Project Report

The implementation of text categorization with term association

Winter – 2003
Instructor: Dr. Vlado Keselj

Team Member:	**Miao, Yingbo**	**B00181251**
	Wei, Gang	**B00344693**
	Yu, Zheyuan	**B00182683**
	Sheng, Xin	**B00140586**

Due Date: **April, 7, 2003**

Content

Abstract
Our project is an implementation of Apriori based ARC-BC algorithm to classify text documents. The main idea is based on ARC-BC [3] [4]. This report reviews text categorization and traditional approaches to learning algorithms. Then we propose an association rule approach to text categorization. We test the text categorization with several different size data sets and support values. Accuracy is evaluated as well. Experiment results shows our implementation is efficient and comparable.

1. Introduction
Amazing development of Internet and digital library has triggered a lot of research areas. Text categorization is one of them. Text categorization is a process that group text documents into one or more predefined categories based on their contents [1]. It has wide applications, such as email filtering, category classification for search engines and digital libraries.

Basically there are two stages involved in text categorization. Training stage and testing stage. In training stage, documents are preprocessed and are trained by a learning algorithm to generate the classifier. In testing stage, a validation of classifier is performed. There are many traditional learning algorithms to train the data, such as Decision trees, Naïve-Bayes (NB), Support Vector Machines (SVM), k-Nearest Neighbor (kNN), Neural Network (NNet), etc. In this project, we apply association rule based ARC-BC algorithm to generate a set of rules associated with each category, and from these rules classifier is further generated. In this way, training time is relatively fast and the rules generated are understandable and can be manually updated or adjusted if needed.

In this project, data documents are preprocessed and ARC-BC algorithm is implemented to generate the classifier. Then the classifier is tested for validation. This project report is organized as follows: Section 2 is an introduction of text document categorization. A detail description of Apriori algorithm and ARC-BC algorithm are given in section 3. Our design document is in section 4, which includes our design flow chart and code description. Programs are tested on different data sets and support values. Accuracy was evaluated as well. Experiment results are described in section 5. Then a summary is given in section 6. In section 7, user documentation is presented.

2. Text Document Categorization Introduction

2.1 Overview of Text Categorization

With the increasing of information on the internet and development of digital articles, people urgently need an efficient tool to automatically classify the information into categories. In this way, we can easily search, filter and store the large amount of resources. Automated text categorization is a process that assigning pre-defined category labels to new documents based on the contents [2].

Text categorization has many applications. For example, we can classify web pages into different categories to speed up the internet search, which is very useful for some search engines like Yahoo. Text categorization can be applied to filter emails to judge if it is spam email and further folder the emails. For news agencies, such as Globe and Mail, they receive thousands of articles a day. Articles can be classified to several categories like sports, politics, medical and etc by text categorization methods. In digital library, people use key words to index articles, text categorization can also be used to classify the digital articles according to the subjects or key words.

Usually there are two stages involved in text categorization, training stage and testing stage. In training stage, we need a learning algorithm to learn the training documents to build the classifier. In testing stage, classifier is used to categorize documents.

2.2 Traditional Approaches for Learning Algorithm

Most of the researches in text categorization come from the machine learning and information retrieval communities such as decision trees, naïve-Bayes (NB) [9], Support Vector Machines (SVM) [11], k-Nearest Neighbor (kNN) [10], Neural Network (NNet) and etc. Among these methods, SVM has the best performance. KNN is a simple statistic method and it also shows very good performance. NB is relatively underperforming the others [2].

2.3 Association Rule Approach

Association rule approach to categorize the documents is relatively new. The concept is to discover the strong patterns that are associated with the class labels and then take advantage of these patterns to build the classifier. Once classifier is built, new documents are categorized into the proper class. We will introduce this approach in more detail in Part 3.

3. Association Rule Algorithm

3.1 Association Rule and Apriori Algorithm

Association rule mining is a data mining task that discovers relationships among items in a transactional database [12]. It is described as follows: Let I= { i_1, i_2,…i_m}, be a set of items. Let D, the task relevant data, be a set of database transactions where each transaction T is a set of items such that T $\subseteq$ I. Each transaction is associated with an identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if $A \subseteq T$. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$ and $A \cap B$ = NULL.

Following key parameters are used to generate valuable rules:

- Support (s)

Support (s) of an association rule is the ratio (in percent) of the records that contain $X \cup Y$ to the total number of records in the database: support($X \Rightarrow Y$) = Prob { $X \cup Y$ }

- Confidence (c)

For a given number of records, confidence (c) is the ratio of the number of records that contain $X \cup Y$ to the number of records that contain X.
confidence($X \Rightarrow Y$) = Prob { Y| X } = (support ($X \cup Y$)) / (support (X))

- Strong Association Rules:

Rules that satisfy both a minimum support threshold (min_sup) and a minimum confidence threshold (min_conf) are called strong rules. Strong rules are what we are interested in.

There are two main steps to process association rule mining: Step 1 is to use prior knowledge find all frequent itemsets by Apriori algorithm. It uses iterative search and use k-itemsets to find (k+1) itemsets.[5] Every itemset occurs at least more than the min_support value. Step 2 is to generate strong association rules from frequent itemsets, which means these rules must satisfy both min_support value and min_confidence value.

