12-2011

Study of feature selection algorithms for text-categorization

Kandarp Dave
University of Nevada, Las Vegas

STUDY OF FEATURE SELECTION ALGORITHMS FOR TEXT-CATEGORIZATION

By

Kandarp Dave

Bachelor of Science
University of Nevada, Las Vegas
2009

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2011

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Kandarp Dave

entitled

Study of Feature Selection Algorithms for Text-Categorization

be accepted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Howard R. Hughes College of Engineering

Kazem Taghva, Committee Chair

Laxmi Gewali, Committee Member

Ajoy Datta, Committee Member

Venki Mukhukumar, Graduate College Representative

Ronald Smith, Ph. D., Vice President for Research and Graduate Studies
and Dean of the Graduate College

December 2011

ABSTRACT

STUDY OF FEATURE SELECTION ALGORITHMS FOR TEXT-CATEGORIZATION

By

Kandarp Dave

Dr. Kazem Taghva, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

This thesis will discuss feature selection algorithms for text-categorization. Feature selection algorithms are very important, as they can make-or-break a categorization engine. The feature selection algorithms that will be discussed in this thesis are *Document Frequency*, *Information Gain*, *Chi Squared*, *Mutual Information*, *NGL (Ng-Goh-Low) coefficient*, and *GSS (Galavotti-Sebastiani-Simi) coefficient*. The general idea of any feature selection algorithm is to determine importance of words using some measure that can *keep* informative words, and *remove* non-informative words, which can then help the text-categorization engine categorize a document, D, into some category, C. These feature selection methods are explained, implemented, and are provided results for in this thesis. This thesis also discusses how we gathered and constructed training and testing data, along with the setup and storage techniques we used.

TABLE OF CONTENTS

ABSTRACT iii

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 DATA COLLECTION 4
- 2.1 Setup 4
- 2.2 Categories 5
- 2.3 Collecting Data 6
- 2.4 Tagging Documents 13
- 2.5 Database Setup 14
  - 2.5.1 Table “folder” 15
  - 2.5.2 Table “selection” 16
  - 2.5.3 Table “selection_folder” 18
  - 2.5.4 Table “selection_test” 19
  - 2.5.5 Table “selection_folder_test” 21
  - 2.5.6 Table “bow” 22
  - 2.5.7 Table “doc_bow_raw” 24
  - 2.5.8 Table “doc_bow_raw_test” 27
- 2.6 Creating Bag of Words for Training Dataset - “bow” Table 29
  - 2.6.1 The b8 Lexer 29
  - 2.6.2 Putting Code Together and Creating BOW 32
  - 2.6.3 Importance of the “BOW” Table 36
- 2.7 Creating Bag of Words for Training Dataset - “doc_bow_raw” Table 37
- 2.8 Creating Bag of Words for Test Dataset - “doc_bow_raw_test” Table 39

CHAPTER 3 COUNTING DOCUMENTS 42
- 3.1 The A, B, C, D Values 42
  - 3.1.1 Explanation of the A, B, C, D Values 42
  - 3.1.2 Calculating and Updating “doc_bow_raw” with the A, B, C, D Values 43

CHAPTER 4 FEATURE SELECTION ALGORITHMS 47
- 4.1 Why Use Feature Selection? 47
- 4.2 Feature Selection Algorithm Explanations 49
  - 4.2.1 Document Frequency - Explanation 49
  - 4.2.2 Information Gain - Explanation 50
  - 4.2.3 Mutual Information - Explanation 50
  - 4.2.4 Chi Square - Explanation 51
  - 4.2.5 NGL (Ng-Goh-Low) Coefficient - Explanation 51
  - 4.2.6 GSS (Galavotti-Sebastiani-Simi) Coefficient - Explanation 52

CHAPTER 5 IMPLEMENTATION OF FEATURE SELECTION ALGORITHMS 53
- 5.1 Document Frequency - Calculation Implementation Details 55
- 5.2 Information Gain - Calculation Implementation Details 56

5.3 Mutual Information - Calculation Implementation Details 57
5.4 Chi Square - Calculation Implementation Details 58
5.5 NGL - Calculation Implementation Details 59
5.6 GSS - Calculation Implementation Details 60

CHAPTER 6 RESULTS 61
6.1 Using All Features 61
6.2 Using Selected Features 64
6.2.1 Document Frequency - Selection Partial Features 64
6.2.2 Information Gain - Selection Partial Features 65
6.2.3 Mutual Information - Selection Partial Features 65
6.2.4 χ^2 - Selection Partial Features 67
6.2.5 NGL Coefficient - Selection Partial Features 69
6.2.6 GSS Coefficient - Selection Partial Features 71
6.3 Explanation of Results 73

CHAPTER 7 CONCLUSION AND FUTURE WORK 74

BIBLIOGRAPHY 77

VITA 79

CHAPTER 1

INTRODUCTION

With the growth of online information, text-categorization has become a very important technology to categorize a large number of documents. The idea of text-categorization, or text-classification, is to categorize textual data into one or more pre-defined categories [1, 2, 3, 4]. Given a set of documents, D, and some pre-determined set of categories, C, the idea of text-categorization is to categorize documents, D, into appropriate categories, C, as best as possible. Text-categorization is a "supervised technique that uses labeled training data to learn the classification system and then automatically classifies the remaining text using the learned system" [4].

Feature selection is an important part of text-categorization, and much research has been done on various feature selection algorithms. Feature selection can be thought of as selecting the best words of a document that can help categorize that document. As a very simple example, when a human is reading some document that contains words such as "iPhone", "iPad", "iPod", or "Mac", he or she can easily determine that if a category related to *technology* named "Apple" exists, then this document must belong in that category. The idea of feature selection, in simple words, is to determine importance of words using some measure that can *keep* informative words, and *remove* non-informative words, which can then help the text-categorization engine.

As adult humans, we have already been trained to put, for example, "iPhone" related documents in "Apple" (technology related) category, but to train an algorithm to pick up *informative* words is a different story, and there are many steps to this process. First, some categories have to be pre-defined that can be used for both training and testing. Once the training documents are gathered, and categories are determined, documents have to be tagged with an appropriate category. In the real world, a document can belong to multiple categories, but for simplicity, we will tag a document with only one category. Once the training documents are tagged, a *bag of words*, or *BOW*, can be created, which can be used to categorize test documents. From the *BOW*, we can *keep* informative words, and *remove* non-informative words, and the idea of choosing informative words, and removing the rest is called Feature Selection. The feature selection methods that are studied, implemented, and provided results for, are the following: *Document Frequency*, *Information Gain*, *Mutual Information*, *Chi Square*, *NGL (Ng-Goh-Low) Coefficient*, and *GSS (Galavotti-Sebastiani-Simi) Coefficient*. These algorithms have been studied before, mainly on Reuters and Newsgroup input data. We did not use Reuters or Newsgroup data, instead, for our needs, we built custom (mixed) data ourselves.

Gathering these training and testing documents is a difficult task, but keeping all the gathered data organized is also difficult. What setup we used to address document organization issue is discussed in chapter 2. Chapter 2 also gives details about how we gathered data, how we stored all information, and how appropriate data-structures were created so that feature selection algorithms can be run easily and efficiently. Chapter 3 builds the core of this thesis that helps all the feature selection algorithms mentioned. Chapter 3 talks about how documents are counted, and how the *counts* are stored. Chapter 4 explains why feature selection algorithms need to be used, and gives explanations about each algorithm. Chapter 5 gives implementation details for each feature selection algorithm. Chapter 6 shows all the results we achieved. The thesis is concluded in chapter 7. Before we begin to explain, please note that the following pairs of words are used interchangeably throughout this thesis:

- "selection" and "document"
- "folder" and "category"

CHAPTER 2

DATA COLLECTION

2.1 Setup

Choosing a correct setup for text-categorization is an important step, as explained by Dasgupta, Drineas, Harb, Josifovski, and Mahoney that “challenges associated with automated text categorization come from many fronts: one must choose an appropriate data structure to represent the documents” [5]. There were multiple factors involved in choosing what environment to use for training data and testing against the trained model. We chose to use PHP, as it takes care of minute details of implementation by providing high-level interfaces, and objects such as arrays that can be associative. It would not have made much difference whether we had used C++, as most of the calculations had to be done database side. There were parts where calculation had to be done using an application layer and not the database layer. Such calculations include calculating Chi Square, Information Gain, Mutual Information, NGL, and GSS values. Most of these calculations took only about 5 seconds, including retrieving data from storage, calculating values, and updating the result values back in the storage space.

This brings me to the next part: Storage. For storage, we used a MySQL database. We chose not to use plain text files due to the amount of data we knew we had to deal with. MySQL and PHP work very well together as PHP has a built in connector that can easily access MySQL database. Also, it is much easier to insert, update, and retrieve thousands of rows of data into and out of MySQL, especially with transaction ability, and SQL. Another big positive point with using MySQL is that we can easily index data, which makes searching of the indexed columns really fast. MySQL turned out to be the best choice to use than to store data in plain text files for the provided reasons.

2.2 Categories

Before any data collection could begin, we had to determine what categories we wanted to work with. We did not want to have categories that were all completely separate from each other. Categories “Technology” and “Food” are considered very separate from each other. Categories such as “JavaScript” and “PHP” are considered very close to each other. Meaning, we wanted have a mix of categories where some categories would be very close to each other and some other categories that would be very separate from each other. Here is the list of categories we worked with:

List of Categories - List 1

- Chinese - Food.
- Indian - Food.
- Italian - Food.
- India - General news.
- Apple - Technology news.
- Google - Technology news.
- Facebook - Technology news.
- PHP - Technology.
- JavaScript - Technology.

As can be seen from the list of categories, “Chinese”, “Indian”, and “Italian” are very close to each other; consider them in group 1. Consider category “India” in group 2. Categories “Apple”, “Google”, and “Facebook” are related as well; so we will put them in group 3. Categories “PHP”, and “JavaScript” will go in group 4 as those categories are related. Now it's easy to see that group 1, 2, 3, and 4 are all very separate from each other. Having such a mix of categories helps determine how well a categorization algorithm is, and how well feature selections methods are. Now that we have categories, we can collect data.

2.3 Collecting Data

Data was manually gathered from various online sources (websites) with the help of some utilities, made by us, that could help us gather data faster. 1,010 training documents and 338 testing documents were manually collected and tagged.

Here is the unique list of subdomain-domain names from which we gathered *training* data:

List of Subdomain-Domain Names Used for Training Data - List 2

- rasamalaysia.com
- homechineserecipes.com
- food.com
- indianfoodforever.com
- thanksgiving.food.com
- chinese.food.com
- eatingchina.com
- allrecipes.com
- manjulaskitchen.com
- sanjeevkapoor.com
- italianfoodforever.com
- timesofindia.feedsportal.com (*)
- zeenews.com
- feedproxy.google.com (*)
- computerworld.com
- mashable.com
- feeds.appleinsider.com (*)

- php.net
- w3schools.com

Here is the unique list of subdomain-domain names from which we gathered *testing* data:

List of Subdomain-Domain Names Used for Testing Data - List 3

- chinese-food-recipes.net
- recipesindian.com
- italianfoodsrecipes.com
- italianhomerecipes.com
- timesofindia.feedsportal.com (*)
- zeenews.india.com
- feedproxy.google.com (*)
- mashable.com
- php.net
- developer.mozilla.org

Items marked with (*) are not actual webpages, but each serves as a pointer to some other webpage. Such subdomain-domain names are *NOT* very descriptive compared to other non-marked domains. For example, consider a feed URL

```
http://timesofindia.feedsportal.com/fy/8at2EtY0RyNP70tD/story01.htm
```

and a non-feed URL

```
http://www.zeenews.com/news/nation/india-china-to-see-growth-in-
n-energy-sector_732607.html.
```

It is very clear that the latter, the non-feed URL, is much more descriptive, and some information can already be guessed about the content of the page from the URL.

To gather training data, 1,010 documents were determined that were appropriate enough to be categorized under the categories listed in List 1. First the URL for each of these these training documents was gathered. Initially, we wrote a utility program that went through all 1,010 documents, and tried to gather text-content from them. It was noticed that most documents' text-content included much unnecessary data that would break the text-categorization engine and would not be of any help to any feature selection algorithm. This data was not used. So, we decided to manually go through all 1,010 training documents, and gathered text-content from each document that best represented that document's category. Below are 3 sample documents with URL, title, meta-information, and text-content.

Sample document 1 - Category "Google":

URL: `http://mashable.com/2011/09/08/google-acquires-zagat/`
Title: `Google Acquires Zagat`
Meta: `Google has placed one of its biggest bets on location to date, acquiring local reviews giant Zagat.`
Text-content:

```
Google has placed one of its biggest bets on location to date,
acquiring local reviews giant Zagat.

Writing on the companys official blog, Marissa Mayer, Googles vice
president of Local, Maps and Location Services, wrote, Moving forward,
Zagat will be a cornerstone of our local offering  delighting people
with their impressive array of reviews, ratings and insights, while
enabling people everywhere to find extraordinary (and ordinary)
experiences around the corner and around the world.

Zagat is far cry from the startups typically mentioned in the location
space. The company was founded 32 years ago and started as a printed
guide to restaurants, with Zagat Ratings becoming an industry standard.

More recently, however, Zagat has reinvented itself on the web and
with mobile apps, bringing it into competition with the likes of Foursquare
and Yelp.
```

Location has been a tough nut for Google to crack. The company acquired early location-based social networking service Dodgeball in 2005, only to eventually shut it down and see founder Dennis Crowley leave to start Foursquare. More recent attempts include Latitude, a largely forgotten Foursquare competitor, and Hotpot, a recommendation engine thats baked into Google Places. The company also appointed Mayer, one of its most prominent executives, to lead its location efforts in late 2010.

While we dont have a price tag on the Zagat acquisition yet, its safe to call the buy one of Googles biggest to date in the content business. Heres a look at some of Googles largest acquisitions through the years:

Sample document 2 - Category “JavaScript”:

URL: http://www.w3schools.com/js/js_obj_string.asp
Title: JavaScript String object
Meta: No meta attached with this document.
Text-content:

The String object is used to manipulate a stored piece of text.

Try it Yourself - Examples

Return the length of a string
How to return the length of a string.

Style strings
How to style strings.

The toLowerCase() and toUpperCase() methods
How to convert a string to lowercase or uppercase letters.

The match() method
How to search for a specified value within a string.

Replace characters in a string - replace()
How to replace a specified value with another value in a string.

The indexOf() method
How to return the position of the first found occurrence of a specified value in a string.

Complete String Object Reference

For a complete reference of all the properties and methods that can be used with the String object, go to our complete String object reference.

The reference contains a brief description and examples of use for each property and method!

String object

The String object is used to manipulate a stored piece of text.

