T.C. DOGUS UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY
COMPUTER AND INFORMATION SCIENCES MASTER PROGRAM

A MULTITHREADED WEB CRAWLER AND TEXT SEARCH ENGINE

M.S. Thesis

Arzu Behiye TARIMCI
200591002

ADVISOR:

Prof. Dr. Selim AKYOKUŞ

SEPTEMBER 2009
İstanbul

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

First of all; I would like thank my advisor Prof. Dr. Selim Akyokuş for his guidance, help and patience throughout this research. He helped me to shape my ideas and knowledge in many respects of computer and information systems. I also feel the need to thank Prof. Dr. Mitat Uysal. I am very grateful to Doguş University and the people who built this institution.

I thank to my family. Especially my mother Sevim Gamsiz who believes me more than anyone else she helped me in economical and emotional aspect ignoring her own problems from the moment I was born, my brother Ahmet Salt and her wife Melek Nazlı who shared their computer facilities with me. I cannot forget my cousin, Asuman Savasçıhabeş. I am grateful all of them.

Hikmet Erdoğan who was my classmate in Doguş University. He was really very good friend. We started Doguş University at the same time. I also always remember Hikmet Erdoğan with great respect.

ABSTRACT

Without a doubt, internet is one of the best inventions in the last era. Number of internet users is more than millions. When internet users need information about something or somewhere, they visit search web sites or personal blog pages on the internet. For this purpose, many internet applications have been developed.

Search Engines and data mining have shown a big improvement in the last 20 years. The developments on the internet increased the need of accessing and finding correct web resources. Raise of search engines caused to differentiation of search engine services. More intelligent search engines are important for accessing to the correct data.

Search engines scan contents of the web sites and create indexes for their contents into own database using robots. Advances in search engines enable classification of subjects of the documents besides words or terms used in a document. Such search engines which have document classification property are called “Clustered Search Engines”. For determination of page categories, the data mining methods are used.

In this thesis study, a web crawler and classification system has been developed. The Open Directory Project (DMOZ) is used as a training set for the classification system. The labeled (categorized) web pages which are stored in the DMOZ directory are used as an input for the classification algorithms. We used classification algorithms available in WEKA Data Mining Tool. The web crawler developed in this thesis classifies web pages according to their subjects while scanning the web pages.

ÖZET

Hiç süphe yok ki, son yüzyılımızın en iyi icatlarından bir tanesi İnternet'dir. Milyonlarca insan İnternet kullanıcısıdır. Bir bilgi aradıklarında çeşitli arama sitelerini ya da blog ismi verilen kişisel web sayfalarını ziyaret etmektedirler. Bu amaçla bir çok İnternet uygulaması geliştirilmiştir.

Son yirmi yılın gelişme kaydetmiş calışma alanlarından ikisi, arama motorları ve veri madenciliğidir. İnternetin gelişmesi, web kaynaklarının erişimine, aranıp bulunmasına olan ihtiyacı arttırmıştır. Bu kullanılan arama motorlarının sayısının artmasına ve arama motoru servis tiplerinin farklılaşmasına neden olmuştur. Daha akıllı arama motorları, kullanıcının aradığına kolay ulaşabilmesi için önemlidir.

Arama motorları kullandıkları robotlar ile web üzerinde bulunan kaynakları taramakta ve içeriklerini indeklemektedir. Bu indeksleri kullanarak, kullanıcılar istedikleri sayfalara erişebilmektedirler. Kullanılan indeks yapıları gelişmekte, bu indekslerde bir sayfadaki kelimeler yanında sayfanın hangi alanda olduğu konusunda da bilgi içermektedir. Konulara gore indekleme yapan arama motorları Kümeli (Clustered) Arama Motoru olarak adlandırılmaktadır. Bir sayfanın hangi konuda olduğunu belirlemek icin veri madenciliği sınıflama yöntemleri kullanılmaktadır.

Bu tez calışmasında, Java platformu kullanılarak bir ağ robotu ve sınıflandırma sistemi geliştirilmiştir. Sınıflandırma sisteminde eğitim kümesi olarak Açık Dizin Projesi(DMOZ) kullanılmıstır. Açık Dizin Projesinde konularına göre etiketlenen web sayfaları taranmış ve bu sınıflandırma öğrenme algoritmalarında kullanılmıştır. Sınıflandırma için, açık kaynaklı bir veri madenciliği yazılımı olan WEKA Sınıflandırma Kütüphaneleri kullanılmıştır. Bu tezde geliştirilmiş olan ağ robotu web sayfalarını tararken, sınıflandırma algoritmalarını kullanarak aynı zamanda bu sayfaların konularını belirleyebilmektedir.

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

DTC	Dogus Turtle Crawler
DTE	Dogus Turtle Engine
NLP	Natural Language Processing
MLP	Multi-Layer Perceptron
API	Application Programming Interface
GUI	Graphical User Interface
HTML	Hypertext Markup Language
WEBSPHINX	Website-Specific Processors for HTML Information Extraction
MVC	Model View Controller
SE	Search Engine
GUI	Graphical User Interface
HTTP	Hyper Text Transport Protocol
IT	Informations Technology
JDBC	Java Database Connectivity
MB	Megabyte
SQL	Structured Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WSDL	Web Service Description Language
XML	Extensible Markup Language
SEO	Search Engine Optimizations
DOC	Document
IR	Information Retrieval
FIFO	First In First Out

1 INTRODUCTION

Data and Information plays a big role for human life. From Sumerian time to today, writing and archiving history such as natural and cultural events and everything that belongs to our past can be learned from the records allow us to understand some information and sometimes help us to decide what we do. For instance; if Isaac Newton had not been wrote his studies, may be Neil Armstrong could not have been go to Moon. Thanks to his findings, some improvements have been continued and will continue in the future.

Humans have been structuring, organizing, and labeling information for centuries. Back in 660 B.C., an Assyrian king had his clay tablets organized by subject. In 330 B.C., the Alexandria Library housed a 120-scroll bibliography. In 1873, Melvil Dewey conceived the Dewey Decimal System as a tool to organize and provide access to the growing number of books. In modern times, most of us become familiar with the basics of information organization through our experiences with books and libraries (Morville, 2007).

The objective of many information systems is to extract information and/or knowledge from large collection of data stored in databases and other mediums. The meaning of word "data" is usually accepted as raw data that is stored as records in databases, document retrieval systems or world wide web. Information can be defined as a manipulated data which is shaped and become a meaningful in other words processed data which has used to make a decision etc.

Information and knowledge are generally used for overlapping concepts. Data is the lowest level of abstraction, information is the next level, and finally, knowledge is the highest level among all three.

According to some sources Knowledge is defined as 'actionable information'."Knowledge is defined by the Oxford English Dictionary (i) expertise, and skills acquired by a person through experience or education; the theoretical or prac curtical understanding of a subject, (ii) what is known in a particular field or in total; facts and information or (iii) awareness or familiarity gained by experience of a fact or situation. Philosophical debates in general start with Plato's formulation of knowledge as 'justified true belief' (Wikipedia 1, 2009).

Web search engines extracts useful index information from the raw document data stored in the World Wide Web. Data mining systems allow discovery of knowledge from raw data or information. The objective of this thesis to analyze the current commercial and open-source search engines and develop a web robot, indexing and classification system that extracts useful information and knowledge from the raw web data resources.

This thesis is organized as follows. The chapter II, analyzes current commercial search engines, architecture of search engines and new developments in search engines. Chapter III describes a web mining, web mining algorithms and tools. In addition, it introduces open-source data mining library WEKA. In Chapter IV, we describe a web search engine, called as DOGUS TURTLE ENGINE (DTC) DTC consists of a web crawler and search engine GUI. Web Crawler includes a text classification module that classifies web pages according to their subjects. In Chapter V, we present a results obtained from web crawler and text classification algorithms. The Last chapter contains conclusion and suggestions for the future work.

2 SEARCH ENGINES

Without any doubt; people cannot know all of the necessary the web sites or addresses they need on the internet. Search engines help them to find best pages that match their query. Last decade, GOOGLE have been become as the most popular web search engine on the internet world. Success of GOOGLE depends on its ranking algorithm that produce better searching results for a query then its competitors.

The process of gathering information about web pages is performed by an agent called a crawler, spider, or robot. The words “ant”, “automatic-indexer”, “bot” and “worm” are also used rarely instead of a crawler. The crawler looks at every pre-determined site on the web, and parsers each page. Each parsed page, terms(words) on each page and its link is stored in a database or a special information retrieval system.

A traditional search engine consists of three major parts: a crawler, an indexer, and a query processor. Crawlers traverse the WWW by following the hyperlinks to fetch web pages from the WWW for indexing (Hu, 2003).

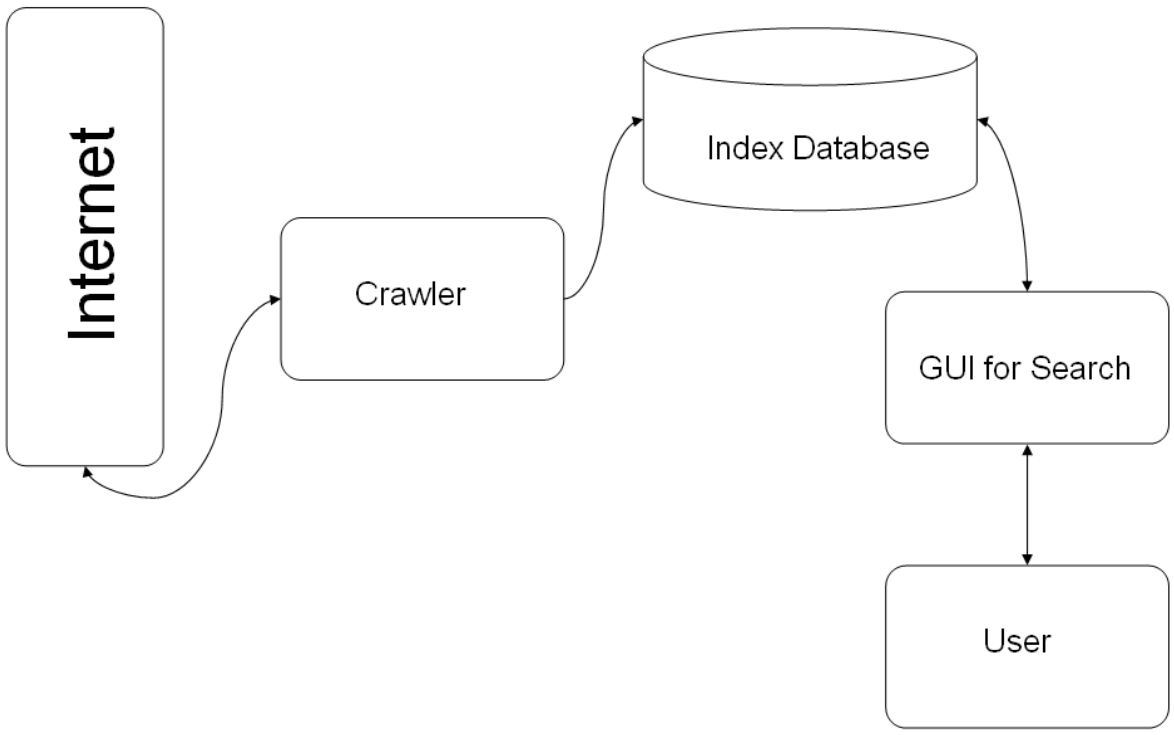


Figure 2.1 A Search Engine High Level Structure

In the Figure 2.1, a sample flow of data, among search engine, crawler and query processor is shown. Some of the well known and popular web search engines are GOOGLE, YAHOO, ALTAVISTA, Overture, HOTBOT, ABOUT, DOGPILE SPLUT, EXCITE, ALLTHEWEB, MIVA, MSN, LOOKSMART and AOL. Now content specific search engines are becoming a more popular. KIDSCLICK is one example of it and its web address is "http://www.kidsclick.org/". Another sample web search engine in this category is "http://www.babycrawler.com".

2.1 Classification of Search Engines

Search engines can be categorized (as Targeted, Primary and Secondary) depending on their the scope of contents. The diversity of search engines is increased day by day. In the following sections the next generation search engines are detailed.

2.1.1 Primary Search Engines

The primary search engine uses a robot to travel the Internet to retrieve documents; it has a computer program that will search these documents; and it covers a significant part of the World Wide Web (ENGINE).

Google can be categorized as primary search engine. Some index most or all sites on the Web.

2.1.2 Secondary Search Engines

Secondary search engines are targeted at smaller, more specific audiences, although the search engine's content itself is still general. They don't generate as much traffic as the primary search engines, but they're useful for regional and more narrowly focused searches. Examples of secondary search engines include Lycos, LookSmart, Miva, Ask.com, and Espotting. Secondary search engines, just like the primary ones, will vary in the way they rank search results. Many users of secondary search engines are users because they have some loyalty to that specific search engine. For example, many past AOL users who have moved on to broadband Internet service providers still use the AOL search engine whenever possible, because it's comfortable for them (Poremsky, 2004).

2.1.3 Targeted Search Engines

Targeted search engines—sometimes called topical search engines—are the most specific of them all. These search engines are very narrowly focused, usually to a general topic, like

medicine or branches of science, travel, sports, or some other topic. Examples of targeted search engines include CitySearch, Yahoo! Travel, and MusicSearch, and like other types of search engines, ranking criteria will vary from one to another (Poremsky, 2004).

2.2 Search Engines in the internet

2.2.1 Google

Without a doubt; Google is the best known search web site. This web site can be used a search purposes by the other applications or web sites. It allows to making full text search (Figure 2.2).

It is seen that, most of the web sites use Google as a search provider. For example, one of the web sites that uses the Google to make searches on site. Like Stanford University, many web sites use Google to power their own searches.

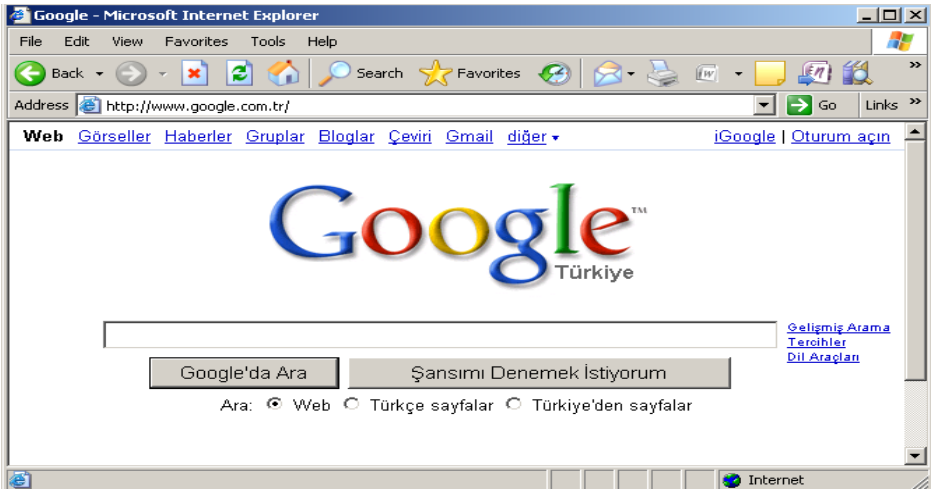


Figure2.2 Google Web Page (Google 1, 2009)

In Google, the web crawling (downloading of web pages) is done by several distributed crawlers. There is a URLserver that sends lists of URLs to be fetched to the crawlers. The web pages that are fetched are then sent to the storeserver. The storeserver then compresses and stores the web pages into a repository. Every web page has an associated ID number called a docID which is assigned whenever a new URL is parsed out of a web page. The indexing function is performed by the indexer and the sorter. The indexer performs a number of functions. It reads the repository, uncompresses the documents, and parses them. Each document is converted into a set of word occurrences called hits. The hits record the word, position in document, an approximation of font size, and capitalization. The indexer

distributes these hits into a set of "barrels", creating a partially sorted forward index. The indexer performs another important function. It parses out all the links in every web page and stores important information about them in an anchors file. This file contains enough information to determine where each link points from and to, and the text of the link (Figure 2.3).

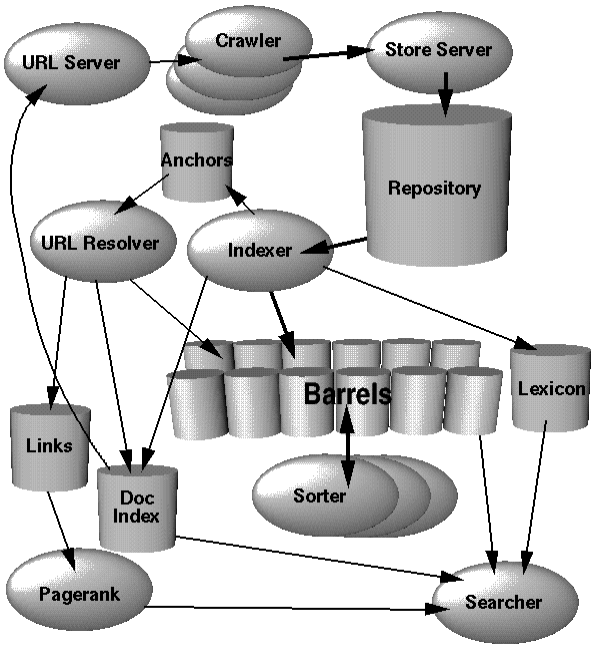


Figure2.3 Google High Level Design (Google 2, 2009)

Countless improvements are occurred in the web search area but undoubtedly; the brightest star of the last decade was Google. Thanks to Google and similar services, search economy is improved and countless innovations built and continued.

2.2.2 Yahoo and MSN

One of the most known web sites is Yahoo which is a portal, search engine, and online content provider. As it is known, Yahoo and Google use different indexing and listing strategies. Yahoo indexes only a site's main URL, title, and description, while Google builds full-text indexes of entire sites. MSN web site is one of the oldest portal, search engine, and online content provider.

2.2.3 RedZee

RedZee is a search engine and it works in a extremely different way than all the other search engines do. After doing search, it brings a snapshot of the web sites which related with the search-key.

2.2.4 MetaSearch Search Engine

Clusty is one of the meta-search engines for internet users. Clusty is a meta-search engine developed by Vivísimo which offers clusters of results. The main purpose of this search engine approach is to organize a large amount of search results into several sub categories.

A view of Clusty web site is seen in the Figure 2.4.

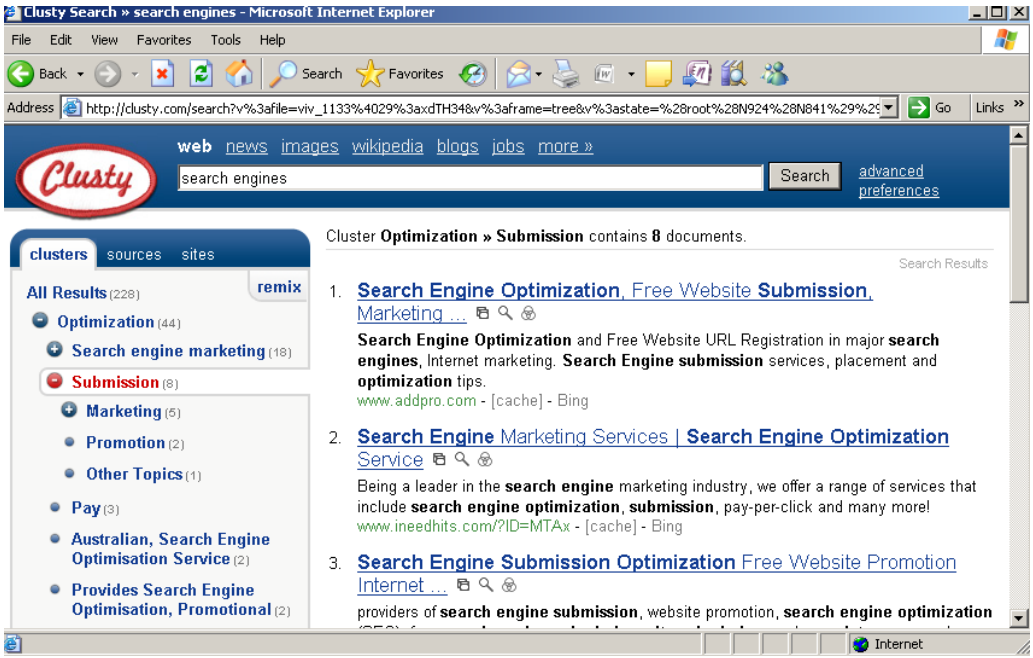


Figure2.4 Clusty Search Engine (Clusty, 2009)

2.2.5 Open Directory Project

One of the internet's largest directory system and it is in partnership with AOL. Google uses the DMOZ results as known. According to categories, sub links are prepared (Figure 2.5).

Since 1998, DMOZ have established and it is maintained by a passionate and volunteer editors. The Open Directory Project, also known as the ODP, is a searchable subject index of over four million sites. Like Yahoo, it's a searchable subject index. Unlike Yahoo, it contains no other properties—no news search, no mailing lisints, no nothin'. Furthermore, it's a volunteer effort, maintained by an army of over 60,000 editors managing over half a million

categories.Volunteers maintain the site, quality sometimes appears uneven; some categories are more active than others. Search is with plain keywords, with a Boolean default of AND (Ledford, 2007). The main categories of DMOZ are listed in the Table 2.1. Each category may also contain sub categories.

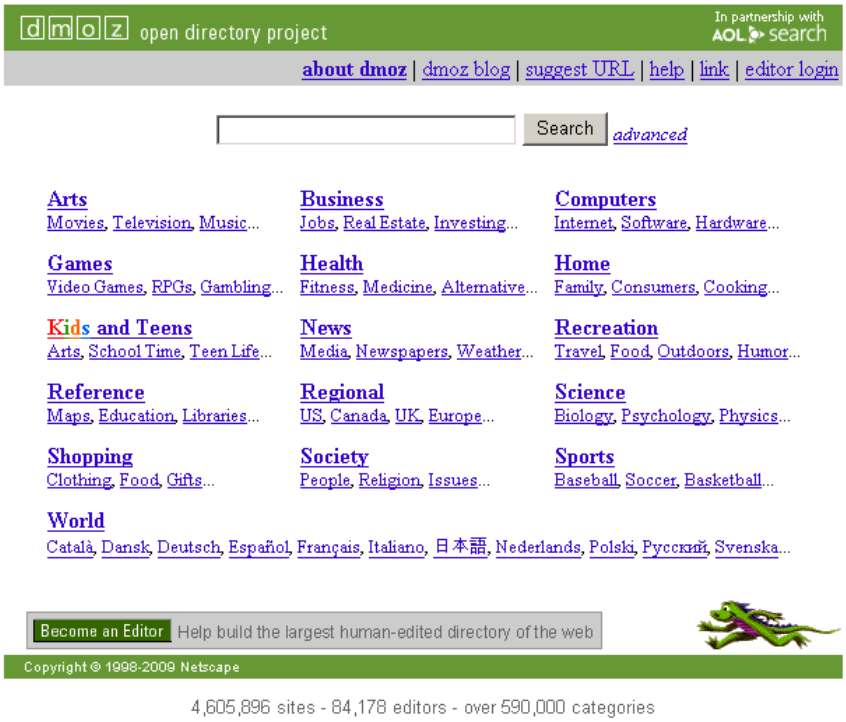


Figure2.5 DMOZ Web Site(Dmoz, 2009)

Table 2.1 DMOZ Main Categories

Category Name	Web Address
Business	http://www.dmoz.org/Business/
Computers	http://www.dmoz.org/Computers/
Health	http://www.dmoz.org/Health/
Science	http://www.dmoz.org/Science/
Sports	http://www.dmoz.org/Sports/
Arts	http://www.dmoz.org/Arts/
Games	http://www.dmoz.org/Games/
Home	http://www.dmoz.org/Home/
Kids and Teens	http://www.dmoz.org/Kids_and_Teens/

News	http://www.dmoz.org/News/
Recreation	http://www.dmoz.org/Recreation/
Reference	http://www.dmoz.org/Sports/
Regional	http://www.dmoz.org/Regional/
Shopping	http://www.dmoz.org/Shopping/
Society	http://www.dmoz.org/Society/

2.2.6 The Other Web Sites as Search Engine

Lycos and Excite are portal and search engine. AOL is Portal and search engine; for subscribers only Altavista is a search engine. Ask Jeevs is a search engine; includes children's search engine. Hotbot is a search engine; includes links to other search engines.