3.2 Apriori Based ARC-BC Algorithm

A new document categorization algorithm was proposed by M. Antonie and Osmar R. Zaiane [3]. It has following advantages: it makes no assumption of term independence and it is fast during both training and categorización. ARC-BC is an Apriori based algorithm that only interested in rules that indicate a category label. As shown in Figure 3.1, in this algorithm each set of documents that belong to one category is considered as a separate text collection to generate association rules. If a document belongs to more than one category, this document will be present in each set associated with the categories that the document falls into. [4]

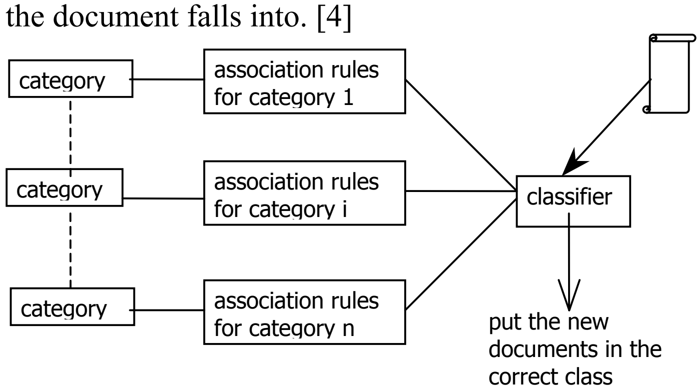

Figure 3.1 ARC-BC Algorithm [4]

ARC-BC Algorithm is described as follows: [3]

- Input:

A set of documents (D) of the form $D_i = \{c_i, t_1, t_2, \ldots, t_n\}$ where c_i is the category attached to the document and , t_n are the selected terms for the document; *min_support* is the threshold for support;

- Output:

Association rules like: $t_1 \wedge t_2 \wedge \ldots \wedge t_n$ => Ci

t_j is a term and *Ci* is the category

- Pseudo Code:

```
(1) C1 = {Candidate 1 term-sets and their support }
(2) F1 = { Frequent 1 term-sets and their support}
(3)  for ( i = 2; Fi-1  != Æ; i++){
(4)        Ci = candidates generated from Fi-1
(5)       Di = FilterTable (Di-1 , Fi-1)
(6)       foreach document d in Di {
(7)                 foreach c in Ci  {
(8)                           c.support += Count(c, d)
(9)                 }
(10)     }
(11)      Fi  = {c Є Ci | c.support  > min_support}
(12)   }
(13)  Sets = Ui  {c Є Fi | i  > 1}
(14)  foreach itemset I in Sets{
(15)     R += { I => Category}
(16)  }
```

In step 2, it generates the frequent 1-itemset. In steps 3-12, it generates all the k-frequent itemsets. In step 14-16, it generates the association rule. It's almost same as the Apriori, but there are some differences:

1) In step 5, The filtertable function removes the terms not in Frequent i-1 sets, which are not useful in the next loop.
2) In step 14-16, it generates rules by combining the frequent itemsets with category.

4. Design and Implementation

4.1 System Overview

Basically, there are two stages involved in text categorization as shown in Figure 4.1: training stage and testing stage. In training stage, we have some predefined documents and we will try to learn these documents and generate the document classifier. In this stage, we preprocess the documents to represent them suitable for learning algorithm. Then we will implement an algorithm to learn the training documents to generate the document classifier. In our project, association rule mining algorithm is applied to generate the associative classifier. In testing stage, we put the new documents into the document classifier and get the classified documents. In project implementation, there are three parts: data preprocessing, generate association classifier and validation.

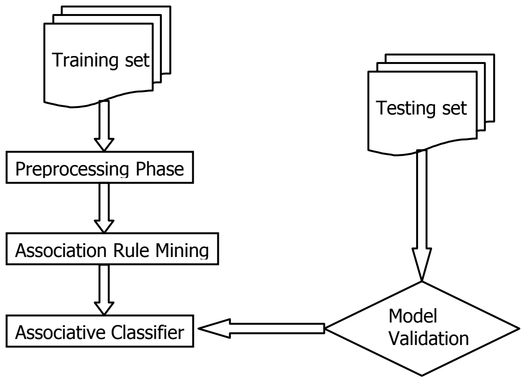

Figure 4.1 Text categorization flow chart

4.1.1 Part 1: Data Preprocessing

In preprocessing phase, test documents are regarded as transactions where items are words or phrases from the document as well as the categories to which the document belongs. Given a text document, which is represented by strings or characters, first we should change it into the format that is suitable for automatic text categorization learning algorithm. Reduce the feature set or data cleaning is usually the approach by selection a set of noise words. From Zipf's law [8], we know that the high frequency terms in top of term list such as "a", "the" and the low frequency terms with one or two occurrences in a document are useless to represent a document. Such kind of terms is usually treated as noise for document representation, and usually is removed before processing. As described above, a data cleaning phase is required to weed out those words that are of no interest in building the associative classifier. our processing includes: remove the tags from the text and retrieve the document, remove any words appearing in a stop list found in Van Rijsbergen [7] the remaining terms are applied to Porter's stemming algorithm [6], which aims to get the stem of the word, such as removing the word suffix. It is only after the terms are selected from the cleaned documents that the transactions are formed.

The size of our news data file is 27.8 M bytes. The documents are extracted and saved into "Repository" file, which is 13.4 M bytes after compression using ZLIB algorithm. Terms for each document are extracted and stemmed, stop words are removed. The frequency of all the terms in a document and in a category are counted and stored in vector and term files respectively. "Category" file is generated to store the document Ids in each category. After data preprocessing, three files, "Category", "Vector" and "Term" will be used for classifier generating phase.

Since the term frequency in Vector file is not used in current version of ARC-BC, and for the convenience of programming, the Documents file is generated based on Vector file.

File format:

Index File

GDBM Key

Term (String)	Total number Of documents (2bytes)	docid (2bytes)	docid (2bytes)	…

Vector File

GDBM Key

Doc ID (String)	Term ID (3 bytes)	Frequency (3bytes)	Term ID (3bytes)	Frequency (3bytes)	….

