
```

SVM Tuning Observations

Since SVM kernel functions and their parameters play import roles in terms of prediction results, we conduct many experiments for SVM tuning. When we try to use different kernel function and different combinations of parameter values, we always observe that, even though some features in the feature set are dominant compared to other features, the prediction result tends to be wrong direction. For example, we use one identified domain parking web site to run test. Its name server matches our reference list. Its text content shows "this domain is for sale". After SVM predicting, the result is false negative. The other five features may play a role to mistakenly unrecognized this domain parking web site.

We want to figure out: If there is any better classifier that, higher confidence feature should be dominant for the final decision. Finally, we have our own proposed learning algorithm based on Feature Confidence Index.

3.3 Feature Confidence Index

In this section, we introduce the concept of Feature Confidence Index (FCI). We introduce a Feature Confidence Graph, which is a graphic demonstration to guide us to finally have FCI classifier. FCI algorithm is presented accordingly. At last, we show that, in terms of comparative results, FCI classifier is an appropriate choice for DPR.

3.3.1 Feature Confidence Index Motivations

From SVM tuning observation, we have some intuitive thinking:

(1) More features may "disturb" a classifier to do right prediction for domain parking web site. For example, we have five rules as five features. They may lead a prediction to a wrong direction even other two features show a very strong evidence. It gives us an idea, some "confident" features should be dominant features in the feature set. If we need to assign weight to each feature, compared to Text Content Match feature, the weight of Rule 1 may be smaller.

(2) A classifier should consider various size of a feature set, rather than a fixed number of features. For example, when we prepare a feature set for SVM, we have to put all seven features for each web site. For some of unavailable features, e.g. Rule 1 is not matched, we still need to set it as zero value in the feature set. DPR is a special application compared with GWCC. Sometimes, one or two dominant feature(s) may be good enough to identify domain parking web site.

Grouping Features

The point (1) is an overfitting problem. We have applied a parameter C of SVM to control overfitting because it is the way to provide soft-margin to trade off relative important maximum margin. However, overfitting still occurs due to (i) small size of training data set, or (ii) complex parameters applied in SVM classifier. Since overfitting generally occurs when a model is excessively complex, for domain parking web sites, we want to have a simple model to reduce this overfitting problem. Therefore, we combine five rules as one feature, naming Association Rules Match. Then, we have a definition for the new feature:

Definition 7. ***Association Rules Match Feature:*** *The feature is presented if and only if all five association rules are discovered.*

Thus, we now have three features to recognize domain parking web sites: (i) Name Server Match - NSM, (ii) Association Rules Match - ARM, and (iii) Text Content Match - TCM.

Feature Confidence

Based on three features, let us assume that, we have a two-class linear classification function $f(x) = w^T x - \sigma$, where x is a feature availability set that contains $\{NSM, ARM, TCM\}$, and $w = \{w_{NSM}, w_{ARM}, w_{TCM}\}$ is a normalized weight vector $\sum_{i \in \{NSM,ARM,TCM\}} w_i = 1$, which is learned from training data set. The parameter σ is a threshold, thus, if $w^T x \geq \sigma$, the function return True, meaning domain parking web site is identified. DPR have a prediction function that:

$$DPR_{predict} = \begin{cases} True & w^T x \geq \sigma \\ False & otherwise \end{cases}$$

According to the point (2), if only one features in x, e.g. TCM, are available, the weight w_{NSM} and w_{ARM}, which are learned from training data set, will not be counted. Thus, the final prediction may not be correct. The reason is that, when TCM feature is available, we have a higher confidence to believe the web site belongs to domain parking web sites. Therefore, we have a feature confidence illustrated as follows.

Feature Confidence is defined as a conditional probability of each feature

$$FC_{feature} = Pr(feature|D) \tag{3.2}$$

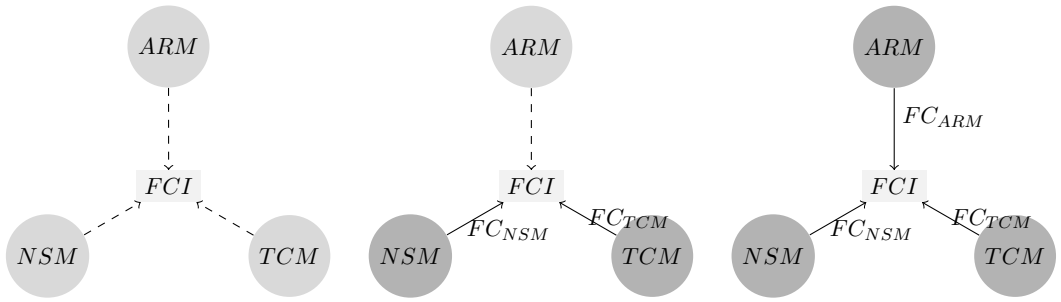
where $feature \in \{NSM, ARM, TCM\}$ and D stands for domain parking web sites. With feature confidence in hand, if there is only one feature TCM available for an unknown web site, we still can compare FC_{TCM} and σ to have a final prediction.