Examples of use:

The following example uses the length property of the String object to find the length of a string:

```
var txt="Hello world!";
document.write(txt.length);
```

The code above will result in the following output:

```
12
```

The following example uses the toUpperCase() method of the String object to convert a string to uppercase letters:

```
var txt="Hello world!";
document.write(txt.toUpperCase());
```

The code above will result in the following output:

```
HELLO WORLD!
```

Sample document 3 - Category “Indian” (food):

URL: http://www.manjulaskitchen.com/2008/08/12/malai-kofta/
Title: Malai Kofta | Manjula’s Kitchen | Indian Vegetarian Recipes | Indian Cooking Videos
Meta: Malai kofta is a delicious and rich main dish for any special occasion.
Text-content:

Malai kofta is a delicious and rich main dish for any special occasion.

Recipe serves 4 to 6.

Ingredients:

Kofta:

1 cup boiled mash potatoes
1 cup mash paneer
2 tablespoon minced cilantro (hara dhania)
1/2 teaspoon cumin seed (jeera)
1/4 teaspoon salt
1 small finally chopped green chili
For Batter:

2 tablespoon all purpose flour (maida, plain flour)
4 tablespoon water
Also needed:

Oil to fry
Gravy:
2 tablespoon oil
Generous pinch asafetida (hing)
1 teaspoon cumin seed (jeera)
2 medium tomatoes
1 tablespoon shredded ginger (adrak)
1 green chili
1 tablespoon coriander powder (dhania)
1/2 teaspoon turmeric (haldi)
1/4 teaspoon red chili powder
1 teaspoon all purpose flour (maida, plain flour)
1/4 cup cream
1/2 teaspoon salt (adjust to taste)
1/4 teaspoon garam masala
2 tablespoon minced cilantro (hara dhania)
Method

Kofta

Mix all the ingredients together for kofta,
With oiled hands, divide the mixture into 14 to 16 equal parts.
Make them in round balls.
Mix flour with about 4 tablespoons of water and mix well until batter is smooth.
Heat the oil in a frying pan on medium high heat.
The frying pan should have at least 1 1/2 inch of oil. To check if the oil is ready, just put one small piece of mix in the oil, it should come up right away but not change color.
Dip the paneer balls in the batter one at a time and slowly drop into the frying pan.
Turn them occasionally. Fry koftas until golden-brown all around.
Gravy:

```
Blend the tomatoes, green chilies and ginger to make a puree. If
you prefer mild take the seeds out of green chili before blending.
Mix cream and flour and keep aside.
Heat the oil in a saucepan. Test the heat by adding one cumin
seed to the oil; if it cracks right away oil is ready.
Add the hing and cumin seeds.
Add the tomato puree, coriander powder, turmeric, red chili powder
and cook for about 4 minutes on medium heat.
Tomato mixture will start leaving the oil and will reduce to about half
in quantity.
Add milk and flour mix, salt and one cup of water and let it cook
covered for 7 to 8 minutes on medium heat.
Add the garam masala, salt and cilantro. Let it cook for another minute.
Add koftas as soon it comes to boil turn off the heat. Note: koftas will
expand to about 11/2 times, if you like more gravy this is the time to
add some more boiled water and adjust salt.
Koftas are very soft they should be added to the gravy when you are
ready to serve, otherwise koftas will break.
```

Call this text-content *SELECTION_DETAIL*. Each webpage also has some other helpful properties associated with it. Properties such as URL, meta, and title can also help in categorization of the document. To gather these properties, a JavaScript-PHP utility program was developed. This utility contained URL for each of the 1,010 documents. The utility would go through each item in the list, visit the webpage, and bring back meta information, title, and URL of the document, and update the database with this retrieved information.

The same strategy was used for test data. 338 documents were determined, and URL for each document was collected. We manually went through 338 documents to collect text-content, and using the same utility mentioned above, meta information, title, and URL were retrieved for each document, and data was updated in the database.

2.4 Tagging Documents

Tagging of training documents is one of the most important part of text-categorization, as it will help in training of the text-categorization engine. If done incorrectly, text-categorization engine will give wrong results. Thus, all documents that we gathered had to be tagged manually by the experts, Kandarp Dave and Dr. Taghva. For each of the 1,010 training and 338 test documents, the JavaScript utility mentioned above also included an array slot for the best possible Category in which the document should belong. The following is the sample code:

Listing 1: Sample JavaScript URL-Category Array Code

```
var linkArray = new Array(
   new Array(
      'http://allrecipes.com/recipe/indian-chapati-bread/detail.aspx',
      '2'
   ),
   new Array(
      'http://allrecipes.com/recipe/indian-sweet-bread/detail.aspx',
      '2'
   ),
   new Array(
      'http://allrecipes.com/recipe/naan/detail.aspx'
      ,'2'
   ),
   new Array(
      'http://allrecipes.com/recipe/naan-bread/detail.aspx',
      '2'
   ),
   new Array(
      'http://www.php.net/manual/en/language.operators.string.php',
      '22'
   ),
   new Array(
      'http://www.php.net/manual/en/language.operators.array.php',
      '22'
   ),
   new Array(
      'http://www.php.net/manual/en/language.control-structures.php',
      '22'
   ),
   new Array(
      'http://www.php.net/manual/en/control-structures.elseif.php',
      '22'
   )
);
```

As it can be seen from the sample code, documents of “allrecipes.com” belong to category “2”, which is the primary key for category “Indian” food in the database. Documents of “php.net” belong to category “22”, which is the primary key for category “PHP” in the database. This is only sample code, but the *linkArray* contained 1,010 tagged-items for training. All 338 test documents were also manually tagged the same way to determine if the results of the text-categorization algorithm matched with what is expected.

2.5 Database Setup

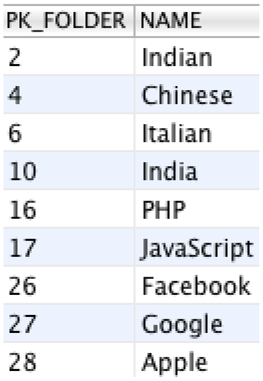
As mentioned, we used a database for storage, specifically a MySQL database. The name of the schema is “thesis”. In this database, we set up appropriate tables. These tables, along with column names and their descriptions, are explained in each section below.

2.5.1 Table “folder”

“folder” table can be thought of as one that contains categories.

The columns are:

- PK_FOLDER - A primary key column for table “folder”.
- NAME - Name of the category.



PK_FOLDER	NAME
2	Indian
4	Chinese
6	Italian
10	India
16	PHP
17	JavaScript
26	Facebook
27	Google
28	Apple

Figure 1: Table “folder”

2.5.2 Table “selection”

“selection” table is the main table that contains all training documents. A selection item can be thought of as a document.

The columns are:

- PK_SELECTION - A primary key column for table “selection”.

- URL - The document’s Universal Resource Locator. Text in this column is used in training of the text-categorization engine.

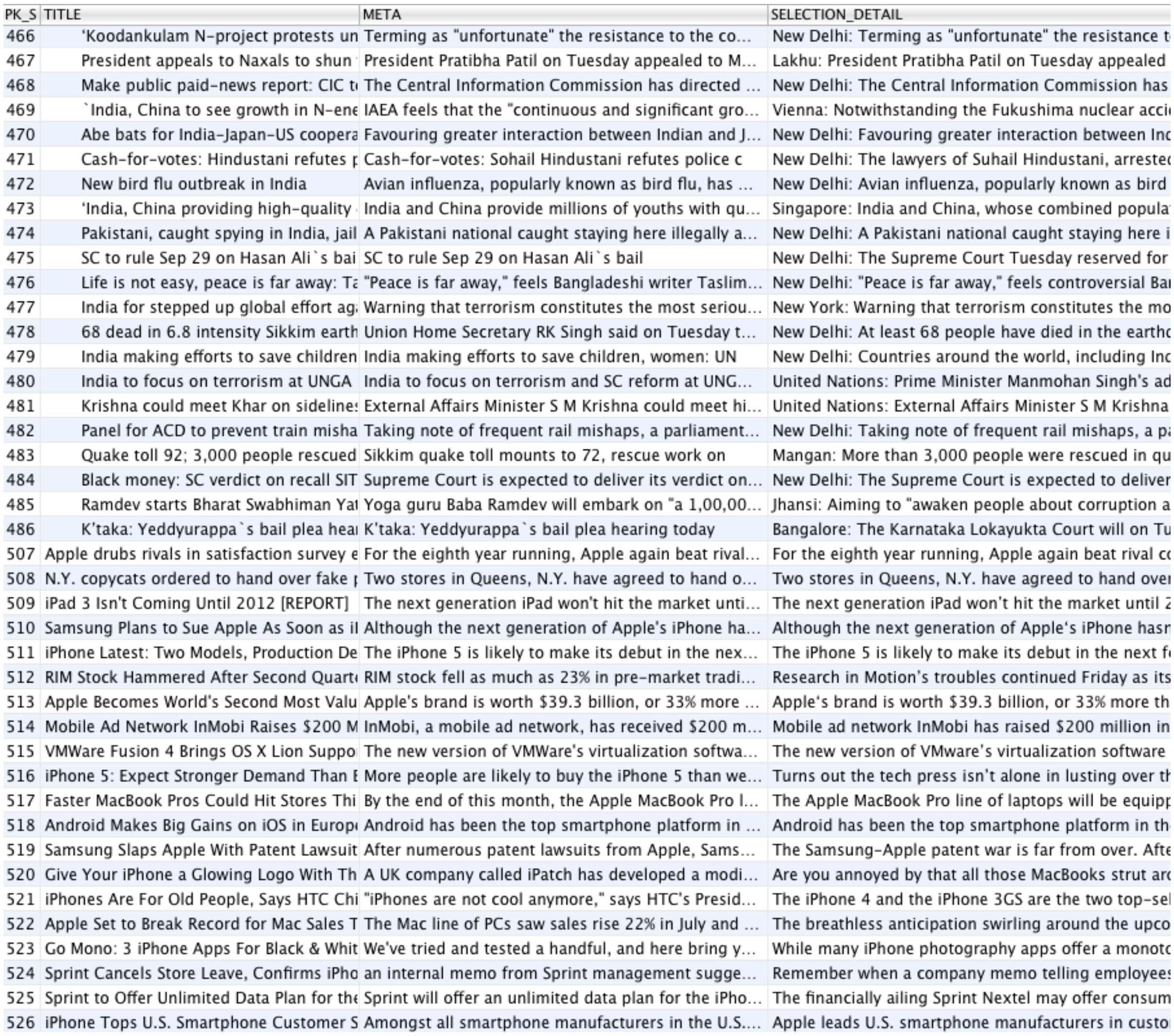
- TITLE - Title of the webpage. Text in this column is used in training of the text-categorization engine.

- META - A webpage can have many meta tags. We use the value of *content* attribute of the meta tag whose *name* attribute has the value “description”. For example,

  ```
  <meta name="description" content="This text will be extracted
  and used for text-categorization." />
  ```

  Text in this column is used in training of the text-categorization engine.

- SELECTION_DETAIL - This column contains the actual text-content extracted manually. Each row contains text-content of that document that can best help the text-categorization engine. Text in this column is used in training of the text-categorization engine.



PK_S	TITLE	META	SELECTION_DETAIL
466	'Koodankulam N-project protests un	Terming as "unfortunate" the resistance to the co…	New Delhi: Terming as "unfortunate" the resistance t
467	President appeals to Naxals to shun	President Pratibha Patil on Tuesday appealed to M…	Lakhu: President Pratibha Patil on Tuesday appealed
468	Make public paid-news report: CIC t	The Central Information Commission has directed …	New Delhi: The Central Information Commission has
469	`India, China to see growth in N-ene	IAEA feels that the "continuous and significant gro…	Vienna: Notwithstanding the Fukushima nuclear acci
470	Abe bats for India-Japan-US coopera	Favouring greater interaction between Indian and J…	New Delhi: Favouring greater interaction between Inc
471	Cash-for-votes: Hindustani refutes p	Cash-for-votes: Sohail Hindustani refutes police c	New Delhi: The lawyers of Suhail Hindustani, arrestec
472	New bird flu outbreak in India	Avian influenza, popularly known as bird flu, has …	New Delhi: Avian influenza, popularly known as bird
473	'India, China providing high-quality	India and China provide millions of youths with qu…	Singapore: India and China, whose combined popula
474	Pakistani, caught spying in India, jail	A Pakistani national caught staying here illegally a…	New Delhi: A Pakistani national caught staying here i
475	SC to rule Sep 29 on Hasan Ali`s bai	SC to rule Sep 29 on Hasan Ali`s bail	New Delhi: The Supreme Court Tuesday reserved for
476	Life is not easy, peace is far away: Ta	"Peace is far away," feels Bangladeshi writer Taslim…	New Delhi: "Peace is far away," feels controversial Bai
477	India for stepped up global effort ag	Warning that terrorism constitutes the most seriou…	New York: Warning that terrorism constitutes the mo
478	68 dead in 6.8 intensity Sikkim earth	Union Home Secretary RK Singh said on Tuesday t…	New Delhi: At least 68 people have died in the earthc
479	India making efforts to save children	India making efforts to save children, women: UN	New Delhi: Countries around the world, including Inc
480	India to focus on terrorism at UNGA	India to focus on terrorism and SC reform at UNG…	United Nations: Prime Minister Manmohan Singh's ad
481	Krishna could meet Khar on sideline:	External Affairs Minister S M Krishna could meet hi…	United Nations: External Affairs Minister S M Krishna
482	Panel for ACD to prevent train misha	Taking note of frequent rail mishaps, a parliament…	New Delhi: Taking note of frequent rail mishaps, a pa
483	Quake toll 92; 3,000 people rescued	Sikkim quake toll mounts to 72, rescue work on	Mangan: More than 3,000 people were rescued in qu
484	Black money: SC verdict on recall SIT	Supreme Court is expected to deliver its verdict on…	New Delhi: The Supreme Court is expected to deliver
485	Ramdev starts Bharat Swabhiman Yat	Yoga guru Baba Ramdev will embark on "a 1,00,00…	Jhansi: Aiming to "awaken people about corruption a
486	K'taka: Yeddyurappa`s bail plea hea	K'taka: Yeddyurappa`s bail plea hearing today	Bangalore: The Karnataka Lokayukta Court will on Tu
507	Apple drubs rivals in satisfaction survey e	For the eighth year running, Apple again beat rival…	For the eighth year running, Apple again beat rival cc
508	N.Y. copycats ordered to hand over fake p	Two stores in Queens, N.Y. have agreed to hand o…	Two stores in Queens, N.Y. have agreed to hand over
509	iPad 3 Isn't Coming Until 2012 [REPORT]	The next generation iPad won't hit the market unti…	The next generation iPad won't hit the market until 2
510	Samsung Plans to Sue Apple As Soon as il	Although the next generation of Apple's iPhone ha…	Although the next generation of Apple's iPhone hasr
511	iPhone Latest: Two Models, Production De	The iPhone 5 is likely to make its debut in the nex…	The iPhone 5 is likely to make its debut in the next fi
512	RIM Stock Hammered After Second Quarte	RIM stock fell as much as 23% in pre-market tradi…	Research in Motion's troubles continued Friday as its
513	Apple Becomes World's Second Most Valu	Apple's brand is worth $39.3 billion, or 33% more …	Apple's brand is worth $39.3 billion, or 33% more th
514	Mobile Ad Network InMobi Raises $200 M	InMobi, a mobile ad network, has received $200 m…	Mobile ad network InMobi has raised $200 million in
515	VMWare Fusion 4 Brings OS X Lion Suppo	The new version of VMWare's virtualization softwa…	The new version of VMware's virtualization software
516	iPhone 5: Expect Stronger Demand Than E	More people are likely to buy the iPhone 5 than we…	Turns out the tech press isn't alone in lusting over th
517	Faster MacBook Pros Could Hit Stores Thi	By the end of this month, the Apple MacBook Pro l…	The Apple MacBook Pro line of laptops will be equipp
518	Android Makes Big Gains on iOS in Europe	Android has been the top smartphone platform in …	Android has been the top smartphone platform in th
519	Samsung Slaps Apple With Patent Lawsuit	After numerous patent lawsuits from Apple, Sams…	The Samsung-Apple patent war is far from over. Afte
520	Give Your iPhone a Glowing Logo With Th	A UK company called iPatch has developed a modi…	Are you annoyed by that all those MacBooks strut arc
521	iPhones Are For Old People, Says HTC Chi	"iPhones are not cool anymore," says HTC's Presid…	The iPhone 4 and the iPhone 3GS are the two top-sel
522	Apple Set to Break Record for Mac Sales T	The Mac line of PCs saw sales rise 22% in July and …	The breathless anticipation swirling around the upco
523	Go Mono: 3 iPhone Apps For Black & Whit	We've tried and tested a handful, and here bring y…	While many iPhone photography apps offer a monotc
524	Sprint Cancels Store Leave, Confirms iPho	an internal memo from Sprint management sugge…	Remember when a company memo telling employees
525	Sprint to Offer Unlimited Data Plan for the	Sprint will offer an unlimited data plan for the iPho…	The financially ailing Sprint Nextel may offer consum
526	iPhone Tops U.S. Smartphone Customer S	Amongst all smartphone manufacturers in the U.S.…	Apple leads U.S. smartphone manufacturers in custo