Documents File

GDBM Key

Doc ID (3bytes)	Terms (HashSet, each item in it is Integer, reprensenting the Term ID)

Repository File

GDBM Key

Doc ID (String)	Text of Document (String)

Category File

GDBM Key

Category (String)	Doc ID (2 bytes)	Doc ID (2 bytes)	….

Term File

GDBM Key

Category (String)	Term ID (3bytes)	Frequency (3bytes)	….

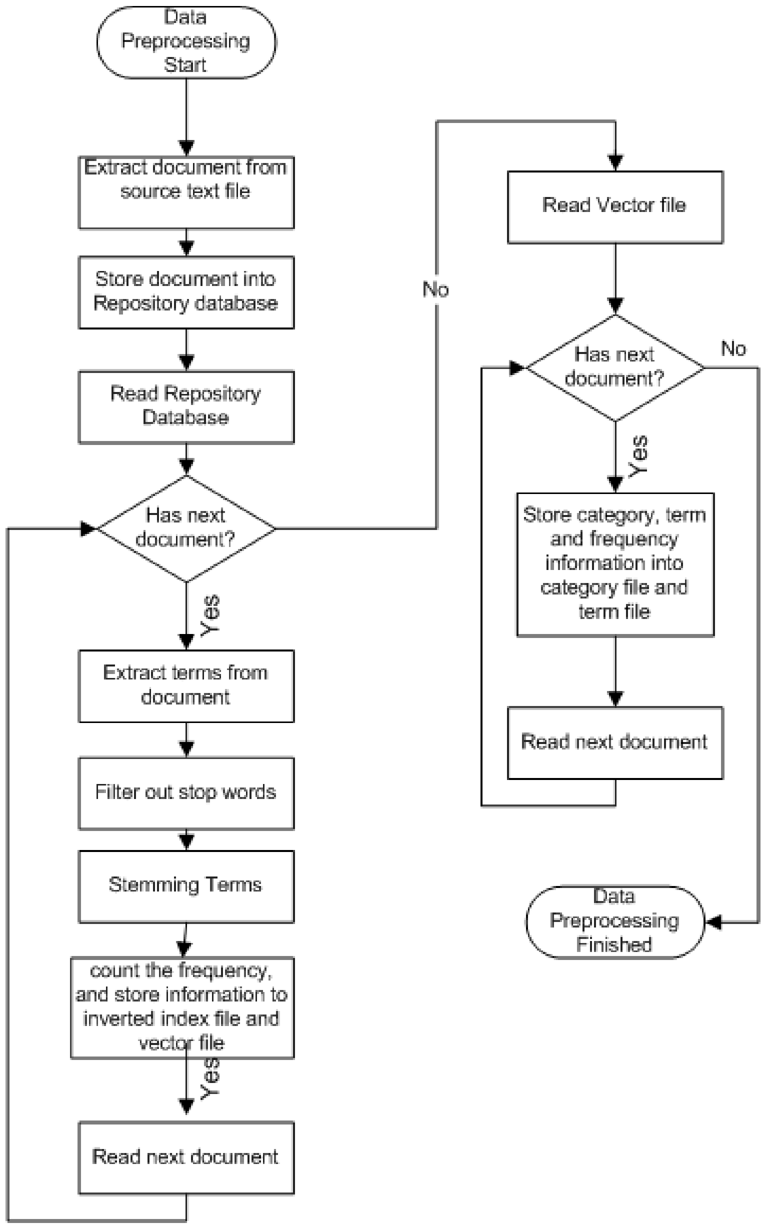

Figure 4.2 Flow Chart of Data Preprocessing

4.1.2 Part 2: Generate Classifier by ARC-BC

After preprocess the data, we start to generate the classifier. For an automatic text categorization process, we should transform documents into transaction to represent them suitable for learning algorithm. That is, for a certain document Di, we divide it into two sets: first is the categories set of this certain document, cc1 to ccm, which is the subset of the whole categories set. And the second set of the document is the term set, tt1 to ttn, which is the subset of the whole term set. We can represent them as:

categories set $C = \{c_1, c_2, \ldots, c_m\}$

term set $T = \{t_1, t_2, \ldots, t_n\}$

document $D_i = \{c_{c1}, c_{c2}, \ldots, c_{cm}, t_{t1}, t_{t2}, \ldots, t_{tn}\}$

For each category, we apply ARC-BC algorithm to generate the candidate sets, calculate the support and generate the frequent sets. The general flow chart for part 2 is shown in Figure 4.3. Detail ARC-BC algorithm implementation flow chart is shown in Figure 4.4.

At the beginning, it gets 1-itemset from the category. Then it generates all the k-frequent itemsets from candidate sets. The third step is to calculate the support and delete those candidates less than min_support value, therefore generate the frequent set and we get the classifier.

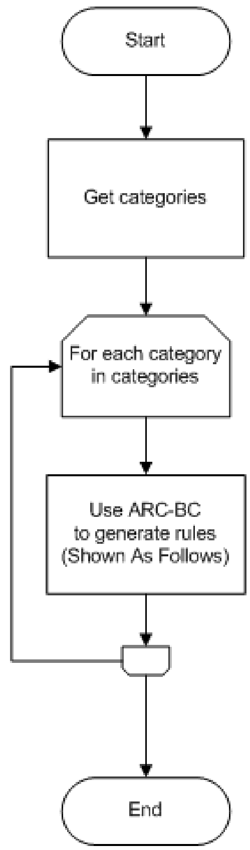

Figure 4.3 Flow Chart for Generate Rules

For the details of “Using ARC-BC to generate rules”, please reference Figure 4.4.