3.3.2 Feature Confidence Graph

Before we have a FCI classifier, feature confidence graph gives us a graphic demonstration for how FCI classifier works. Feature confidence graph G contains two kinds of node: (i) feature node and (ii) class-determine node. Feature nodes represent each feature in the G. In this thesis study, we have three feature nodes. Class-determine node contains a prediction function $FCI : FCs \rightarrow D$, so that, by giving feature confidence FCs, it is able to predict the label of domain parking web site D.

Let v belongs to the one of feature nodes. Let u belongs to the class-determine node. The directed edge $\vec{E} = (u, v)$ is presence if and only if the feature v is available. For example, v represents feature NSM. The edge $\vec{E} = (u, v)$ is presence if and only if the feature NSM is true. The direction of edge $\vec{E}$ is from feature nodes to the class-determine node. It is the channel to convey feature confidence from v to u.

Figure 3.2 shows how feature confidence graph works. For diagram (i), all three feature nodes are false, the edges $\vec{E}$ are not existed. We use dashed arrows to represent. For diagram (ii), because two feature nodes are true, the edges between these two feature nodes and the class-determine node are created. The feature confidence of these two features, FC_{NSM} and FC_{TCM}, are delivered to the class-determine node. For diagram (iii), all three feature nodes are true. All feature confidence, FC_{NSM}, FC_{ARM} and FC_{TCM}, are contributed to the class-determine node for final prediction.



(i) All three feature nodes are false. (ii) Two feature nodes are true. (iii) Three feature nodes are true.

Figure 3.2: Three cases of feature confidence graph.

3.3.3 FCI Algorithm

According to FCG demonstration, we come up with an algorithm to generate Feature Confidence Index - FCI. In terms of formula (3.2), FCI is defined as the probability that any of the $FC_{feature}$ is true, while $feature \in \{NSM, TCM, ARM\}$. We have a FCI formula defined as:

$$FCI = Pr(\bigvee_{feature \in \{NSM,TCM,ARM\}} FC_{feature} | feature = True) \tag{3.3}$$

Then, we have an algorithm to generate a FCI value for a given web site. Algorithm 4 shows that, the input data are feature confidence for each feature, which is calculated by

formula (3.2). We also have the product of the probabilities of two features as input data, which are defined as:

$$\begin{aligned} FC_{NSM \wedge TCM} &= Pr(NSM|TCM, D) \times Pr(TCM|D) \\ FC_{NSM \wedge ARM} &= Pr(NSM|ARM, D) \times Pr(ARM|D) \\ FC_{TCM \wedge ARM} &= Pr(TCM|ARM, D) \times Pr(ARM|D) \end{aligned} \tag{3.4}$$

The input data also include the product of the probability of three features, which is defined as:

$$FC_{NSM \wedge TCM \wedge ARM} = Pr(NSM|D) \times Pr(TCM|NSM, D) \times Pr(ARM|TCM, NSM, D) \tag{3.5}$$

Based on formula (3.4) and (3.5), we extend formula (3.3) to:

$$\begin{aligned} FCI = FC_{NSM} + FC_{TCM} + FC_{ARM} \\ -FC_{NSM \wedge TCM} - FC_{NSM \wedge ARM} - FC_{TCM \wedge ARM} \\ +FC_{NSM \wedge TCM \wedge ARM} \end{aligned} \tag{3.6}$$

The output result is FCI for the web site. The variables, n,t and a, are coefficient of feature confidences when calculating FCI. They are set to zero initially. The algorithm goes through each feature. If one feature appears, the corresponding variable is set to one.

Finally, the FCI is calculated by input data and coefficient variables.