Looksmart is one of the topic directories. Dogpile, About, Splut, Gigablast, Overture, Miva, Netscape and Alltheweb sites are the other search web sites. And as last, CUIL is the web search engine which is published in 2008.

2.3 Architecture of Search Engines

A search engine can contain the following steps;

- Gathering Information (Retrieval)
- Information Extraction
- Querying-Answering

Each step mentioned above, sometimes is parted in two different processes. For Information gathering and information extraction, a web crawler is used and a web API used for providing querying-answering. The more information was given in the following sub sections about component issues of search engine works.

The diversity of engines is depends on the design and architectural differences among them. In this section, search engine and its properties were explained.

2.4 Search Engine Components

2.4.1 Web Crawlers

Crawler applications are used for collecting data. Crawler generally starts from a specific starting list. Each visited page is parsed according to chosen parser and the document is

indexed by indexer. If the visited page contains links, this links are added to crawler's queue in order to visit. Usually, First-In-First-Out (FIFO) process logic is used to visit pages in the used queue.

Crawlers have some common behavioral properties. A sample flow of data in the web crawlers are illustrated in the Figure 2.6.

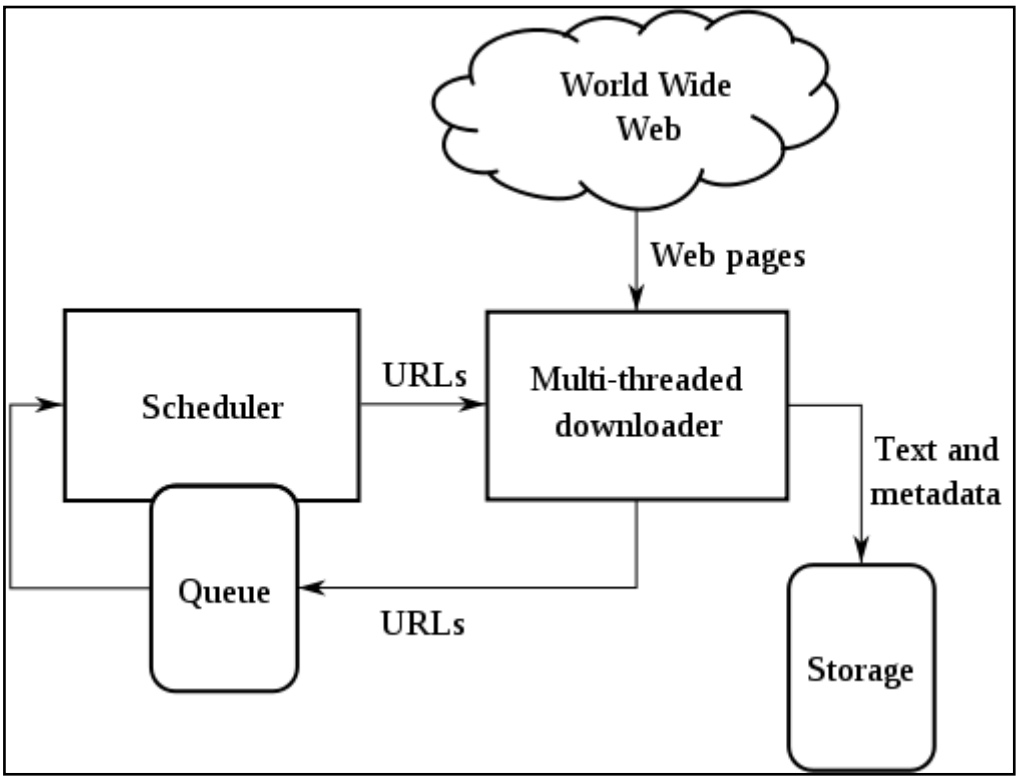


Figure2.6 High-Level Design of web crawler (Wikipedia 2, 2009)

In the internet a web page contains a lot of links which correspond to another web page. A sample illustration is shown in the Figure2.7. Crawlers receive page content and Crawler parses html page and receives other web pages for visiting according to working strategy. In the literature, when a relationship between links is defined, it means Web Graph Structure is defined for the pages.

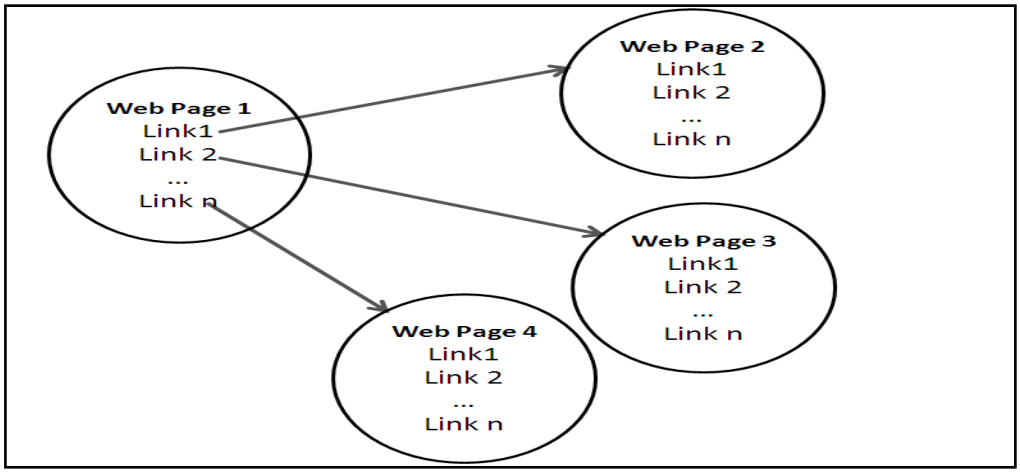


Figure 2.7 Link Structure of Web Pages

2.4.1.1 Policies of Web Crawlers

The behavior of a Web crawler is the outcome of a combination of policies: (Wikipedia 3, 2008)

- A selection policy
- A re-visit policy
- A politeness policy
- A parallelization policy

A Selection policy means a determination process of visited pages by crawler. The crawler cannot download all pages on the Web.

The crawler to carefully select the pages and to visit "important" pages first by prioritizing the URLs in the queue properly, so that the fraction of the Web that is visited is more meaningful (Arasu, 2001). But, general and simple method is starting visit pages according to a list and recursively visiting the links in the each visited page.

Usually, the crawler has downloaded a significant number of pages, it has to start revisiting the downloaded pages in order to detect changes and refresh the downloaded collections. Revisiting is required because web pages can be changed at very different rates. Before visiting a webpage, a crawler downloads the URL list, identifies the URLs that were modified since its last visit, and retrieves only the modified pages. This approach is very efficient but requires modifications to the server-side implementation. Most current web servers do not support this (Tan ea., 2007).

The speed of visiting page of web crawlers is very high. When a lot of page requests are sent to a visited web page, it causes a problem in the network traffic and the site moderators can block an access of a web crawler. Because of that reason, a crawler should not cause a network problem.

Parallelization is often necessary in order to download a large number of pages in a reasonable amount of time. It has an importance for coordination of distributed crawler architecture.

2.4.1.2 Popular Web Crawlers

In the Internet commercial and open source web crawlers are available. There are several hundred commercial web crawlers. AltaVista, Excite, Google, HotBot, Lycos and Northern Light are some of the most popular web crawlers (Umd Library, 2009).

The following crawlers are in the open source crawlers (Wikipedia 3, 2008):

- Aspseek is a crawler, indexer and a search engine written in C and licenced under the GPL.
- "www.arachnode.net" is a .NET web crawler written in C# using SQL 2005 and Lucene and is released under the GNU General Public License.
- DataparkSearch is a crawler and search engine released under the GNU General Public License.
- GNU Wget is a command-line-operated crawler written in C and released under the GPL..
- GRUB is an open source distributed search crawler that Wikia Search uses to crawl the web.
- Heritrix is the Internet Archive's archival-quality crawler, designed for archiving periodic snapshots of a large portion of the Web. It was written in Java.
- ht://Dig includes a Web crawler in its indexing engine.
- HTTrack which uses a crawler to create a mirror of a web site for off-line viewing. It is written in C and released under the GPL.
- ICDL Crawler is a cross-platform web crawler written in C++.
- mnoGoSearch is a crawler, indexer and a search engine written in C and licenced under the GPL.
- Nutch is a crawler written in Java and released under an Apache License.

- Pavuk is a command-line Web mirror tool with optional X11 GUI crawler and released under the GPL.
- YaCy is a free distributed search engine, built on principles of peer-to-peer networks and it is licensed under GPL.
- WebSPINX is composed of a java class library that implements multithreaded web page retrieval and HTML parsing and provides a GUI interface. In the following sections WebSPHINX is detailed.

2.4.2 Searcher Component

Searcher component is the interface between the client and the database of web crawler. In the literature, some engines can allow for complex strategy development with Boolean operators, phrase and proximity searching, and nesting. Some special searcher components search multiple crawler databases in the same time with a single interface.

Each search engine interprets the terms the user enters into the search box in different ways. Variable features which can affect your search results include (Umich, 2009).

Search criteria of search engines depends on their search policies. Most search engines allow user to use operators such as AND, OR, and NOT in order to create complex search statements. The terms may need to be entered in upper case. Many Search Criterias can be categorized as follow:

- Plain-English searches
- AND searches
- OR searches
- NOT searches
- NEAR searches
- Nested searches
- Full Text Search

If user enters two words, a search engine gets results the pages contain two entered words, we say that search engine supports AND search. At least of the words exists in the page, it means that search engine supports OR search.

Phrase Searching; Search engines will generally search for words as phrases when quotation marks are placed around the phrase. Some search engines may use a drop down menu which offers the option of searching the terms as a phrase.

Truncation; Some search engines will automatically truncate the terms you enter. This means that the search engine will not only search for the term exactly as you spelled it, but will also search on that term with alternate endings and as a plural. Some search engines will only search for variable endings on certain common words.

2.5 Future Search Engines

Using search engine is one of the most important activities for internet users, so search engines became one of the crucial applications on the Internet. Future search engines will have new features to that enable to make complex searches on many types of data.

2.5.1 Multimedia Search Engines

As it is known, images, sounds, videos are defined as multimedia data types. Videos and mp3 music, there is a rapidly increasing number of non-textual "documents" available to users and became a popular in the internet. Next generation search engines are provided this type of files for search. Many web sites are supported image search in the internet now.

Image search engines generally don't have the extensive special syntaxes that text search engines have, but they have other limiting options, such as image size. If you have an image search that fits within one image size, be sure to use the special image search restrictions (Ledford, 2007).

2.5.2 Content Specific Search Engines

Smart and content specific search engines will be a part of our life. Beside from smart search engines purpose specific search engines are published for many providers. For instance Google has five levels of special searches that help to find very specific topics.

The sample web links are listed:

- http://www.google.com/options/specialsearches.html
- http://froogle.google.com
- http://www.yahooligans.com

2.5.3 Next Generation Search Engines

As mentioned in the previous sections, the first generation of web search engines used text-only retrieval techniques. After WEB2.0, the search engines will provide determination of user purposes for searching.

Smart Search engines can be provided by understanding user behaviors. Next generation web search engines seem to be personal assistant of internet users by engaging with supporting question-answering and using Semantic Web technologies. The Figure 2.9 shows a page layout of Google's Smart Search Engine which is under development.

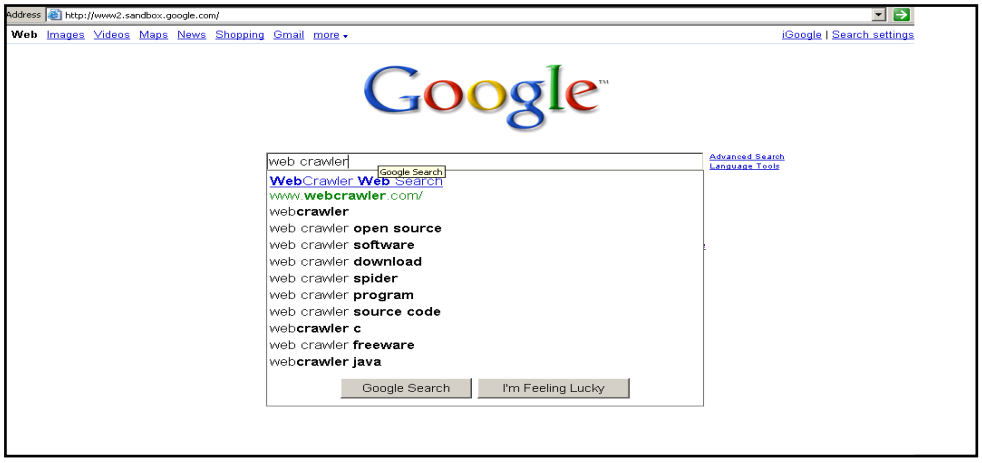


Figure 2.9 Smart Google (Google 3, 2009)

3 WEB MINING

A major topic of this thesis is web mining which is a sub research field of Data Mining. Data Mining can be thought as interdisciplinary in the science and research areas. Data always have thought as crucial for human being and future. Data mining methods have a common usage such as the medical explorations, financial estimations and fraud detection cases in the real life. Applying data mining methods to text document collection make easy to accessing data for instance, book indexes, digital library catalogues can be prepared by using these methods. According to usage purpose, descriptive and deterministic methods are selected and used.

Data mining can be defined as discovery of knowledge from the structured or non- structured data by using scientific techniques. Data mining can be defined as a process of finding patterns and relationships in data, preparing model and applying that composed model to new data. A sample knowledge discovery process is shown in the Figure 3.1.

In the literature, the following steps are applied for text mining: text preprocessing, term selection, applying data mining or information extraction methods. Applying data mining for text contains specific tasks such as Natural Language Processing (NLP) and information extraction.

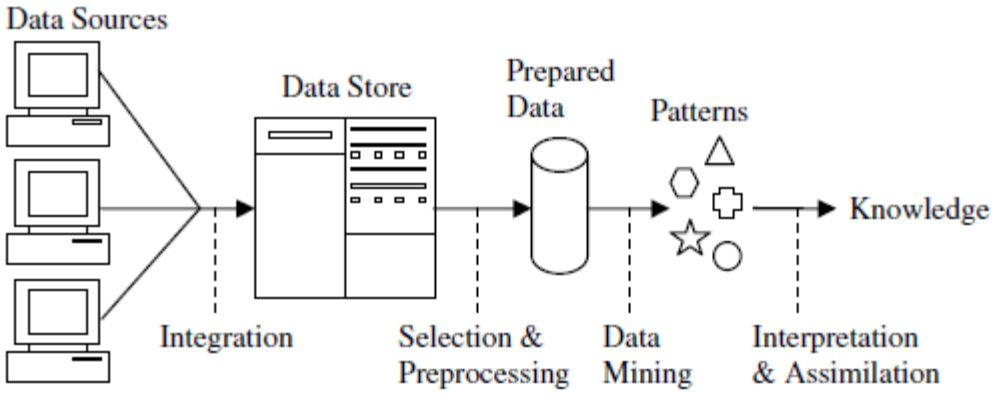


Figure 3.1 Knowledge Discovery Process (Bramer, 2007)

Classical applications in text mining come from the data mining community, like document clustering and document classification. For both the idea is to transform the text into a structured format based on term frequencies and subsequently apply standard data mining techniques. Typical applications in document clustering include grouping news articles or

information service documents; whereas text categorization methods are used in, e.g., e-mail filters and automatic labeling of documents in business libraries (Feinerer, 2008).

Web mining is the use of data mining techniques to automatically discover and extract information from Web documents/services (Tripaty, 2008)**.** Web mining aims to discover useful information or knowledge from the Web hyperlink structure, page content, and usage data (Liu, 2006). The field of web mining is very close to text mining. Web mining can be put into Text mining category but data in the web are different format when it compared with text data. Text data contains clauses. Web pages contain HTML format data. Web mining is can be defined as Knowledge Discovery from web.

Web mining tasks can be categorized into three types: Web structure mining, Web content

mining and Web usage mining (Feinerer, 2008).

- **Web structure mining**: Web structure mining discovers useful knowledge from hyperlinks (or links for short), which represent the structure of the Web. For example, from the links, we can discover important Web pages, which, incidentally, is a key technology used in search engines.
- **Web content mining:** Web content mining extracts or mines useful information or knowledge from Web page contents. For example, we can automatically classify and cluster Web pages according to their topics.
- **Web usage mining:** Web usage mining refers to the discovery of user access patterns from Web usage logs, which record every click made by each user. Web usage mining applies many data mining algorithms. One of the key issues in Web usage mining is the pre-processing of clickstream data in usage logs in order to produce the right data for mining.

In the literature, a lot of research and usages exist for web mining methods such as email spam filtering, clustered search engines. For processing data and deriving information and knowledge, text data should be prepared for manipulation. For the text mining, the main input is words or clauses as text in the document. In the following section, this process is detailed.

3.1.1 Web Mining Process

As a web mining technique, data is changed or converted according to data manipulation steps in order to processing for data mining. After applying data mining techniques, the results are evaluated. In this section, text preprocessing methods are detailed. In the following figure of "Web Usage Mining", the process steps are detailed for web mining.

Guo et all modeled their text categorization architecture is shown in the following Figure 3.2. As it is shown in the Figure, the first step in web mining is data preprocessing.

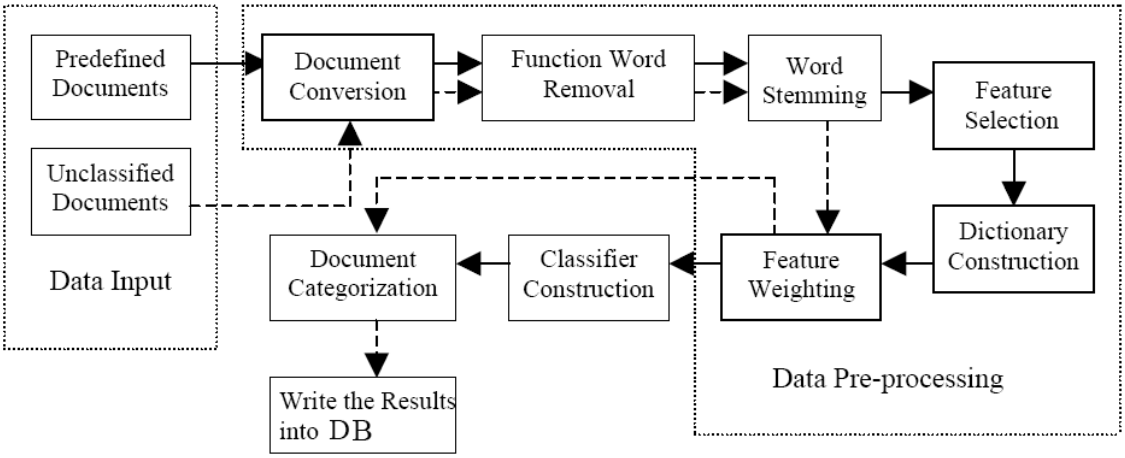


Figure 3.2 The General Architecture of Text Categorization (Guo, 2004)

3.1.1.1 Preprocessing

Data Format of web documents and content of text data (in the web or documents such as reports, newspapers) can contain many stop-words (such as is, are, the, and etc.) or may be formatted such as html, xml, etc. For that reason, preprocessing methods is used for cleaning up and structuring the text for further analysis, is a crucial part of text mining studies.

3.1.1.2 Filtering

Text data can be a lot of formats which are mentioned parent section. Some terms in the text do not have any important meaning content such as common words like "a", "is", "about", "for" for English. These terms are called as stop words. Text data are filtered by removing the stop words and whitespace characters. Web data is different from text, so HTML tags should be removed from the web content data.

3.1.1.3 Stemming

In the literature, Stemming is a technique which aims to extract common suffixes of words. Thus, words which are literally different but have a common stem, may be abstracted by their common stem. The underlying goal when using a stemming technique is to improve recall, at the possible expense of precision loss. A well known technique for stemming text is Porter's algorithm, which is based on a set of rules extracted from the English language (Nascimento, 2008).

Snowball, in which stemmers can be exactly defined, and from which fast stemmer programs in ANSI C or Java can be generated. A range of stemmers is presented in parallel algorithmic and Snowball form, including the original Porter stemmer for English.

Snowball is a string processing language which is used for creating stemming algorithms. Snowball source code is covered by the BSD License. Weka application contains Snowball stemmer. The web address of application is in the following link: "http://snowball.tartarus.org/texts/introduction.html"

3.1.1.4 Applying Web Mining Algorithms

General problem of Text mining is defined as assigning a pre-defined category or categories to a document. In the literature, text classification is used to grouping text document(s) to pre-defined sub categories, an example would be to automatically label each incoming news story with a topic like "sports", "Business", or "Computer" (Hotho, 2005). Many machine learning methods are developed to solve this problem.

For text classification (Lam, Ruiz, & Srinivasan, 1999), an abundance of machine learning algorithms (Sebastiani, 2002; Yang, 1999) such as k-nearest neighbor (Han, Karypis, & Kumar, 2002), naive Bayesian (McCallum & Nigam, 1998), neural networks (Ng, Goh, & Low, 1997), decision trees (Lewis & Ringuette,1994), and support vector machines (Sun, Lim, & Ng, 2002) are used in the literature. In Web page classification (Kan, 2004), due to its performance and quality, naive Bayesian classifier is usually preferred. A number of machine learning tools such as Weka (Witten & Frank, 2005), Grid WEKA (Khoussainov, Zuo, & Kushmerick, 2004), and the Harbinger machine learning toolkit (Cambazoglu & Aykanat, 2005) are readily available for use in text classification (Cambazoglu, 2007).

Some researchers have demonstrated that the classification process greatly improves the efficiency of information retrieval. One group of the researchers compared the results of

automatic categorization and manual categorization of documents in Spanish. They concluded that the results obtained from manual classification did not differ greatly from that of automatic categorization mechanism. One group of the researchers performed automatic web classification by link and context analysis (Metikurke, 2006).