ARC-BC Algorithm Flow Chart

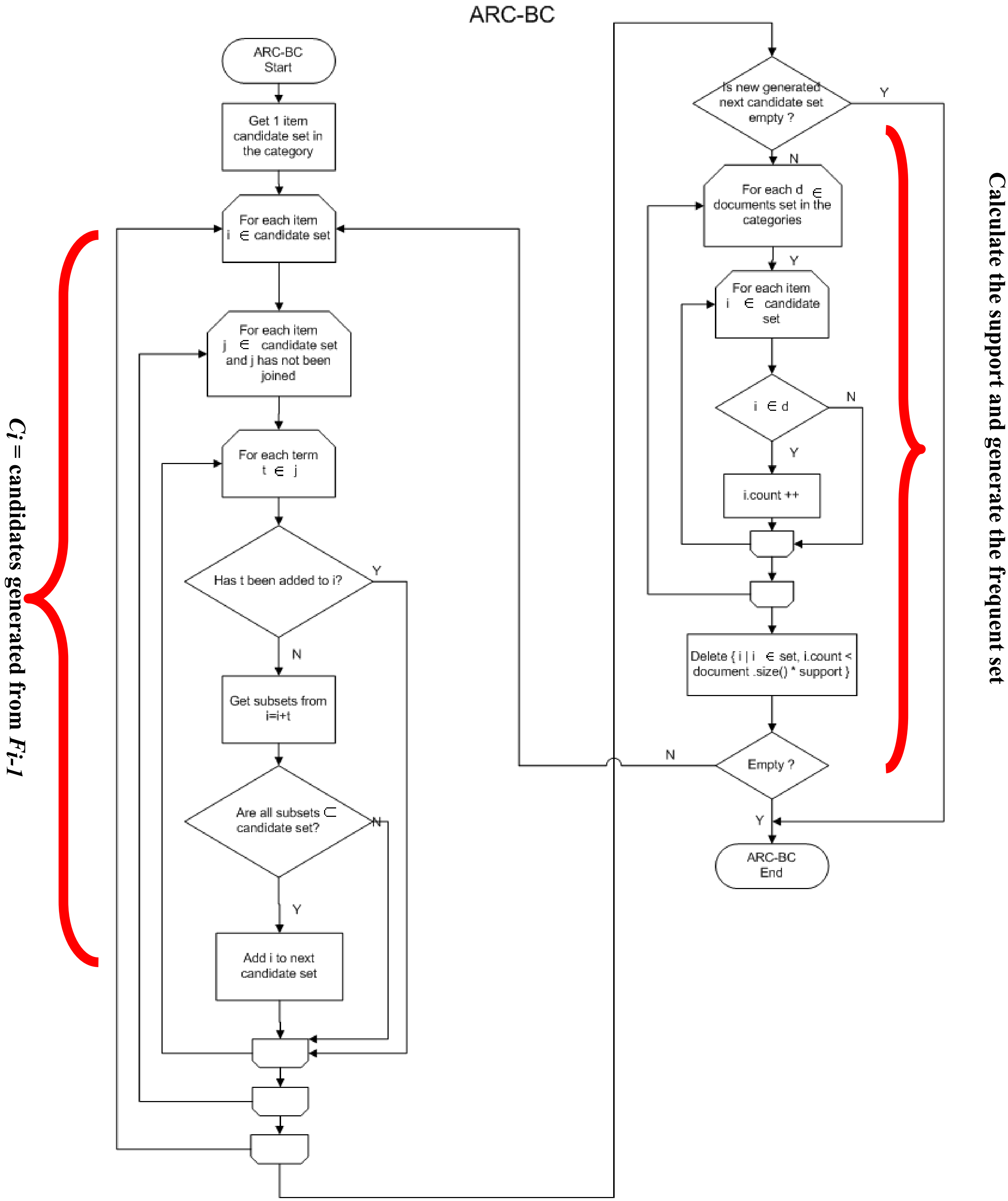

Figure 4.4 Flow Chart for the ARC-BC

Also we generate all k item frequent sets (k = 1, 2, 3…), we only save the last none empty k frequent set as the rules that will be used in text classifier, because the items in the last none empty k item show up most frequently in the documents.

If there are too few documents in a category, there may be too many noisy rules. For example, if there is only one document in a category, all term combinations will be kept, but they are noisy rules. The program simply bypass this kind of category, and the threshold can be set in the configure file.

4.1.3 Part 3: Validate text classifier

Our classification method is: for a new document, first, we count the number of rules this document satisfy for each category, then label the document with category that has max count number. The result shows the accuracy of classifier. Flow chart is shown as Figure 4.5.

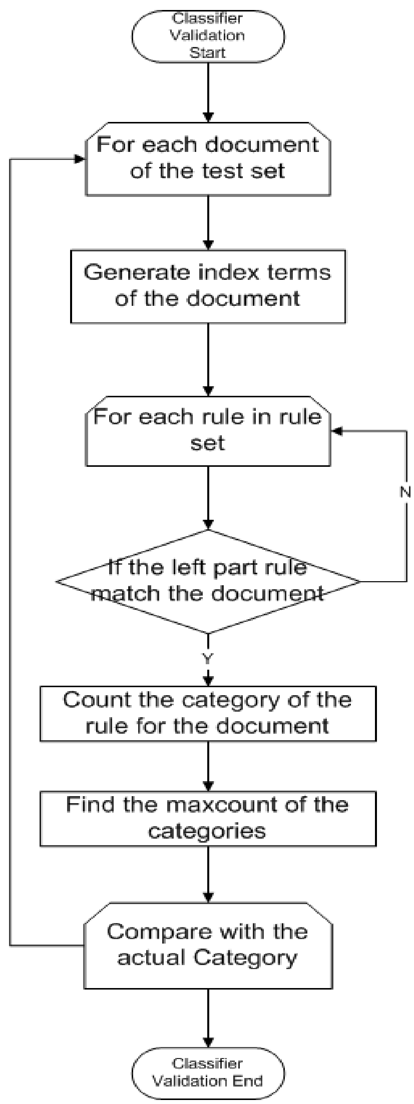

Figure 4.5 Flow Chart for Classifier Validation

4.2 Program Structure

4.2.1 Part 1: data preprocessing

1) class IRZIPG → A class compress a string

- Method:
  - public static String compress(String s)
  - public static String uncompress(String s)

2) class NewString → Compile it, import the Porter class into you program and create an instance. Then use the stripAffixes method of this method which takes a String as input and returns the stem of this String again as a String.