Data: FC_{NSM}, FC_{TCM}, FC_{ARM}, $FC_{NSM \wedge TCM}$, $FC_{TCM \wedge ARM}$, $FC_{NSM \wedge ARM}$, $FC_{NSM \wedge TCM \wedge ARM}$
Result: Feature Confidence Index FCI
n, t, a $\leftarrow$ 0;
if *Feature Name Server Match is True* **then**
  n $\leftarrow$ 1;
end
if *Feature Text Content Match is True* **then**
  t $\leftarrow$ 1 ;
end
if *Feature Association Rules Match is True* **then**
  a $\leftarrow$ 1 ;
end
$FCI = n \times FC_{NSM} + t \times FC_{TCM} + a \times FC_{ARM} - n \times t \times FC_{NSM \wedge TCM} - n \times a \times FC_{NSM \wedge ARM} - t \times a \times FC_{TCM \wedge ARM} + n \times t \times a \times FC_{NSM \wedge TCM \wedge ARM}$;
return FCI

Algorithm 4: Feature Confidence Index

We have a threshold σ. The DPR prediction function $DPR_{predict}$ returns True if $FCI \geq \sigma$. Then, DPR prediction function is defined as:

$$DPR_{predict} = \begin{cases} True & FCI \geq \sigma \\ False & otherwise \end{cases}$$

3.4 Summary

In this chapter, we proposed several features that are able to recognize domain parking web sites. During classifier selection, In order to see if those features are independently present, we generate five association rules as five new features. When we use SVM classifier to treat totally seven features equally, we have some intuitive thinking to design our own classifier. Finally, we proposed the Feature Confidence Index algorithm, which is more appropriate for DPR.

Chapter 4

Experiments and Results

With the domain parking web site features defined, we conduct experiments for Naïve Bayes, SVM and FCI classifiers. We first introduce the experiment preparation, including experiment environment, data set selection and training data set preparation. Then, we describe experiments on text-based classifiers: Naïve Bayes and SVM, for domain parking web sites. After that, The experiments on FCI classifier is conducted. Finally, we show the comparative results among these three classifier.

4.1 Experiments Preparation

4.1.1 Experiment Environment

For DPR, we use the same experiment environment as case study of GWCC. We have one computer to include all modules of DPR, from database system to classifier tools. The computer is running Mac OS Mountain Lion, which is UNIX-like operating system. The CPU is Intel(R) Core(TM)2 Duo at 2.40GHz. We have 6G memory space. Most of modules, including classifier tools, are implemented by Python language.

4.1.2 Data Set Preparation

Data Set Selection

Data set selection is deterministic for this thesis study. The decision we made is based on the scope of data that we are going to process regularly. Initially, our motivation is to solve the problem that domain parking webs sites can not be recognized by GWCCE for

daily released domain web sites. Therefore, all domain web sites we selected are based on zone files data from six major top-level domains: (i) *.com* zone files, (ii) *.org* zone files, (iii) *.gov* zone files, (iv) *.us* zone files, (v) *.biz* zone files, and (vi) *.info* zone files. Data source provider is PremiumDrops.com.

Training Data

Unlike preparing training data set in case study of GWCC, building up domain parking web sites data set is labor intensive. In order to minimize reviewing time for each web site, we applied a Python Selenium module to (i) launch web browser, (ii) visit the web sites, and (iii) take screen shots automatically. All screen shots were taken at the night time. We then can review the screen shots and pick up the right one. By using this approach, we finally have 2000 domain parking web sites. For non-domain-parking web sites, we select top 2000 sports web sites in Alexa, which were used in case study.

Testing Data

For testing data preparation, we continuously use Python Selenium module to randomly pick up new added domains daily. After reviewing the screen shots, we finally have 200 domain parking web sites each day to do the test. We also select another 200 non-domain-parking web sites each day from Forbes top 2000 business and top U.S. universities.

According to experiment results, those non-domain-parking web sites test data have a bias. We noticed that, if we select all test data from business web sites for one experiment, the false positive number will be higher than that we select all test data from universities web sites. Intuitively, the reason is that, domain parking web sites are created to pretend to be business web sites.

We adopt false positive rate to represent the bias for two groups of test data. Let n be the number of non-domain-parking web sites for one experiment. Let fp be the number of web sites that FCI classifier mistakenly predict them as domain parking web sites. Then, false positive rate is defined as:

$$false\ positive\ rate = \frac{fp}{n}$$

Table 4.1 shows partial experiment results for two groups of test data. Overall, test data from top U.S. universities have lower false positive rate than that from Forbes top 2000 business.

Experiments	Data Source	False Positive Rate
1	Top U.S. universities	5.0%
2	Top U.S. universities	7.0%
3	Top U.S. universities	0.0%
4	Top U.S. universities	8.0%
5	Forbes top 2000 business	14.43%
6	Forbes top 2000 business	12.89%
7	Forbes top 2000 business	14.50%
8	Forbes top 2000 business	11.17%

Table 4.1: Support Values

In order to avoid test data bias, we randomly mix web sites from these two groups as test data.

4.1.3 Training Data Feature Extraction

We have a fetching module implemented by Python to retrieve useful information from web sites to local database.