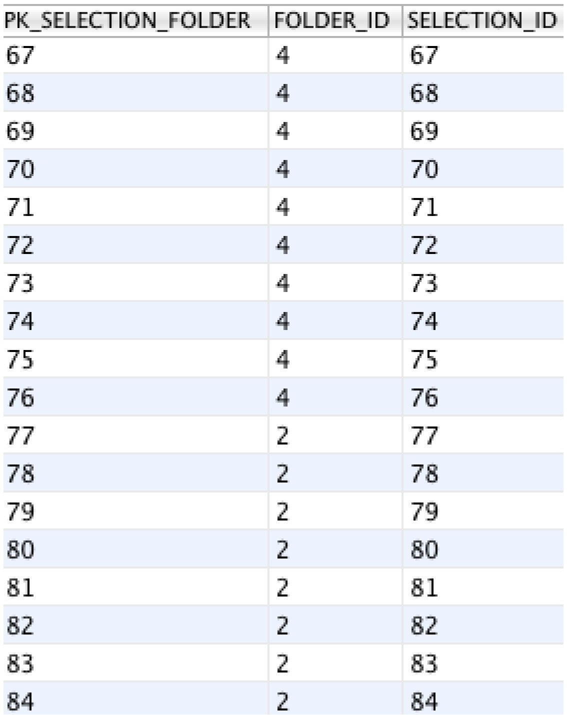
Figure 2: Table “selection”

2.5.3 Table “selection_folder”

“selection_folder” table is a linking table that links training documents to categories.

The columns are:

- PK_SELECTION_FOLDER - A primary key column for table “selection_folder”.
- FOLDER_ID - ID of the folder to which this training document belongs.
- SELECTION_ID - The ID of the document.



PK_SELECTION_FOLDER	FOLDER_ID	SELECTION_ID
67	4	67
68	4	68
69	4	69
70	4	70
71	4	71
72	4	72
73	4	73
74	4	74
75	4	75
76	4	76
77	2	77
78	2	78
79	2	79
80	2	80
81	2	81
82	2	82
83	2	83
84	2	84

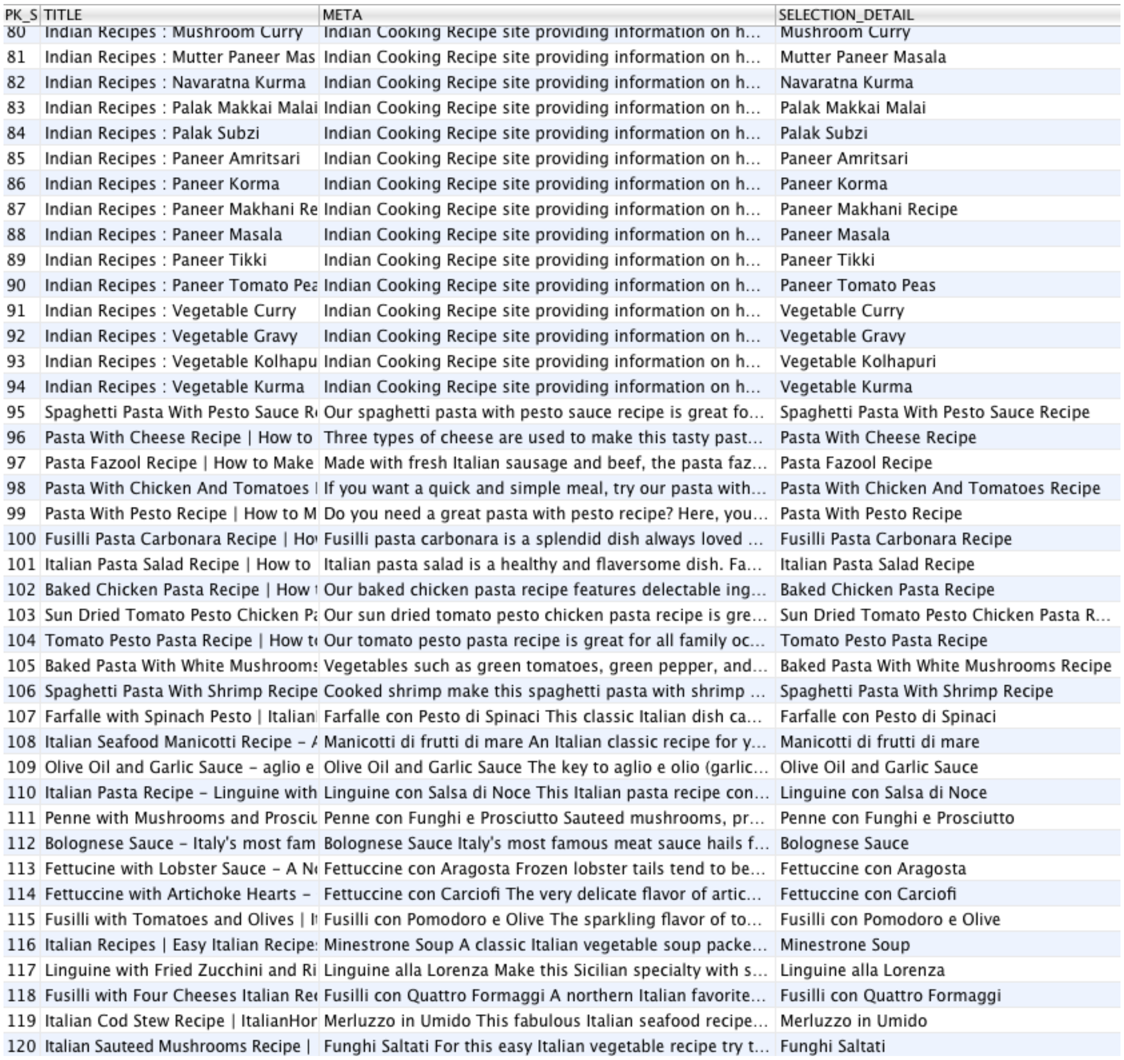
Figure 3: Table “selection_folder”

2.5.4 Table “selection_test”

“selection_test” is the table that contains all test documents. Again, a selection item can be thought of as a document.

The columns are:

- PK_SELECTION_TEST - A primary key column for table “selection_test”.
- URL - Text in the URL column is used in creating a test Bag of Words table to test against using the text-categorization engine.
- TITLE - Text in the TITLE is the title of the webpage used in testing data. This column is also used in creating a test Bag of Words table.
- META - Meta of the webpage, extracted the same way as it is done with training data, and used in creating a test Bag of Words table.
- SELECTION_DETAIL - Text for this column was extracted from webpages manually for the purpose of testing. This column is used in creating a test Bag of Words table.



PK_S	TITLE	META	SELECTION_DETAIL
80	Indian Recipes : Mushroom Curry	Indian Cooking Recipe site providing information on h...	Mushroom Curry
81	Indian Recipes : Mutter Paneer Mas	Indian Cooking Recipe site providing information on h...	Mutter Paneer Masala
82	Indian Recipes : Navaratna Kurma	Indian Cooking Recipe site providing information on h...	Navaratna Kurma
83	Indian Recipes : Palak Makkai Malai	Indian Cooking Recipe site providing information on h...	Palak Makkai Malai
84	Indian Recipes : Palak Subzi	Indian Cooking Recipe site providing information on h...	Palak Subzi
85	Indian Recipes : Paneer Amritsari	Indian Cooking Recipe site providing information on h...	Paneer Amritsari
86	Indian Recipes : Paneer Korma	Indian Cooking Recipe site providing information on h...	Paneer Korma
87	Indian Recipes : Paneer Makhani Re	Indian Cooking Recipe site providing information on h...	Paneer Makhani Recipe
88	Indian Recipes : Paneer Masala	Indian Cooking Recipe site providing information on h...	Paneer Masala
89	Indian Recipes : Paneer Tikki	Indian Cooking Recipe site providing information on h...	Paneer Tikki
90	Indian Recipes : Paneer Tomato Pea	Indian Cooking Recipe site providing information on h...	Paneer Tomato Peas
91	Indian Recipes : Vegetable Curry	Indian Cooking Recipe site providing information on h...	Vegetable Curry
92	Indian Recipes : Vegetable Gravy	Indian Cooking Recipe site providing information on h...	Vegetable Gravy
93	Indian Recipes : Vegetable Kolhapu	Indian Cooking Recipe site providing information on h...	Vegetable Kolhapuri
94	Indian Recipes : Vegetable Kurma	Indian Cooking Recipe site providing information on h...	Vegetable Kurma
95	Spaghetti Pasta With Pesto Sauce R	Our spaghetti pasta with pesto sauce recipe is great fo...	Spaghetti Pasta With Pesto Sauce Recipe
96	Pasta With Cheese Recipe | How to	Three types of cheese are used to make this tasty past...	Pasta With Cheese Recipe
97	Pasta Fazool Recipe | How to Make	Made with fresh Italian sausage and beef, the pasta faz...	Pasta Fazool Recipe
98	Pasta With Chicken And Tomatoes |	If you want a quick and simple meal, try our pasta with...	Pasta With Chicken And Tomatoes Recipe
99	Pasta With Pesto Recipe | How to M	Do you need a great pasta with pesto recipe? Here, you...	Pasta With Pesto Recipe
100	Fusilli Pasta Carbonara Recipe | Ho	Fusilli pasta carbonara is a splendid dish always loved ...	Fusilli Pasta Carbonara Recipe
101	Italian Pasta Salad Recipe | How to	Italian pasta salad is a healthy and flaversome dish. Fa...	Italian Pasta Salad Recipe
102	Baked Chicken Pasta Recipe | How	Our baked chicken pasta recipe features delectable ing...	Baked Chicken Pasta Recipe
103	Sun Dried Tomato Pesto Chicken Pa	Our sun dried tomato pesto chicken pasta recipe is gre...	Sun Dried Tomato Pesto Chicken Pasta R...
104	Tomato Pesto Pasta Recipe | How t	Our tomato pesto pasta recipe is great for all family oc...	Tomato Pesto Pasta Recipe
105	Baked Pasta With White Mushrooms	Vegetables such as green tomatoes, green pepper, and...	Baked Pasta With White Mushrooms Recipe
106	Spaghetti Pasta With Shrimp Recipe	Cooked shrimp make this spaghetti pasta with shrimp ...	Spaghetti Pasta With Shrimp Recipe
107	Farfalle with Spinach Pesto | Italian	Farfalle con Pesto di Spinaci This classic Italian dish ca...	Farfalle con Pesto di Spinaci
108	Italian Seafood Manicotti Recipe – A	Manicotti di frutti di mare An Italian classic recipe for y...	Manicotti di frutti di mare
109	Olive Oil and Garlic Sauce – aglio e	Olive Oil and Garlic Sauce The key to aglio e olio (garlic...	Olive Oil and Garlic Sauce
110	Italian Pasta Recipe – Linguine with	Linguine con Salsa di Noce This Italian pasta recipe con...	Linguine con Salsa di Noce
111	Penne with Mushrooms and Prosciu	Penne con Funghi e Prosciutto Sauteed mushrooms, pr...	Penne con Funghi e Prosciutto
112	Bolognese Sauce – Italy's most fam	Bolognese Sauce Italy's most famous meat sauce hails f...	Bolognese Sauce
113	Fettucine with Lobster Sauce – A N	Fettuccine con Aragosta Frozen lobster tails tend to be...	Fettuccine con Aragosta
114	Fettuccine with Artichoke Hearts –	Fettuccine con Carciofi The very delicate flavor of artic...	Fettuccine con Carciofi
115	Fusilli with Tomatoes and Olives | I	Fusilli con Pomodoro e Olive The sparkling flavor of to...	Fusilli con Pomodoro e Olive
116	Italian Recipes | Easy Italian Recipe	Minestrone Soup A classic Italian vegetable soup packe...	Minestrone Soup
117	Linguine with Fried Zucchini and Ri	Linguine alla Lorenza Make this Sicilian specialty with s...	Linguine alla Lorenza
118	Fusilli with Four Cheeses Italian Re	Fusilli con Quattro Formaggi A northern Italian favorite...	Fusilli con Quattro Formaggi
119	Italian Cod Stew Recipe | ItalianHor	Merluzzo in Umido This fabulous Italian seafood recipe...	Merluzzo in Umido
120	Italian Sauteed Mushrooms Recipe |	Funghi Saltati For this easy Italian vegetable recipe try t...	Funghi Saltati

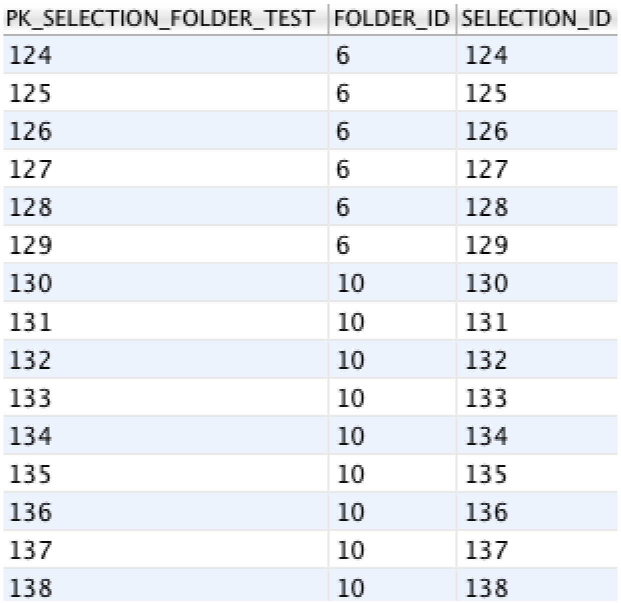
Figure 4: Table "selection_test"

2.5.5 Table “selection_folder_test”

“selection_folder_test” is a linking table that links testing documents to categories. When the text-categorization algorithm is run, the results from the algorithm are compared to the documents in this table. This helps determine how many documents are *true positives*, *false positives*, and *false negatives*.