- Methods:
  - private String Clean(String str)
  - private boolean hasSuffix(String word, String suffix, NewString stem)
  - private boolean vowel(char ch, char prev)
  - private int measure(String stem)
  - private boolean containsVowel(String word)
  - private boolean cvc(String str)
  - private String step1(String str)
  - private String step2(String str)
  - private String step3(String str)
  - private String step4(String str)
  - private String step5(String str)
  - private String stripPrefixes (String str)
  - private String stripSuffixes(String str)
  - public String stripAffixes(String str)

3) class Preprocessing → This class is used to do preprocessing for Association Rule for Document Categorization, CS6405 Data Mining project.The documents will be readed from source text file, which are in SGML format. All the tags will be removed. Stop words will be filtered out. The terms in the documents set are extracted and the term frequencies are calculated. Some files, vector, category, term are generated.

- Methods:
  - public static void main(String[] args)

4) class Repository → A class extracting documents from source text file. Create the repository database which stores all document one by one with tags. In repository database, each document is saved as an entry in GDBM file

- Attribute:
  static public int BUFF_SIZE → BUFF_SIZE is the default size of Buffer
  static byte[] buffer → Buffer is used to improve the performance of I/O
- Methods:
  - public Repository(String fileFrom, String fileTo) throws IOException
  - static public void InsertEntry(int intKey, String strItem, GdbmFile db)
  - static public String GetDoc(int intKey,GdbmFile db)

5) **class Indexer** → A class building inverted index and vector file

- Attribute:
    - private String strFileFrom;
    - private Vector v;
    - public StopWord sw;
    - private TreeMap m;
    - private int intStopWordCount
    - static GdbmFile vectorDatabase, indexDatabase, repositoryDatabase
    - private int intTermNo=0;
    - private HashMap termHashMap;
    - private Porter st;
    - public Indexer () throws IOException
- Methods:
    - public void CreateIndexAndVector(String strIndexFile,String strVectorFile,String strRepositoryFile) throws IOException
    - public void InsertTermToInvertedIndex(String strNo, int intDocID) throws IOException
    - private void InsertToVector(int docid) throws IOException
    - private void InsertOrAppendToMapOfTermList(String strTerm) throws IOException
    - public String Stemming(String strTerm)
    - public void BuildTermListProcessing (int docid, String s) throws IOException
    - public void BuildTermCode(String strTermCodeFile) throws IOException
    - public void BuildTermList() throws IOException

6) **class CategoryIndexe** → A class building category index file

- Attribute:
    - private GdbmFile indexDatabase;
    - private GdbmFile vectorDatabase;
    - private GdbmFile repositoryDatabase;
    - private GdbmFile categoryDatabase;
    - private GdbmFile termDatabase;
- Methods:
    - public CategoryIndexer ()
    - public String getTermInCategory (byte[] byteTermList, byte[] byteTermNo, int intTermFrequency)
    - public String getDoc(int intDocID)
    - public String getCategory(String strDoc)

7) **class StopWord**

- Attribute:
    - private String strFileFrom="stopwords";

        - private InputStream in = null;
        - private Vector v;
        - private int intSize=0;
    - Methods:
        - public StopWord () throws IOException
        - public int size()
        - static public boolean compare(byte[] s, byte[] d)
        - public boolean greater(byte[] s, byte[] d) throws IOException
        - public boolean IsStopWord(byte[] term) throws IOException

8) class StopWord → Convert Vector file format to Document file format

- Methods: main()

4.2.2 Part 2: Generate classifier by ARC-BC

1) class GenRules → This class implements the ARC-BC algorithms

- Methods: public static void main(String args[])

2) class Arcbc → This class implements the ARC-BC algorithms

- Attribute:

CanSet candidates → The candidates set as well as the frequent set
int minSupport → The minimum support
int minCount → minCoun t= minSupport * transactions count
DocSet docSets → The DocSet object to commucate with data sets

- Methods:
    - public Arcbc(int aMinSupport, DocSet aDocSet)
    - public Set generateAllRules()
    - public boolean genFreqSet()

3) class CanSet → This class is used to generate and save rules.

a. Attribute:

Set canSet → The set to store candidates

- Method:
    - public CanSet(Set aSet)
    - public Set getCandidates()
    - public void add(CanSetItem aItem)
    - ArrayList genSubSets(CanSetItem aOrgSet, Integer addingTermID)
    - public boolean genNextCanSet()
    - public String toString()

4) class CanSetItem.java extends HashSet → The candiate or frequent set, including the support(count) of each item.

- Attribute:

int support → The support(count) of this item

- Method:

  - public int getSupport()
  - public void setSupport(int aSupport)
  - public void addSupport(int aAdd)

5) interface DocSet → The DocSet interface is designed to provid a common protocol for operating data sets.

- Method:
  - public String[] getAllCategories()
  - public void saveRules(String aCategory, HashSet aRules);
  - public void setCategory(String aCategory)
  - public void initialize()
  - public Set getOneItemCanSet()
  - public boolean hasNext()
  - public Doc next()
  - public int docsCount()

6) class GdbmDocSet implements DocSet → The GdbmDocSet class implements the DocSet interface using Gdbm file format.