Web page classification algorithms have two important phases which are "Training" and "Categorization of text". Training section is the preparing models of predefined categories for the text. Categorization phase is the labeling text documents according to trained data. When text preprocessing methods are applied to the input web data, the input data for training are ready. A training model is composed by implementing machine learning techniques on the text data. After Trained Model is prepared, the new text inputs can be categorized according to training model. A sample flow is shown in the Figure 3.3.

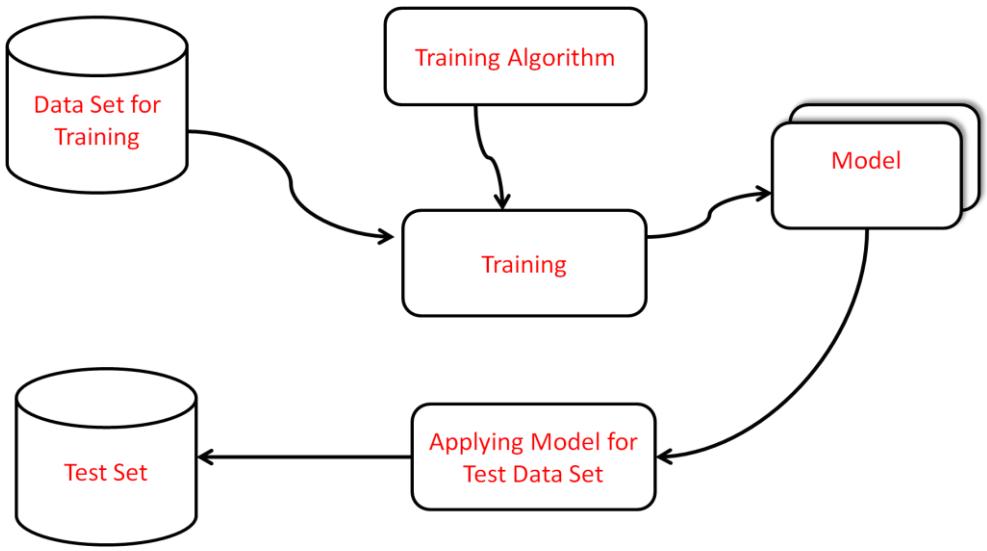


Figure 3.3 A Sample of Classification (Data Mining, 2009)

3.1.2 The Used Text Mining Algorithms

In this thesis, we developed a crawler called DOGUS Turtle Crawler (DTC) and applied two text categorization(classification) algorithms. DTC Crawler uses a WEKA java library for text classification. NaiveBayes and MultilayerPerceptron classes of WEKA library are used for classification. In addition InfoGainFilter class of WEKA is used for term reduction. In this section, NaiveBayes. MultilayerPerceptron and InfoGainFilter are explained.

3.1.2.1 Naïve Bayes

Naïve Bayes is a statistical classification method which uses Bayes Theorem. A classification problem is defined as finding the class with maximum probability for given data set. Such probability is seen as the posterior probability of the class given the data, and is usually computed using the Bayes theorem, as shown in the Formula 3.1, where C is any of the possible class values, X is a vector of $N_{feature}$ attribute values, while $P(C)$ and $P(X|C)$ are the prior probability of the class and the conditional probability of the attribute values given the class, respectively. Usually Bayesian classifiers maximize $P(X|C)P(C)$, which is proportional to $P(C|X)$, being $P(X)$ constant given a dataset(N.Bayes, 2009).

$$P(C|X) = \frac{P(X|C)\,P(C)}{P(X)}$$

Figure 3.4 The Bayes Theorem

3.1.2.2 Multi Layer Perceptron

In the literature, Multi-Layer Perceptron is also known as "The Multilayered Back-Propagation Association Networks".

A popular Artificial Neural Network classifier is the Multi-Layer Perceptron (MLP) architecture using the Backpropagation algorithm. A Description of the algorithm is given (Multi-Layer Perceptron, 2009).

- In overview, a MLP is composed of layers of processing units that are interconnected through weighted connections.
    - The first layer consists of the input vector
    - The last layer consists of the output vector representing the output class.
    - Intermediate layers called `hidden` layers receive the entire input pattern that is modified by the passage through the weighted connections. The hidden layer provides the internal representation of neural pathways.

- The network is trained using backpropagation with three major phases.
    - First phase: an input vector is presented to the network which leads via the forward pass to the activation of the network as a whole. This generates a difference (error) between the output of the network and the desired output.
    - Second phase: compute the error factor (signal) for the output unit and propagate this factor successively back through the network (error backward pass).
    - Third phase: compute the changes for the connection weights by feeding the summed squared errors from the output layer back through the hidden layers to the input layer.
- Continue this process until the connection weights in the network have been adjusted so that the network output has converged, to an acceptable level, with the desired output.
- Assign "unseen" or new data
    - The trained network is then given the new data and processing and flow of information through the activated network should lead to the assignment of the input data to the output class.

3.1.2.3 Information Gain Filter

Information gain is attribute selection measure is based on pioneering work by Claude Shannon on information theory, which studied the value or "information content" of messages. This method enables the reduction of number of attributes that will be used in a classification algorithm. The reduction of terms (words, keywords) provides description of documents with less number of terms.

3.1.3 Text Mining Tools

Many universities and big companies are researching text mining techniques and developing applications in text mining. A number of text mining tools and products developed. Figure 3.5 gives an overview of text mining products and available features.

Product	Preprocess	Associate	Cluster	Summarize	Categorize	API
Commercial						
Clearforest	✓	✓	✓	✓		
Copernic **Summarizer**	✓			✓		
dtSearch	✓	✓		✓		
Insightful **Infact**	✓	✓	✓	✓	✓	✓
Inxight	✓	✓	✓	✓	✓	✓
SPSS **Clementine**	✓	✓	✓	✓	✓	
SAS **Text Miner**	✓	✓	✓	✓	✓	
TEMIS	✓	✓	✓	✓	✓	
WordStat	✓	✓	✓	✓	✓	
Open Source						
GATE	✓	✓	✓	✓	✓	✓
RapidMiner	✓	✓	✓	✓	✓	✓
Weka/KEA	✓	✓	✓	✓	✓	✓
R/tm	✓	✓	✓	✓	✓	✓

Figure 3.5 Overview of Text Mining Products and Available Features (Feinerer, 2008)

Commercial products include Clearforest, a text-driven business intelligence solution, Copernic Summarizer, a summarizing software extracting key concepts and relevant sentences, dtSearch, a document search tool, Insightful Infact, a search and analysis text mining tool, Inxight, an integrated suite of tools for search, extraction, and analysis of text, SPSS Clementine, a data and text mining workbench, SAS Text Miner, a suite of tools for knowledge discovery and knowledge extraction in texts, TEMIS, a tool set for text extraction, text clustering, and text categorization, and WordStat, a product for computer assisted text analysis. R/tm is the another framework for text mining by Feinerer, Hornik and Meyer (Feinerer, 2008)

PASW is a program used for statistical analysis. Before 2009 it was called SPSS, but in 2009 it was re-branded as PASW (Predictive Analytics SoftWare). The company announced July 28, 2009 that it was being acquired by IBM for US$1.2 billion (Wikipedia 4, 2009).

RapidMiner is a platform for machine learning and data mining experiments. The initial version has been developed by the University of Dortmund since 2001. It is public under a GNU license, and has been hosted by SourceForge.

According to sources, RapidMiner is well suited for analyzing data generated by high-throughput instruments, e.g., genotyping, proteomics, and mass spectrometry (Stanford, 2009).

Cluster property shows the tool has a property for clustering of similar texts into the same groups. Summarize means the tool has summarization of important terms in a text. Categorize shows the tool can classify texts into predefined categories. API means the application has a graphical user interface.

Oracle Data Mining, Oracle provides a data mining tool named as Oracle Data Mining (ODM). Business domain knowledge and knowledge of the available data are the most important factors in the process for data mining. Oracle Data Mining automates the mechanics of building, testing, and applying a model. Tool can be downloaded in the official web site of Oracle.

In addition to the tools given above, there are also other companies that provide commercial text mining programs or tools (Wikipedia 5, 2009):

- AeroText which is provides a suite of text mining applications for content analysis.
- AlchemyAPI is SaaS-based text mining platform that supports 6+ languages. It includes named entity extraction, keyword extraction, document categorization, etc.
- Autonomy which is suite of text mining, clustering and categorization solutions for a variety of industries.
- Fair Isaac is leading provider of decision management solutions powered by advanced analytics.
- LanguageWare contains Text Analysis libraries and customization tooling from IBM
- Linguamatics is the text mining software for getting meaningful information from unstructured data

3.1.4 WEKA

WEKA is an open source Data Mining software package written in Java and available from www.cs.waikato.ac.nz/~ml/weka/index.html (web site of University of Waikato). It is developed at the University of Waikato, New Zealand. The project started when the authors needed to apply machine learning techniques on an agricultural problem. The project is developed and distributed under the GPL license and has a sub domain on the SOURCEFORGE portal (Dimov, 2009).

The WEKA system provides a rich set of powerful Machine Learning algorithms for Data Mining tasks, some not found in commercial data mining systems. These include basic

statistics and visualization tools, as well as tools for pre-processing, classification, and clustering, all available through an easy to use graphical user interface (Markov, 2006).

WEKA API allows to data preprocessing, processing and clustering for datasets to users. For text categorization, DTC Crawler uses the WEKA libraries and classification algorithms.

For information gain, InfoGainFilter is used and for classification two methods are implemented into DTC. In the Figure 3.6, the Classifier Section of Explorer window in the WEKA is shown.

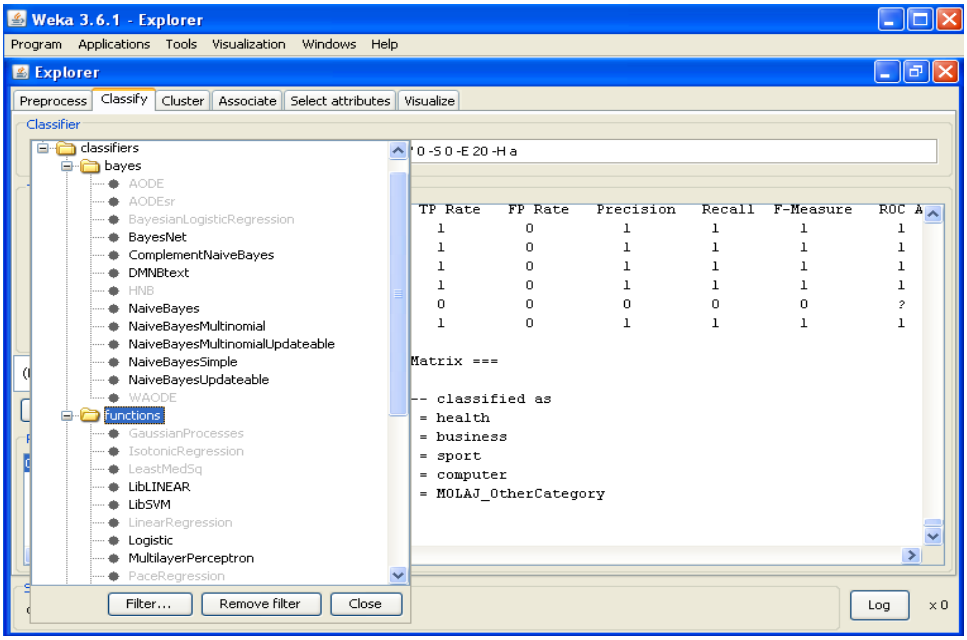


Figure 3.6 Sample View of WEKA API

4 DOGUS TURTLE ENGINE

In this thesis, we developed a web crawler called as DOGUS TURTLE Engine (DTE). DTE application is designed for collecting pages and indexing the words by starting with given determined initial pages. We developed a web crawler component and a search component. Their API enables users to develop search systems easily.

The Crawler is a multithreaded JAVA standalone application and it is platform independent so that it can run any machine that supports JAVA. The search component is a web application which runs on the Oracle (Bea) Weblogic Server.

The more details are given in the following sub chapters.

A General architecture of DTE is shown in the Figure 4.1.

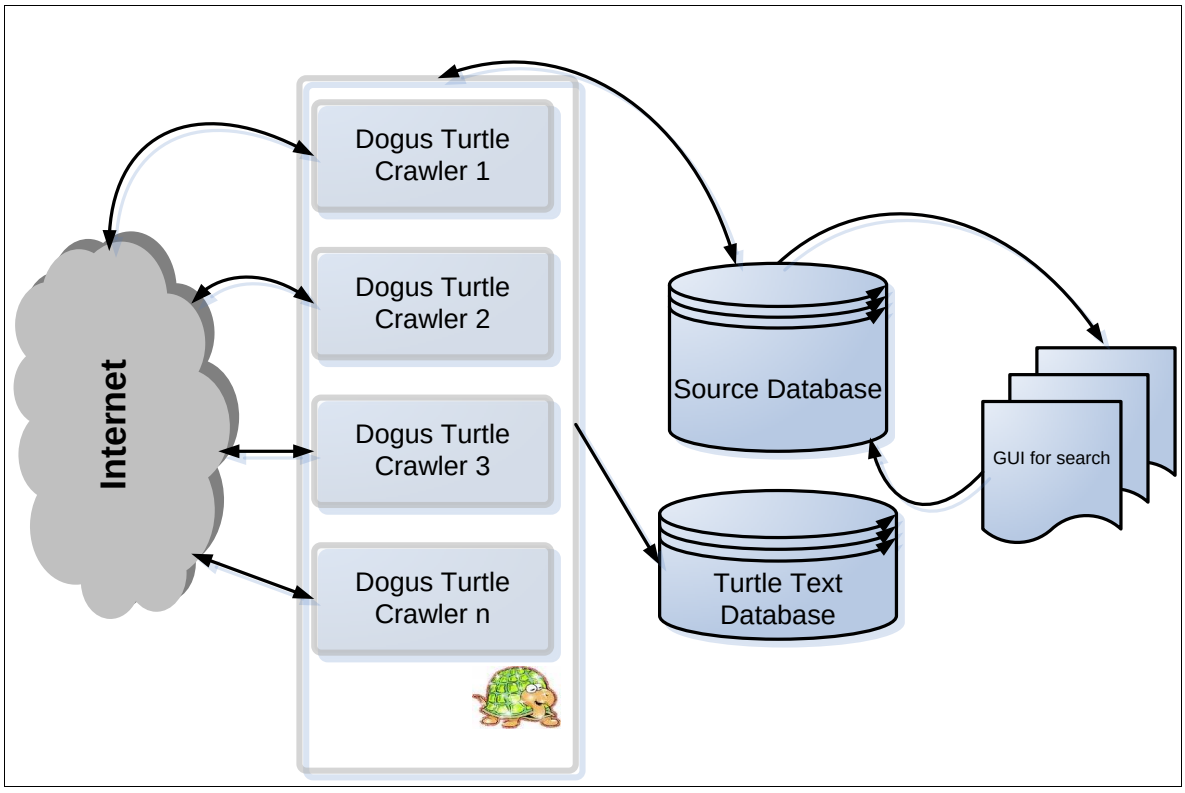


Figure 4.1 DTE Model Architecture

4.1 Java Multithreaded Crawler

DTE is a standalone multithreaded java application. According to startup links, DTE visits pages and collects information about the text content and links, saves them into a database. Search component of the DTE allows search of documents in database.

The engine runs according to the following startup parameters.

- Depth: gives the number of levels of links that can be visited during the search process.
- Thread Number: gives the number of parallel threads that crawls web in parallel.
- classifyEnabled: this parameter enables classification of documents into categories using classification algorithm. To run this feature, classification algorithm should be trained using DMOZ data set.
- stemEnabled: this parameters enables the stemming of words. This parameter is taken into consideration only classifyEnabled is set as true.

The starting URL information is got from the DME_LINK_STARTUP table in the database. Every visited page is updated in this table and LINK_STATUS is set from 1 to 0. Each visited page is parsed and the links in the page are populated into DME_DOCUMENT_LINKS table. For each link a new DOC_ID is generated and this links are added to sub link queue for each thread. The Depth of the link controls whether the crawler visit the link or not. In the Figure 4.2, a flow diagram is detailed. A crawling strategy is FIFO for DTC.

When a web crawler has a content of page, the stop words are cleaned from the content and the remained words are populated into database table sequentially. In the following sections, some UML diagrams are given. This diagram is generated by using Borland JDeveloper.

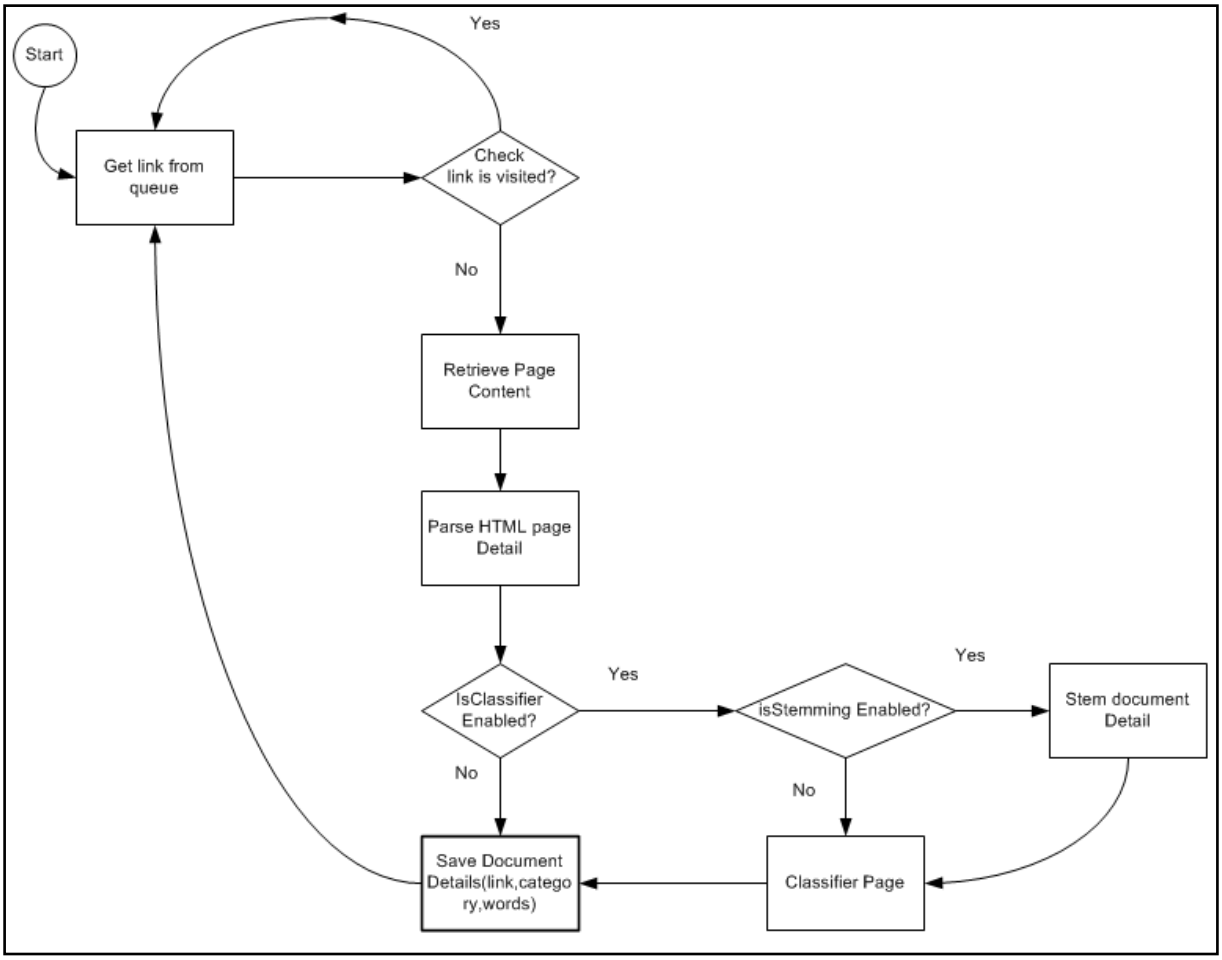


Figure 4.2 A DTC Page Visit Flow

4.1.1 Model of Crawler

There are three important classes in DTC: WinMain, Crawler and StopWordRemover. In this section each of these classes is explained below. In addition a Java Documentation of all the classes is given in the appendix section.

4.1.1.1 Main Class of Dogus Turtle Crawler

WinMain is the main class of crawler. This class enables the execution of DTC Crawler in multiple threads. This class first reads the startup parameters from command line and parses them. Startup parameters specify number of threads, depth, classification and stemming flags. After reading startup parameters, this class starts the execution of threads in parallel. Each thread retrieves a URL from DME_LINKS_STARTUP table and then retrieves the page specified by the URL. After that, it parses the page and stores the retrieved an information into DME_DOCUMENTS, DME_DOCUMENT_LINKS, DME_WORD_DETAIL tables. The object model of WinMain class is shown in the Figure 4.3.

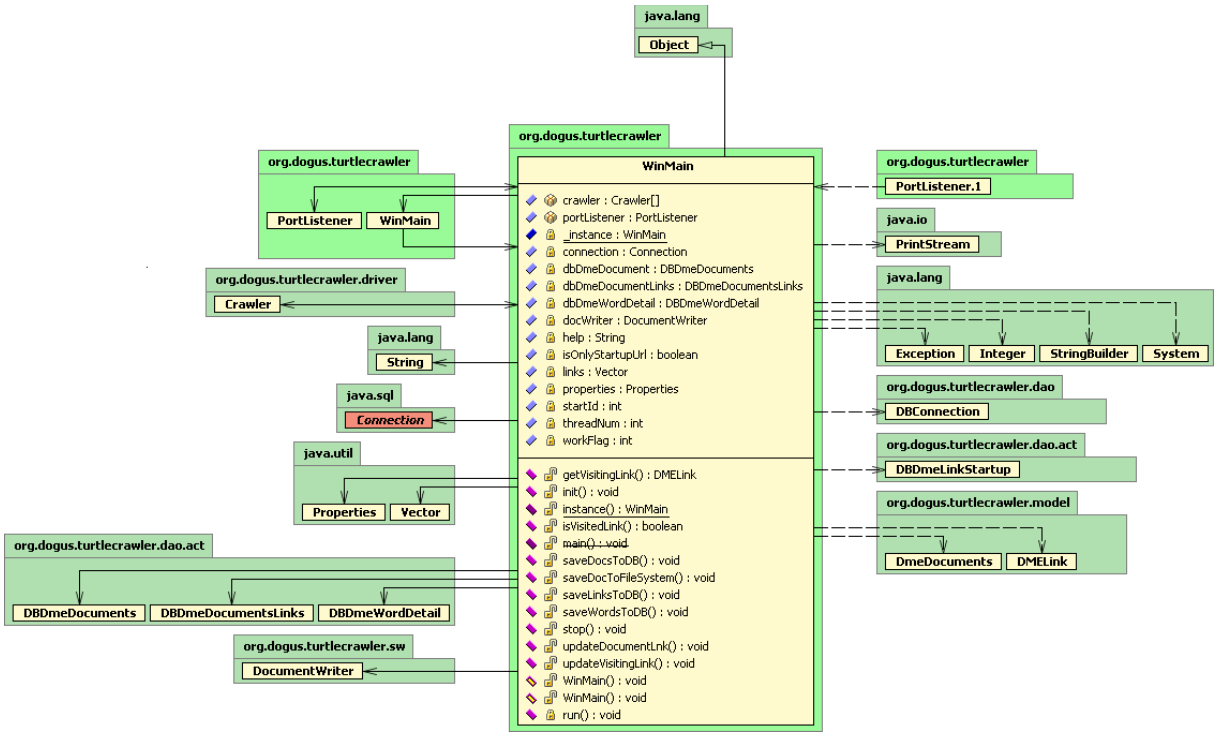


Figure 4.3 WinMain Java Model Diagram

4.1.1.2 Crawler Class of Dogus Turtle Crawler

One of the important classes is Crawler. It contains html parser class which is imported from WEBSPINX library. Given a web page, the HTML Parser returns an object instance which stores terms and links contained in that web page.