The columns are:

- PK_SELECTION_FOLDER_TEST - A primary key column for table “selection_folder_test”.
- FOLDER_ID - ID of the folder to which this test document belongs.
- SELECTION_ID - The ID of the document.



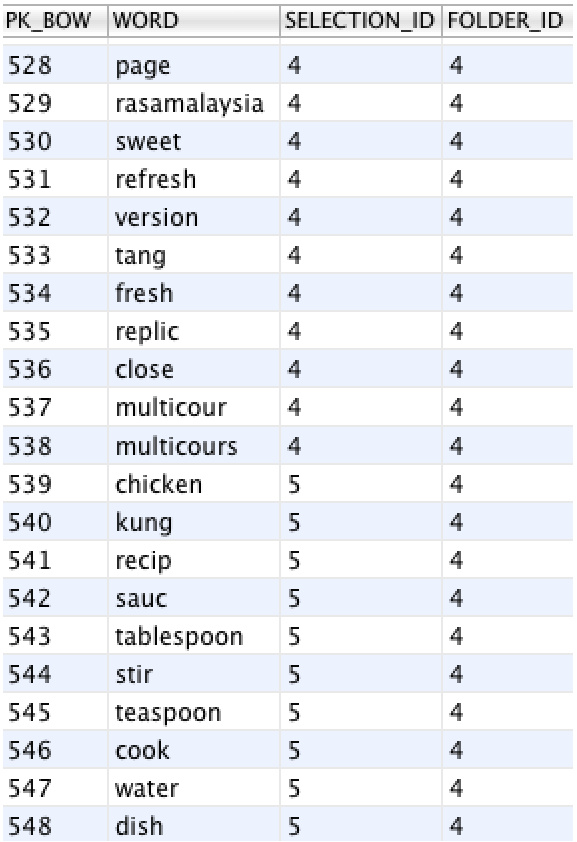
PK_SELECTION_FOLDER_TEST	FOLDER_ID	SELECTION_ID
124	6	124
125	6	125
126	6	126
127	6	127
128	6	128
129	6	129
130	10	130
131	10	131
132	10	132
133	10	133
134	10	134
135	10	135
136	10	136
137	10	137
138	10	138

Figure 5: Table “selection_folder_test”

2.5.6 Table “bow”

“bow” is the Bag of Words table. Once each training document is cleaned, and a word list is created, the word list is also cleaned and words are stemmed. This clean list of words is put in “bow” table. Here words are not grouped, meaning duplicate words can appear. This table is very helpful in determining document counts.
The columns are:

- PK_BOW - A primary key column for table “bow”.
- WORD - This column contains a non-unique list of words. Meaning words may be repeated.
- SELECTION_ID - Document ID for the given word. Meaning, given some word, w, this column will represent in which document ID that word belongs. If a word, w, appears in 5 *different* documents, 5 different rows will be created.
- FOLDER_ID - ID of the folder or category in which this document belongs.



PK_BOW	WORD	SELECTION_ID	FOLDER_ID
528	page	4	4
529	rasamalaysia	4	4
530	sweet	4	4
531	refresh	4	4
532	version	4	4
533	tang	4	4
534	fresh	4	4
535	replic	4	4
536	close	4	4
537	multicour	4	4
538	multicours	4	4
539	chicken	5	4
540	kung	5	4
541	recip	5	4
542	sauc	5	4
543	tablespoon	5	4
544	stir	5	4
545	teaspoon	5	4
546	cook	5	4
547	water	5	4
548	dish	5	4

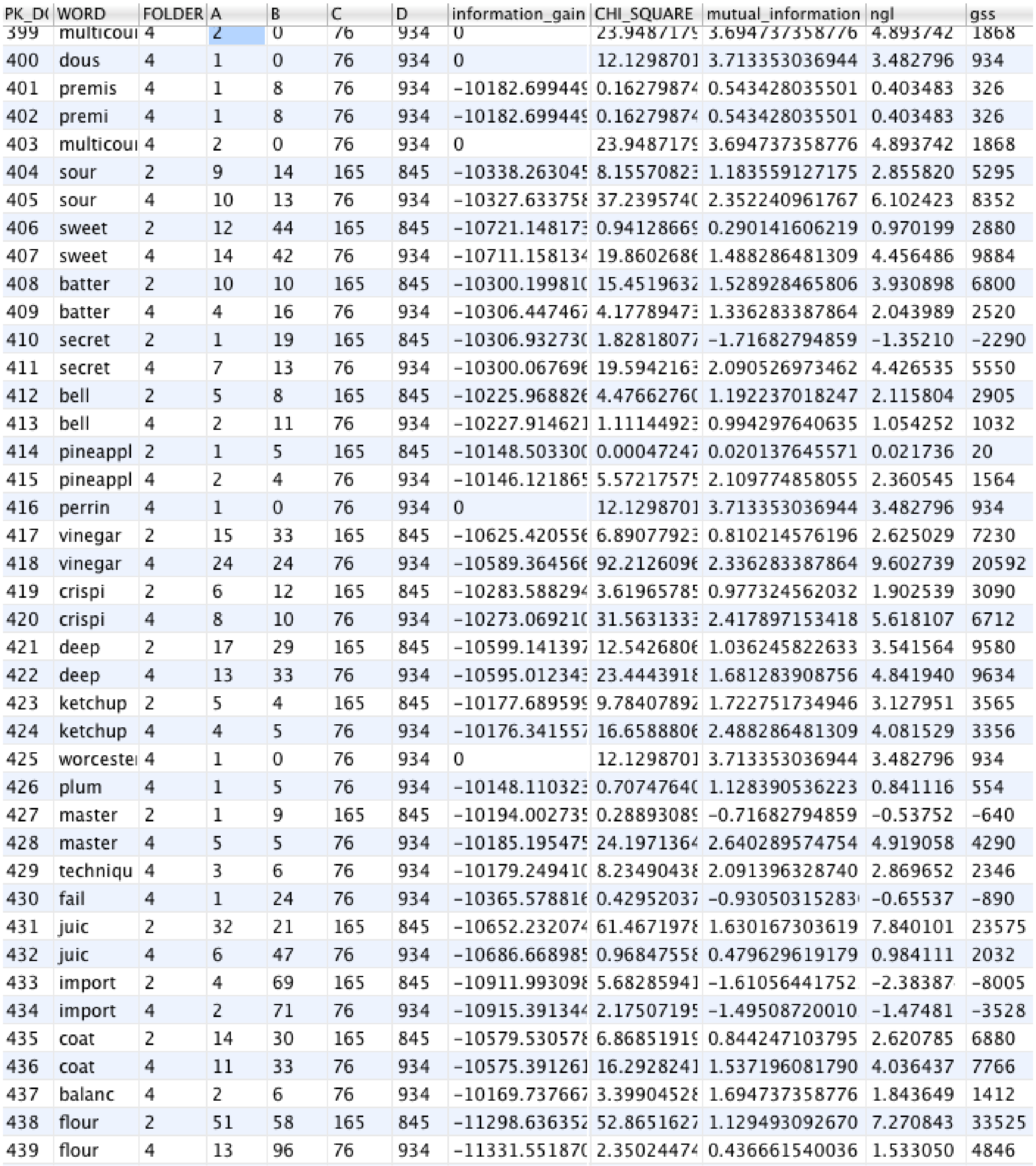
Figure 6: Table “bow”

2.5.7 Table “doc_bow_raw”

“doc_bow_raw” contains a raw list of Bag of Words for training documents. This table will have less or equal amount of rows (data) than the “bow” table. In this table, words are grouped. Meaning, if the word “rice” appears in more than one document in category named “Chinese”, then in the “doc_bow_raw” table, “rice” and “Chinese” pair will appear only once. Column named “A” will represent in how many documents the word appears in. Columns A, B, C, and D are the most important columns of this table as from them, required values for feature selection algorithms are calculated. The columns are:

- PK_DOC_BOW_RAW - A primary key column for table “doc_bow_raw”.
- WORD - A list of words gathered from “bow” table, grouped by FOLDER_ID.
- FOLDER_ID - ID of the folder in which the word for this row belongs.
- A - Number of documents in FOLDER_ID, C, containing WORD, w.
- B - Number of documents *not* in FOLDER_ID, C, containing WORD, w.
- C - Number of documents in FOLDER_ID, C, *not* containing WORD, w.
- D - Number of documents *not* in FOLDER_ID, C, *not* containing WORD, w.
- INFORMATION_GAIN - Calculated Information Gain values using the A, B, C, and D columns.
- CHI_SQUARE - Calculated Chi Square values using the A, B, C, and D columns.
- MUTUAL_INFORMATION - Calculated Mutual Information values using the A, B, C, and D columns.

- NGL - Calculated Ng-Goh-Low coefficient values using the A, B, C, and D columns.
- GSS - Calculated Galavotti-Sebastiani-Simi coefficient values using the A, B, C, and D columns.



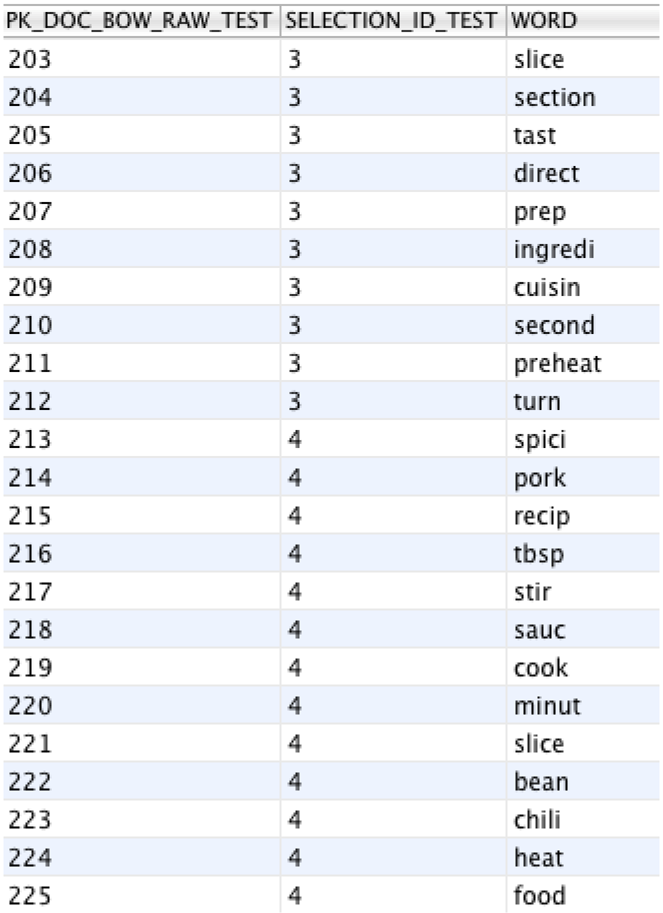
PK_D(	WORD	FOLDER	A	B	C	D	information_gain	CHI_SQUARE	mutual_information	ngl	gss
399	multicoui	4	2	0	76	934	0	23.9487179	3.694737358776	4.893742	1868
400	dous	4	1	0	76	934	0	12.1298701	3.713353036944	3.482796	934
401	premis	4	1	8	76	934	-10182.699449	0.16279874	0.543428035501	0.403483	326
402	premi	4	1	8	76	934	-10182.699449	0.16279874	0.543428035501	0.403483	326
403	multicoui	4	2	0	76	934	0	23.9487179	3.694737358776	4.893742	1868
404	sour	2	9	14	165	845	-10338.263045	8.1557082	1.183559127175	2.855820	5295
405	sour	4	10	13	76	934	-10327.633758	37.2395740	2.352240961767	6.102423	8352
406	sweet	2	12	44	165	845	-10721.148173	0.94128669	0.290141606219	0.970199	2880
407	sweet	4	14	42	76	934	-10711.158134	19.8602686	1.488286481309	4.456486	9884
408	batter	2	10	10	165	845	-10300.199810	15.4519632	1.528928465806	3.930898	6800
409	batter	4	4	16	76	934	-10306.447467	4.1778947	1.336283387864	2.043989	2520
410	secret	2	1	19	165	845	-10306.932730	1.82818077	-1.71682794859	-1.35210	-2290
411	secret	4	7	13	76	934	-10300.067696	19.5942163	2.090526973462	4.426535	5550
412	bell	2	5	8	165	845	-10225.968826	4.4766276	1.192237018247	2.115804	2905
413	bell	4	2	11	76	934	-10227.914621	1.1114492	0.994297640635	1.054252	1032
414	pineappl	2	1	5	165	845	-10148.503300	0.00047247	0.020137645571	0.021736	20
415	pineappl	4	2	4	76	934	-10146.121865	5.5721757	2.109774858055	2.360545	1564
416	perrin	4	1	0	76	934	0	12.1298701	3.713353036944	3.482796	934
417	vinegar	2	15	33	165	845	-10625.420556	6.8907792	0.810214576196	2.625029	7230
418	vinegar	4	24	24	76	934	-10589.364566	92.2126096	2.336283387864	9.602739	20592
419	crispi	2	6	12	165	845	-10283.588294	3.6196578	0.977324562032	1.902539	3090
420	crispi	4	8	10	76	934	-10273.069210	31.5631333	2.417897153418	5.618107	6712
421	deep	2	17	29	165	845	-10599.141397	12.5426806	1.036245822633	3.541564	9580
422	deep	4	13	33	76	934	-10595.012343	23.4443918	1.681283908756	4.841940	9634
423	ketchup	2	5	4	165	845	-10177.689599	9.7840789	1.722751734946	3.127951	3565
424	ketchup	4	4	5	76	934	-10176.341557	16.6588806	2.488286481309	4.081529	3356
425	worceste	4	1	0	76	934	0	12.1298701	3.713353036944	3.482796	934
426	plum	4	1	5	76	934	-10148.110323	0.7074764	1.128390536223	0.841116	554
427	master	2	1	9	165	845	-10194.002735	0.28893089	-0.71682794859	-0.53752	-640
428	master	4	5	5	76	934	-10185.195475	24.1971364	2.640289574754	4.919058	4290
429	techniqu	4	3	6	76	934	-10179.249410	8.2349043	2.091396328740	2.869652	2346
430	fail	4	1	24	76	934	-10365.578816	0.42952037	-0.93050315283	-0.65537	-890
431	juic	2	32	21	165	845	-10652.232074	61.4671978	1.630167303619	7.840101	23575
432	juic	4	6	47	76	934	-10686.668985	0.9684755	0.479629619179	0.984111	2032
433	import	2	4	69	165	845	-10911.993098	5.6828594	-1.61056441752	-2.38387	-8005
434	import	4	2	71	76	934	-10915.391344	2.1750719	-1.49508720010	-1.47481	-3528
435	coat	2	14	30	165	845	-10579.530578	6.8685191	0.844247103795	2.620785	6880
436	coat	4	11	33	76	934	-10575.391261	16.2928241	1.537196081790	4.036437	7766
437	balanc	4	2	6	76	934	-10169.737667	3.3990452	1.694737358776	1.843649	1412
438	flour	2	51	58	165	845	-11298.636352	52.8651627	1.129493092670	7.270843	33525
439	flour	4	13	96	76	934	-11331.551870	2.3502447	0.436661540036	1.533050	4846