We have introduced HTML parser in Chapter 3. The features: (i) the number of internal links, (ii) the number of non-local JavaScript source, and (iii) text phrases of the web page, are extracted by HTML parser.

We have a DNS module provided by Python library. The module provide a query function that can specify SOA record. Thus, we can get authority name server for any web site by using this module.

We also have a Python library - *urllib*2. It is a HTTP client that is able to connect HTTP server. By using this tool, we can extract cookie set from HTTP headers that HTTP servers send to us. Thus, the number of tracking cookie set is stored to our database.

4.1.4 Evaluation Metrics

We have introduced F_1 score in Chapter 2 as a metric to evaluate case study of GWCC. F_1 score considers both the precision and the recall. For DPR, we continuously use F_1 score as a metric to evaluate DPR prediction.

4.2 Text-based Classifier Experiments

In this section, we conduct two experiments by text-based classifiers introduced in GWCC case study. We first conduct Naïve Bayes classifier experiments by using five original features. After association rules generated, we conduct SVM classifier experiments by using seven features. We also show the experiment results for both classifiers.

4.2.1 Naïve Bayes Classifier Experiments

We design and implement Naïve Bayes classifier tool by Python and its libraries. We conduct the experiments by using five original features we defined, including (i) Name Server Match, (ii) Text Content Match, (iii) JavaScript Source, (iv) Internal Links, and (v) Tracking Cookies.

The NBC tool has two major parts. One part is a NBC learner module. We keep all training data information in a database system. This module is a major function to load data from training database, and calculate probabilities for five features, including (i) prior probabilities, (ii) likelihood probabilities and (iii) evidence probabilities. When DPR is launched, all features prior probability, likelihood and evidence are loaded to the memory as a key-value dictionary. The other part is a NBC prediction module. This module has two major functions. One function is to fetch information from a given web site and construct a feature set. After the web site information is retrieved, another function is to calculate posterior probabilities in terms of the key-value dictionary.

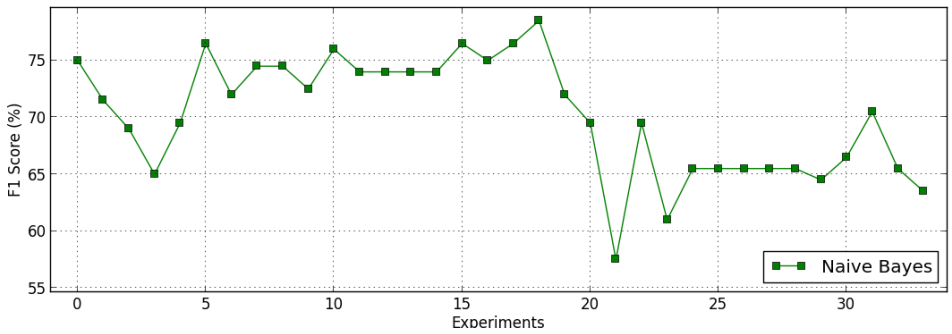


Figure 4.1: Experiment results for Naïve Bayes classifier

We conduct roughly 30 experiments in one month. We have 200 domain parking web sites and 200 non-domain-parking web sites for each day. Figure 4.1 shows experiment

results. As we can see, the best F_1 score is 78%. However, the worst F_1 score is only 57%. The average F_1 score is 70%.

We have illustrated the limitations of NBC in Chapter 2 and Chapter 3. We believe that, there must be some relationship between features. In order to find relations among these five features, we conduct association rule experiments.

Features	Supports
Name Server Match	82%
Text Content Match	73%
Internal Links	89%
JavaScript Source	66%
Tracking Cookies	100%
JavaScript Source ⇒ Tracking Cookies	66%
JavaScript Source ⇒ Internal Links	66%
Tracking Cookies ⇒ Internal Links	100%
Tracking Cookies ⇒ JavaScript Source	100%
{Tracking Cookies, JavaScript Source } ⇒ Internal Links	66%

Table 4.2: Support Values

Table 4.2 shows support values for the features. Then we have five rules among Internal Links, JavaScript Source and Tracking Cookies. Plus Name Server Match and Text Content Match, we have totally seven features. Due to the limitations of NBC, we conduct experiments on SVM classifier.

4.2.2 SVM Classifier Experiments

We have demonstrated SVM classifier in Chapter 3. For SVM classifier experiments, we first introduce kernel function selection. We then use Radial basis function as a kernel function. At last, we show the experiment results.