This class also controls the depth of the web page links that will be visited. A Java model of Crawler class is shown in the figure Figure 4.4.

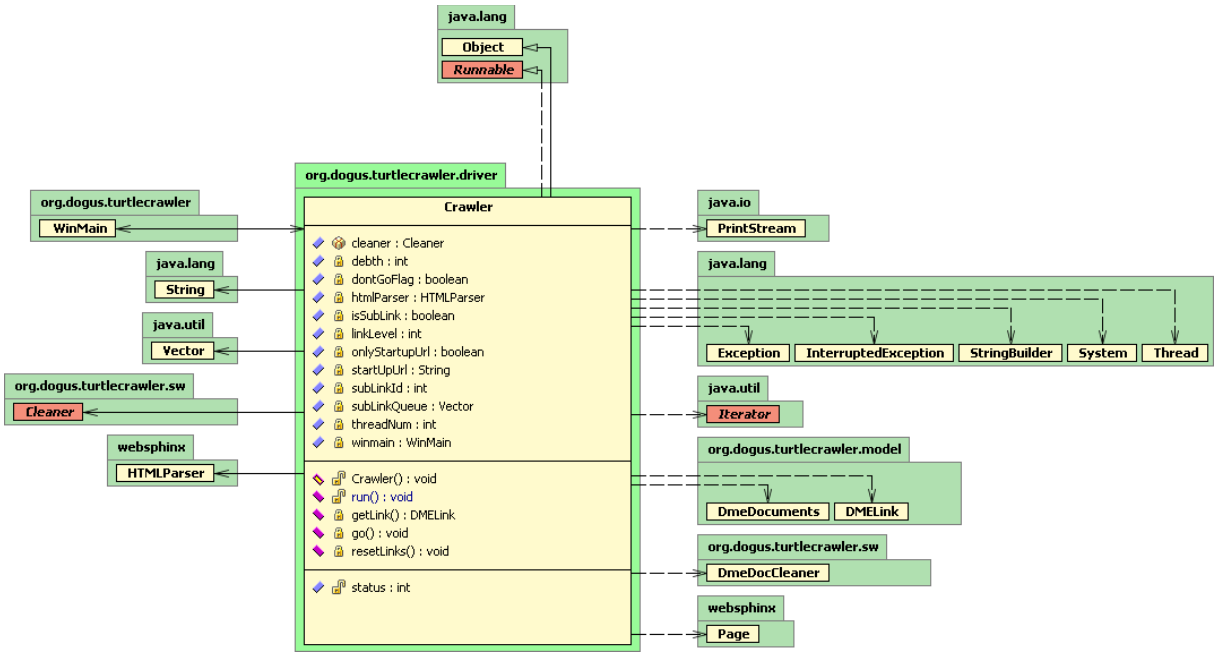


Figure 4.4 Crawler Class of DTC

4.1.1.3 StopWordRemover Class of Dogus Turtle Crawler

The StopWordRemover Class is used to clean stop words in the visited web page. Java model diagram is shown in the Figure 4.5. The list of stop-word will be taken from "http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/a11-smart-stop-list/english.stop" web site and the lists of values are enhanced. The all of the words will be detailed in the appendix section.

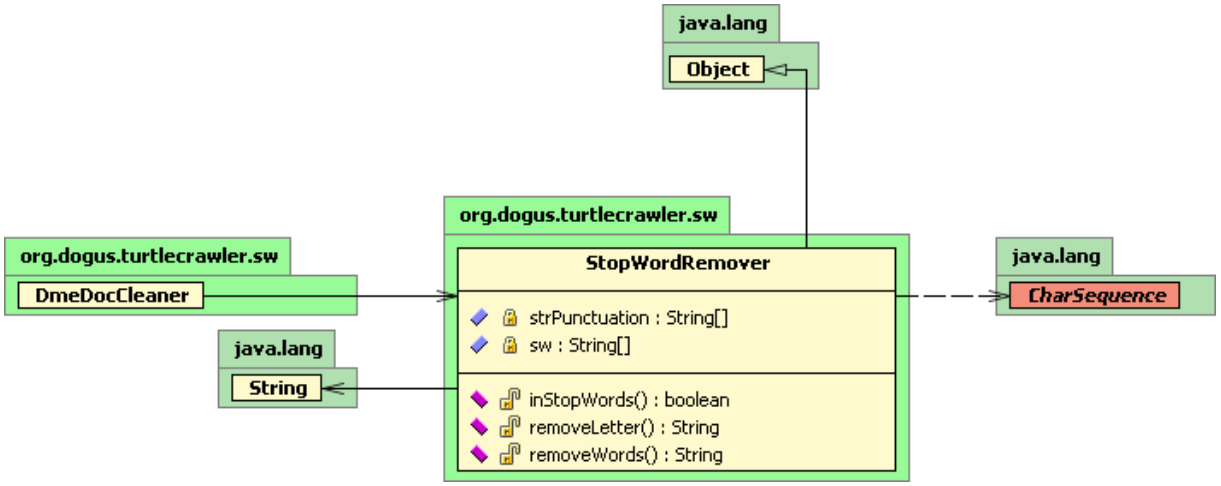


Figure 4.5 StopWordRemover Class of DTC

4.1.2 Text Classification While Crawling

DTC Crawler can perform text classification (categorization) while crawling as shown in the Figure 4.6. For this purpose, DTC crawler contains a categorization java library which is prepared to classify documents. This library is named as DOGUS Categorization library. Before running this library all the documents in DMOZ Open Directory are retrieved and stored in database tables which start with "DME_DMOZ". This information is used as training data.

DTC Crawler classification module uses WEKA's classifiers. The version of WEKA is 3.6.1 is used for this research. First, the training data contained database tables is used to train classifiers. When classifier flag is enabled in DTC Crawler, DTC Crawler uses one of the WEKA's specified classifiers to categorize each web page it retrieved.

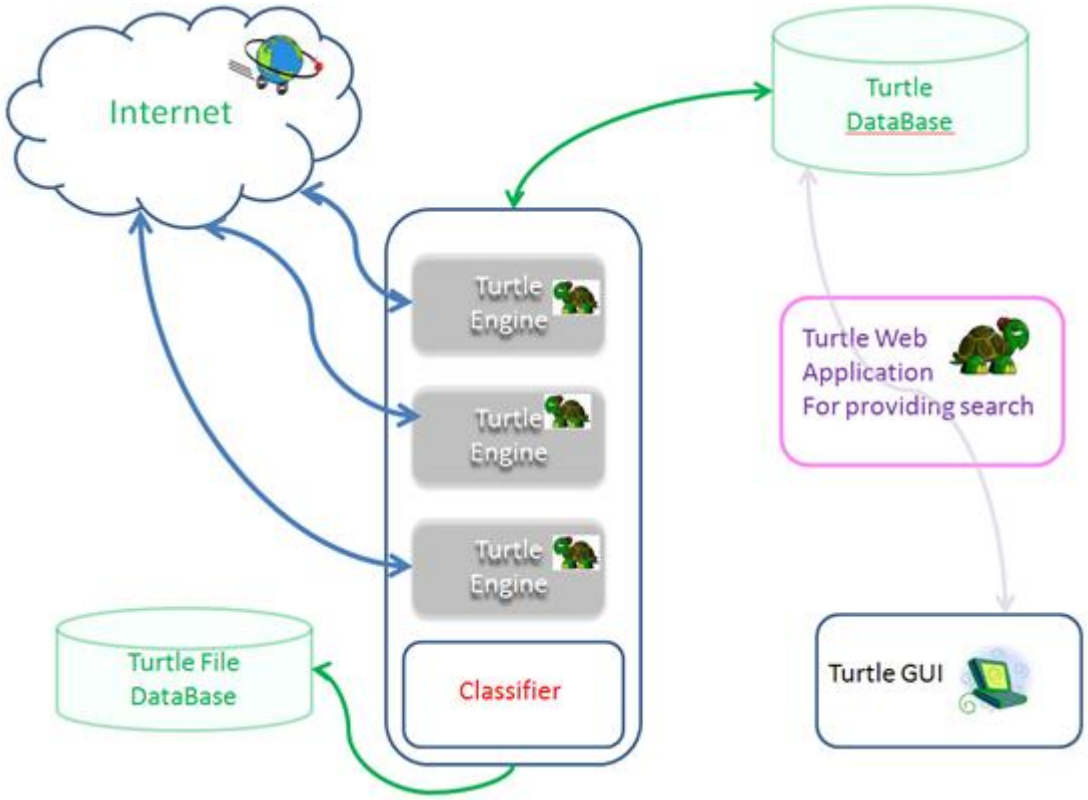


Figure 4.6 Classifier Enabled DOGUS Turtle Engine

4.1.3 Database Model of DTC Crawler

DTC Crawler stores the information about the visited web pages in an ORACLE database. The Database Model of DOGUS Turtle Engine is explained in the following sub sections. When the DTC Crawler engine visits a page; it collects information about links and words in a page. After parsing a page, each link in a page is added to the visited list table (DME_DOCUMENT_LINKS) and also is inserted into visiting queue so that the DTC Crawler will retrieve documents in those links depending on the given depth parameter.

4.1.3.1 Table for Starting Links

The DME_LINK_STARTUP Table is used to define initial startup links that will be visited by the DTC Crawler. The startup links should be provided by the user. This table stores URL of the link and link status. The link status field is set to "0" whenever pages are visited in that link (Table 4.1).

Table 4.1 The DME_LINK_STARTUP Table

COLUMN NAME	DATA TYPE	DESCRIPTION
LINK_ID	NUMBER	Link sequence number
LINK_URL	VARCHAR2(500)	Web address of page for visiting
LINK_STATUS	INTEGER	1:will be visited, 0: was visited

4.1.3.2 Documents Table

The DME_DOCUMENTS table stores document ID and URL of all the pages visited by DTC Crawler. This table contains a unique document ID each different page (Table 4.2).

Table 4.2 The DME_DOCUMENTS Table

COLUMN NAME	DATA TYPE	DESCRIPTION
DOC_ID	NUMBER	Primary Key
URL	VARCHAR2(500)	Web page URL information

4.1.3.3 Document Sub Links Table

The DME_DOCUMENT_LINKS table is used to store the links contained in each web page. The DOC_ID field corresponds to the DOC_ID in the DME_DOCUMENTS table. If there are several links in a web page, this table will store a record for each link with the same DOC_ID (Table 4.3).

Table 4.3 The DME_DOCUMET_LINKS Table

COLUMN NAME	DATA TYPE	DESCRIPTION
DOC_ID	NUMBER	Foreign Key
URL	VARCHAR2(500)	Web page URL information

4.1.3.4 Word Detail Table

The DME_WORD_DETAIL table stores words (terms) contained in each web page. This tables has three fields DOC_ID, WORD and TERMNO (Table 4.4). The TERMNO field stores the order of words stored in a document.

Table 4.4 The DME_WORD_DETAIL Table

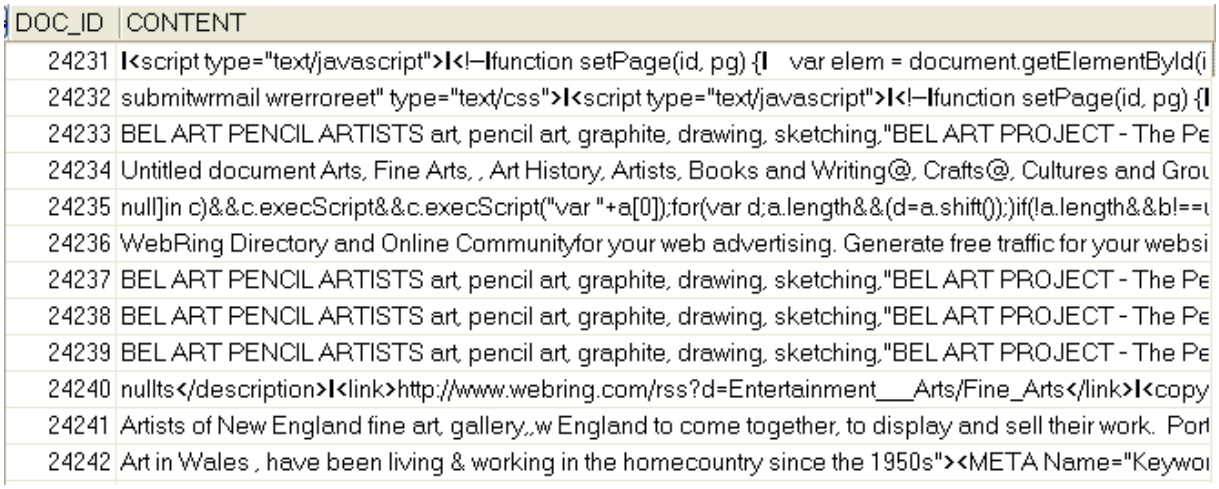
COLUMN NAME	DATA TYPE	DESCRIPTION
DOC_ID	NUMBER	Foreign Key
WORD	VARCHAR2(150)	Word in the visited page
TERMNO	NUMBER	Word sequence number

4.1.3.5 Document Content Table

The DME_DOCUMENT_CONTENT table stores the head section of a HTML document. If a web page does not have head section, it stores the first 500 words of the document. The data in this table is used in the Search results interface to show a part of each web page found after a search query (Table 4.5). An instance of this table is shown in the Figure 4.7.

Table 4.5 The DME_DOCUMENT_CONTENT Table

COLUMN NAME	DATA TYPE	DESCRIPTION
DOC_ID	NUMBER	PK
CONTENT	VARCHAR2(500)	Document Content



DOC_ID	CONTENT
24231	I<script type="text/javascript">I<!–Ifunction setPage(id, pg) {I var elem = document.getElementById(i
24232	submitwrmail wrerroreet" type="text/css">I<script type="text/javascript">I<!–Ifunction setPage(id, pg) {I
24233	BEL ART PENCIL ARTISTS art, pencil art, graphite, drawing, sketching,"BEL ART PROJECT - The Pe
24234	Untitled document Arts, Fine Arts, , Art History, Artists, Books and Writing@, Crafts@, Cultures and Grou
24235	null]in c)&&c.execScript&&c.execScript("var "+a[0]);for(var d;a.length&&(d=a.shift());)if(!a.length&&b!==u
24236	WebRing Directory and Online Communityfor your web advertising. Generate free traffic for your websi
24237	BEL ART PENCIL ARTISTS art, pencil art, graphite, drawing, sketching,"BEL ART PROJECT - The Pe
24238	BEL ART PENCIL ARTISTS art, pencil art, graphite, drawing, sketching,"BEL ART PROJECT - The Pe
24239	BEL ART PENCIL ARTISTS art, pencil art, graphite, drawing, sketching,"BEL ART PROJECT - The Pe
24240	nullts</description>I<link>http://www.webring.com/rss?d=Entertainment___Arts/Fine_Arts</link>I<copy
24241	Artists of New England fine art, gallery,,w England to come together, to display and sell their work. Port
24242	Art in Wales , have been living & working in the homecountry since the 1950s"><META Name="Keywo

Figure 4.7 The Document Content of Web Pages visited by DTC

4.1.4 About WebSPHINX

WebSPHINX is a crawler developed in Carnique Mellon University (Sphinx, 2008). The HTML parser of the WebSPHINX crawler is used to parse pages in this study.

WebSPHINX (Website-Specific Processors for HTML INformation eXtraction) is a Java class library and interactive development environment for web crawlers. WebSPHINX is composed of two parts: the Crawler Workbench and the WebSPHINX class library.The Crawler Workbench is a GUI that allows configuring and controlling a customizable web crawler.

The WebSPHINX class library provides support for writing web crawlers in Java. The class library offers a number of features (Sphinx, 2008):

- Multithreaded Web page retrieval in a simple application framework
- An object model that explicitly represents pages and links
- Support for reusable page content classifiers

- Tolerant HTML parsing
- Support for the robot exclusion standard
- Pattern matching, including regular expressions, Unix shell wildcards, and HTML tag expressions. Regular expressions are provided by the Apache jakarta-regexp regular expression library.
- Common HTML transformations , such as concatenating pages , saving pages to disk, and renaming links

4.2 Search Interface

The web data stored in databases can be queried in two interfaces:

1. Standard Search
2. Clustered Search

4.2.1 Standard Search Interface

Given a search query as shown in Figure 4.8, the search interface retrieves the web pages related with words in the query. The search interface is executed on ORACLE WebLogic Server. It uses Model-View-Controller framework as shown in the Figure 4.9.

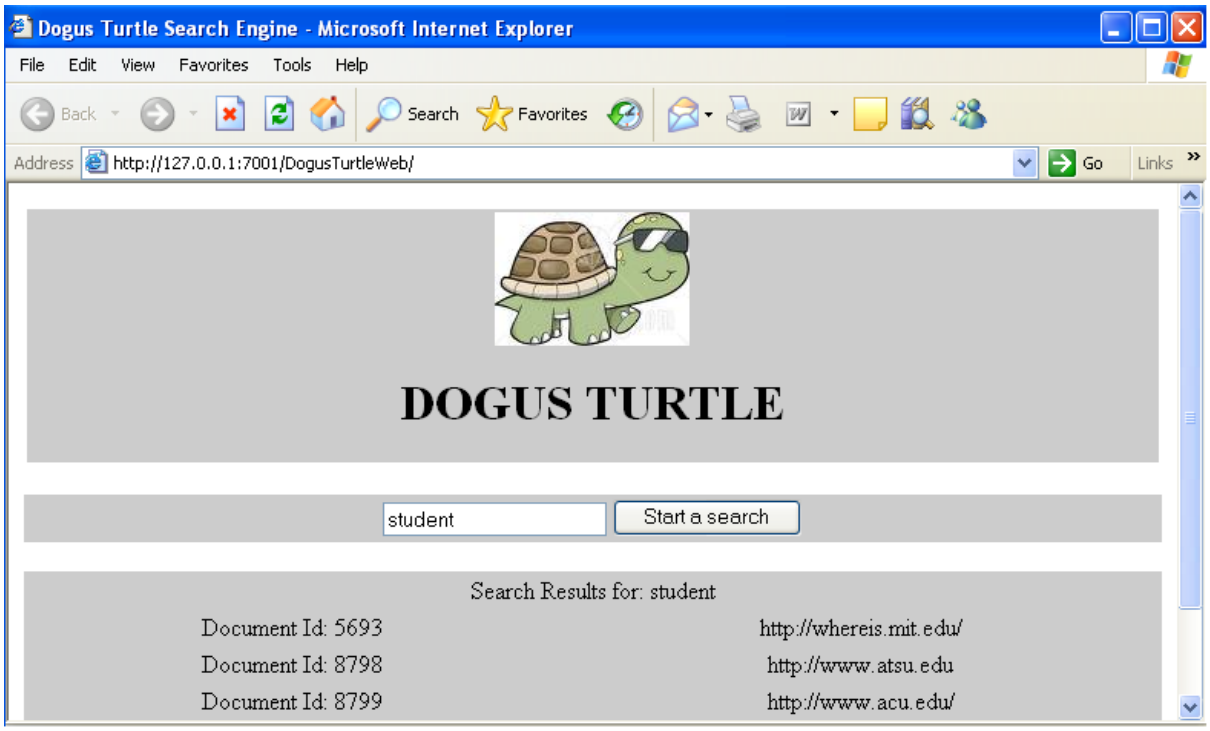


Figure 4.8 Search Results view for DTC

In the MVC pattern, three objects provide separation of application data and a presentation layer. The model maintains the application data and the business rules that govern manipulation of this data. The model is the closest of these objects to a real-world entity. The

view, on the other hand, has to understand the particulars of the display technology that it is rendering output for. The controller is the interface into the model. It helps to translate user interaction with the view into actions to be handled by the model. The controller may be somewhat different depending on what actions the view is capable of generating. The action may be an HTTP GET action when the user clicks a button on a browser, or it may be an HTTP POST from a Web service consumer.

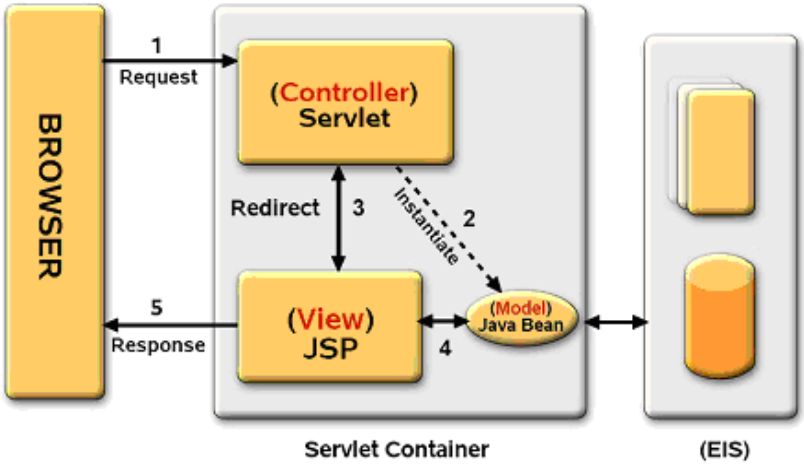


Figure 4.9 MVC Design (Web Controller, 2009)

The Search Interface which uses MVC framework has 3 packages: MODEL, SERVLET, DAO. The MODEL stores search data. The SERVLET is the actual component that makes the search. The DAO is the database interface object. Figure 4.10 shows a part of SERVLET code that makes a search.

```
public class TurtleController extends  HttpServlet {

    private static final long serialVersionUID = -5169232742312398342L;

    public void service(HttpServletRequest oReq, HttpServletResponse oRes) {

        String page = "/index.jsp";
        if (oReq.getParameter("actUrl").equals("doSearch")) {
            page = "/results.jsp";
            List<Document> results = getResults(oReq,oRes);
                oReq.setAttribute("results", results);
        }
        try {
            getServletContext().getRequestDispatcher(page).forward(oReq, oRes);
        } catch (ServletException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    /****
     * According to search criteria
     * making search
     * @param req
     * @param res
     * @return
     */
    private List<Document> getResults(HttpServletRequest req,
            HttpServletResponse res) {

        DBResult dbr = new DBResult();
        String criteria = req.getParameter("criteria");

        if (criteria != null) {
            try {
                return dbr.getSearchResults(criteria);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return null;
    }

}
```

Figure 4.10 Servlet Class detail for DTC

4.2.2 Clustered Search Interface

As explained before, DTC Crawler can classify pages while crawling. The classification property of the crawler enables a new kind of search called Clustered Search. The clustered search provides not only the documents that are related with the search query, also a list of categories. As shown in the Figure 4.11, clustered search lists the categories of web pages for a given query. This enables the user to narrow search query results on a specific category.