Figure 7: Table “doc_bow_raw”

2.5.8 Table “doc_bow_raw_test”

“doc_bow_raw_test” contains a raw list of Bag of Words for test documents. In actual testing, this table would *not* be created. The only reason for creating this table is so that we can run multiple tests on testing data easily. Creating a Bag of Words for 338 documents for each test run would take much more time than creating the BOW once, and reusing it for all tests. The columns are:

- PK_DOC_BOW_RAW_TEST - A primary key column for table “doc_bow_raw_test”.
- SELECTION_ID_TEST - ID of the test document in which this word belongs.
- WORD - A cleaned word, which will help in testing of the text-categorization engine and the feature selection algorithms.



PK_DOC_BOW_RAW_TEST	SELECTION_ID_TEST	WORD
203	3	slice
204	3	section
205	3	tast
206	3	direct
207	3	prep
208	3	ingredi
209	3	cuisin
210	3	second
211	3	preheat
212	3	turn
213	4	spici
214	4	pork
215	4	recip
216	4	tbsp
217	4	stir
218	4	sauc
219	4	cook
220	4	minut
221	4	slice
222	4	bean
223	4	chili
224	4	heat
225	4	food

Figure 8: Table “doc_bow_raw_test”

2.6 Creating Bag of Words for Training Dataset - “bow” Table

Creating an initial bag of words list is an important step, as it will help in determining in how many documents each *cleaned* word appears. But to do create the list, we first need to determine what those *cleaned* words are. The next subsections explain each step of the process in creating the Bag of Words, or the BOW, to be inserted in the “bow” table.

2.6.1 The b8 Lexer

In this section, we will explain how the b8 lexer creates tokens, or BOW, that helps in text-categorization. b8 is a Naive Bayesian Spam filter library written by Tobias Leupold. We downloaded the library, and extracted the lexer file out of it for our needs. Written in PHP, this lexer helps create tokens from a string of words. We updated the original lexer by adding stopword removal capability. Stopwords, or “overly common words”, are not helpful in categorization, as they cannot differentiate between categories [6]. Words such as *the*, *a*, *of*, *is*, *at*, *on* and many more are considered stopwords. The *stopwords* file we used containd 573 stopwords. Forman also tells us that stopwords are language and domain specific, and he says, “depending on the classification task, they may run the risk of removing words that are essential predictors, e.g. the word ‘can’ is discriminating between ‘aluminum’ and ‘glass’ recycling” [6, 7].

Our version of the b8 lexer also uses Porter stemmer. “The Porter stemming algorithm (or ‘Porter stemmer’) is a process for removing the commoner morphological and inflexional endings from words in English. Its main use is as part of a term normalisation process that is usually done when setting up Information Retrieval systems" [8]. Stemming of words is an important step in creating useful tokens. To humans, words "cook", "cooking", and "cooked" mean the same action. Using three different words in text-categorization for the same action can lead to wrong or bad results. Porter stemming algorithm stems words and makes them all same. Meaning three different words "cook", "cooking", and "cooked" would end up becoming one word, "cook". Porter stemming algorithm improves results of text-categorization. George Forman also says that "the common practice of stemming or lemmatizing - merging various word forms such as plurals and verb conjugations into one distinct term - also reduces the number of features to be considered. It is properly, however, a feature engineering option" [6].

The b8 lexer first receives the stopwords file as a list of stopword tokens. The lexer has one function that takes in the string to be tokenized, call this string *$text*. The lexer begins by modifying all punctuation from *$text*. Following punctuation symbols are not removed, but are converted into a space.

```
~ ‘ ! @ # $ % ^ & * ( ) - _ = + [ { ] } \ \ | \ ’ " ; : , < . > / ?
```

The reason for converting punctuation into spaces is so that the lexer would work as intended. For example, removing punctuation from phrase "`MySQL's awesome`" would make the phrase "`MySQLs awesome`". The stemming algorithm may or may not remove the trailing 's', which would cause problems for the categorization algorithm. Replacing punctuation in the same phrase with a space would make it "`MySQL s awesome`". Letter 's' by itself is a stopword, and would be removed, leaving a descriptive phrase "`MySQL awesome`".

The b8 lexer also receives the document ID, and the category ID at initialization, so that it can be determined for which document and category *$text* is being tokenized. This information is passed to the lexer per document, so that at the end when a BOW is created in the lexer, it knows what words belong to what document and what category.

$text is then split using b8's built in regular expressions.

Listing 2: b8 Regular Expressions

```
public $regexp = array(
    'ip'         => '/([A-Za-z0-9\_\-\.]+)/',
    'raw_split'  => '/[\s,\.\/"\:;\|<>\-_\[\]{}\+=\)\(\*\&\^%]+/',
    'html'       => '/(<.+?>)/',
    'tagname'    => '/(.+?)\s/',
    'numbers'    => '/^[0-9]+$/'
);
```

From our example, "MySQL" and "awesome" would be the resulting words from the lexer. These resulting words will be added to a *static* BOW array managed by the b8 lexer itself. The token list, *$tokens*, which in our example contains words "MySQL" and "awesome", is passed to the function *AddToBagOfWords*. Here is the code:

Listing 3: AddToBagOfWords Function

```
private function AddToBagOfWords ( $tokens ) {
    $arr = array();
    foreach ( $tokens as $token => $tokenCount ) {
        $arr['token'] = $token;
        $arr['folder_id'] = $this->currentCategory;
        $arr['selection_id'] = $this->currentSelectionID;
        array_push ( self::$bagOfWords , $arr );
    }
}
```

The function *AddToBagOfWords* first creates an array called *$arr*. Then it goes through all the tokens in *$tokens*, and adds the token, the current document ID, and the current category ID to *$arr*. *$arr* is then pushed to the global static array *$bagOfWords*. Note that this approach does not create a unique list of words, but also note that it does not create duplicates of word-folderID-selectionID. This is how BOW is created for given *$text* data.

Explained above is the core functionality of the b8 lexer, and what it does. How everything is tied together is explained in the next section.

2.6.2 Putting Code Together and Creating BOW

The b8 lexer is a core component that creates the BOW for given *$text* data. However, it is the *createBOW.php* program, referred to as *createBOW*, that is responsible for bringing data from the database, sending *$text* to the lexer, and handing over the finished BOW list to the database-update utility for updating of the "bow" table in the database.

createBOW's first responsibility is to retrieve data from the database. Specifically, retrieve tagged documents for the purpose of training. *createBOW* begins by running the following query:

Listing 4: Retrieve Documents SQL Query

```sql
select s.pk_selection, s.url, s.title, s.meta, s.selection_detail, f.*
from selection as s, folder as f, selection_folder as sf
where
   s.pk_selection=sf.selection_id and
   f.pk_folder=sf.folder_id limit 200 offset 0
```

This SQL query retrieves the document ID, the URL, the title, the meta, and the text-content of 200 documents at a time from the "selection" table. The same query also retrieves, for those documents, the category ID in which they belong. The reason for retrieving 200 documents at a time is because not all 1,010 documents can fit in the memory at one time. Although, the offset can be increased by 200 automatically, since there were only 1,010 documents, we manually ran the query updating the offset by 200 each iteration. The *createBOW* then retrieves the list of stopwords and gives the list to the b8 lexer as shown below:

Listing 5: Retrieving and Setting Stopwords in the b8 Lexer

```
1  $stopwords = file_get_contents ( 'stopwords' );
2  $stopwords = explode ( "\r\n" , $stopwords );
3
4  $numStopwords = count($stopwords);
5  // \b for word-boundary "u" for UTF-8. "i" for insensitive.
6  for ( $i=0 ; $i<$numStopwords ; $i++ ) {
7     $stopwords[$i] = "/\b" . $stopwords[$i] . "\b/ui";
8  }
9
10 $lexer = new b8_Lexer();
11 $lexer->SetStopwords ( $stopwords );
```

Line 1 gets the contents of the "stopwords" file in the *$stopwords* string. On line 2, the *explode* function breaks the *$stopwords* string into an array specified by the delimiter string "\r\n". The for loop prepares each stopword for a regular-expression in the b8 lexer so that each stop word can be easily removed. In the loop, each stopword is wrapped by '\b' for word-boundary, and appended with the "ui" flags for UTF-8 and CASE INSENSITIVE. Lexer is then initialized on line 10, and the *$stopwords* are set in line 11.

Once the documents are retrieved and the lexer is initialized, the actual processing for tokenizing of the documents can begin. Code is given below. The variable *$selections* is a list of documents.

Listing 6: Tokenize Documents using the b8 Lexer

```
foreach ( $selections as $selectionItem ) {
   $selectionID = $selectionItem['pk_selection'];
   $category = $selectionItem['PK_FOLDER'];

```

```
    $url = strtolower($selectionItem['url']);
    $title = strtolower($selectionItem['title']);
    $meta = strtolower($selectionItem['meta']);
    $selectionDetail = strtolower($selectionItem['selection_detail']);

    $strToTokenize = $url .' '. $title .' '. $meta .' '. $selectionDetail;
    $lexer->SetCategory ( $category );
    $lexer->SetSelectionID ( $selectionID );
    $lexer->get_tokens ( $strToTokenize );
}
```

In the code given above, the loop goes through each document item, and stores the document ID, the category ID, and the URL, the title, the meta information and the text-content of the document in appropriate variables. Note that the URL, the title, the meta, and the text-content are all first converted to lower case, so that the lexer treats words with different cases in a uniform way. These fields are then concatenated together into one string. Dot (`.`) is the concatenation operator in PHP. The reason for making one string is because, I'm considering the URL, the title, the meta, and the text-content as a document's *joined* information. This is a naïve approach, however. Better approach would be to assign each field different weights, which would then improve categorization dramatically. Next, for each document, the category ID and the document ID are set in the lexer. Then the *get_tokens* function is passed the concatenated string to be tokenized.

The *get_tokens* tokenizes the string *$strToTokenize* and creates a BOW for those tokens given the category ID, and the document ID. Once the loop is done, a BOW in the lexer is fully created. The *createBOW* retrieves BOW from the lexer, and inserts all the tokens in the "bow" table in the database. Code is given below. Notice that the *$tokenList* is passed to the InsertBOW, which actually performs inserts to the table.

Listing 7: Get Tokens and Call InsertBOW to Save BOW

```
$tokenList = $lexer->GetBagOfWords();

$bowInsertFacade = new BOWInsertFacade();
$bowInsertFacade->InsertBOW ( $tokenList );
```

Below is the *InsertBOW* function that creates and performs inserts on the "bow" table.

Listing 8: InsertBOW Function

```
1  public function InsertBOW ( $bow ) {
2      $num = count($bow);
3
4      for ( $i=0 ; $i<$num ; $i++ ) {
5          $query =
6          "insert into bow ( word , selection_id , folder_id ) values (
7              '" . mysql_real_escape_string($bow[$i]['token'],$this->dbLink) . "',
8              '" . $bow[$i]['selection_id'] . "',
9              '" . $bow[$i]['folder_id'] . "'
10         )";
11
12         $result = mysql_query ( $query , $this->dbLink );
13     }
14 }
```

In the *InsertBOW* function above, line 2 first counts how many items need to be inserted. The for-loop then runs through all the items, and builds an insert-query-string, which is then run using the *mysql_query* command of PHP. Here are a few sample queries that were ran:

Listing 9: Sample InsertBOW Queries

```
insert into bow ( word , selection_id , folder_id )
    values ( 'chop' , '77' , '2' );
insert into bow ( word , selection_id , folder_id )
    values ( 'pepper' , '77' , '2' );
insert into bow ( word , selection_id , folder_id )
    values ( 'bake' , '77' , '2' );
insert into bow ( word , selection_id , folder_id )
    values ( 'stir' , '77' , '2' );
insert into bow ( word , selection_id , folder_id )
    values ( 'photo' , '673' , '26' );
insert into bow ( word , selection_id , folder_id )
    values ( 'network' , '673' , '26' );
```

```
insert into bow ( word , selection_id , folder_id )
    values ( ’social’ , ’673’ , ’26’ );
insert into bow ( word , selection_id , folder_id )
    values ( ’cast’ , ’772’ , ’22’ );
insert into bow ( word , selection_id , folder_id )
    values ( ’object’ , ’772’ , ’22’ );
insert into bow ( word , selection_id , folder_id )
    values ( ’string’ , ’772’ , ’22’ );
```

I have shown only 10 sample queries above. After running all insert queries to the “bow” table, the “bow” table contained *88,230* rows.

Now the “bow” table is fully created, and has data that can be used. How data from the “bow” table will be used is briefly explained in the next section.

2.6.3 Importance of the “BOW” Table

The “bow” table is ready to be used. There are two main purposes for using the “bow” table: reusability, and counting of documents.

1. Reusability - Creating the “bow” table takes a long time. Once the “bow” table is created, the training documents do not need to be touched again, because the “bow” table contains all needed information. This saves a lot of time.

2. Counting of documents - For text-categorization, it is important to determine in how many documents a word appears, or in how many documents the word does not appear, and more variations explained in detail later. And because the “bow” table contains ungrouped list of word - document ID - folder ID, counting of documents can be easily queried right from the “bow” table without touching the documents again. Again, saving a lot of time.