- Attribute:

String categoryPatht → The paths and filename of category GDBM file.
String documentsPath → The paths and filename of document GDBM file.
String canFirstPath → The paths and filename of 1 item candidates GDBM file.
String rulesPath → The paths and filename of rules GDBM file to save rules.
String category→ The label of category
List documents → Documents in the category
int nextDocument → The pointer points to the next document

- Method:
  - GdbmDocSet()
  - public String[] getAllCategories()
  - public void saveRules(String aCategory, HashSet aRules)
  - public void setCategory(String aCategory)
  - public void initialize()
  - public Set getOneItemCanSet()
  - public boolean hasNext()
  - public Doc next()
  - public int docsCount()

7) interface Doc → The Doc interface is designed to provid a common protocol for interface DocSet to operate a document, which is the a of a terms.

- Method:
  - public Set getTerms();
  - public boolean isContain(Collection aTerms)

8) class GdbmDoc implements Doc → The GdbmDoc class implements the Doc interface.

b. Attribute:

Set terms → The terms of the doucment

- Method:
    - public Set getTerms()
    - public boolean isContain(Collection aTerms)

4.2.3 Part 3: Validate text classifier

1) class Classfier → This program is a simple classifier use the generated association rules

- Method:
    - public Classfier()
    - static HashMap readCategory(GdbmFile categorydb)
    - public static void main(String[] args)

5. Experimental Result

5.1 Experiment Data

Data source of our experiments is news documents from "The Herald" from July 1, 2002 to August 31, 2002. The total number of documents is 7837, each document with a pre-assigned category label and total number of term occurrences is 3950848. Stop words are used as filter and then words are stemmed by Porter stemmer. There are 1741183 words removed from the data set. After preprocessing, there are 76682 unique terms stored in the database. The corpus is split into two parts, 5000 document training set and 2837 document testing set respectively. We also picked different number of documents from the training data set.

Our experiment is executed on Locutus server of Dalhousie University Computer Science Department. The server is a Sun Enterprise 4500 with eight 400Mhz Sparc processors and three gigabytes of RAM.

5.2 Experimental Results and Analysis

Table 6.1 shows the result of generated rules with different training time based on three support thresholds 70%, 80%, 90%.

Data Set Size	*1000 documents*		*2000 documents*		*3000 documents*		*5000 documents*	
Support (%)	**# of Generated Rules**	**Training Time (sec)**	**# of Generated Rules**	**Training Time (sec)**	**# of Generated Rules**	**Training Time (sec)**	**# of Generated Rules**	**Training Time (sec)**
70	8	1.93	31	199	40	276	36	460
80	5	1.86	30	184	32	249	29	422
90	5	1.7	22	123	27	251	24	268

Table 5.1 Generated Rules and Training Time

From Table 5.1, firstly, we can see that the number of generated rules varies with the size of training set. From 1000 to 3000 documents, the number of rules increases correspondingly, and for 5000 documents, it decreases comparing with the 3000 documents.

Secondly, when the support thresholds are going up (70%, 80%, and 90%), the number of rules and training time are reduced. It is because that the higher support threshold prunes more items in the procedure of association rule generating so processing time and generated rules are reduced as expected. Figure 5.1 shows the chart of dataset vs. training time compared by two support values, whose data is derived from the above table, the training time with higher support value 90% (bottom line) has a better performance, and we notice that when the size of data set is greater than 3000, the difference of training time is remarkably increased. The trend was clearly showed in Figure 5.2

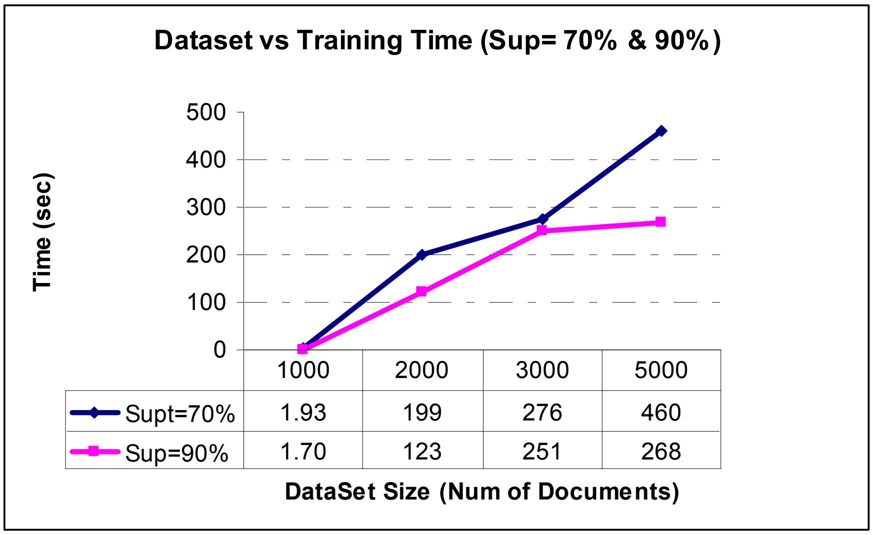

Figure 5.1 Dataset vs. Training Time (Sup=70% and 90%)