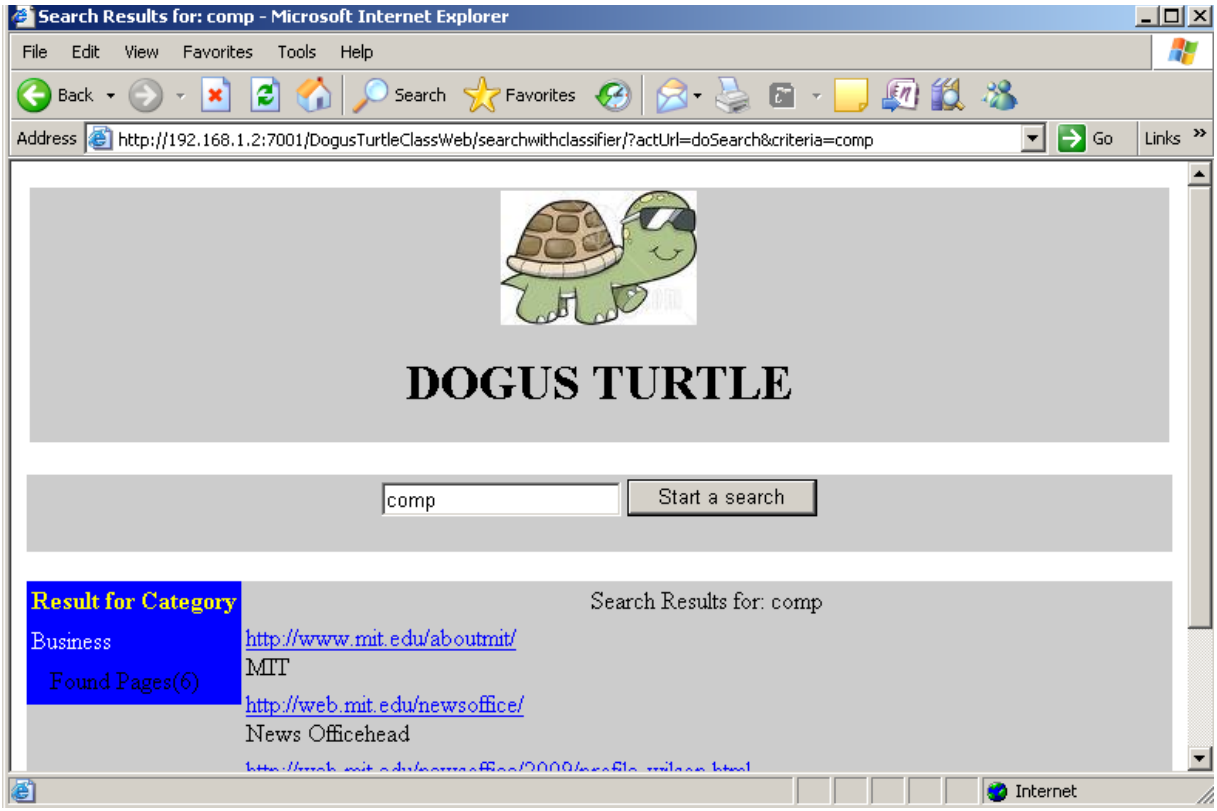


Figure 4.11 Search Results view for DTC

4.3 Text Classification Module

Text classification module enables classification of web pages while crawling. For classification several classification learning algorithms can be used. Each classification algorithm should be trained using a training data set. We obtained training data set from DMOZ Open Directory Project.

4.3.1 DMOZ

DMOZ is known as one of the biggest web directories in the internet. The Open directory project contains main directories and sub directories. Each directory lists documents or sub categories related with a subject.

As shown in the Figure 4.12, a DMOZ page contains sub sections. For example, business category contains Accounting, Management, Inventing and the other sub categories. The Accounting sub-category contains Firms, Outsourcing, Small Business and the other sub categories, too.

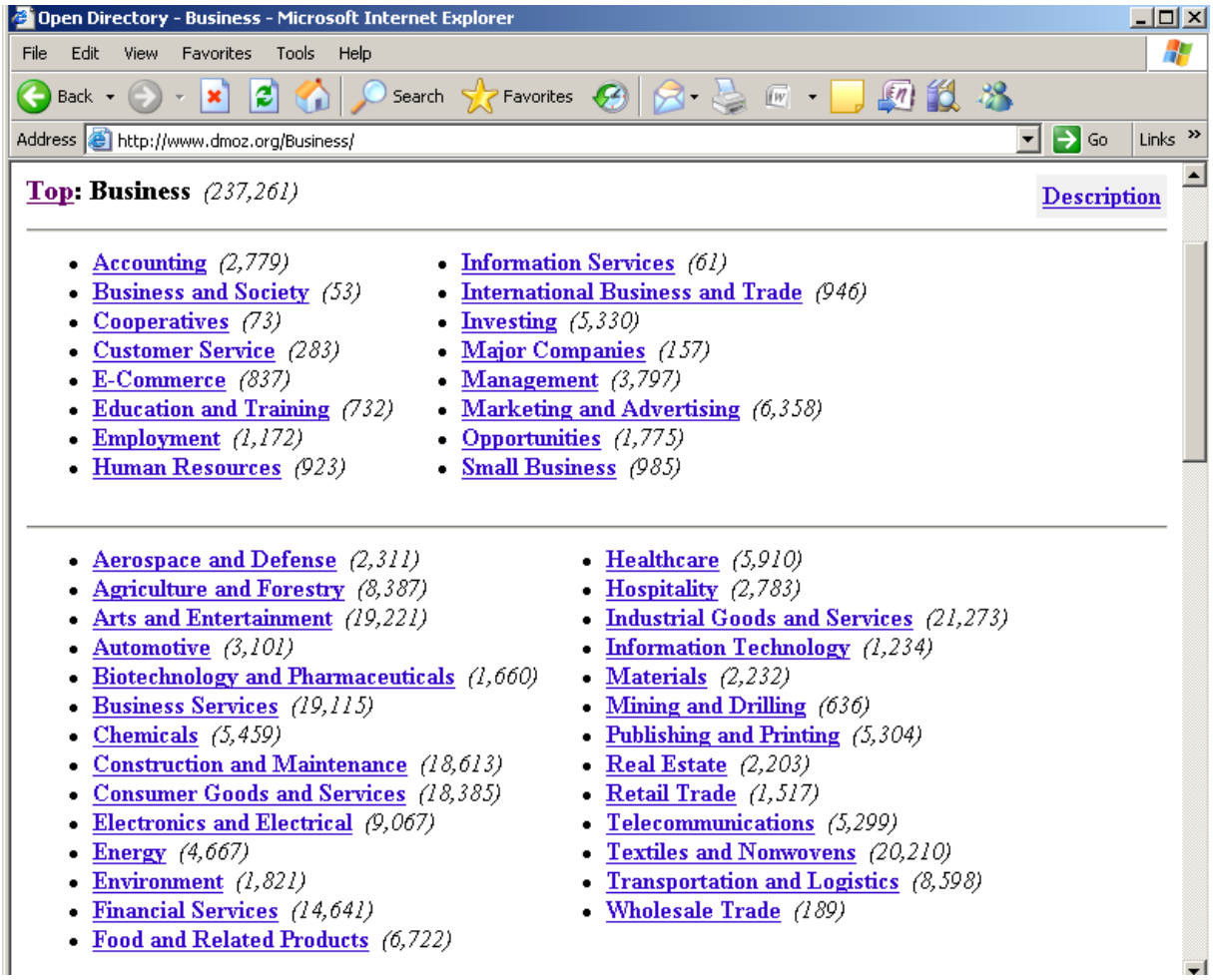


Figure 4.12 Business Category (Dmoz 1, 2009)

Some of categories contain a document links that are related with that category. For instance Business/Associations category contains document links (Figure 4.13) related with business associations.

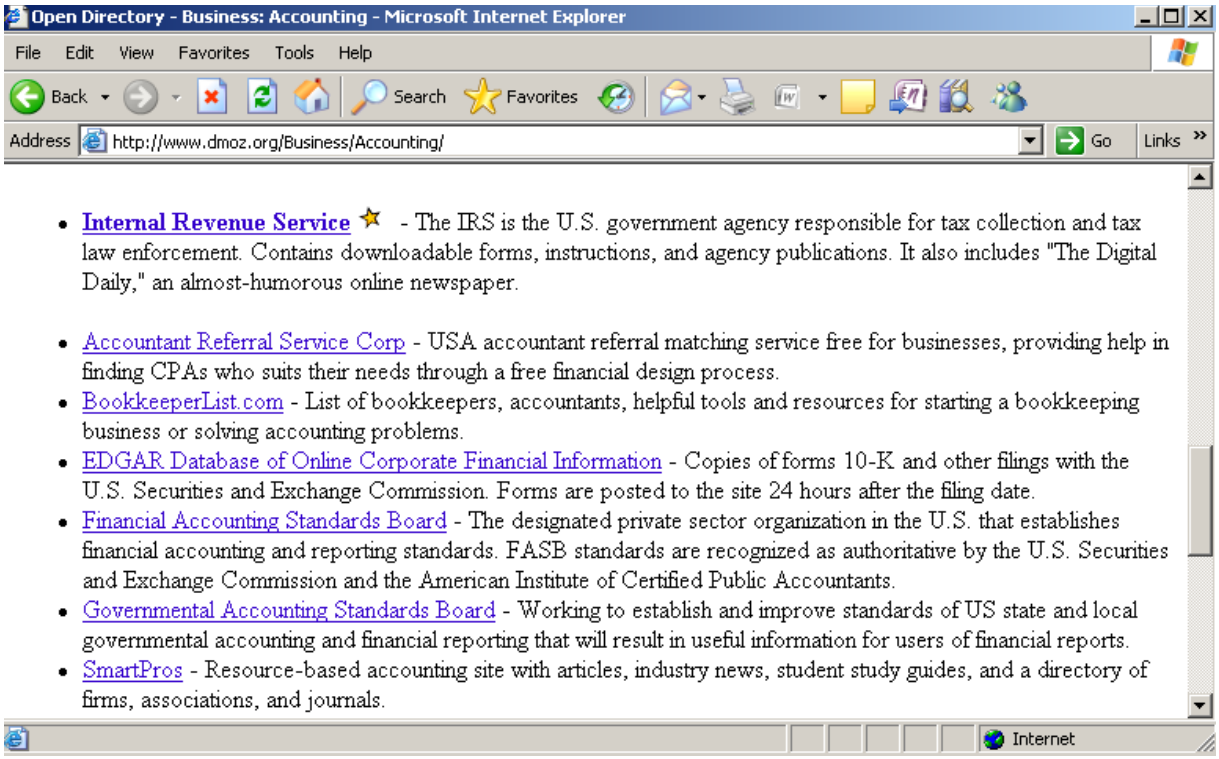


Figure 4.13 Business- Associations Category (Dmoz 2, 2009)

4.3.2 Text Preprocessing

Two text preprocessing methods are used in this study:

- Stemming
- Term(Attribute) Reduction

Stemming algorithms find the stems of words in given document. We used snowball stemmer in our study. If the DTC Crawler is started with stemming enabled property, snowball library is used and all terms are stemmed before the classification step. If stemming property is enabled, both training and test sets are stemmed. The Stemming Java library included to DTC Crawler by downloading from the following link: "http://snowball.tartarus.org/texts/introduction.html".

Attribute reduction (selection) algorithms enable the reduction of attributes in a data set. This can increase the classification success rates. We used an Info Gain Filter Algorithm of WEKA java library.

InfoGainFilter class (Figure 4.15) contains two getList methods. According to input object(Vector<String> or String[]), methods apply the infoGain filtering for a given threshold value by using classes in the WEKA java library.

The class diagram is illustrated in the Figure 4.14.

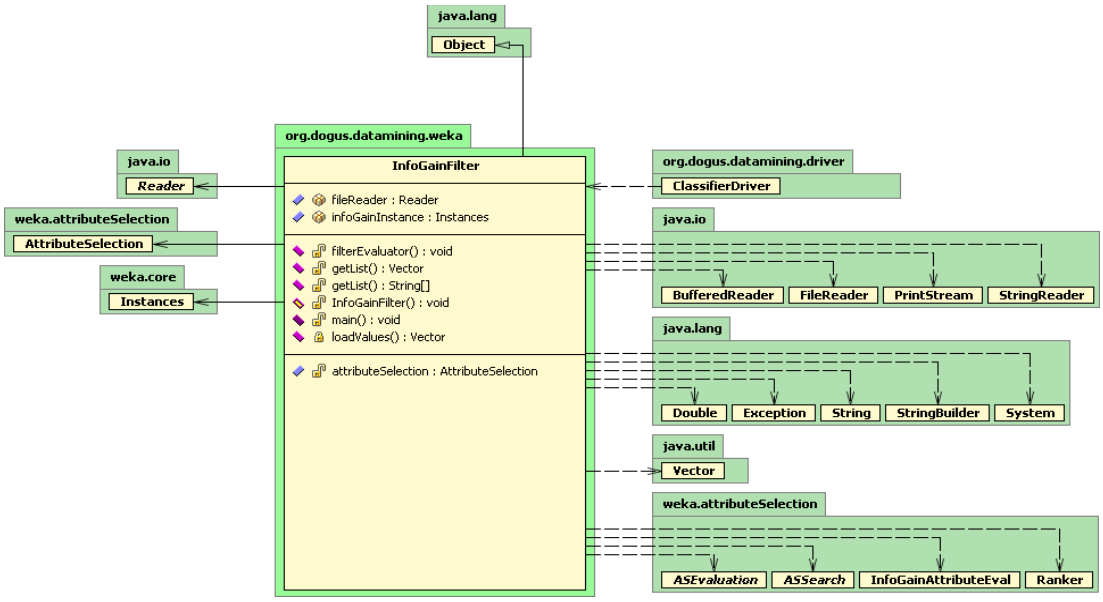


Figure 4.14 The Model Diagram of InfoGainFilter Class

```
package org.dogus.datamining.weka;

import java.io.BufferedReader;

 *  @author ArzuBT

public class InfoGainFilter {

    Instances           infoGainInstance    = null;
    Reader              fileReader          = null;
    AttributeSelection  attributeSelection;

    public InfoGainFilter(){
        super();
    }

    public void filterEvaluator(String strPathOfArrf,

    public Vector<String> getList(Vector<String> values, float rate) throws Exception {

    public String[]  getList(String [] values, float rate) throws Exception {


    public AttributeSelection getAttributeSelection(){
        return this.attributeSelection;
    }


     * For Testing class
    public static void main(String args[]) throws Exception {

     * For Testing class
    private Vector<String> loadValues(Vector<String> values) {
}
```

Figure 4.15 The InfoGainFilter Class

4.3.3 Obtaining DMOZ Training Data Using DMOZ Crawler

A Crawler is developed to crawl the pages in DMOZ Directory. This Crawler is very similar to DOGUS Turtle Engine (DTC). The objective of this crawler is to obtain training a dataset.

According to startup DMOZ links, DMOZ Crawler visits pages and collects information about the text content and links saves them into a database.

The DMOZ Crawler engine runs according to the following startup parameters.

- Depth: gives the number of levels of links that can be visited during the search process.
- Thread Number: gives the number of parallel threads that crawls web in parallel.
- Startup URL Only: this parameter enables visiting pages only contain web addresses in the startup link table.

The starting URL information is got from the DME_DMOZ_LINK_STARTUP table in the database. Every visited page is updated in this table and LINK_STATUS is set from 1 to 0. Each visited page is parsed and the links in the page are populated into DME_DMOZ_LINKS table. For each link a new DOC_ID is generated and this links are added to sub link queue for each thread. The Depth of the link controls whether the crawler visit the link or not. Figure 4.16 shows the class diagram of DMOZ Web Crawler.

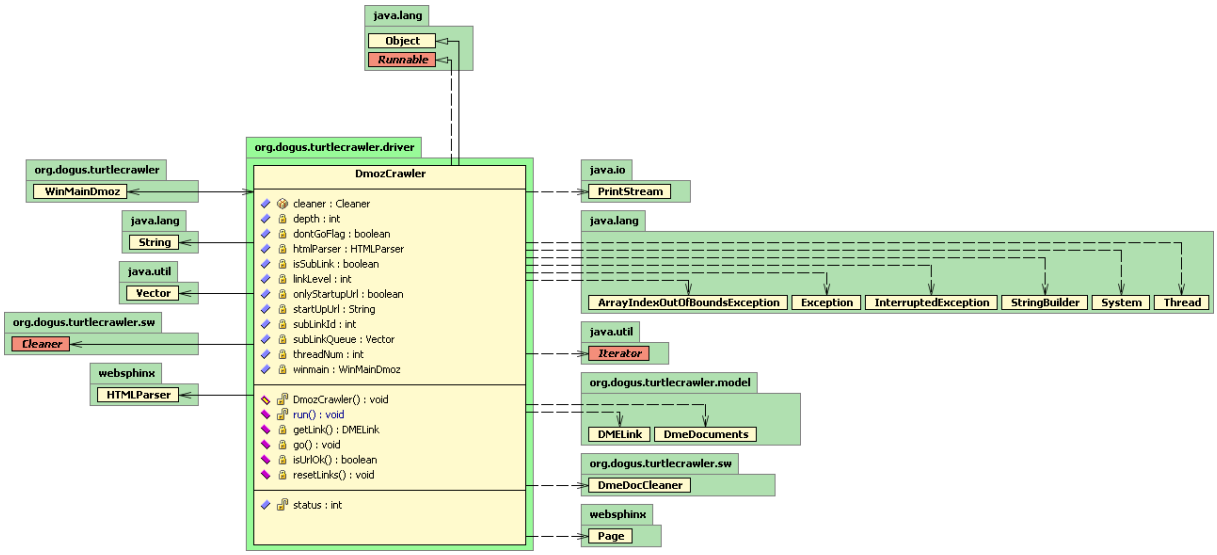


Figure 4.16 The DmozCrawler Class

When the DMOZ Web Crawler is started, the steps and visited pages are displayed in the console. In the Figure 4.17, console log messages can be seen.

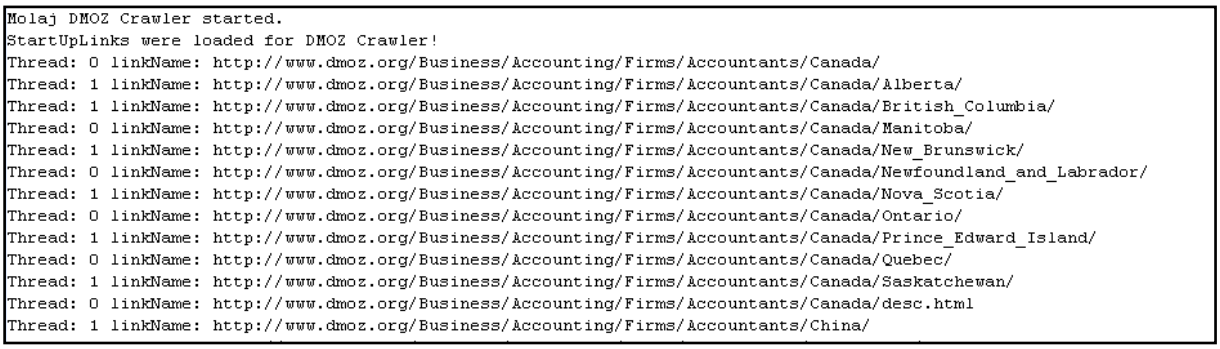


Figure 4.17 DmozCrawler Console log details

4.3.4 Database Model of DMOZ Training Data Set

DMOZ training data set is obtained by visiting the pages in DMOZ Open Directory. We use a crawler called "DMOZ Crawler" to visit pages in the DMOZ directory. The information about the visited pages and categories stored in the ORACLE Database. This database has 4 tables that stores information from the DMOZ Open Directory. When the DMOZ Crawler is run, the startup DMOZ site links are contained in the DME_DMOZ_LINK_STARTUP table with LINK_STATUS equals to one initially. DME_DMOZ_DOCUMENTS table is used to collect visited pages. DME_DMOZ_WORD_DETAIL contains the words of visited pages. The DME_DMOZ_LINKS table contains the links in visited pages. The table formats are shown in the following tables.

4.3.4.1 Table for Starting Links

The DME_DMOZ_LINK_STARTUP Table is used to define initial startup links that will be visited by the DTC Crawler for DMOZ. The startup links should be provided by the user. This table stores URL of the link and link status. The link status field is set to "0" whenever pages are visited in that link (Table 4.7).

Table 4.7 The DME_DMOZ_LINK_STARTUP table

COLUMN NAME	DATA TYPE	DESCRIPTION
LINK_ID	NUMBER	Link sequence number
LINK_URL	VARCHAR2(500)	Web address of page for visiting
LINK_STATUS	INTEGER	1:will be visited, 0: was visited

4.3.4.2 The DMOZ Documents Table

The DME_DMOZ_DOCUMENTS table stores document ID and URL of all the pages visited by DTC Crawler. This table contains a unique document ID each different page (Table 4.8).

Table 4.8 The DME_DMOZ_DOCUMENTS Table

COLUMN NAME	DATA TYPE	DESCRIPTION
DOC_ID	NUMBER	PK
URL	VARCHAR2(500)	Web page URL information

4.3.4.3 The DME_DMOZ_LINKS Table

The DME_DMOZ_LINKS table (Table 4.9) stores the entire URL addresses in the document visited by DTC Crawler.

Table 4.9 The DME_DMOZ_LINKS Table

COLUMN NAME	DATA TYPE	DESCRIPTION
DOC_ID	NUMBER	DOC_ID corresponds to the DOC_ID in the DME_DMOZ_DOCUMENTS table.
URL	VARCHAR2(500)	Web page URL information

4.3.4.4 The DMOZ Word Detail Table

The DME_DMOZ_WORD_DETAIL table stores words (terms) contained in each web page. This tables has three fields DOC_ID, WORD and TERMNO (Table 4.10). The TERMNO field stores the order of words stored in a document.