The “bow” table contains ungrouped list of word - document ID - folder ID. Next, we have to create a list of words that grouped by word and category ID, and insert into a table that we call “doc_bow_raw”. Counts of the documents will be found in this table as well. The procedure is extremely easy, as we have to run only one SQL query. This procedure of creating the “doc_bow_raw” table is explained next.

2.7 Creating Bag of Words for Training Dataset - “doc_bow_raw” Table

The “doc_bow_raw” table has the following columns available: word, folder_id, multiple columns for different types of document counts, and columns available for values for feature selection algorithms. The “bow” table had ungrouped data, whereas the “doc_bow_raw” table has grouped data, grouped by word - folder ID. Because we’re using MySQL, we can just run a simple query to insert data from the “bow” table into the “doc_bow_raw” table. Again, reusability of the “bow” table becomes very helpful.

To insert into the “doc_bow_raw” table from the “bow” table, we will run the following simple query.

Listing 10: Query to Insert Into “doc_bow_raw” Table

```
insert into doc_bow_raw ( word , folder_id )
  select word, folder_id from bow
    group by word, folder_id
```

A total of *20,419* records were inserted into the “doc_bow_raw” table from the “bow” table. Even though inserting data this way is more correct than any other way, one way would be to check integrity of these insertions is by checking against another table that we know for fact contained grouped word - folder ID pairs.

This second table we checked against was actually created using PHP-MySQL using the same way the “bow” table was created. The only difference would be to replace the *AddToBagOfWords* function explained above with the following new function:

Listing 11: AddToBagOfWords Function

```
   private function AddToBagOfWords ( $tokens ) {
      foreach ( $tokens as $token => $tokenCount ) {

         if ( !isset(self::$bagOfWords[$token]) ) {
            self::$bagOfWords[$token] = array();
            for ( $i=0 ; $i<$this->numCategories ; $i++ ) {
               $category = $this->categoryList[$i];
               self::$bagOfWords[$token][$category] = 0;
            }
         }
      }
   }
```

Here, the only difference is that, if the word (token) is already set in the *$bagOfWords* global static array, then the word is not added to the list again.

Now we have two different versions of the “doc_bow_raw” table. One using the query, another using PHP-MySQL. To check correctness, we will run the following query:

Listing 12: Checking Correctness of the Data in Table “doc_bow_raw” Inserted Using a Query on the “bow” Table

```
select word, folder_id from doc_bow_raw
   where ( word , folder_id ) not in
   (
      select word , folder_id from doc_bow_raw_PHP
   );
```

Running this query returned an empty set, as expected. This means, data inserted using PHP and data inserted using the query on the “bow” table match 100%. Next, we create the bag of words for the test dataset.

2.8 Creating Bag of Words for Test Dataset - “doc_bow_raw_test” Table

As mentioned earlier in section 2.3, and 2.4, the test dataset was collected and tagged the same was as the training dataset. The only difference between the training and the testing dataset is that the BOW created for them has to go in different tables. The BOW for the training data needs to go in the “doc_bow_raw” table, and the BOW for the test data needs to go in the “doc_bow_raw_test” table. There is no difference in how the lexer builds the BOW for the training and the testing dataset. How the BOW is created has already been explained, and exactly the same process applies to the test dataset. How data for the test BOW is inserted is explained here.

The program *createDocRawBOWTest* is used to create BOW for test data. Here a call is made to the lexer to get the BOW:

Listing 13: Get BOW from the b8 Lexer

```
$bow = $lexer->GetBagOfWords();
```

Then a call to another function is made to insert *$bow* with a given document ID. If you recall, in listing 6, we talked about how the selectionID, or the document ID, is retrieved from a *$selectionItem*. For each test document, the document ID is retrieved almost the same way. This *$selectionID* along with *$bow* are passed to the *InsertBOWTest* function. Note that *pk_selection_test* is a column in the table *selection_test*. Code is given below.

Listing 14: Get Document ID for Test Data

```
$selectionID = $selectionItem['pk_selection_test'];

$bowTest->InsertBOWTest ( $selectionID , $bow );
```

Below is the code that will insert word - document ID pairs in the “doc_bow_raw_test” table.

Listing 15: InsertBOWTest Function

```
public function InsertBOWTest ( $selectionID , &$bow ) {
   foreach ( $bow as $word => $nothing ) {

      $word = mysql_real_escape_string ( $word , $this->dbLink );

      $query =
      "insert into doc_bow_raw_test
         ( selection_id_test , word )
      values (
         '" . $selectionID . "',
         '" . $word . "'
      )";

      $result = mysql_query ( $query , $this->dbLink );
   }
}
```

A total of *26,789* rows were inserted in the *doc_bow_raw_test* table. Here are a few sample queries that were ran:

Listing 16: Sample InsertBOWTest Queries

```
insert into doc_bow_raw_test
   ( selection_id_test , word )
values (
   '1' , 'carrot'
);
insert into doc_bow_raw_test
   ( selection_id_test , word )
values (
   '1' , 'potato'
);
insert into doc_bow_raw_test
   ( selection_id_test , word )
values (
   '1' , 'stew'
);
insert into doc_bow_raw_test
   ( selection_id_test , word )
values (
   '1' , 'tbsp'
);
insert into doc_bow_raw_test
   ( selection_id_test , word )
values (
   '1' , 'cook'
);
```

```
insert into doc_bow_raw_test
   ( selection_id_test , word )
values (
   ’1’ , ’food’
);
```

Now that the training and the testing datasets have been inserted in proper tables, we can proceed to the counting of the documents. Document counting is the most crucial process for feature selection algorithms. Now we have to calculate the A, B, C, D. Calculating these A, B, C, D values is explained in the next section.

CHAPTER 3

COUNTING DOCUMENTS

Document counting is an important task in feature selection algorithms. Document counts can help determine how important or not-important a word token is. The next few sections explain the A, B, C, D values, what they are, and how they are calculated.

3.1 The A, B, C, D Values

3.1.1 Explanation of the A, B, C, D Values

The A, B, C, D values have been mentioned many times so far in this thesis, and they are the most important values in the feature selection algorithms we have tested. Now, we will describe what each part of A, B, C, D means:

- A - the number of documents in category, C, containing word/token, t.
- B - the number of documents not in category, C, containing word/token, t.
- C - the number of document in category, C, not containing word/token, t.
- D - the number of documents not in category, C, not containing word/token, t.

[4]

These A, B, C, D values are used in all feature selection algorithms we've tested in one way or another. These A, B, C, D values are stored in the "doc_bow_raw" table, but the actual calculations happen on the "bow" table. Calculating them is a major task, and how they are calculated is explained next.

3.1.2 Calculating and Updating “doc_bow_raw” with the A, B, C, D Values

Calculating the A, B, C, D values first requires me to get word - folder ID information from the “doc_bow_raw” table. The “doc_bow_raw” table has *20,419* records. Retrieving only 2 columns of *20,419* records works fine, as it can fit in the memory. Because the actual document count calculations have to be done using the “bow” table, our initial plan was to retrieve all *88,230* rows of the “bow” table in PHP, and do the counts on this retrieved data in the memory itself. This way turned out to be much slower. Another, faster, way was implemented by running the *count* queries on the database side. This approach did not need the “bow” table to be retrieved. Code for the loop that goes thorough each word - category ID, calls *CalculateABCDValues* and *UpdateABCDValues* is given below:

Listing 17: Loop to Calculate and Update A B C D Values

```
foreach ( $resultList as $key => $details ) {
   $word = $details['word'];
   $folder = $details['folder'];

   $values = $dbDocBOWRaw->CalculateABCDValues ( $word , $folder );

   $dbDocBOWRaw->UpdateABCDValues ( $word , $folder , $values );

}
```

The loop above goes thorough each word - category ID and calls the *CalculateABCDValues* function, passing word and category ID. Code for the *CalculateABCDValues* function is given below:

Listing 18: CalculateABCDValues Function

```
   public function CalculateABCDValues ( $word , $folderID ) {
      $word = mysql_real_escape_string($word,$this->dbLink);
      $folderID = mysql_real_escape_string($folderID,$this->dbLink);

      $query =
6          "select count(0) as A from (
7             select distinct selection_id
8             from bow
9             where folder_id='" . $folderID . "' and word='" . $word . "'
10         ) as tbl";
11         $row = mysql_fetch_assoc ( mysql_query ( $query , $this->dbLink ) );
12         $A = $row['A'];
13
14         $query =
15         "select count(0) as B from (
16            select distinct selection_id
17            from bow
18            where folder_id<>'" . $folderID . "' and word='" . $word . "'
19         ) as tbl";
20         $row = mysql_fetch_assoc ( mysql_query ( $query , $this->dbLink ) );
21         $B = $row['B'];
22
23         $query =
24         "select count(0) as C from (
25            select distinct selection_id
26            from bow
27            where folder_id='" . $folderID . "' and word<>'" . $word . "'
28         ) as tbl";
29         $row = mysql_fetch_assoc ( mysql_query ( $query , $this->dbLink ) );
30         $C = $row['C'];
31
32         $query =
33         "select count(0) as D from (
34            select distinct selection_id
35            from bow
36            where folder_id<>'" . $folderID . "' and word<>'" . $word . "'
37         ) as tbl";
38         $row = mysql_fetch_assoc ( mysql_query ( $query , $this->dbLink ) );
39         $D = $row['D'];
40
41         $values = array();
42         $values['A'] = $A;
43         $values['B'] = $B;
44         $values['C'] = $C;
45         $values['D'] = $D;
46         return $values;
47      }
```

In the *CalculateABCDValues* function, the *$folderID* variable represents the ID of the category. Notice that all count queries are done on the "bow" table.

Explanation of SQL query for 'A' (lines 6-10): First get all document ID where category ID matches *$folderID* and word matches *$word*. Then do a count of these documents, and store the result in variable *$A*.

Explanation of SQL query for ‘B’ (lines 15-19): First get all document ID where category ID does *NOT* match *$folderID* and word matches *$word*. Then do a count of these documents, and store the result in variable *$B*.

Explanation of SQL query for ‘C’ (lines 24-28): First get all document ID where category ID matches *$folderID* and word does *NOT* match *$word*. Then do a count of these documents, and store the result in variable *$C*.

Explanation of SQL query for ‘D’ (lines 33-37): First get all document ID where category ID does *NOT* match *$folderID* and word does *NOT* match *$word*. Then do a count of these documents, and store the result in variable *$D*.

Once the variables *$A*, *$B*, *$C*, and *$D* are set, store them in *$values* array, and return this array.

Then, as shown in Listing 17, the function *UpdateABCDValues* is called, and is passed the word, the category ID, and the *$values* array to be updated in the “doc_bow_raw” table. Code for the function *UpdateABCDValues* is given below:

Listing 19: UpdateABCDValues Function

```
public function UpdateABCDValues ( &$word , &$folder , &$values ) {
   $a = $values['A'];
   $b = $values['B'];
   $c = $values['C'];
   $d = $values['D'];

   $query =
   "update doc_bow_raw set
   A='" . $a ."',
   B='" . $b . "',
   C='" . $c . "',
   D='" . $d . "'
   where
      word='" . mysql_real_escape_string($word,$this->dbLink) . "' and
      folder_id='" . $folder . "'";
   $result = mysql_query ( $query , $this->dbLink );
}
```

As mentioned earlier, calculating these A, B, C, D values is a big task. Even with *indexed* columns in all the tables, calculating and updating these A, B, C, D values in the "doc_bow_raw" table took more than 30 minutes on a computer, which has the following configuration:

- Processor - 2.2 GHz Intel Core i7
- Memory - 4 GB 1333 MHz DDR3
- HD - 500 GB. 468.35 GB Available.

These A, B, C, D values, however, have to be calculated only once for training. Once these values are stored in the table, they do not need to be modified unless more training needs to be done. The training and the testing datasets used for this thesis are very, very small. Also, as mentioned in section 2.2, we have only 9 categories, which is a very small number of categories. In the real world, there could be hundreds of categories, and more categories would have to be included dynamically. New documents have to be included in the training set, and so, training would have to be done dynamically as well. In such large-scale cases, a very efficient system would have to be developed where these document counts could be updated much faster! This area could be further researched. For now, we can move on to explaining the core of this thesis, the feature selection algorithms.

CHAPTER 4

FEATURE SELECTION ALGORITHMS

Six different feature selection algorithms were implemented. This section describes why to use a feature selection algorithm, and describes each algorithm.

4.1 Why Use Feature Selection?

Much research has been done on why feature selection is important and should be used when categorizing textual data. Shang, Huang, Zhu, Lin, Qu, Wang state the following:

> A major problem of text categorization is the high dimensionality of the feature space. For many learning algorithms, such high dimensionality is not permitted. Moreover most of these dimensions are not relative to text categorization; even some noise data hurt the precision of the classifier.

[2]

It is not practical to use all features gathered from the training documents to use in text-categorization. "Reduction of the features used for the representation of documents is an absolute requirement for using most of the machine learning algorithms." [9]

For this reason, one or more feature selection techniques have to be used, so that by using less amount of features/tokens/words, new documents can be categorized easily, faster, and by using less computation power.

Feature selection is "selecting a subset of the features available for describing the data" [5], or in other words it is a method to reduce "the dimensionality of the dataset by removing features that are considered irrelevant for the classification" [9].

There are many benefits to using feature selection. The benefits of using feature selection are described below:

1. Simplifying or speeding up computations with only little loss in classification quality. [5]

2. Reduce dimensionality of feature space and improve the efficiency, performance gain, and precision of the classifier. [2, 10, 9]

3. Improves classification effectiveness, computational efficiency, and accuracy. [10, 1]

4. Helps remove non-informative and noisy features and helps reduce the feature space to a manageable size. [11]

5. Helps keep computational requirements and dataset size small, especially for those text-categorization algorithms that do not scale with the feature set size. [9]

George Forman describes that,

> The overall feature selection procedure is to score each potential feature according to a particular feature selection metric, and then take the best k features. Scoring involves counting the occurrences of a feature in training positive- and negative-class training examples separately, and then computing a function of these.

[6]

What feature selection algorithms were used, and their explanations are given next.

4.2 Feature Selection Algorithm Explanations

This section describes the feature selection algorithms we used. The algorithms are: *Document Frequency*, *Information Gain*, *Mutual Information*, *Chi Square*, *NGL (Ng-Goh-Low) Coefficient*, and *GSS (Galavotti-Sebastiani-Simi) Coefficient*. Each subsection below explains an algorithm.

4.2.1 Document Frequency - Explanation

Document frequency is a very simple feature selection method. Document frequency for a term can be found by counting the number of documents in which a term/feature occurs. [1, 10, 3]. Document frequency assumes that rare terms are "non-informative for category prediction, or non-influential in global performance" [3], and "terms with higher document frequency are more informative for classification" [1].