Kernel Function Selection

As we illustrated in Chapter 3, kernel functions provide us an inner product of transferred high-dimensional feature space $\phi(x)$. Because the dimension of the feature set is seven, it is hard to imagine what the separating hyperplane looks like. Therefore, we conduct an experiment for selecting appropriate kernel function for DPR.

Kernel Function	Average F_1 Score
Linear	55%
Polynomial	63%
Radial Basis Function	77%
Sigmoid	76%

Table 4.3: SVM kernel function selection

Table 4.3 shows the final result of the experiment result. We first generate four learning models by using *svmlight.learn*() function based on the same training data set. These four learning models corresponds to (i) Linear function, (ii) Polynomial function, (iii) Radia basis function, and (iv) Sigmoid function. Then, we pick up several small group of testing data set. For each testing group, we have a F_1 score for each kernel function. At last, we have average F_1 scores.

According to Table 4.3, we finally choose Radial Basis function as the kernel function for SVM classifier.

SVM^{light} Experiments

The SVM^{light} tool has two parts. One part is done by learning function *svmlight.learn*(). The input parameters include (i) training data feature sets, (ii) the type of learning, (iii) the type of kernel function, (iv) verbosity level, and (v) other parameters related with the type of learning and the type of kernel function. The output is a learning model, which is the data to describe what is separating hyperplane. The learning model is used for the classification. Another part is done by classification function *svmlight.classify*(). The input parameters include (i) test data feature sets, and (ii) the learning model generated in part one.

We conduct around 30 experiments in one month. We keep using 200 domain parking web sites and 200 non-domain-parking web sites for each day. Figure 4.2 shows experiment results. The best F_1 score is 84%, which is better than 78% of NBC. The worst F_1 score is only 70%, which is still better than 57% of NBC. The average F_1 score is 77%, while NBC is only 70%.

We see the improvement by introducing five rules as five features. However, we noticed that, the classification is wrong even though the feature, e.g. Text Content Match, shows very obviously indication of domain parking web sites. Therefore, we start to conduct FCI

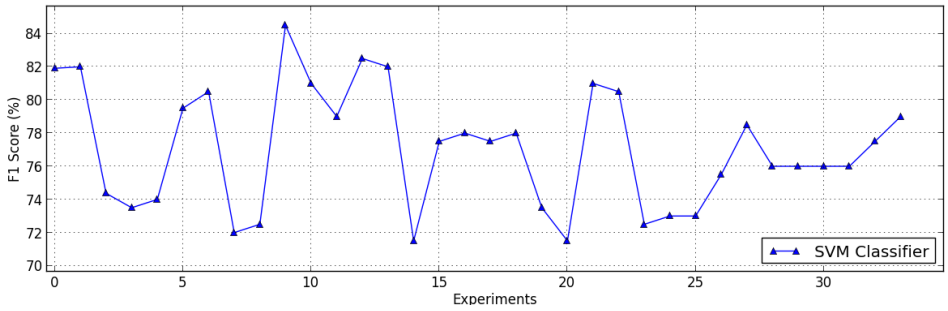


Figure 4.2: Experiment results for SVM classifiers

classifier experiments.

4.3 FCI Classifier Experiments

4.3.1 FCI Threshold Selection

We have defined DPR prediction function in Chapter 3. In the function, σ plays an important role to make a decision. The selection of σ is based on the experiments.

Feature Confidence Values

We have feature confidence values FC_{NSM}, FC_{ARM} and FC_{TCM}, which are calculated by formula (3.2). Additionally, we have formula (3.4) and (3.5) to calculate the products of the probabilities of features. Table 4.4 shows the results according to the training data set we collected.

Feature Confidence	Values
FC_{NSM}	82%
FC_{TCM}	73%
FC_{ARM}	85%
$FC_{NSM \vee TCM}$	88%
$FC_{NSM \vee ARM}$	95%
$FC_{TCM \vee ARM}$	91%
$FC_{NSM \vee TCM \vee ARM}$	99%

Table 4.4: Feature confidence values

σ Selection

We decide to use five levels of σ value to be thresholds. Level 1 (σ_1) is defined as minimum feature confidence among three features. Level 2 (σ_2) is defined as the mean value of three feature confidence. Level 3 (σ_3) is defined as a minimum value of the probabilities of any two features that happened. The probabilities can be calculated by formula (3.3). Level 4 (σ_4) is defined as the mean value of the probabilities of any two features that happened. Level 5 (σ_5) is defined as the probabilities of any three features that happened. Table 4.5 shows the definitions, which are used to calculate σ values.