Table 4.10 The DME_DMOZ_WORD_DETAIL Table

COLUMN NAME	DATA TYPE	DESCRIPTION
DOC_ID	NUMBER	PK
WORD	VARCHAR2(150)	Word in the visited page
TERMNO	NUMBER	Word sequence number

4.3.4.5 Training Data Set Table

The DME_TRAINING Table (Table 4.11) is the main table that contains the training data set for classification. This table is loaded with the data by running the query in the Figure 4.14.

```
INSERT INTO MOLAJ.DME_TRAINING
   (DOC_ID, URL, WORD, UP_URL, UP_DOC_ID, CAT_URL)
SELECT D.DOC_ID, D.URL, DE.WORD, DC.CAT_DESC,D.DOC_ID AS UP_DOC_ID, D.URL AS CAT_URL
  FROM DME_DMOZ_DOCUMENTS D, DME_DMOZ_WORD_DETAIL DE, DME_CATEGORY DC
 WHERE D.DOC_ID = DE.DOC_ID AND D.URL=  DC.CAT_START_URL
 UNION
SELECT D2.DOC_ID,D2.URL, DE.WORD, DC.CAT_DESC, D.DOC_ID AS UP_DOC_ID,D.URL AS CAT_URL
  FROM DME_DMOZ_DOCUMENTS D, DME_CATEGORY DC, DME_DMOZ_LINKS DL, DME_DMOZ_DOCUMENTS D2,DME_DMOZ_WORD_DETAIL DE
 WHERE D.URL=  DC.CAT_START_URL AND DL.DOC_ID = D.DOC_ID  AND D2.URL = DL.URL AND DE.DOC_ID=D2.DOC_ID
```

The Figure 4.18: SQL Script for the DME_TRAINING Table

Table 4.11 The DME_TRAINING Table

COLUMN NAME	DATA TYPE	DESCRIPTION
DOC_ID	NUMBER	
WORD	VARCHAR2(150)	Word in the visited page
TERMNO	NUMBER	Word sequence number
UP_DOC_ID	NUMBER	
URL	VARCHAR2(750)	URL information
UP_URL	VARCHAR2(750)	URL category information

The Figure 4.14 shows the loadingProcess method which prepares input data for WEKA Classifiers. ExportManager class in the program given in the Figure 4.19 selects the data from the DME_TRAINING table and convert it into DmeMainCategory Vector. Then data in DmeMainCategory Vector is loaded into two arrays (arr[], arrCat[]) which are used as an input to WEKA Classifiers.

```
public void loadingProcess(){

    exportManager = new ExportManager();
    Vector<DmeMainCategory> trainData   = exportManager.getDataForTraining();

    int size    = 0; // number of elements in traininit set
    for (DmeMainCategory dme: trainData) {
         size += dme.getValues().size();
    }

    String arr[]        = new String[size];
    String arrCat[]     = new String[size];

    int i=0;
    for (DmeMainCategory dme: trainData) {
         for (String str: dme.getValues()){
             arr[i] = str;
             arrCat[i] = dme.getCategoryName();
             i++;
         }
    }

    if (stemEnabled){
        arr = StringBuilder.getCommaStemWords(arr);
    }
```

Figure 4.19 The Part of Loading Process Method

4.3.4.6 DME_WEKA Classifiers Table

The DME_WEKA table stores names and class paths of WEKA Classifiers algorithms used in this study. The identification values of WEKA_CLASS_ID are also defined in the definition class of DTC DMOZ Crawler. A sample view of DME_WEKA table is shown in the Figure 4.20.

WEKA_CLASS_ID	DESCRIPTION	WEKA_CLASS
2	NaiveBayes	weka.classifiers.bayes.NaiveBayes
1	MultilayerPerceptron	weka.classifiers.functions.MultilayerPerceptron

Figure 4.20 The DME_WEKA Table

4.3.4.7 The DME_CATEGORY Table

The DME_CATEGORY table contains the main categories and the URL of main categories in DMOZ directory. This table is manually populated with data. A sample view of DME_CATEGORY table is in the Figure 4.21.

CAT_ID	CAT_DESC	CAT_START_URL
0	OthersCategory	does not exist
1	Business	http://www.dmoz.org/Business/
2	Sport	http://www.dmoz.org/Sports/
3	Science	http://www.dmoz.org/Science/
4	Health	http://www.dmoz.org/Health/
5	Computers	http://www.dmoz.org/Computers/
6	Kids	http://www.dmoz.org/Kids_and_Teens/
7	Shopping	http://www.dmoz.org/Shopping/
8	Arts	http://www.dmoz.org/Arts/
9	Games	http://www.dmoz.org/Games/
10	Home	http://www.dmoz.org/Home/
11	News	http://www.dmoz.org/News/
12	Regional	http://www.dmoz.org/Regional/
13	Society	http://www.dmoz.org/Society/
14	Recreation	http://www.dmoz.org/Recreation/

Figure 4.21 The Detail of DME Category Table

4.3.4.8 The DME_DOCUMENT_CAT Table

The DME_DOCUMENT_CAT table contains DOC_ID, CAT_ID, WEKA_CLASS_ID, STEMMED columns. The WEKA Classifier fills data into this table. Given a document with DOC_ID, a WEKA Classifier with WEKA_CLASS_ID determines the class of the given document and stores it into CAT_ID column. A sample view of the table is shown in the Figure 4.22.

DOC_ID	CAT_ID	WEKA_CLASS_ID	STEMMED
23206	8	1	1
23207	8	1	1
23208	8	1	1
23209	8	1	1
23204	8	1	1
23205	8	1	1
23210	8	1	1
23211	8	1	1
23212	8	1	1
23213	8	1	1
23214	8	1	1
23215	8	1	1
23216	8	1	1
23218	8	1	1

Figure 4.22 The Detail of DME Document Category Table

4.3.5 Text Classification Algorithms

Text classification module enables classification of web pages while crawling. For classification, several classification learning algorithms can be used. We use the following classification algorithms in our study:

a. Naïve Bayes
b. MultiLayerPerceptron

These algorithms are imported from the WEKA Machine Learning Software Package. New classes are developed to train and make predictions using the classification algorithms.

4.3.5.1 The ClassifierDriver Class

Classification driver is a class for running different types of classification algorithms. For different classification algorithms, the usage of classifier can be changed. A class view is shown in the Figure 4.23. The loadingProcess method is called by the WinMain Class of DTC. The loadClasses method prepares classifier classes using the given configuration parameters. The loadClassifiers method returns the trained classifiers as a List.

```
package org.dogus.datamining.driver;

import java.util.Vector;


 * @author ArzuBT

public class ClassifierDriver {
    Vector<DMEClassifier>   classifiers     = null;
    DataClassifier          dataClassifier  =  new DataClassifier();
    ExportManager           exportManager;
    UtilDriver              utilDriver      = new UtilDriver();
    ClassAdapter            clsAdapter      =   null;
    String [] categoryValues;
    String [] categoryNames;
    String[] testDataValues;

    boolean stemEnabled = false;

    public ClassifierDriver(boolean stemmerEnabled) {

    public ClassifierDriver() {

    public DMEClassifier getDriver(String desc) {

    public void classifierDocument(DmeDocuments doc) {

     * For each classifier is loaded from database
    public Vector<DMEClassifier> loadClassifiers() {

     * Data Selected from DB.
    public void loadingProcess(){

    private void writeFile(String[] categoryNames2, String[] categoryValues2,

    public void setStemStatus(boolean stemEnabled) {

    public void setDmeClassifier(Vector<DMEClassifier> loadClassifiers) {

    public Vector<DMEClassifier> loadClassifiers(String[] testText) {

    public void loadingProcessForInputs(    String arr[] ,  String arrCat[] ){


    public String [] getCategoryValues(){

    public String [] getCategoryNamess(){

}
```

Figure 4.23 The ClassifierDriver in the Data Mining Library of DTC

4.3.5.2 The Classifier Classes

For his Project two classes are implemented from DMEClassifier interface. The MultiLayerPerceptronClassifier and the NaiveBayesClassifier classes are used for this purpose. Methods of interface are shown in the Figure 4.24. The MultiLayerPerceptronClassifier class is shown in the Figure 4.18. The train method is called in the only first time when DTC is starting up. After a web page is visited, page category is determined by calling the predictClause method. This method is declared in the DmeClassifier interface. Each classifier classes is implemented from that Interface.

```
package org.dogus.datamining.driver;

import weka.classifiers.Classifier;

 * @author ArzuBT

public interface DMEClassifier {

    public void setClassifier(Classifier classifier);

    public Classifier getClassifier();

    public int getClassId();

    public String getClassDefinition(String val);

    public void train(String[] categoryNames, String[] categoryValues);

    public boolean isClassifierLoaded();

    public String predictClause(String  valueStr) ;

    public StringBuffer testIt(String  []value);
}
```

Figure 4.24 The DME Classifier Interface

```
package org.dogus.datamining.driver;

import java.util.Arrays;

 * @author ArzuBT Dogus University,2009
public class MultiLayerPerceptronClassifier extends MainClassifier implements DMEClassifier {
    private MLPConfiguration config;
    private MultilayerPerceptron classifier;
    private int layerSize = 40;

    private Attribute classAttribute = null;

    public MultiLayerPerceptronClassifier(MLPConfiguration _config) {

    public MultiLayerPerceptronClassifier() {

    public String getClassDefinition(String val) {

    public Classifier getClassifier() {

    public MultilayerPerceptron getMultiLayerPerceptronClassifier() {

    public void setClassifier(MultilayerPerceptron classifier) {

    public void train(String[] categoryNames, String[] categoryValues) {

    public void setClassifier(Classifier classifier) {

    public String predictClause(String  valueStr) {

    public StringBuffer testIt(String  []value) {

    public boolean isClassifierLoaded() {

    public int getClassId() {

    public void testItExtra(String[] testText) {

}
```

Figure 4.25 The MultiLayerPerceptronClassifier Class

5 WEB CRAWLER AND TEXT CLASSIFICATION RESULTS

5.1 Web Crawler Results

A sample dataset is constructed in order to find the most frequently used words on the data set obtained by DOGUS Turtle Crawler. The DOGUS Turtle Crawler is run with 402 different startup links. Each link is visited with a depth size 1. It means that, only startup web pages and links in those pages are visited and the contents of documents are populated into related database tables. A total of 20678 links is collected from the visited pages. A total of 68487 words is populated into the DME_WORD_DETAIL table.

The most frequently used words in the visited sites are calculated by using a SQL aggregation query. In the table 5.1, the first top ten most frequently used words are displayed. “directory” is the most common word in the collected words which is found in the 902 different documents.

Table 5.1 Top word counts

Count	Word
902	directory
681	dmoz
666	sports
595	services
561	open
500	information
468	tax
462	accounting
436	business
395	netscape

5.2 Text Classification Results

As it is stated before two classification algorithms are used in our study. In this section, the accuracy rates of these two algorithms on different types of data sets are compared. We obtained eight different training data sets and two testing data sets. These data sets are

obtained from DMOZ Directory. Each data set has five categories: Games, Arts, Health, Business and Computers. The Table 5.2 shows the properties of these data sets. The first column shows number of documents in the Training set. The second column gives information filtering rate which reduces the number of attributes on the training data sets. The third column gives the number of attributes (terms) in a given data set after attribute reduction by using information gain algorithm. The last column shows that whether the stemming algorithm is applied to the given training set or not.

Table 5.2 Table of Training Data

Training Data Set	Page Counts	Information Filtering Rate	Number of Attributes	Stemming Enabled(Yes/No)
TRAINSET1	496	0.02	837	No
TRAINSET2	479	0.03	390	No
TRAINSET3	301	0.05	104	No
TRAINSET4	514	-	4124	No
TRAINSET5	506	0.02	682	Yes
TRAINSET6	497	0.03	342	Yes
TRAINSET7	422	0.05	98	Yes
TRAINSET8	514	-	4647	Yes

We have two different testing data sets. The first data set contains 195 documents about computer category. This testing data set is obtained from several sites. This data set is called as TESTSET1. The second data set are taken from the standard twenty newsgroups database (http://www.cs.cmu.edu/afs/cs/project/theo-11/www/naive-bayes/20_newsgroups.tar.gz). The second data set consists of messages about "computer.windows.x" category. The second data set contains 1000 messages. This is the second data set is called TESTSET2. Table 5.3 shows the number of documents in each of the data sets.

Table 5.3 Testing Data Sets

Testing Data Set	Description	Number of Pages
TESTSET1	Pages from the Computer category (64217 words)	195
TESTSET2	News Data Set from www.cmu.edu web site (computer.windows.x)	1000

5.2.1 Using Naïve Bayes Classifier

In this section, Naïve Bayes Classifier results are given for test and training model.

5.2.1.1 Results for Training Model

We run the Naïve Bayes Classifier algorithm with eight training sets. In these experiments, each training set is also used as a test data set. A confusion matrix is obtained for each of the experiments which show the prediction rate of the Naïve Bayes Classifier. Tables 5.4 - 5.11 shows the confusion matrix for each of the data sets.

Table 5.12 shows classification rates of data sets. As it is seen in Table 5.12, the best results are obtained in training sets TRAINSET4 and TRAINSET8. The TRANSET4 contains all the terms without attribute reduction and without application of stemming algorithm. The TRAINSET8 contains all the terms without attribute reduction but stemming algorithm is applied.

Table 5.4 Confusion Matrix of Trainset1

Category Names	**Unknown**	**Computers**	**Arts**	**Health**	**Business**	**Games**
Unknown	0	0	0	0	0	0
Computers	19	68	0	1	1	5
Arts	10	1	83	0	2	0
Health	13	0	0	49	3	0
Business	15	0	0	0	67	0
Games	36	3	0	1	3	116

Table 5.5 Confusion Matrix of Trainset2

Category Names	Unknown	Computers	Arts	Health	Business	Games
Unknown	0	0	0	0	0	0
Computers	24	60	0	0	1	5
Arts	12	1	78	0	3	2
Health	20	1	0	40	3	0
Business	20	1	0	1	57	1
Games	32	7	1	0	6	103

Table 5.6 Confusion Matrix of Trainset3

Category Names	Unknown	Computers	Arts	Health	Business	Games
Unknown	0	0	0	0	0	0
Arts	0	72	0	7	1	10
Health	0	0	31	2	3	19
Computers	0	0	1	34	2	9
Business	0	0	0	4	40	13
Games	0	0	0	7	5	41

Table 5.7 Confusion Matrix of Trainset4

Category Names	Unknown	Computers	Arts	Health	Business	Games
Unknown	0	0	0	0	0	0
Computers	5	87	0	0	3	6
Arts	7	0	86	0	1	2
Health	3	0	0	60	1	1

Business	6	0	0	0	78	1
Games	7	3	0	1	2	154

Table 5.8 Confusion Matrix of Trainset5

Category Names	Health	Computers	Business	Games	Arts	Unknown
Health	52	0	2	0	0	11
Computers	0	69	4	4	0	21
Business	0	0	65	1	0	16
Games	1	3	8	119	2	32
Arts	1	4	0	3	82	6
Unknown	0	0	0	0	0	0

Table 5.9 Confusion Matrix of Trainset6

Category Names	Health	Computers	Business	Games	Arts	Unknown
Health	48	0	2	0	0	14
Computers	1	66	3	2	0	25
Business	0	2	59	1	0	17
Games	1	4	9	107	6	34
Arts	2	6	0	2	77	9
Unknown	0	0	0	0	0	0

Table 5.10 Confusion Matrix of Trainset7

Category Names	Health	Computers	Business	Games	Arts	Unknown
Health	39	3	4	10	0	5
Computers	3	49	0	18	0	8
Business	1	8	35	13	0	10
Games	2	7	2	82	4	25
Arts	0	3	0	16	72	3
Unknown	0	0	0	0	0	0

Table 5.11 Confusion Matrix of Trainset8

Category Names	Health	Computers	Business	Games	Arts	Unknown
Health	61	0	0	1	0	3
Computers	1	83	3	7	1	6
Business	0	0	77	1	0	7
Games	2	1	4	149	0	11
Arts	0	0	0	6	86	4
Unknown	0	0	0	0	0	0

Table 5.12 Classification Rates for Naïve Bayes Classifier

	Correctly Classified	Incorrectly Classified	Total	Percentage of Correctly Classified Instances
TRAINSET1	383	113	496	77.22
TRAINSET2	338	141	479	70.56
TRAINSET3	218	83	301	72.43
TRAINSET4	465	49	514	90.47
TRAINSET5	387	119	506	76.48

TRAINSET6	357	140	497	71.83
TRAINSET7	277	145	422	65.63
TRAINSET8	456	58	514	88.72

5.2.1.2 Results for TestSet1

In this section, the classification rates of Naïve Bayes Classifier are given for the TESTSET1. After training Naïve Bayes Classifier with training sets 1 to 8, the TESTSET1 is applied to the Naïve Bayes Classifier to obtain classification rates. Table 5.13-5.20 shows results of these experiments.

Table 5.13 Classification Results for TrainSet1

Category Names	Number of Instance	Rate of Instances (%)
Unknown	33	16.92
Art	68	34.87
Business	5	2.56
Computers	41	21.03
Games	22	11.28
Health	26	13.33

Table 5.14 Classification Results for TrainSet2

Category Names	Number of Instance	Rate of Instances (%)
Unknown	33	16.92
Art	106	54.36
Business	11	5.64
Computers	25	12.82
Games	3	1.54
Health	17	8.72

Table 5.15 Classification Results for TrainSet3

Category Names	Number of Instance	Rate of Instances (%)
Unknown	33	16.92
Art	106	54.36
Business	11	5.64
Computers	25	12.82
Games	3	1.54
Health	17	8.72

Table 5.16 Classification Results for TrainSet4

Category Names	Number of Instance	Rate of Instances (%)
Unknown	0	0
Art	129	66.15
Business	12	6.15
Computers	7	3.59
Games	45	23.08
Health	2	1.03

Table 5.17 Classification Results for TrainSet5

Category Names	Number of Instance	Rate of Instances (%)
Unknown	18	9.23
Art	126	64.62
Business	4	2.05
Computers	39	20.00
Games	4	2.05
Health	4	2.05

Table 5.18 Classification Results for TrainSet6

Category Names	Number of Instance	Rate of Instances (%)
Unknown	9.23	18
Art	65.64	128
Business	1.54	3
Computers	18.46	36
Games	2.56	5
Health	2.56	5

Table 5.19 Classification Results for TrainSet7

Category Names	Number of Instance	Rate of Instances (%)
Unknown	19	9.74
Art	124	63.59
Business	9	4.62
Computers	19	9.74
Games	20	10.26
Health	4	2.05

Table 5.20 Classification Results for TrainSet8

Category Names	Number of Instance	Rate of Instances (%)
Not Found	15	7.69
Art	113	57.95
Business	4	2.05
Computers	39	20.00
Games	19	9.74
Health	5	2.56

5.2.1.3 Naïve Bayes Results for TestSet2

In this section, the classification rates of Naïve Bayes Classifier are given for the TESTSET2. After training Naïve Bayes Classifier with training sets 1 to 8, the TESTSET2 is applied to the Naïve Bayes Classifier to obtain classification rates. Table 5.21-5.28 shows results of these experiments.

Table 5.21 Classification Results for TrainSet1

Category Names	Number of Instance	Rate of Instances (%)
Unknown	526	52.6
Art	285	28.5
Business	23	2.3
Computers	101	10.1
Games	29	2.9
Health	36	3.6

Table 5.22 Classification Results for TrainSet2

Category Names	Number of Instance	Rate of Instances (%)
Unknown	580	58
Art	200	20
Business	77	7.7
Computers	69	6.9
Games	38	3.8
Health	36	3.6

Table 5.23 Classification Results for TrainSet3

Category Names	Number of Instance	Rate of Instances (%)
Unknown	0	0
Art	195	19.5
Business	40	4
Computers	69	6.9
Games	691	69.1
Health	5	0.5

Table 5.24 Classification Results for TrainSet4

Category Names	Number of Instance	Rate of Instances (%)
Unknown	215	21.5
Art	5	0.5
Business	128	12.8
Computers	560	56
Games	74	7.4
Health	18	1.8

Table 5.25 Classification Results for TrainSet5

Category Names	Number of Instance	Rate of Instances (%)
Unknown	763	76.3
Art	94	9.4
Business	39	3.9
Computers	81	8.1
Games	20	2
Health	3	0.3

Table 5.26 Classification Results for TrainSet6

Category Names	Number of Instance	Rate of Instances (%)
Unknown	796	79.6
Art	84	8.4
Business	16	1.6
Computers	90	9
Games	12	1.2
Health	2	0.2

Table 5.27 Classification Results for TrainSet7

Category Names	Number of Instance	Rate of Instances (%)
Unknown	750	75
Art	50	5
Business	5	0.5
Computers	73	7.3
Games	121	12.1
Health	1	0.1

Table 5.28 Classification Results for TrainSet8

Category Names	Number of Instance	Rate of Instances (%)
Unknown	380	38
Art	56	5.6
Business	62	6.2
Computers	134	13.4
Games	338	33.8
Health	30	3

5.2.2 Using Multi Layer Perceptron Classifier

In this section, Multi Layer Perceptron results are given for test and training model.

5.2.2.1 Results for Training Model

We run the Multi Layer Perceptron algorithm with eight training sets. In these experiments, each training set is also used as a test data set. A confusion matrix is obtained for each of the experiments which show the prediction rate of the Multi Layer Perceptron. Tables 5.29 - 5.36 shows the confusion matrix for each of the data sets.

Table 5.37 shows classification rates of data sets. As it is seen in Table 5.37, the best results are obtained in training sets TRAINSET2,TRAINSET3 and TRAINSET7. The TRANSET2 contains all the terms with attribute reduction (0.03) and without application of stemming

algorithm. The TRAINSET8 contains all the terms without attribute reduction but stemming algorithm is applied.

Table 5.29 Confusion Matrix of Trainset1

Category Names	Unknown	Computers	Arts	Health	Business	Games
Unknown	0	0	0	0	0	0
Computers	0	0	0	0	0	94
Arts	0	0	64	0	0	32
Health	0	0	0	20	0	45
Business	0	0	0	1	0	81
Games	0	0	0	0	0	159

Table 5.30 Confusion Matrix of Trainset2

Category Names	Unknown	Computers	Arts	Health	Business	Games
Unknown	0	0	0	0	0	0
Computers	0	71	0	7	11	1
Arts	0	0	87	7	2	0
Health	0	2	0	56	6	0
Business	0	21	0	3	55	1
Games	0	18	0	7	54	70

Table 5.31 Confusion Matrix of Trainset3

Category Names	Unknown	Computers	Arts	Health	Business	Games
Unknown	0	0	0	0	0	0
Computers	0	86	0	1	0	3
Arts	0	0	42	0	0	13
Health	0	0	0	18	0	28

Business	0	0	3	2	37	15
Games	0	1	1	0	1	50

Table 5.32 Confusion Matrix of Trainset4

Category Names	**Unknown**	**Computers**	**Arts**	**Health**	**Business**	**Games**
Unknown	0	0	0	0	0	0
Computers	0	0	101	0	0	0
Arts	0	0	96	0	0	0
Health	0	0	64	0	0	0
Business	0	0	85	0	0	0
Games	0	0	167	0	0	0

Table 5.33 Confusion Matrix of Trainset5

Category Names	**Health**	**Computers**	**Business**	**Games**	**Arts**	**Unknown**
Health	35	0	0	30	0	0
Computers	0	9	0	89	0	0
Business	0	0	4	78	0	0
Games	0	0	0	165	0	0
Arts	0	4	0	20	72	0
Unknown	0	0	0	0	0	0

Table 5.34 Confusion Matrix of Trainset6

Category Names	Health	Computers	Business	Games	Arts	Unknown
Health	25	4	18	11	6	0
Computers	0	48	1	45	3	0
Business	0	5	44	10	20	0
Games	0	1	0	155	5	0
Arts	0	0	0	7	89	0
Unknown	0	0	0	0	0	0

Table 5.35 Confusion Matrix of Trainset7

Category Names	Health	Computers	Business	Games	Arts	Unknown
Health	47	0	2	12	0	0
Computers	6	42	5	25	0	0
Business	4	3	47	13	0	0
Games	5	4	2	111	0	0
Arts	0	2	0	10	82	0
Unknown	0	0	0	0	0	0

Table 5.36 Confusion Matrix of Trainset8

Category Names	Health	Computers	Business	Games	Arts	Unknown
Health	0	0	0	0	65	0
Computers	0	0	0	0	101	0
Business	0	0	0	0	85	0
Games	0	0	0	0	167	0
Arts	0	0	0	0	96	0
Unknown	0	0	0	0	0	0

Table 5.37 Classification Rates

	Correctly Classified	Incorrectly Classified	Total	Percentage of Correctly Classified Instances
TRAINSET1	243	253	496	48.99
TRAINSET2	339	140	479	70.77
TRAINSET3	233	68	301	77.41
TRAINSET4	97	417	514	18.87
TRAINSET5	285	221	506	56.32
TRAINSET6	361	136	497	72.64
TRAINSET7	329	93	422	77.96
TRAINSET8	96	418	514	18.68

5.2.2.2 Results for TestSet1

In this section, the classification rates of Multi Layer Perceptron are given for the TESTSET1. After training Naïve Bayes Classifier with training sets 1 to 8, the TESTSET1 is applied to the Multi Layer Perceptron Classifier to obtain classification rates. Table 5.38-5.45 shows results of these experiments.