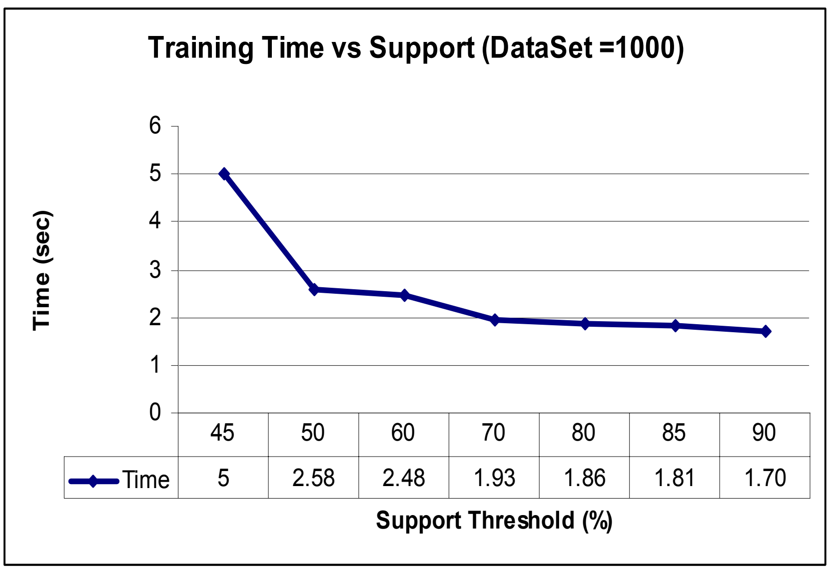

Figure 5.2 Training Time vs. Support (Data Set size = 1000)

The validation of the association rule classifier is implemented by our simplified algorithm, and table 5.2 shows the experiment result and table 5.3 shows the sample association rules composing the classifier. Two testing data sets of 100 and 1000 are applied on two training data sets with three support thresholds: 70%, 80%, and 90%. The validation is measured by accuracy which is the percentage of correctly classified data on whole test data.

First, we see that there is no significant difference of accuracy between the two test data sets. They almost achieved all the same result for different testing. Second, the accuracy is almost constant around 50% for 2000 training set, whereas the accuracy has a significant increase when the support threshold is set up to 90% for the 5000 data set. The difference shows that increasing the number of files in training set and the support threshold can increase the accuracy of classification.

Number of documents in Training Set	Number of documents in Testing Set	Support	70%	80%	90%
2000	100	Accuracy	49%	48%	50%
	1000	Accuracy	49%	50%	50%
5000	100	Accuracy	53%	52%	**79%**
	1000	Accuracy	52%	50%	**80%**

Table 5.2: Accuracy of classifier for different support and data set

people ^ halifax ^ editor ^ dear → Letter
press ^ look ^ include ^ fashion → LivingFashion
People ^ new ^ disease ^ medical → LivingHealth

Table 5.3 Examples of association rules composing the classifier.

Our experimental results are not as good as the results shown in paper [3]. The results are not contributed by the association rule generating algorithm, but by our simple implementation of model validation. This may also be caused by our data set. The document distribution for each category may not be uniform. Some categories have small number of documents, for example, category “funny” has only one document, and this document has high possibility to generate a noisy rule, thus further affect the accuracy of our classifier.

Because of limited time, we implemented a simple classifier, which classify documents by counting the rules that satisfy and return the category with maximum count. Our implementation ignores the confidence, which is an important measure that can be used to generate rules, prune rules and apply rules to build text classifier. The number of rules generated by our system is relatively small since we prune the rules and only keep the last none empty k frequent sets as the final rules, which we think is the most representable feature of a certain category. Since each document in our data set is pre-labeled with one category, we only implement single-class categorization. A document can be assigned to multi-classes, if more that one category exceeds a threshold for the count of rules that this documents satisfied. So by introducing the “count threshold”, our algorithm can also be used for multi-class categorization after slightly change the method of classifier.

6 Conclusion and future work

In this project, we employed association rule in the text categorization. Our study provides evidence that association rule can be used in automatic text categorization efficiently and effectively. One major advantage of the association rule based classifier is that it doesn't assume that terms are independent and its training is relatively fast. Furthermore, the rules is human understandable and easy to be maintained or pruning by human being. Since time is limited, we simplified the pruning and classification method, and the result is comparable to the methods mentioned in the [3] [4].

In the future, we would add following features to the project:

- Feature selection: Adding the weight of each term in the documents and pruning the terms with lower weight. The feature selection will reduce the number of terms as well as reduce the noisy of the terms.
- Classification: Improve our simple classifier by using algorithms based on confidence.

7. User Instruction

7.1 User Instruction Overview

This system can be used to classify text documents. Our system can be applied in many fields, such as web site categorization, email filtering, digital library etc.

Java 1.3.1 and GNU dbm (GDBM) need to be installed in the computer to run our project programs. GDBM is a set of database routines that use extensible hashing. It works similar to the standard UNIX dbm routines and is idea for storing and retrieving small dataset. JDK 1.3.1 can download from www.sun.com. GDBM can be downloaded from GNU website http://www.gnu.org/software/gdbm/gdbm.html. Both of them are free.

Our programs run well on the Locutus server. We can use javac *.java to compile the programs.

There are three steps shown as following:

7.2 Part 1: User instruction for data preprocessing

Run
java Preprocessing

This program reads the source document, generate three data files, “Vector”, “Category” and “Term”, these files will be used in the training phase.

7.3 Part 2: User instruction for generating classifier by ARC-BC

Run
java GenRules

Configure
There are to configure files.
1. configure
1) min_support: Set the mininum support.
Examples: min_support=90, min_support=80
2) min_Category_#files: The minum number of files in a category. If the number of files in a category is lower than iMinFilesInCategor, system will not generate rules for it.
Examples: min_Category_#files=5, min_Category_#files=7

2. dataconfigure
1)category_file: the path and filename of category file, which is in Gdbm file format, created in data preprocessing phase.
Examples: category_file=./data/category1k
category_file=./data/category2k

2)document_file: the path and filename of doucment file, which is in

Gdbm file format, created in data preprocessing phase.
Example: document_file=./data/documents

3)first_item_candidates_file: the path and filename of the first item candidates file, which is in Gdbm file format, created in data preprocessing phase.
Examples: first_item_candidates_file=./data/candidates1k,
first_item_candidates_file=./data/candidates2k

4)rules_file: the path and filename of the file in Gdbm file format, to save the rules that will be generated.
Exapmle: rules_file=./data/rules

7.4 Part 3: User instruction for validating text classifier

Run
java Classfier

Input
Rule set generated from part 2

Output
accuracy & error

References

[1] K.Aas and A.Eikvil, Text Categorization: A survey. *Technical report, Norwegian Computing Center*, June, 1999.