Thresholds	Definitions
σ_1	$min(FC_{TCM}, FC_{NSM}, FC_{ARM})$
σ_2	$mean(FC_{TCM}, FC_{NSM}, FC_{ARM})$
σ_3	$min(FC_{TCM \vee NSM}, FC_{TCM \vee ARM}, FC_{NSM \vee ARM})$
σ_4	$mean(FC_{TCM \vee NSM}, FC_{TCM \vee ARM}, FC_{NSM \vee ARM})$
σ_5	$FC_{TCM \vee NSM \vee ARM}$

Table 4.5: The σ levels definition

Based on table 4.4 and 4.5, we have five σ values shown on table 4.6.

Thresholds	Values
σ_1	73%
σ_2	80%
σ_3	88%
σ_4	91%
σ_5	99%

Table 4.6: Threshold values

Experiments on σ Values

We conduct 10 experiments in 10 days. We have 50 domain parking web sites and 50 non-domain-parking web sites for each day. Each experiment applies five σ values. To evaluate the experiment results, we also record the values of *precisions* and *recalls*. In order to

simplify decision making, we use the mean of 10 experiment results. The table 4.7 shows the mean values for *precisions*, *recalls* and F_1 scores.

Thresholds	*Precisions*	*Recalls*	F_1 scores
σ_1	66.67%	95.24%	78.43%
σ_2	74.07%	91.74%	81.97%
σ_3	89.29%	90.91%	90.09%
σ_4	90.91%	74.07%	81.63%
σ_5	98.04%	58.82%	68.18%

Table 4.7: The mean values of *precisions*, *recalls* and F_1 scores

Based on table 4.7, Figure 4.3 shows three diagrams, so that, we can see *precisions* and *recalls* affected by different level of σ values.

The left diagram shows that, the *precision* values are increased along with the σ values. The reason is that, the higher σ values, the lower false positive numbers. For example, during the experiments, we observed that, some business web sites have NSM feature or TCM feature. Lower σ value will be more likely to cause false positive. However, there is a few of business web sites have NSM feature and TCM feature at the same time. Thus, the higher σ value, the lower false positive, accordingly the higher *precision* value.

The middle diagram shows that, the *recall* values are decreased along with the σ values. The reason is that, the higher σ values, the higher false negative numbers. For example, we observed during the experiments that, some domain parking web sites only have one NSM feature. They don't have TCM or ARM feature. If σ threshold is higher, they are missed to be identified as domain parking web sites.

The right diagram shows the curve of F_1 scores. We can see that, when $\sigma_2 = 80\%$ and $\sigma_4 = 91\%$, the F_1 scores are close. The highest F_1 score is 90.09% when $\sigma_3 = 88\%$. When $\sigma_5 = 99\%$, the F_1 score dropped dramatically. We finally decide to use $\sigma_3 = 88\%$ as the threshold.

4.3.2 DPR Prediction Experiments

With formula (3.6) and threshold $\sigma = 88\%$, we start to conduct DPR prediction experiments. DPR has two parts: learning module and prediction module. Learning module is to calculate feature confidence from training data set, and to load results to the shared

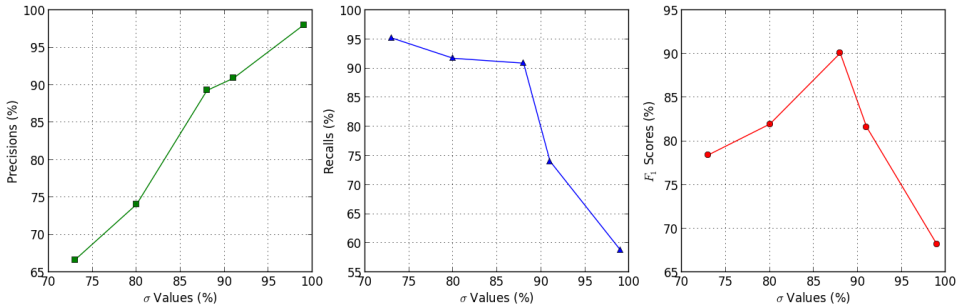


Figure 4.3: Experiments on FCI threashold σ selection

memory. Prediction module is to fetch information from a given web site, and to construct features and their values. Following the Algorithm 4 in Chapter 3, DPR is able to predict whether or not the given web site is domain parking web site.