Document frequency is already calculated, because we have already calculated the A, B, C, D values, so document frequency is:

$$DF = A$$

4.2.2 Information Gain - Explanation

Information gain value measures “the number of bits of information obtained for category prediction by knowing presence of absence of a term in a document”. [3, 1, 10].

According to Mukras *et al.*, “the idea behind IG is to select features that reveal the most information about the classes” [12].

Information gain values were calculated as follows:

$$IG(t,c) = \sum_{c \in \{c_i, \bar{c_i}\}} \sum_{t \in \{t_k, \bar{t_k}\}} P(t,c) \cdot log \frac{P(t,c)}{P(t)P(c)}$$

[9]

4.2.3 Mutual Information - Explanation

Mutual information method assumes that the “term with higher category ratio is more effective for classification” [1].

Mutual information can be calculated as follows using our already calculated A, B, C, D values:

$$MI = \log \frac{A \times N}{(A+C)(A+B)}$$

[1]

Here, A is the number of documents that contain the term, *t*, and also belong to category, *c*. B is the number of documents that contain the term, *t*, but do not belong to category, *c*. C is the number of documents that do not contain the term, *t*, but belong to category, *c*. N is the number of training documents. [1, 3].

4.2.4 Chi Square - Explanation

Chi square measures the lack of independence between a term, t, and the category, c [10, 3].

Chi square, χ^2, can be calculated as follows, again, using our previously calculated A, B, C, D values:

$$\chi^2 = \frac{N(AD - CB)^2}{(A + C)(B + D)(A + B)(C + D)}$$

[4]

Again, A is the number of documents that contain the term, t, and also belong to category, c. B is the number of documents that contain the term, t, but do not belong to category, c. C is the number of documents that do not contain the term, t, but belong to category, c. D is the number of documents that do not contain the term, t, and do not belong to category, c. N is the number of training documents. [4].

4.2.5 NGL (Ng-Goh-Low) Coefficient - Explanation

NGL Correlation Coefficient (CC) is a variant of χ^2 metric. A positive NGL CC value indicates that word, w, is a possible feature word and correlates with category, c, while a negative value means word, w, correlates with category, $\bar{c}$. The NGL CC value can be computed as follows:

$$NGL = \frac{\sqrt{Tr} \cdot [P(t_k, c_i) \cdot P(\bar{t_k}, \bar{c_i}) - P(t_k, \bar{c_i}) \cdot P(\bar{t_k}, c_i)]}{\sqrt{P(t_k) \cdot P(\bar{t_k}) \cdot P(c_i) \cdot P(\bar{c_i})}}$$

[9]

Which we can easily compute using the A, B, C, D values as:

$$NGL = \frac{\sqrt{N} \cdot (AD - CB)}{\sqrt{(A+C)(B+D)(A+B)(C+D)}}$$

Uchyigit and Ma tell us that, "the NGL coefficient is reported to have better performance than χ^2" [13] They say so, because NGL "selects words that correlate with c (i.e. are positive) and does not select those words which correlate with $\bar{c}$, unlike the χ^2 statistic" [13].

4.2.6 GSS (Galavotti-Sebastiani-Simi) Coefficient - Explanation

Galavotti-Sebastiani-Simi propose a *simplified* χ^2 statistic. They remove the $\sqrt{N}$ factor, and the denominator completely. They describe the $\sqrt{N}$ factor as being unnecessary. They also remove the denominator, $\sqrt{(A+C)(B+D)(A+B)(C+D)}$, by giving the reason that the denominator gives high Correlation Coefficient score to rare words, and rare categories [13]. The GSS CC value can be computed as follows:

$$GSS = P(t_k, c_i) \cdot P(\bar{t_k}, \bar{c_i}) - P(t_k, \bar{c_i}) \cdot P(\bar{t_k}, c_i)$$

Which we can easily compute using the A, B, C, D values as:

$$GSS = AD - CB$$

Now that we have understanding of these feature selection algorithms, we can move on to implementation. Next chapter describes how values of these algorithms were calculated.

CHAPTER 5

IMPLEMENTATION OF FEATURE SELECTION ALGORITHMS

This section gives implementation details for each algorithm described above. Before we begin implementation details on any algorithm, we will first look at a common task that is required for all algorithms, which is to retrieve the A, B, C, D values from the database. A utility *DBDocBowRaw* is created, which includes many functions related to the "doc_bow_raw" table, and also contains the function *RetrieveABCDValues*. Implementation of this function is given below, which is used to get the A, B, C, D values out of the database and into the application.

Listing 20: RetrieveABCDValues Function

```
1   public function RetrieveABCDValues() {
2     $query =
3     "select pk_doc_bow_raw , a, b, c, d from doc_bow_raw";
4
5     $result = mysql_query ( $query , $this->dbLink );
6
7     if ( $result ) {
8       $list = array();
9       while ( $row = mysql_fetch_assoc ( $result ) ) {
10        $pk = $row['pk_doc_bow_raw'];
11        if ( !isset($list[$pk]) ) {
12          $list[$pk] = array();
13        }
14        unset($row['pk_doc_bow_raw']);
15        $list[$pk] = $row;
16      }
17
18      return $list;
19    }
20    else {
21      return false;
22    }
23  }
```

The while loop (lines 9-16) fetches rows from the resource, *$result*. The purpose of the code inside the while loop is to create a new list, called *$list*, with keys being the primary key values of the “doc_bow_raw” table. Meaning, given the following array data-structure:

Listing 21: Sample RAW Array Result

```
Array (
    [0] => Array (
        [pk_doc_bow_raw] => 404
        [a] => 2
        [b] => 9
        [c] => 14
        [d] => 165
    )
    [1] => Array (
        [pk_doc_bow_raw] => 405
        [a] => 4
        [b] => 10
        [c] => 13
        [d] => 76
    )
    [2] => Array (
        [pk_doc_bow_raw] => 406
        [a] => 2
        [b] => 12
        [c] => 44
        [d] => 165
    )
)
```

The while loop will convert the structure into:

Listing 22: Sample Converted Array Result

```
Array (
    [404] => Array (
        [a] => 2
        [b] => 9
        [c] => 14
        [d] => 165
    )
    [405] => Array (
        [a] => 4
        [b] => 10
        [c] => 13
        [d] => 76
    )
    [406] => Array (
        [a] => 2
        [b] => 12
        [c] => 44
        [d] => 165
    )
)
```

Note that *pk_doc_bow_raw* were dropped from each sub-array, and became *keys* in the array. The following code shows how we call the *RetrieveABCDValues* and retrieve the A, B, C, D values.

Listing 23: Calling RetrieveABCDValues Function

```
$dbDocBOWRaw = new DBDocBOWRaw();
$dbr = $dbDocBOWRaw->RetrieveABCDValues();
```

Now that we have the A, B, C, D values stored in the variable *$dbr* for our feature selection methods, we can begin implementation details for each algorithm. Let us begin with Document Frequency.

5.1 Document Frequency - Calculation Implementation Details

We have already calculated document frequency values as the ‘A’ column in the “doc_bow_raw” table in the database. No further calculations are necessary. Let us look at *information gain* next.

Now that we have the A, B, C, D values in the *$dbr* variable, we can loop through all items in *$dbr* and calculate information gain using the IG formula. Here's the code to calculate and update the information gain values:

Listing 24: Calculate and Update Information Gain Values

```
foreach ( $dbr as $pk => $counts ) {
  $a = $counts['a'];
  $b = $counts['b'];
  $c = $counts['c'];
  $d = $counts['d'];

  $t_c = $a * log ( $a/(($a+$c)*($a+$b)) , 2 );
  $tBar_c = $b * log ( $b/(($b+$d)*($a+$b)) , 2 );
  $t_cBar = $c * log ( $c/(($a+$c)*($c+$d)) , 2 );
  $tBar_cBar = $d * log ( $d/(($b+$d)*($c+$d)) , 2 );

  $dbr[$pk]['ig'] = $t_c + $tBar_c + $t_cBar + $tBar_cBar;
}

$dbDocBOWRaw->UpdateInformationGain ( $dbr );
```

The code above goes through each item in *$dbr*, and calculates the IG values according to the formula, and puts the result back in *$dbr*. At the end, *UpdateInformationGain* is called, and is passed *$dbr* by reference to update the information gain values in the database. The code below shows the *UpdateInformationGain* function.

Listing 25: UpdateInformationGain Function

```
  public function UpdateInformationGain ( &$dbr ) {
    foreach ( $dbr as $pk => $details ) {
      $query =
      "update doc_bow_raw
        set information_gain='" . $details['ig'] . "'
        where pk_doc_bow_raw='" . $pk . "'";
      $result = mysql_query ( $query , $this->dbLink );
    }
  }
```

5.3 Mutual Information - Calculation Implementation Details

Code is given below to calculate and update the mutual information values from the *$dbr* variable. Because the MI formula requires the number of training documents, a variable, *$numTrainingDocuments*, is created with the value of $1,010$.

Listing 26: Calculate and Update Mutual Information Values

```
$numTrainingDocuments = 1010;
foreach ( $dbr as $pk => $counts ) {
   $A = $counts['a'];
   $B = $counts['b'];
   $C = $counts['c'];
   $mi = log( ($A*$numTrainingDocuments) / (($A+$C)*($A+$B)) ,2);

   $dbDocBOWRaw->UpdateMIValue ( $pk , $mi );
}
```

The function *UpdateMIValue* is given below that updates the mutual information values in the database. Note that this function is executed at every iteration in the loop given above.

Listing 27: UpdateMIValue Function

```
   //Passed by reference.
   public function UpdateMIValue ( &$pk , &$mi ) {
      $query =
      "update " . $this->table . " set mutual_information='" . $mi . "'
         where pk_doc_bow_raw='" . $pk . "'";
      $result = mysql_query ( $query , $this->dbLink );
   }
```

5.4 Chi Square - Calculation Implementation Details

Code is given below to calculate and update the χ^2 values from the *$dbr* variable. Again, *$numTrainingDocuments* is used with the value of $1,010$, as the χ^2 formula requires the number of training documents..

Listing 28: Calculate and Update χ^2 Values

```
$numTrainingDocuments = 1010;
foreach ( $dbr as $pk => $counts ) {
   $A = $counts['a'];
   $B = $counts['b'];
   $C = $counts['c'];
   $D = $counts['d'];
   $chiValue = ( $numTrainingDocuments * pow($A*$D-$C*$B,2) )
                  /    ( ($A+$C)*($B+$D)*($A+$B)*($C+$D) );

   $dbDocBOWRaw->UpdateChiValue ( $pk , $chiValue );
}
```

The function *UpdateChiValue* is given below that updates the χ^2 values in the database. Note that this function is also executed at every iteration in the loop given above.

Listing 29: UpdateChiValue Function

```
   //Passed by reference.
   public function UpdateChiValue ( &$pk , &$chiValue ) {
      $query =
      "update " . $this->table . " set chi_square='" . $chiValue . "'
         where pk_doc_bow_raw='" . $pk . "'";
      $result = mysql_query ( $query , $this->dbLink );
   }
```

5.5 NGL - Calculation Implementation Details

Code is given below to calculate and update the NGL coefficients from the *$dbr* variable. The number of training documents is also required to calculate the NGL coefficients. So, a variable, *$numTrainingDocuments*, is created with the value of 1, 010.

Listing 30: Calculate and Update NGL Coefficients

```
$numTrainingDocuments = 1010;
foreach ( $dbr as $pk => $counts ) {
    $A = $counts['a'];
    $B = $counts['b'];
    $C = $counts['c'];
    $D = $counts['d'];
    $ngl = ( sqrt($numTrainingDocuments) * ($A*$D-$C*$B) )
                        /   sqrt( ($A+$C)*($B+$D)*($A+$B)*($C+$D) );

    $dbDocBOWRaw->UpdateNGLValue ( $pk , $ngl );
}
```

The function *UpdateNGLValue* is given below that updates the NGL coefficients in the database. Again, this function is executed at every iteration in the loop given above.

Listing 31: UpdateNGLValue Function

```
    //Passed by reference.
    public function UpdateNGLValue ( &$pk , &$ngl ) {
        $query =
        "update " . $this->table . " set ngl='" . $ngl . "'
            where pk_doc_bow_raw='" . $pk . "'";
        $result = mysql_query ( $query , $this->dbLink );
    }
```

5.6 GSS - Calculation Implementation Details

Code is given below to calculate and update the GSS coefficients from the *$dbr* variable.

Listing 32: Calculate and Update GSS Coefficients

```
foreach ( $dbr as $pk => $counts ) {
   $A = $counts['a'];
   $B = $counts['b'];
   $C = $counts['c'];
   $D = $counts['d'];
   $gss = ($A*$D-$C*$B);

   $dbDocBOWRaw->UpdateGSSValue ( $pk , $gss );
}
```

The function *UpdateGSSValue* is given below that updates the GSS coefficients in the database. Again, this function is executed at every iteration in the loop given above.

Listing 33: UpdateGSSValue Function

```
   //Passed by reference.
   public function UpdateGSSValue ( &$pk , &$gss ) {
      $query =
      "update " . $this->table . " set gss='" . $gss . "'
         where pk_doc_bow_raw='" . $pk . "'";
      $result = mysql_query ( $query , $this->dbLink );
   }
```

We have calculated all needed values using our feature selection algorithms. Now, we run our text categorization engine and determine which feature selection algorithms can give us the best results. The next chapter reports the results.

CHAPTER 6

RESULTS

6.1 Using All Features

Recall that we already created the “doc_bow_raw_test” table from the test data. This same table is used in testing. We also calculated *precision*, *recall*, and *F1* values to determine how accurately documents were categorized when using different feature selection methods. We can calculate *recall* and *precision* as follows:

$$recall = \frac{TP}{TP + FN}$$

$$precision = \frac{TP}{TP + FP}$$

Here, *TP* is the number of true-positives, *FP* is the number of false-positives, and *FN* is the number of false-negatives.

A true-positive is when a human and the categorization algorithm both agree that a document belongs in the exact same category. For example, if we believe a document is about “Indian” food recipe, then the categorizer must also give “Indian” category as the result.

A false-positive is when a human *knows* that a document must belong to some category, C_A, but the categorizer gives category, C_B, as the result. Both results belong to some parent category, *C*, but category $C_A \neq C_B$. For example, a human knows that a document is about “Indian” food recipe, and categorizes that document into “Indian” category, but the categorizer puts the same document into “Italian” food recipe category. Here, “Indian” and “Italian” categories are close to each other, because they both can belong to a parent category, *Food*. This is a case of false-positive.

A false-negative is when a human *knows* that a document must belong to some category, X, but the categorizer gives a completely different category, Y, as the result. Both the human and the categorizer completely disagree on the result. This can happen, for example, when a human knows that a document is a technology related article, and, say, he or she believes, the document must belong to the "Google" category, but categorizer says the document is about "Chinese" food recipe, and categorizes the same document into "Chinese" category. Even though this is a bit extreme example, categorizer can give terribly wrong results.