Table 5.38 Classification Results for TrainSet1

Category Names	Number of Instance	Rate of Instances (%)
Unknown		
Art	4	2.05
Business	160	82.05
Computers	31	15.90
Games		
Health		

Table 5.39 Classification Results for TrainSet2

Category Names	Number of Instance	Rate of Instances (%)
Unknown		
Art	42	21.54
Business	27	13.85
Computers	79	40.51
Games	6	3.08
Health	41	21.03

Table 5.40 Classification Results for TrainSet3

Category Names	Number of Instance	Rate of Instances (%)
Unknown		
Art	123	63.08
Business	5	2.56
Computers	3	1.54
Games	55	28.21
Health	9	4.62

Table 5.41 Classification Results for TrainSet4

Category Names	Number of Instance	Rate of Instances (%)
Unknown		
Art	193	98.97
Business		
Computers		
Games		
Health	2	1,02

Table 5.42 Classification Results for TrainSet5

Category Names	Number of Instance	Rate of Instances (%)
Unknown		
Art	117	60.00

Business		
Computers	45	23.08
Games	33	16.92
Health		

Table 5.43 Classification Results for TrainSet6

Category Names	Number of Instance	Rate of Instances (%)
Unknown		
Art	75	38.46
Business	24	12.31
Computers	73	37.44
Games	23	11.79
Health		

Table 5.44 Classification Results for TrainSet7

Category Names	Number of Instance	Rate of Instances (%)
Unknown		
Art	124	63.59
Business	17	8.72
Computers	19	9.74
Games	35	17.95
Health		

Table 5.45 Classification Results for TrainSet8

Category Names	Number of Instance	Rate of Instances (%)
Unknown		
Art	193	98.97
Business		
Computers	2	1.03
Games		
Health		

5.2.2.3 Results for TestSet2

In this section, the classification rates of Multi Layer Perceptron are given for the TESTSET2. After training Naïve Bayes Classifier with training sets 1 to 8, the TESTSET2 is applied to the Multi Layer Perceptron Classifier to obtain classification rates. Table 5.46-5.52 shows results of these experiments.

Table 5.46 Classification Results for TrainSet1

Category Names	Number of Instance	Rate of Instances (%)
Unknown		
Art	15	1.50
Business		
Computers		
Games	985	98.50

Table 5.47 Classification Results for TrainSet2

Category Names	Number of Instance	Rate of Instances (%)
Unknown	99	9.9
Art	62	6.20
Business	561	56.10
Computers	252	25.20
Games	26	2.60

Table 5.48 Classification Results for TrainSet3

Category Names	Number of Instance	Rate of Instances (%)
Unknown		
Art	15	1.50
Business		
Computers		
Games	985	98.50

Table 5.49 Classification Results for TrainSet4

Category Names	Number of Instance	Rate of Instances (%)
Unknown		
Art	1000	100
Business		
Computers		
Games		

Table 5.50 Classification Results for TrainSet5

Category Names	Number of Instance	Rate of Instances (%)
Unknown		0
Art	8	0.8
Business	2	0.2
Computers	8	0.8
Games	982	98.2

Table 5.51 Classification Results for TrainSet6

Category Names	Number of Instance	Rate of Instances (%)
Unknown		0
Art	72	7.2
Business	2	0.2
Computers	8	0.8
Games	918	91.8

Table 5.51 Classification Results for TrainSet7

Category Names	Number of Instance	Rate of Instances (%)
Unknown		0
Art	6	0.6
Business	1	0.1
Computers	38	3.8
Games	941	94.1

Table 5.52 Classification Results for TrainSet8

Category Names	Number of Instance	Rate of Instances (%)
Unknown		0
Art	1000	100
Business		0
Computers		0
Games		0

5.2.3 Comparing Results and Disccussion

Table 5.53 shows the comparison of Naïve Bayes and Multi Layer Perceptron classifiers on training sets 1 to 8. As it is seen in the table, Naïve Bayes seems to make a more accurate classification than Multi Layer Perceptron for the training data sets in most of the cases. The Multilayer Perceptron Classifier only works better only on training sets 3, 6 and 7.

When we look at the Info Gain and Stemming results; Info Gain Filter helps to decrease of number of attributes, but for our data set, filtering does not enhance classification ratios. In addition, the application of stemming algorithms does not change results, either. The experiments done with test data sets 1 and 2 usually do not produce good classification ratios.

Table 5.53 Comparison Classifiers

	Naïve Bayes				**Multi Layer Perceptron**				
	Correctly Classified	%	Incorrectly Classified	%		%	Incorrectly Classified	%	Total
TRAINSET1	383	77.22	113	22.78	243	48.99	253	51.01	496
TRAINSET2	338	70.56	141	29.44	339	70.77	140	29.23	479
TRAINSET3	218	72.43	83	27.57	233	77.41	68	22.59	301
TRAINSET4	465	90.47	49	9.53	97	18.87	417	81.13	514
TRAINSET5	387	76.48	119	23.52	285	56.32	221	43.68	506
TRAINSET6	357	71.83	140	28.17	361	72.64	136	27.36	497
TRAINSET7	277	65.63	145	34.36	314	74.40	108	26.59	422
TRAINSET8	456	88.72	58	11.28	96	18.68	418	81.32	514

6 CONCLUSION AND SUGGESTIONS

Web search engines extracts useful index information from the raw document data stored in the World Wide Web. The objective of this thesis to analyze the current commercial and open-source search engines and develop a web robot, indexing and classification system that extracts useful information and knowledge from the raw web data resources.

In this thesis, current commercial search engines, architecture of search engines and new developments in search engines are overviewed. Then, text mining algorithms, tools and Open Source WEKA Data Mining library is introduced and described.

We developed a multi threaded web crawler (called DOGUS Turtle Crawler) and Search Engine Query interface using Java. Web Crawler includes a text classification module that classifies web pages according to their document contents. We obtained several training sets from the DMOZ directory. These training sets are used to train two classification algorithms. We evaluated the performance of these algorithms on the given training and testing sets. The use of classifiers provides the automatic categorization of web pages while crawling. This enables clustered search of document in our document database collected by our crawler.

As a future work, other classification algorithms might be added to classifier library of DOGUS Turtle Engine. DOGUS Turtle Crawler only uses a HTML parser for parsing html document in the visited web pages. In the Internet, there are many types of text documents other than the HTML documents like MS WORD and PDF files. These types of files can also be indexed by including parsers that supports different types of file formats.

REFERENCES

Books

Geisler,Eliezer, (2008), *Knowledge and Knowledge Systems:Learning from the Wonders of the Mind, Illionois:IDI Publishing*

Keyes, Jessica, (2006), *Knowledge Management, Business Intelligence, and Content Management:* Auerbach Publications

Manning ,Christopher D., *Introduction to Information Retrieval,* Cambridge University Press

Morville,P. and Rosenfeld,L., (2007) *Information Architecture for the World Wide Web:* O'Reilly Media, Inc.

Lucey, T., (2005) *Management information systems,* Cengage Learning EMEA

Lucas, H. (2000) *Information technology for management, Newyork:* Irwin McGraw-Hili

Ledford, Jerri L. (2007), *Search Engine Optimization Bible,* John Wiley & Sons

Poremsky, D. (2004), *Google and Other Search Engines: Visual Quickstart Guide***,** Peachpit Press

Calishain, T. ea, (2003), *Spidering Hacks,* O'Reilly

Bramer, M. (2007), *Principles of Data Mining*, Springer

Liu, B., (2006), *Web Data Mining - Exploring Hyperlinks, Contents and Usage Dat*, Springer

Kantardzic M. (2003), *Data Mining: Concepts, Models, Methods, and Algorithms*, Wiley-IEEE Press

Symposiums

Lin, Shan, ea., (2008), "Information mining system design and implementation based on web crawler" *System of Systems Engineering, 2008. SoSE '08. IEEE International Conference*, June 2008, Singapore

Tan, Qingzhao , ea, (2007), "Designing Clustering-Based Web Crawling Policies for Search Engine Crawlers", *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, Lisbon

Sahami,M. ea., (2001), "The Happy Searcher: Challenges in Web Information Retrieval", *Proceedings of the 8th PRICAI Conference*

Hu, ea, (2003), "A Probabilistic Model for Intelligent Web Crawlers", *Proceedings of the 27th Annual International Computer Software and Applications Conference*

Tripaty A. ea, (2008) "A Web Mining Architectural Model of Distributed Crawler for Internet Searches Using PageRank Algorithm ", *IEEE Asia-Pacific Services Computing Conference*

Guo G. ea. (2008), "A kNN Model-based Approach and its application in text categorization", *Computational linguistics and intelligent text processing. International Conference*

Nascimento, M.A. ea. (1998), "An experiment stemming non-traditional text", *String Processing and Information Retrieval: A South American Symposium*

Metikurke S. ea. (2006), "Grid-Enabled Automatic Web Page Classification", *IEEE International Conference on Fuzzy Systems*

Markov Z., (2006), "An introduction to the WEKA data mining system", *Annual Joint Conference Integrating Technology into Computer Science Education*

Articles

Arasu, A. ea. (2001) "Searching the Web", ACM Transactions on Internet Technology (TOIT), Pages: 2 - 43

Feinerer, Ingo ea., (2008) "Text Mining Infrastructure in R",Journal of Statistical Software

Cambazoglu B. ea. (2007) "Architecture of a grid-enabled Web search engine", Information Processing and Management, Pages: 609–623

Hotho, A. (2005), A Brief Survey of Text Mining ,LDV Forum, Pages: 19-62

Internet Resources

Wikipedia 1, http://en.wikipedia.org/wiki/Knowledge, 2009

Google 1, http://www.google.com.tr, 2009

Google 2, http://infolab.stanford.edu/~backrub/google.html, 2009

Clusty, http://www.clusty.com, 2009

Dmoz, http://www.dmoz.org, 2009

Wikipedia 2, http://en.wikipedia.org/wiki/File:WebCrawlerArchitecture.svg, 2009

Wikipedia 3, http://en.wikipedia.org/wiki/Web_crawler, 2008

UMD Library, www.cs.umd.edu/Library/TRs/CS-TR-4291/CS-TR-4291.pdf, 2009

Umich, https://practice.sph.umich.edu/micphp/Retrieving_Online_Info/R_O_I/CD_Master/CD/content/Search_Engines.pdf , 2009

Google 3, http://www2.sandbox.google.com/, 2009

Data Mining, http://www3.itu.edu.tr/~sgunduz/courses/verimaden/slides/d3.pdf, 2009

Multi-Layer Perceptron, http://www.ncgia.ucsb.edu/giscc/units/u188/u188.html, 2009

Wikipedia 4, http://en.wikipedia.org/wiki/SPSS, 2009

Stanford , http://lane.stanford.edu/howto/index.html?id=4077, 2009

Wikipedia 5, http://en.wikipedia.org/wiki/Text_mining, 2009

Dimov, http://www.dfki.de/~kipp/seminar_ws0607/reports/RossenDimov.pdf, 2009

Sphinx, http://www.cs.cmu.edu/~rcm/websphinx, 2008

Web Controller, http://www.javapassion.com/j2ee/MVCPatternAndFrameworks.pdf, 2009

Dmoz 1, http://www.dmoz.org/Business, 2009

Dmoz 2, http://www.dmoz.org/Business/Accounting, 2009

Wikipedia 6, http://en.wikipedia.org/wiki/data, 2009

Search Engine 1, http://searchenginewatch.com, 2009

Search Engine 2, http://www.searchengineshowdown.com, 2009

Meta Crawler, http://www.metacrawler.com, 2009

Profusion, http://www.profusion.com, 2009

Search Engine 3, http://www2.hawaii.edu/~rpeterso/engine_.htm, 2009

Web Crawler, http://www.webcrawler.com, 2009

Google 4, http://research.google.com/pubs/InformationRetrieval.html , 2009

Oracle, http://www.oracle.com, 2009

Snowball, http://snowball.tartarus.org/texts/introduction.html, 2009

Search Engine 4, http://searchdatamanagement.techtarget.com/generic/0,295582,sid91_gci1264550,00.html, 2009

Statistic, http://www.jstatsoft.org/v25/i05, 2009

N.Bayes, http://www.biomedcentral.com/1471-2105/7/514, 2009

APPENDIX I. JAVADOCS

Packages	
org.dogus.data	
org.dogus.datamining	
org.dogus.datamining.driver	
org.dogus.datamining.driver.bulk	
org.dogus.datamining.main	
org.dogus.datamining.text	
org.dogus.datamining.util	
org.dogus.datamining.weka	
org.dogus.model	
org.dogus.molaj	
org.dogus.servlet	
org.dogus.servlet.dao	
org.dogus.turtlecrawler	
org.dogus.turtlecrawler.dao	
org.dogus.turtlecrawler.dao.act	
org.dogus.turtlecrawler.dao.dmoz	
org.dogus.turtlecrawler.driver	
org.dogus.turtlecrawler.export	
org.dogus.turtlecrawler.model	
org.dogus.turtlecrawler.sw	
org.tartarus.snowball	
org.tartarus.snowball.ext	
snippet	

Package org.dogus.datamining.driver

Interface Summary	
DMEClassifier	

Class Summary	
ClassifierDriver	
DataClassifier	
MainClassifier	
MultiLayerPerceptronClassifier	
NaiveBayesClassifier	
UtilDriver	

Package org.dogus.datamining.util

Interface Summary	
Configuration	

Class Summary	
ClassAdapter	
InstanceHolder	
MLPConfiguration	
NBayesConfiguration	

Package org.dogus.datamining.weka

Class Summary	
InfoGainFilter	
WordSetUtil	

Package org.dogus.turtlecrawler

Class Summary	
DATAUsage	
PortListener	
WinMain	
WinMainDmoz	

Package org.dogus.turtlecrawler.export

Class Summary	
ExportManager	

Package org.dogus.turtlecrawler.model

Class Summary	
DmeDocumentDetail	
DmeDocumentEmails	
DmeDocuments	
DMELink	
DmeMainCategory	
DmeWordDetail	

Package org.dogus.turtlecrawler.sw

Interface Summary	
Cleaner	

Class Summary	
DmeDocCleaner	
DocumentWriter	
StopWordRemover	

org.dogus.turtlecrawler

Class WinMain

java.lang.Object
└ **org.dogus.turtlecrawler.WinMain**

---

public class **WinMain**
extends java.lang.Object

Author:
ArzuBT Dogus University,2009 **Class Name:** WinMain ARZUBT 200591002

Constructor Summary
WinMain()
WinMain(java.lang.String[] args)

Method Summary	
void	**classifyDocument**(DmeDocuments doc)
DMELink	**getVisitingLink**(int threadNum) This method gives a link which will be visited
void	**init**() Loads DB Objects
static WinMain	**instance**(java.lang.String[] args)
boolean	**isVisitedLink**(java.lang.String link)
static void	**main**(java.lang.String[] args)
void	**saveDocsToDB**(DmeDocuments dmeDoc) In order to having new doc_id this method is called.

void	**saveDocToFileSystem**(DmeDocuments doc)
void	**saveLinksToDB**(DmeDocuments dmeDoc)
void	**saveWordsToDB**(DmeDocuments dmeDoc) For each visited page's words are saved into DB
void	**stop**() // watch to threads // stop them // when every thread finished // call System.exit(0);
void	**updateDocumentDetails**(DmeDocuments doc)
void	**updateDocumentLnk**(DmeDocuments doc)
void	**updateVisitingLink**(int threadNum, int linkId)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

WinMain
public **WinMain**()

WinMain
public **WinMain**(java.lang.String[] args)

Method Detail

main
public static void **main**(java.lang.String[] args)

---

init
public void **init**()
Loads DB Objects

---

instance
public static WinMain **instance**(java.lang.String[] args)

---

stop
public void **stop**()
// watch to threads // stop them // when every thread finished // call System.exit(0);

---

getVisitingLink
public DMELink **getVisitingLink**(int threadNum)
This method gives a link which will be visited

---

updateVisitingLink
public void **updateVisitingLink**(int threadNum,
int linkId)

---

saveLinksToDB
public void **saveLinksToDB**(DmeDocuments dmeDoc)
Parameters:
dmeDoc - All found links are populated to table for reporting

---

saveWordsToDB
public void **saveWordsToDB**(DmeDocuments dmeDoc)
For each visited page's words are saved into DB
Parameters:
dmeDoc -

---

saveDocsToDB
public void **saveDocsToDB**(DmeDocuments dmeDoc)
In order to having new doc_id this method is called. DME_DOCUMENTS table is populated. If a doc_id exists do nothing
Parameters:
dmeDoc -

---

isVisitedLink
public boolean **isVisitedLink**(java.lang.String link)

---

saveDocToFileSystem

public void **saveDocToFileSystem**(DmeDocuments doc)

---

updateDocumentLnk

public void **updateDocumentLnk**(DmeDocuments doc)

---

updateDocumentDetails

public void **updateDocumentDetails**(DmeDocuments doc)

---

classifyDocument

public void **classifyDocument**(DmeDocuments doc)

org.dogus.turtlecrawler

Class WinMainDmoz

java.lang.Object
└ **org.dogus.turtlecrawler.WinMainDmoz**

---

public class **WinMainDmoz**
extends java.lang.Object

Author:

ArzuBT Dogus University,2009 **Class Name:** WinMainDmoz ARZUBT 200591002

Constructor Summary
WinMainDmoz()
WinMainDmoz(java.lang.String[] args)

Method Summary	
DMELink	**getVisitingLink**(int threadNum) This method gives a link which will be visited
void	**init**()

	Loads DB Objects for DMOZ
static WinMainDmoz	**instance**(java.lang.String[] args)
boolean	**isVisitedLink**(java.lang.String link)
static void	**main**(java.lang.String[] args)
void	**saveDocsToDB**(DmeDocuments dmeDoc) In order to having new doc_id this method is called.
void	**saveLinksToDB**(DmeDocuments dmeDoc)
void	**saveWordsToDB**(DmeDocuments dmeDoc) For each visited page's words are saved into DB
void	**stop**() // watch to threads // stop them // when every thread finished // call System.exit(0);
void	**updateDocumentLnk**(DmeDocuments doc)
void	**updateVisitingLink**(int threadNum, int linkId)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

WinMainDmoz

public **WinMainDmoz**()

WinMainDmoz

public **WinMainDmoz**(java.lang.String[] args)

Method Detail

main

public static void **main**(java.lang.String[] args)

---

init

public void **init**()

Loads DB Objects for DMOZ

---

instance

public static WinMainDmoz **instance**(java.lang.String[] args)

---

stop

public void **stop**()

// watch to threads // stop them // when every thread finished // call System.exit(0);

---

getVisitingLink

public DMELink **getVisitingLink**(int threadNum)

This method gives a link which will be visited

---

updateVisitingLink

public void **updateVisitingLink**(int threadNum,
int linkId)

---

saveLinksToDB

public void **saveLinksToDB**(DmeDocuments dmeDoc)

Parameters:

dmeDoc - All found links are populated to table for reporting

---

saveWordsToDB

public void **saveWordsToDB**(DmeDocuments dmeDoc)

For each visited page's words are saved into DB

Parameters:

dmeDoc -

---

saveDocsToDB

public void **saveDocsToDB**(DmeDocuments dmeDoc)

In order to having new doc_id this method is called.
DME_DMOZ_DOCUMENTS table is populated. If a doc_id exists do nothing

Parameters:

dmeDoc -

---

isVisitedLink

public boolean **isVisitedLink**(java.lang.String link)

---

updateDocumentLnk

public void **updateDocumentLnk**(DmeDocuments doc)

org.dogus.turtlecrawler.driver

Class Crawler

java.lang.Object
└ **org.dogus.turtlecrawler.driver.Crawler**

All Implemented Interfaces:

java.lang.Runnable

---

public class **Crawler**
extends java.lang.Object
implements java.lang.Runnable

Author:

ArzuBT Dogus University,2009

Class Name: Crawler

Constructor Summary
Crawler(int _threadNum, int _depth, boolean _classifyEnabled, WinMain _winMain)

Method Summary	
void	**run**()
void	**setStatus**(int _status)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Crawler

public **Crawler**(int _threadNum,
int _depth,
boolean _classifyEnabled,
WinMain _winMain)

Method Detail

run

public void **run**()

Specified by:

run in interface java.lang.Runnable

---

setStatus

public void **setStatus**(int _status)

org.dogus.turtlecrawler.driver

Class DmozCrawler

java.lang.Object
└ **org.dogus.turtlecrawler.driver.DmozCrawler**

All Implemented Interfaces:

java.lang.Runnable

---

public class **DmozCrawler**
extends java.lang.Object
implements java.lang.Runnable

Author:

ArzuBT Dogus University,2009 **Class Name:** DmozCrawler

Constructor Summary

DmozCrawler(int _threadNum, int _depth, boolean _onlyStartpUpUrl,
WinMainDmoz _winMain)

Method Summary	
void	**run**()
void	**setStatus**(int _status)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DmozCrawler

public **DmozCrawler**(int _threadNum,
int _depth,
boolean _onlyStartpUpUrl,
WinMainDmoz _winMain)

Method Detail

run

public void **run**()

Specified by:

run in interface java.lang.Runnable

---

setStatus

public void **setStatus**(int _status)

org.dogus.turtlecrawler.model

Class DmeDocuments

java.lang.Object
└ **org.dogus.turtlecrawler.model.DmeDocuments**

---

public class **DmeDocuments**
extends java.lang.Object

Author:

ArzuBT Dogus University,2009 Class Name: DmeDocuments Purpose: Every web page will be kept in this class Links and words also are kept as a vector class

---

Constructor Summary
DmeDocuments(int _link_level)

Method Summary	
void	**evaluate**() This method can be moved for future use 1.
java.util.Vector<DmeMainCategory>	**getDmeCategories**()
java.util.Vector<java.lang.String>	**getDmeDocumentEmails**()
java.util.Vector<java.lang.String>	**getDmeDocumentWords**()
java.util.Vector<DMELink>	**getDmeLinks**()
long	**getDoc_id**()
java.lang.String	**getEncoding**()
java.lang.String	**getIp_no**()
int	**getLevel**() If the given debth parameter is greater than document debth, crawler stops and get a new statrup link

	value from winmain.
websphinx.Page	**getPage**()
java.lang.String	**getTitle**()
java.lang.String	**getUrl**()
boolean	**isProcessStemmed**()
boolean	**isSaved**()
void	**setDmeCategories**(java.util.Vector<DmeMainCategory> dmeCategories)
void	**setDmeDocumentWords**(java.util.Vector<java.lang.String> dmeDocumentWords)
void	**setDmeLinks**(java.util.Vector<DMELink> dmeLinks)
void	**setDoc_id**(long doc_id)
void	**setEncoding**(java.lang.String encoding)
void	**setIp_no**(java.lang.String ip_no)
void	**setLevel**(int level)
void	**setPage**(websphinx.Page page)
void	**setProcessStemmed**(boolean processStemmed)
void	**setSaved**(boolean isSaved)
void	**setTitle**(java.lang.String title)
void	**setUrl**(java.lang.String url)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DmeDocuments

public **DmeDocuments**(int _link_level)

Method Detail

getDoc_id

public long **getDoc_id**()

---

setDoc_id

public void **setDoc_id**(long doc_id)

---

getUrl

public java.lang.String **getUrl**()

---

setUrl

public void **setUrl**(java.lang.String url)

---

getIp_no

public java.lang.String **getIp_no**()

---

setIp_no

public void **setIp_no**(java.lang.String ip_no)

---

getTitle

public java.lang.String **getTitle**()

---

setTitle

public void **setTitle**(java.lang.String title)

---

getEncoding

public java.lang.String **getEncoding**()

---

setEncoding

public void **setEncoding**(java.lang.String encoding)

---

getDmeDocumentWords

public java.util.Vector<java.lang.String> **getDmeDocumentWords**()

---

setDmeDocumentWords

public void
setDmeDocumentWords(java.util.Vector<java.lang.String> dmeDocumentWords)

---

getDmeDocumentEmails

public java.util.Vector<java.lang.String> **getDmeDocumentEmails**()

---

getPage

public websphinx.Page **getPage**()

---

setPage

public void **setPage**(websphinx.Page page)

---

getDmeLinks

public java.util.Vector<DMELink> **getDmeLinks**()

---

setDmeLinks

public void **setDmeLinks**(java.util.Vector<DMELink> dmeLinks)

---

getLevel

public int **getLevel**()

If the given debth parameter is greater than document debth, crawler stops and get a new statrup link value from winmain.