Accuracy Comparison

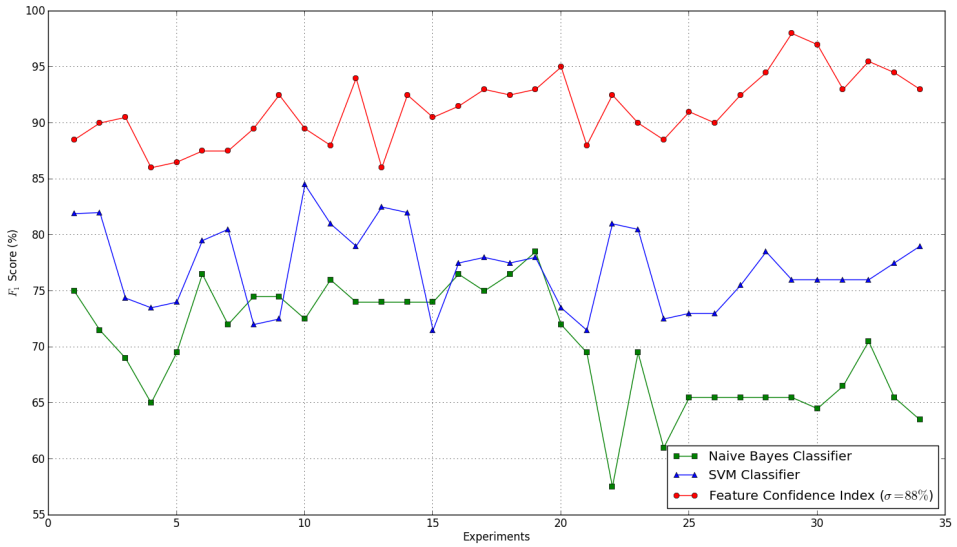


Figure 4.4: Comparative results for NBC, SVM and FCI classifiers

We conduct around 30 experiments in one month. We keep using 200 domain parking web sites and 200 non-domain-parking web sites for each day. Figure 4.4 shows experiment

results. The best F_1 score is 98%, which is better than 84% of SVM. The worst F_1 score is only 86%, which is still better than 70% of SVM. The average F_1 score is 91%, while SVM is only 77%.

Computational Complexity Comparison

We will consider time complexities of learning phase and prediction phase for NBC, SVM and FCI. In the learning phase, let W be a training set. Let F be a feature set for each sample in W (We assume that $|F|$ is the same for all three classifiers). SVM classifier has $O(|W|^2 * |F|)$ running time for RBF kernel[5]. NBC classifier has $O(|W| * |F|)$ running time[9]. FCI classifier has $O(|W| * |F|)$ running time. In the prediction phase, all three classifiers have $O(|F|)$ running time. However, NBC classifier has to predict twice for domain parking and non-domain-parking web sites. Therefore, FCI classifier is more time efficient than NBC and SVM.

4.4 Summary

In this chapter, we demonstrate the experiments for each phase of DPR evolution. Due to popularity and reputation, we conduct experiments based on NBC classifier. In order to find relations among those features, we conduct association rules experiments. We re-define features for SVM classifier due to its reputation on text-based categorization. We notice the result of SVM is better than that of NBC. We observe that, some features may "disturb" SVM classifier. We then come up with FCI classifier. We conduct experiments for the threshold σ selection. Finally, we conduct experiments based on the threshold we selected. The comparative result shows that, our proposed FCI classifier is more suitable for DPR.

Chapter 5

Conclusion and Future Work

In this thesis, we studied a special case of web content data mining. We have researched several features that domain parking web sites owned. We also selected the classifier based on experiment results and finally we came up with our own simple learning algorithm - FCI. During the thesis study, we learned relative topic, including how to select a database system, how to adopt a parallel computing model MapReduce. In the following, the conclusion and contribution of this thesis, and the future work are summarized.

5.1 Conclusion and Contributions

In this thesis study, we found that, to recognize domain parking web sites may cause higher false negative by traditional web text content categorization. We then investigated several features related with domain parking web sites. We proposed a Domain Parking Recognizer as a classification engine with Feature Confidence Index as its classifier. In order to increase F_1 score, we researched on two text-based classifiers, and eventually came up with a FCI learning algorithm. We conclude that, FCI learning algorithm has a better result than text-based classifiers.

We have two contributions in this thesis study. The first one is domain parking web site feature selection. We identified five features for domain parking web sites. We also found the relations among those features. For example, the feature set retrieved from domain parking web sites contains tracking cookie, then it is 100% confidence that internal link is zero. The second one is the FCI classifier that is tailor-made for domain parking web sites. The experiments results show that, the FCI algorithm is more appropriate for DPR than

text-based classifiers, such as Naïve Bayes and SVM.

5.2 Future Work

Although DPR is in a good shape to pre-process domain web sites from zone files in this thesis study, it still has a space to be improved.

First, the data set we have learned have a scope that, we only took zone files of six major top-level domains. Most of domains are registered in North America. To extend this research, we could investigate more zone files data, for example, zone file data from Asia or Europe. We probably will find more interesting features and consequently improve our FCI algorithm.