F1 is the harmonic mean of precision and recall and is calculated as follows:

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

The following results were achieved when we categorized 338 test documents. Here all *20,419* features from the "doc_bow_raw" table were used for calculation. Meaning, feature selection algorithms were used more as a score than as feature selection.

Method	Total	TP	FP	FN	Precision	Recall	F1
DF	338	232	106	0	0.686	1	0.814
IG	338	276	56	6	0.831	0.979	0.899
χ^2	338	313	25	0	0.926	1	0.962
MI	338	299	38	1	0.887	0.997	0.939
MI (*)	338	304	34	0	0.899	1	0.947
NGL	338	301	37	0	0.891	1	0.942
NGL (*)	338	309	29	0	0.914	1	0.955
GSS	338	253	85	0	0.749	1	0.856

Algorithms marked (*) are optimal runs. Meaning, not all features were used, but rather only a small set of features from *20,419* total features.

The 6 *FN* of the IG method are listed below:

Listing 34: False-Negatives of Information Gain

```
Actual: India Result: Facebook
Actual: India Result: Apple
Actual: India Result: Facebook
Actual: JavaScript Result: Google
Actual: JavaScript Result: Google
Actual: JavaScript Result: Facebook
```

The *FN* of the MI (non-optimal) method is listed below:

Listing 35: False-Negatives of Mutual Information

```
Actual: India Result: Google
```

Our main purpose was to get as best *recall* values as possible, as to avoid false-negatives. For most methods, we got recall of 1. Meaning, our results were what we wanted. However, here, we used all *20,419* features, which is not what we should have used. The real purpose of any feature selection algorithm is to score features and *keep* the most informative features, and *remove* all other non-informative features. Next, we address this issue.

6.2 Using Selected Features

Our previous results used all *20,419* features, and we wanted to improve on this count. In other words, we wanted to use as less features as possible, and still maintain a high number of *TP* values. In this section, we address this very issue.

For each feature selection algorithm, we determined what was the min-value and the max-value of that algorithm. Then, we used a loop for each feature selection method from an approximate min value to an approximate max value, and selected feature only in that range, and ran the categorization algorithm, and this gave us results on how many *TP* it found.

Below we explain min to max range for each algorithm, and show the *TP* results after selecting only partial features. Keep in mind that we have a total of *20,419* features, and *338* test documents.

6.2.1 Document Frequency - Selecting Partial Features

We determined document frequency's range to be from 0 to 160. We, then, ran a loop from 0 to 160 and incremented the *cutoff* value by 10. Here, for example, *cutofff* value of 20 would mean that, a feature has to appear in at least 20 documents. In other words, select only those feature that are in 20 or more documents. Below are the sample results:

Cutoff	# of Features Used	# of Features Removed	TP
0	20,419	0	232
10	1,714	18,705	218
20	729	19,690	201
30	412	20,007	182
40	261	20,158	179
50	175	20,244	164
60	127	20,292	135

Document frequency is not a very useful feature selection algorithm. But even for a simple feature selection algorithm, such as DF, after removing $18,705$ features, we still got *TP* of 218. Meaning even after removing $18,705$ features, we lost only 14 *TP*.

Next, we look at how well *information gain* performs.

6.2.2 Information Gain - Selecting Partial Features

Information gain values ranged from $-15,000$ to 0. Below are the two sample results we obtained:

Cutoff	# of Features Used	# of Features Removed	TP
-15,000	20,419	0	276
-10,000	6,184	14,235	276

Notice that we still have the same number of TP after removing $14,235$ features.

6.2.3 Mutual Information - Selecting Partial Features

Mutual information values ranged from -6 to 4, and we ran the loop by incrementing the *cutoff* value by 0.5. Full results are listed below:

Cutoff	# of Features Used	# of Features Removed	TP
-6	20,419	0	299
-5.5	20,418	1	299
-5	20,415	4	299
-4.5	20,398	21	299
-4	20,374	45	298
-3.5	20,322	97	298
-3	20,228	191	298
-2.5	20,043	376	297
-2	19,770	649	299
-1.5	19,398	1,021	300
-1	18,849	1,570	300
-0.5	18,020	2,399	302
0	16,953	3,466	303
0.5	15,382	5,037	303
1	13,509	6,910	304
1.5	11,439	8,980	303
2	9,415	11,004	297
2.5	7,374	13,045	286
3	4,921	15,498	188
3.5	955	19,464	56

Using features with positive *mutual information* value yield better results. The point here to notice is that, even after removing $6,910$ feature, we got even better results. Moreover, when we used all *20,419* features, we got 299 TP. But when we removed $11,004$ features (cutoff 2), we got 297 TP.

6.2.4 χ^2 - Selecting Partial Features

Recall from our previous results that we got 313 TP. For our case, χ^2 is the best feature selection algorithm. Next, we wanted to see how well χ^2 performed when we removed as many features as possible. We determined the range of χ^2 values to be from 0 to about 400. We ran the loop, and incremented the *cutoff* by 10. The results we obtained are shown below:

Cutoff	# of Features Used	# of Features Removed	TP
0	20,419	0	313
10	4,916	15,503	313
20	2,039	18,380	311
30	1,270	19,149	308
40	841	19,578	306
50	635	19,784	306
60	480	19,939	304
70	369	20,050	301
80	310	20,109	305
90	264	20,155	304
100	218	20,201	303

Cutoff	# of Features Used	# of Features Removed	TP
110	186	20,233	303
120	161	20,258	302
130	138	20,281	306
140	123	20,296	300
150	106	20,313	302
160	95	20,324	300
170	90	20,329	297
180	80	20,339	302
190	73	20,346	301
200	65	20,354	303
210	61	20,358	302
220	52	20,367	283
230	47	20,372	282
240	43	20,376	273
250	36	20,383	231
260	33	20,386	231
270	29	20,390	228
280	28	20,391	231
290	26	20,393	213
300	26	20,393	213

Cutoff	# of Features Used	# of Features Removed	TP
310	24	20,395	213
320	21	20,398	184
330	19	20,400	184
340	17	20,402	175
350	16	20,403	145
360	12	20,407	120
370	11	20,408	120
380	9	20,410	66
390	9	20,410	66

Again, χ^2 feature selection algorithm performed very well even with less amount of features. Even when we removed $15,503$ features, and used only $4,916$ features, results (TP) did not change. The number of TP stay above 300 even when we use only 61 features.

6.2.5 NGL Coefficient - Selecting Partial Features

NGL was another algorithm that also performed well. NGL values ranged from -7 to 21, and we ran the loop by incrementing the *cutoff* by 1. Below are the results:

Cutoff	# of Features Used	# of Features Removed	TP
-7	20,419	0	301
-6	20,418	1	301
-5	20,409	10	301
-4	20,358	61	303
-3	20,223	196	305
-2	19,809	610	305
-1	18,877	1,542	307
0	17,061	3,358	307
1	14,163	6,256	306
2	10,737	9,682	306
3	6,078	14,341	309
4	2,572	17,847	305
5	1,540	18,879	304
6	995	19,424	296
7	642	19,777	295
8	443	19,976	281
9	306	20,113	298
10	218	20,201	298
11	157	20,262	304
12	116	20,303	300

Cutoff	# of Features Used	# of Features Removed	TP
13	91	20,328	300
14	70	20,349	304
15	50	20,369	282
16	36	20,383	231
17	26	20,393	213
18	20	20,399	184
19	12	20,407	120
20	9	20410	66

As can be easily seen from these results, NGL performs almost as well as the χ^2 algorithm.

6.2.6 GSS Coefficient - Selecting Partial Features

GSS coefficient values ranged from $-48,754$ to $123,981$, and the loop was ran with the *cutoff* incremented by $10,000$. Here are the results:

Cutoff	# of Features Used	# of Features Removed	TP
-48754	20419	0	253
-38754	20416	3	253
-28754	20398	21	253
-18754	20345	74	252
-8754	20096	323	253
1246	7141	13278	250
11246	879	19540	236
21246	393	20026	220
31246	212	20207	201
41246	132	20287	192
51246	94	20325	176
61246	70	20349	169
71246	50	20369	135
81246	41	20378	125
91246	22	20397	73
101246	8	20411	71
111246	5	20414	73
121246	2	20417	33

6.3 Explanation of Results

We have demonstrated 6 different feature selection algorithms, and χ^2 and *NGL* algorithms have out-performed all other algorithms. These results seem to be in agreement with Kotcz, Prabakarmurthi, Kalita, and Yang & Pedersen's results, as they also found Chi-Squared to be very effective [3, 11]. Dasgupta *et al.* [5] describe that "it is often difficult to claim more than a vague intuitive understanding of why a particular feature selection algorithm performs well when it does" [5].

Kotcz, Prabakarmurthi, and Kalita have also shown that, "by reducing the feature space, the accuracy of a classification method can be increased and, even when only very few of the original features are kept, good accuracy can maintained" [11]. Our results agree. We have shown that by *keeping* only the informative features, and *removing* all other non-informative features, we can either improve results, *TP*, or can get same results by reducing the feature set by a very large degree. As can be seen from the results that even after removing many features, we were still able to get *TP* that were close to the optimal.

Conducting one sample T-Test on precision values, with *hypothetical mean* of 0.9483, we got P value of 0.0362, and this indicates that the difference is statistically significant.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Text categorization is very important, but we believe, the problem of feature selection is as much, or more important than text-categorization. In this thesis, we discussed many important topics ranging from collecting data, to organizing data and ultimately using the organized data to efficiently conduct tests using the feature selection algorithms.

In chapter 2, we showed how we used a MySQL database to efficiently store our collection of documents. We would like to mention again that using a MySQL database to store data was a really good decision, as not only it made implementation easier for us, but it was also much more efficient than using text files. MySQL also has the ability to index data, which can help retrieve and update data speedily. In the same chapter, we described what setup was used for the database, and described the structure of each table in the database. Then we explained the details of the *b8* lexer, and described how we used the *b8* lexer to create bag of words, or BOW, to help us train and test data.

Chapter 3, even though a short one, is the *core* of this thesis. In chapter 3, we described how the counting of the training documents was done using BOW. We explained what the A, B, C, D values were, and how we calculated each of those values using our training data. We also explained why these A, B, C, D values are important and very much needed for the feature selection algorithms.

In chapter 4, we first showed the important of feature selection algorithms, and then we gave explanation on the following feature selection methods: *Document*

Frequency, *Information Gain*, *Mutual Information*, *Chi Square*, *NGL (Ng-Goh-Low) Coefficient*, and *GSS (Galavotti-Sebastiani-Simi) Coefficient*.

In chapter 5, we implemented the above mentioned feature selection algorithms. In this chapter, we gave code listings on how each of the algorithms were implemented.

Results are shown in chapter 6. We showed what feature selection algorithm turned out to be the best algorithm for our case. We then went on to show that even after removing features, and in some cases more than 90%, we were still able to maintain over 99% of *TP* in our results. This study has shown how powerful feature selection algorithms can be.

We have shown some dramatic improvements using our results. And we believe feature selection methods should be researched further, on Very Large Scale Data. The number of training and test documents used in this thesis are very small compared to what is out there on the Internet. Moreover, in this thesis, only 9 categories were used. In the real world, hundreds of categories exist. To have a large-scale categorizer, powerful feature selection algorithm(s) would have to be developed. And we believe, this area could be researched and tested on further.

Another area that we and George Forman believe is that of "hierarchical categories". Forman says the following:

> Hierarchy is among the most powerful of organizing abstractions. Hierarchical classification includes a variety of tasks where the goal is to classify items into a set of classes that are arranged into a tree or directed acyclic graph.

[7]

and Forman believes,

> The problem is cast as a multi-label task to select multiple interior nodes, optionally including all super-classes along the paths to the root.
>
> [7]

We strongly believe, conducting further research on the topics mentioned above would be very helpful, as it can ultimately help categorize all documents in the world.

BIBLIOGRAPHY

[1] S. Li, R. Xia, C. Zong, and C.-R. Huang, "A framework of feature selection methods for text categorization.," in *ACL/AFNLP'09*, pp. 692–700, 2009.

[2] W. Shang, H. Huang, H. Zhu, Y. Lin, Y. Qu, and Z. Wang, "A novel feature selection algorithm for text categorization," *Expert Systems with Applications*, vol. 33, no. 1, pp. 1–5, 2007.

[3] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," pp. 412–420, Morgan Kaufmann Publishers, 1997.

[4] M. Z. Fadi Thabtah, Mohammad Ali H. Eljinini and W. M. Hadi, "Nave bayesian based on chi square to categorize arabic data," *Communications of the IBIMA*, vol. 10, no. 20, pp. 158–163, 2009.

[5] A. Dasgupta, P. Drineas, B. Harb, V. Josifovski, and M. W. Mahoney, "Feature selection methods for text classification," *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining KDD 07*, vol. 21, no. 2, p. 230, 2007.

[6] G. Forman, I. Guyon, and A. Elisseeff, "An extensive empirical study of feature selection metrics for text classification," *Journal of Machine Learning Research*, vol. 3, pp. 1289–1305, 2003.

[7] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *J. Mach. Learn. Res.*, vol. 3, pp. 1289–1305, March 2003.

[8] M. Porter, "The porter stemming algorithm." `http://tartarus.org/martin/PorterStemmer/`, 2006.

[9] D. Fragoudis, D. Meretakis, and S. Likothanassis_aff1n3, "Best terms: an efficient feature-selection algorithm for text categorization," *Knowl. Inf. Syst.*, vol. 8, no. 1, pp. 16–33, 2005.

[10] M. Rogati and Y. Yang, "High-performing feature selection for text classification," pp. 659–661, 2002.

[11] A. Kolcz, V. Prabakarmurthi, J. Kalita, and P. Inc, "Summarization as feature selection for text categorization," 2001.

[12] R. Mukras, N. Wiratunga, R. Lothian, S. Chakraborti, and D. Harper, "Information gain feature selection for ordinal text classification using probability re-distribution."

[13] G. Uchyigit and M. Ma, *Personalization techniques and recommender systems*, p. 310. Series in machine perception and artificial intelligence, World Scientific, 2008.

VITA

Graduate College
University of Nevada, Las Vegas
Kandarp Dave

Degrees:
Bachelor of Science, Computer Science, 2009
University of Nevada, Las Vegas

Thesis Title: Study of Feature Selection Algorithms for Text-Categorization

Thesis Examination Committee:
Chairperson, Dr. Kazem Taghva, Ph.D.
Committee Member, Dr. Ajoy K. Datta, Ph.D.
Committee Member, Dr. Laxmi P. Gewali, Ph.D.
Graduate College Representative, Dr. Muthukumar Venkatesan, Ph.D.