---

setLevel

public void **setLevel**(int level)

---

evaluate

public void **evaluate**()

This method can be moved for future use 1. From the page object get the links

---

isSaved

public boolean **isSaved**()

---

setSaved

public void **setSaved**(boolean isSaved)

---

getDmeCategories

public java.util.Vector<DmeMainCategory> **getDmeCategories**()

---

setDmeCategories

public void **setDmeCategories**(java.util.Vector<DmeMainCategory> dmeCategories)

---

isProcessStemmed

public boolean **isProcessStemmed**()

---

setProcessStemmed

public void **setProcessStemmed**(boolean processStemmed)

org.dogus.turtlecrawler.model
Class DmeDocumentDetail

java.lang.Object
└ **org.dogus.turtlecrawler.model.DmeDocumentDetail**

---

public class **DmeDocumentDetail**
extends java.lang.Object

Author:

ArzuBT Dogus University,2009

Constructor Summary
DmeDocumentDetail()

Method Summary	
java.lang.String	**getContent**()
java.lang.String	**getContent2**()
java.lang.String	**getContent3**()
java.lang.String	**getContent4**()
java.lang.String	**getContent5**()
void	**setContent**(java.lang.String content)
void	**setContent2**(java.lang.String content2)
void	**setContent3**(java.lang.String content3)
void	**setContent4**(java.lang.String content4)
void	**setContent5**(java.lang.String content5)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DmeDocumentDetail

public **DmeDocumentDetail**()

Method Detail

getContent

public java.lang.String **getContent**()

---

setContent

public void **setContent**(java.lang.String content)

---

getContent2

public java.lang.String **getContent2**()

---

setContent2

public void **setContent2**(java.lang.String content2)

---

getContent3

public java.lang.String **getContent3**()

---

setContent3

public void **setContent3**(java.lang.String content3)

---

getContent4

public java.lang.String **getContent4**()

---

setContent4

public void **setContent4**(java.lang.String content4)

---

getContent5

public java.lang.String **getContent5**()

---

setContent5

public void **setContent5**(java.lang.String content5)

org.dogus.turtlecrawler.model

Class DmeMainCategory

java.lang.Object
└ **org.dogus.turtlecrawler.model.DmeMainCategory**

public class **DmeMainCategory**
extends java.lang.Object

Constructor Summary
DmeMainCategory()
DmeMainCategory(java.util.Vector<java.lang.String> values, java.lang.String categoryName)

Method Summary	
java.lang.String	**getCategoryName**()
int	**getCatId**()
int	**getStemFlag**()
java.util.Vector<java.lang.String>	**getValues**()
int	**getWekaClassId**()
void	**setCategoryName**(java.lang.String categoryName)
void	**setCatId**(int catId)
void	**setStemFlag**(int stemFlag)
void	**setValues**(java.util.Vector<java.lang.String> values)
void	**setWekaClassId**(int wekaClassId)
java.lang.String	**toString**()

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

DmeMainCategory

public **DmeMainCategory**(java.util.Vector<java.lang.String> values,
java.lang.String categoryName)

DmeMainCategory

public **DmeMainCategory**()

Method Detail

toString

public java.lang.String **toString**()

Overrides:

toString in class java.lang.Object

---

getValues

public java.util.Vector<java.lang.String> **getValues**()

---

setValues

public void **setValues**(java.util.Vector<java.lang.String> values)

---

getCategoryName

public java.lang.String **getCategoryName**()

---

setCategoryName

public void **setCategoryName**(java.lang.String categoryName)

---

getCatId

public int **getCatId**()

---

setCatId

public void **setCatId**(int catId)

---

getWekaClassId

public int **getWekaClassId**()

---

setWekaClassId

public void **setWekaClassId**(int wekaClassId)

---

getStemFlag

public int **getStemFlag**()

---

setStemFlag

public void **setStemFlag**(int stemFlag)

org.dogus.turtlecrawler.model

Class DMELink

java.lang.Object
└ **org.dogus.turtlecrawler.model.DMELink**

---

public class **DMELink**
extends java.lang.Object

Author:

ArzuBT Dogus University,2009

Constructor Summary
DMELink()
DMELink(int link_id, java.lang.String link_name, int link_status, int link_level)

Method Summary	
int	**getLink_id**()
int	**getLink_level**()
java.lang.String	**getLink_name**()
int	**getLink_status**()
int	**getRun_link_id**()
void	**setLink_id**(int link_id)
void	**setLink_level**(int link_level) If level equals to 0 it means this link got from database statup up table.
void	**setLink_name**(java.lang.String link_name)
void	**setLink_status**(int link_status)
void	**setRun_link_id**(int run_link_id)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DMELink

public **DMELink**()

DMELink

public **DMELink**(int link_id,

java.lang.String link_name,
int link_status,
int link_level)

Method Detail

getLink_id

public int **getLink_id**()

---

setLink_id

public void **setLink_id**(int link_id)

---

getLink_name

public java.lang.String **getLink_name**()

---

setLink_name

public void **setLink_name**(java.lang.String link_name)

---

getLink_status

public int **getLink_status**()

---

setLink_status

public void **setLink_status**(int link_status)

---

getLink_level

public int **getLink_level**()

---

setLink_level

public void **setLink_level**(int link_level)

If level equals to 0
it means this link got from
database statup up table.
-1 if not set.

---

getRun_link_id

public int **getRun_link_id**()

---

setRun_link_id

public void **setRun_link_id**(int run_link_id)

org.dogus.turtlecrawler.model

Class DmeWordDetail

java.lang.Object
└ **org.dogus.turtlecrawler.model.DmeWordDetail**

---

public class **DmeWordDetail**
extends java.lang.Object

Author:

ArzuBT Dogus University,2009

Constructor Summary
DmeWordDetail(long doc_id, java.lang.String word, int termnno, java.lang.String url, java.lang.String upUrl, long upDocId)

Method Summary	
long	**getDoc_id**()
int	**getTermnno**()
long	**getUpDocId**()
java.lang.String	**getUpUrl**()
java.lang.String	**getUrl**()
java.lang.String	**getWord**()
void	**setDoc_id**(long doc_id)

void	**setTermnno**(int termnno)
void	**setUpDocId**(long upDocId)
void	**setUpUrl**(java.lang.String upUrl)
void	**setUrl**(java.lang.String url)
void	**setWord**(java.lang.String word)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DmeWordDetail

public **DmeWordDetail**(long doc_id,
java.lang.String word,
int termnno,
java.lang.String url,
java.lang.String upUrl,
long upDocId)

Method Detail

getDoc_id

public long **getDoc_id**()

---

setDoc_id

public void **setDoc_id**(long doc_id)

---

getWord

public java.lang.String **getWord**()

---

setWord

public void **setWord**(java.lang.String word)

---

getTermnno

public int **getTermnno**()

---

setTermnno

public void **setTermnno**(int termnno)

---

getUrl

public java.lang.String **getUrl**()

---

setUrl

public void **setUrl**(java.lang.String url)

---

getUpUrl

public java.lang.String **getUpUrl**()

---

setUpUrl

public void **setUpUrl**(java.lang.String upUrl)

---

getUpDocId

public long **getUpDocId**()

---

setUpDocId

public void **setUpDocId**(long upDocId)

org.dogus.datamining.driver

Class ClassifierDriver

java.lang.Object
└ **org.dogus.datamining.driver.ClassifierDriver**

---

public class **ClassifierDriver**
extends java.lang.Object

Author:

ArzuBT Dogus University,2009 Usage Data Class: ClassifierDriver This class is used for classification
Some of the defined Weka classifiers, it works.

Constructor Summary
ClassifierDriver()
ClassifierDriver(boolean stemmerEnabled)

Method Summary	
void	**classifierDocument**(DmeDocuments doc)
java.lang.String[]	**getCategoryNamess**()
java.lang.String[]	**getCategoryValues**()
DMEClassifier	**getDriver**(java.lang.String desc)
java.util.Vector<DMEClassifier>	**loadClassifiers**() For each classifier is loaded from database They are trained according to training set in DME_TRAINING table
java.util.Vector<DMEClassifier>	**loadClassifiers**(java.lang.String[] testText)
void	**loadingProcess**() Data Selected from DB.
void	**loadingProcessForInputs**(java.lang.String[] arr, java.lang.String[] arrCat)
void	**setDmeClassifier**(java.util.Vector<DMEClassifier> load Classifiers)
void	**setStemStatus**(boolean stemEnabled)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ClassifierDriver

public **ClassifierDriver**(boolean stemmerEnabled)

ClassifierDriver

public **ClassifierDriver**()

Method Detail

getDriver

public DMEClassifier **getDriver**(java.lang.String desc)

---

classifierDocument

public void **classifierDocument**(DmeDocuments doc)

---

loadClassifiers

public java.util.Vector<DMEClassifier> **loadClassifiers**()

For each classifier is loaded from database They are trained according to training set in DME_TRAINING table

---

loadingProcess

public void **loadingProcess**()

Data Selected from DB. InfoGain Filter is applied

---

setStemStatus

public void **setStemStatus**(boolean stemEnabled)

---

setDmeClassifier

public void **setDmeClassifier**(java.util.Vector<DMEClassifier> loadClassifiers)

---

loadClassifiers

public java.util.Vector<DMEClassifier> **loadClassifiers**(java.lang.String[] testText)

---

loadingProcessForInputs

public void **loadingProcessForInputs**(java.lang.String[] arr,
java.lang.String[] arrCat)

---

getCategoryValues

public java.lang.String[] **getCategoryValues**()

---

getCategoryNamess

public java.lang.String[] **getCategoryNamess**()

org.dogus.datamining.driver

Class MainClassifier

java.lang.Object
└ **org.dogus.datamining.driver.MainClassifier**

All Implemented Interfaces:

DMEClassifier

Direct Known Subclasses:

MultiLayerPerceptronClassifier, NaiveBayesClassifier

---

public class **MainClassifier**
extends java.lang.Object
implements DMEClassifier

Constructor Summary
MainClassifier()

Method Summary	
java.lang.String	**getClassDefinition**(java.lang.String val)
int	**getClassId**()
weka.classifiers.Classifier	**getClassifier**()
boolean	**isClassifierLoaded**()
java.lang.String	**predictClause**(java.lang.String valueStr)
void	**setClassifier**(weka.classifiers.Classifier classifier)
java.lang.StringBuffer	**testIt**(java.lang.String[] value)
void	**train**(java.lang.String[] categoryNames, java.lang.String[] categoryValues)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

MainClassifier

public **MainClassifier**()

Method Detail

getClassDefinition

public java.lang.String **getClassDefinition**(java.lang.String val)

Specified by:

getClassDefinition in interface DMEClassifier

---

getClassId

public int **getClassId**()

Specified by:

getClassId in interface DMEClassifier

---

getClassifier

public weka.classifiers.Classifier **getClassifier**()

Specified by:

getClassifier in interface DMEClassifier

---

isClassifierLoaded

public boolean **isClassifierLoaded**()

Specified by:

isClassifierLoaded in interface DMEClassifier

---

predictClause

public java.lang.String **predictClause**(java.lang.String valueStr)

Specified by:

predictClause in interface DMEClassifier

---

setClassifier

public void **setClassifier**(weka.classifiers.Classifier classifier)

Specified by:

setClassifier in interface DMEClassifier

---

train

public void **train**(java.lang.String[] categoryNames,
java.lang.String[] categoryValues)

Specified by:

train in interface DMEClassifier

---

testIt

public java.lang.StringBuffer **testIt**(java.lang.String[] value)

Specified by:

testIt in interface DMEClassifier

org.dogus.datamining.driver

Class DataClassifier

java.lang.Object
└ **org.dogus.datamining.driver.DataClassifier**

---

public class **DataClassifier**
extends java.lang.Object

Field Summary	
static java.lang.String[]	**categoryValues**
static float	**infoGainRate**
static java.lang.String	**OTHER_CAT**

Constructor Summary
DataClassifier()

Method Summary	
java.lang.String[]	**getClassifierNames**()
int	**getNumberOfCategory**()
void	**setClassifierNames**(java.lang.String[] classifierNames)
void	**setNumberOfCategory**(int numberOfCategory)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

infoGainRate

public static final float **infoGainRate**

See Also:

Constant Field Values

OTHER_CAT

public static final java.lang.String **OTHER_CAT**

See Also:

Constant Field Values

---

categoryValues

public static final java.lang.String[] **categoryValues**

Constructor Detail

DataClassifier

public **DataClassifier**()

Method Detail

getNumberOfCategory

public int **getNumberOfCategory**()

---

setNumberOfCategory

public void **setNumberOfCategory**(int numberOfCategory)

---

getClassifierNames

public java.lang.String[] **getClassifierNames**()

---

setClassifierNames

public void **setClassifierNames**(java.lang.String[] classifierNames)

org.dogus.datamining.driver

Class MultiLayerPerceptronClassifier

java.lang.Object
└ org.dogus.datamining.driver.MainClassifier
└ **org.dogus.datamining.driver.MultiLayerPerceptronClassifier**

All Implemented Interfaces:

DMEClassifier

---

public class **MultiLayerPerceptronClassifier**
extends MainClassifier
implements DMEClassifier

Author:

ArzuBT Dogus University,2009 Usage Data Class:
MultiLayerPerceptronClassifier

Constructor Summary
MultiLayerPerceptronClassifier()
MultiLayerPerceptronClassifier(MLPConfiguration _config)

Method Summary	
java.lang.String	**getClassDefinition**(java.lang.String val)
int	**getClassId**()
weka.classifiers.Classifier	**getClassifier**()
weka.classifiers.functions.MultilayerPerceptron	**getMultiLayerPerceptronClassifier**()
boolean	**isClassifierLoaded**()
java.lang.String	**predictClause**(java.lang.String valueStr)

void	**setClassifier**(weka.classifiers.Classifier classifier)
void	**setClassifier**(weka.classifiers.functions.MultilayerPerceptron classifier)
java.lang.StringBuffer	**testIt**(java.lang.String[] value)
void	**testItExtra**(java.lang.String[] testText)
void	**train**(java.lang.String[] categoryNames, java.lang.String[] categoryValues)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

MultiLayerPerceptronClassifier

public **MultiLayerPerceptronClassifier**(MLPConfiguration _config)

MultiLayerPerceptronClassifier

public **MultiLayerPerceptronClassifier**()

Method Detail

getClassDefinition

public java.lang.String **getClassDefinition**(java.lang.String val)

Specified by:

getClassDefinition in interface DMEClassifier

Overrides:

getClassDefinition in class MainClassifier

---

getClassifier

public weka.classifiers.Classifier **getClassifier**()

Specified by:

getClassifier in interface DMEClassifier

Overrides:

getClassifier in class MainClassifier

---

getMultiLayerPerceptronClassifier

public weka.classifiers.functions.MultilayerPerceptron **getMultiLayerPerceptronClassifier**()

---

setClassifier

public void **setClassifier**(weka.classifiers.functions.MultilayerPerceptron classifier)

---

train

public void **train**(java.lang.String[] categoryNames,
java.lang.String[] categoryValues)

Specified by:

train in interface DMEClassifier

Overrides:

train in class MainClassifier

---

setClassifier

public void **setClassifier**(weka.classifiers.Classifier classifier)

Specified by:

setClassifier in interface DMEClassifier

Overrides:

setClassifier in class MainClassifier

---

predictClause

public java.lang.String **predictClause**(java.lang.String valueStr)

Specified by:

predictClause in interface DMEClassifier

Overrides:

predictClause in class MainClassifier

---

testIt

public java.lang.StringBuffer **testIt**(java.lang.String[] value)

Specified by:

testIt in interface DMEClassifier

Overrides:

testIt in class MainClassifier

---

isClassifierLoaded

public boolean **isClassifierLoaded**()

Specified by:

isClassifierLoaded in interface DMEClassifier

Overrides:

isClassifierLoaded in class MainClassifier

---

getClassId

public int **getClassId**()

Specified by:

getClassId in interface DMEClassifier

Overrides:

getClassId in class MainClassifier

---

testItExtra

public void **testItExtra**(java.lang.String[] testText)

org.dogus.datamining.driver

Class UtilDriver

java.lang.Object
└ **org.dogus.datamining.driver.UtilDriver**

---

public class **UtilDriver**
extends java.lang.Object

Constructor Summary
UtilDriver()
UtilDriver(weka.classifiers.Classifier wekaClassifier)

Method Summary	
void	**evaluate**(weka.core.Instances instance)

java.lang.String[]	**filterData**(java.lang.String[] arrCategory, java.lang.String[] arrCategoryVal) method has a dependency to inValidList method.
static weka.core.Instances	**filterText**(weka.core.Instances theseInstances)
java.lang.String	**getCategory**(weka.classifiers.Classifier classifier, java.lang.String val, weka.core.Instances instances, weka.core.Attribute classAttribute)
static java.lang.String	**getCatName**(weka.core.Instances theseInstances, weka.classifiers.Classifier thisClassifier, weka.core.Attribute thisClassAttribute)
java.lang.String	**getPredicted**(weka.classifiers.Classifier classifier, weka.core.Instances instance)
java.lang.String[]	**getValidList**()
boolean	**inValidList**(java.lang.String value)
static weka.core.Instances	**populateInstances**(java.lang.String[] theseInputTexts, java.lang.String[] theseInputClasses, weka.core.Instances theseInstances, weka.core.Attribute classAttribute, weka.core.Attribute textAttribute)
void	**setValidList**(java.lang.String[] validList)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

UtilDriver

public **UtilDriver**(weka.classifiers.Classifier wekaClassifier)

UtilDriver

public **UtilDriver**()

Method Detail

---

getCategory

public java.lang.String **getCategory**(weka.classifiers.Classifier classifier,
java.lang.String val,
weka.core.Instances instances,
weka.core.Attribute classAttribute)

---

populateInstances

public static weka.core.Instances **populateInstances**(java.lang.String[] theseInputTexts,
java.lang.String[] theseInputClasses,
weka.core.Instances theseInstances,
weka.core.Attribute classAttribute,
weka.core.Attribute textAttribute)

---

filterText

public static weka.core.Instances **filterText**(weka.core.Instances theseInstances)

---

getCatName

public static java.lang.String **getCatName**(weka.core.Instances theseInstances,
weka.classifiers.Classifier thisClassifier,
weka.core.Attribute thisClassAttribute)

---

evaluate

public void **evaluate**(weka.core.Instances instance)

---

getPredicted

public java.lang.String **getPredicted**(weka.classifiers.Classifier classifier,
weka.core.Instances instance)

---

inValidList

public boolean **inValidList**(java.lang.String value)

---

filterData

public java.lang.String[] **filterData**(java.lang.String[] arrCategory,

java.lang.String[] arrCategoryVal)
method has a dependency to inValidList method.

Parameters:

arrCategory -

arrCategoryVal -

Returns:

---

getValidList

public java.lang.String[] **getValidList**()

---

setValidList

public void **setValidList**(java.lang.String[] validList)

org.dogus.servlet

Class TurtleController

java.lang.Object
└ HttpServlet
└ **org.dogus.servlet.TurtleController**

---

public class **TurtleController**
extends HttpServlet

Constructor Summary
TurtleController()

Method Summary	
void	**service**(HttpServletRequest oReq, HttpServletResponse oRes)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TurtleController

public **TurtleController**()

Method Detail

service

public void **service**(HttpServletRequest oReq,
HttpServletResponse oRes)

org.dogus.servlet

Class TurtleClassifierController

java.lang.Object
└ HttpServlet
└ **org.dogus.servlet.TurtleClassifierController**

---

public class **TurtleClassifierController**
extends HttpServlet

Constructor Summary
TurtleClassifierController()

Method Summary	
java.util.List<DocumentCat>	**getResultCat**(HttpServletRequest req, HttpServletResponse res)
java.util.List<Document>	**getResults**(HttpServletRequest req, HttpServletResponse res) According to search criteria making search
void	**service**(HttpServletRequest oReq, HttpServletResponse oRes)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TurtleClassifierController

public **TurtleClassifierController**()

Method Detail

service

public void **service**(HttpServletRequest oReq,
HttpServletResponse oRes)

---

getResults

public java.util.List<Document> **getResults**(HttpServletRequest req,
HttpServletResponse res)

According to search criteria making search

Parameters:

req -

res -

Returns:

---

getResultCat

public java.util.List<DocumentCat> **getResultCat**(HttpServletRequest req,
HttpServletResponse res)

ARZU BEHİYE TARIMCI

E-mail : arzubt@gmail.com
Web Site: http://www.kansersite.info

Personal Information

Date of Birth:	23 / May / 1979
Place of Birth:	İstanbul

Education

2006 -	**Dogus University - Institute of Science and Technology –** Computer and Information Science Master Program
2000(Fall)	**Bogazici Üniversity - MIS Management Information Systems Certificate Program Rumeli Hisarüstü – İstanbul**
1999 (Fall)	**Kocaeli University - Industrial Engineering Department – İzmit Kocaeli** Graduate Thesis : ERP and SAP R/3 MM Module
1995	**Erenköy İntaş HighSchool -** Erenköy İstanbul