Second, since domain web sites may be changed from domain parking web sites to some commercial web sites. Revisiting or aging process is another challenge for DPR. Currently, we only give the result by real-time retrieved the content. As time passing by, the results stored in database may not be correctly represent the current information of web sites.

Appendix A

Domain Parking Reference List

A.1 Domain Parking Service Providers Reference List

- sedoparking.com
- voodoo.com
- parked.com
- easily.net
- buy.internettraffic.com
- sell.internettraffic.com
- domainapps.com
- bodis.com
- 123-parking.co.uk
- cashparking.com
- ns.ultearch.com
- dnsnameserver.org
- parkingspa.com

- hostingnet.com
- afternic.com
- parkingcrew.net
- smartname.com
- googleghs.com
- parking-page.net
- topdomainer.com
- expireddomains.register.com
- park-you-domain.com
- parkingpage.namecheap.com
- expireddomains.register.com

A.2 Name Server Reference List

- dsredirection.com
- monikerdns.com
- fastpark.com
- voodoo.com
- above.com
- parking.com
- dnsfastandeasy.com
- domaincontro.com
- bodis.com
- hostmonster.com

- searchguideinc.com
- hosting.com
- 123-reg.com
- dnsow.com
- ovh.net.com
- registrar-servers.com
- hostneverdie.com
- eurodns.com.com
- namebrightdns.com
- expiringmonitor.com
- renewyourtld.com

A.3 Text Content Reference List

- buy this domain
- domain is for sale
- inquire about this domain
- parked free domain
- interested in this domain
- parked for free
- please check back soon
- under construction

Bibliography

[1] MIKE 2.0. The big data definition, 2013. http://mike2.openmethodology.org/wiki/Big_Data_Definition.

[2] Charu C. Aggarwal and ChengXiang Zhai. A survey of text classification algorithms. 2012.

[3] Michael W. Berry and Jacob Kogan. Text mining applications and theory. 2010.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc. Secaucus, NJ, USA 2006.

[5] Christopher Burges. A tutorial on support vector machines for pattern recognition. 1998.

[6] U. Matzat C. Snijders and U. Reips. big data: Big gaps of knowledge in the field of internet science. 2012.

[7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, December 2004.

[8] Laney Douglas. The importance of 'big data': A definition. June 2012.

[9] Chris Fleizach and Satoru Fukushima. A naive bayes classifier on 1998 kdd cup. 1998.

[10] Blue Coat System Inc. Bluecoat webfilter, December 2012. http://www.bluecoat.com/products/proxysg/addons/webfilter.

[11] Google Inc. Google image search, December 2012. http://image.google.com.

[12] Google Inc. Google trends, December 2012. http://www.google.com/trends.

[13] The Big Data Institute. Introduction to big data and hadoop ecosystem, 2013. http://thebigdatainstitute.wordpress.com.

[14] T. Joachims, B. Schlkopf, C. Burges, and A. Smola. Making large-scale svm learning practical. advances in kernel methods - support vector learning. 1999.

[15] Bing Liu. Information retrieval and web search. In *Web Data Mining*, Data-Centric Systems and Applications, pages 11–236. Springer Berlin Heidelberg, 2011.

[16] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision, 60, 2, pp. 91-110*, 2004.

[17] Tom M. Mitchell. McGraw Hill, 2010.

[18] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (INTERNET STANDARD), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604.

[19] Netcraft. December 2012 web server survey, December 2012. `http://news.netcraft.com/archives/2012/12/04/december-2012-web-server-survey.html`.

[20] Octopy. Easy mapreduce for python, 2012. `http://code.google.com/p/octopy/`.

[21] M. Porter. An algorithm for suffix stripping. 1980.

[22] Anand Rajaraman, Jure Leskovec, and Jeffrey D. Ullman. *Mining of Massive Datasets*. Cambridge University Press (Dec 30 2011).

[23] Irina Rish. An empirical study of the naive bayes classifier. 2001.

[24] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[25] Yizhou Sun, Hongbo Deng, and Jiawei Han. Probabilistic models for text mining. 2012.

[26] L. Valiant. A theory of the learnable. *Communications of the ACM, 27*, 1984.

[27] Wikipedia.org. Precision and recall, December 2012. `http://en.wikipedia.org/wiki/Precision_and_recall`.

[28] Wikipedia.org. Domain parking, 2013. `http://en.wikipedia.org/wiki/Domain_parking`.

[29] Zhanwu Xu and Miaoliang Zhu. Color-based skin detection: survey and evaluation. *Multi-Media Modelling Conference Proceedings*, 2006.