[2] Y. Yang and X. Liu. A re-examination of text categorization methods. *In International ACM-SIGIR Conference on Research and Development in Information Retrieval*, 1999.

[3] Maria-Luiza Antonie, Osmar R. Zaïane. Text Document Categorization by Term Association. *IEEE International Conference on Data Mining (ICDM'2002), pp 19-26,* Maebashi City, Japan, December 9 - 12, 2002.

[4] Osmar R. Zaïane, Maria-Luiza Antonie, Classifying text documents by associating terms with text categories, *in Proc. of the Thirteenth Australasian Database Conference (ADC'02)*, Melbourne, Australia, January 28-February 1, 2002.

[5] Han, J., Kamber, M. (2001). Data Mining: Concepts and Techniques, *Morgan Kaufmann Publishers*, ISBN 1-55860-489-8, 2002

[6] Porter, M. F. An Algorithm for Suffix Stripping. *Program 14*, pages 130–137, 1980.

[7] C.J. Van Rijsbergen. Information Retrieval. *Butterworths*, London, 1979.

[8] Zipf, G., Human Behavior and the Principle of Least Effort, *Reading, MA: Addison-Wesley*, 1949.

[9] D. Lewis. Naïve (bayes) at forty: The independence assumption in information retrieval. In *10th European Conference on Machine Learning* (ECML-98), pages 4-15, 1998.

[10] Y.Yang. An evaluation of statistical approaches to text categorization. *Technical Report CMU-CS-97-127*, Carnegie mellon University, April 1997.

[11] T.Joachims. Text categorization with support vector machines: learning with many relevant features. In *10th European Conference on Machine Learning* (ECML-98), pages 137-142, 1998

[12] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *In Proc.1993 ACM-SIGMOD Int. Conf. Management of Data*, pages 207–216, Washington, D.C., May 1993.

Appendix1: Sample of news document in experimental data set

<ARTICLE>
<HEADER>
<PUBDATE>2002/07/01</PUBDATE>
<CATEGORY>Metro</CATEGORY>
<SUBCAT>None</SUBCAT>
<RATING>3</RATING>
<KEYWORDS>CRI</KEYWORDS>
<COPYRIGHT>Unknown</COPYRIGHT>
</HEADER>
<STORY>
<VERSION>
<PUBINFO>
<COPYRIGHT>Unknown</COPYRIGHT>
<INTERNET>yes</INTERNET>
<PUBMETRO>A3</PUBMETRO>
<INTONLY></INTONLY>
</PUBINFO>
<HEADLINE>Police seek cab robber </HEADLINE>
<SUBHEAD></SUBHEAD>
<BYLINE> </BYLINE>
<CONTENT>Nova Scotia Crime Stoppers is asking for the public's help in finding the man responsible for a Halifax robbery.

On June 17 shortly after 10 a.m., a man asked a cab driver in front of the Casino Nova Scotia Hotel to take him to Dutch Village and Bayers roads.

Near Almon Street and Connaught Avenue, the suspect indicated he had a gun but no money.

The cabbie turned right onto Connaught Avenue and pulled into an Irving station hoping to find help. As he tried to get out, the suspect held him by his coat, pulled out a carpet knife with a hooked blade and jabbed it at the driver's ribs. The cabbie managed to jump out of his car and call police from the gas station.

The suspect was seen running east on Bayers Road and then through the back of a funeral home parking lot.

The robber was biracial, 23 or 24 years old and wore a beige baseball cap and a grey and beige three-quarter-length jacket. Anyone with information on this or any other serious crime in Nova Scotia is asked to call Crime Stoppers anytime at 1-800-222-8477.

If your tip leads to an arrest, you will qualify for a cash award from $50 to $2,000. Your call is anonymous and you will not have to testify in court.</CONTENT>
</VERSION>
</STORY>
<PHOTOS>
</PHOTOS>
<GRAPHICS>
</GRAPHICS>
<INFOBOXES>
</INFOBOXES>
</ARTICLE>

Appendix 2: Stop words list used in the preprocessing.

a
about
above
across
after
afterwa
rds
again
against
all
almost
alone
along
already
also
althoug
h
always
am
among
amongst
amoungs
t
amount
an
and
another
any
anyhow
anyone
anythin
g
anyway
anywher
e
are
around
as
at
back
be
became
because
become
becomes
becomin
g
been
before
beforeh
and
behind
being
below
beside
besides
between
beyond
bill
both

bottom
but
by
call
can
cannot
cant
co
compute
r
con
could
couldnt
cry
de
describ
e
detail
do
done
down
due
during
each
eg
eight
either
eleven
else
elsewhe
re
empty
enough
etc
even
ever
every
everyon
e
everyth
ing
everywh
ere
except
few
fifteen
fify
fill
find
fire
first
five
for
former
formerl
y
forty
found
four
from

front
full
further
get
give
go
had
has
hasnt
have
he
hence
her
here
hereaft
er
hereby
herein
hereupo
n
hers
herself
him
himself
his
how
however
hundred
i
ie
if
in
inc
indeed
interes
t
into
is
it
its
itself
keep
last
latter
latterl
y
least
less
ltd
made
many
may
me
meanwhi
le
might
mill
mine
more

moreove
r
most
mostly
move
much
must
my
myself
name
namely
neither
never
neverth
eless
next
nine
no
nobody
none
noone
nor
not
nothing
now
nowhere
of
off
often
on
once
one
only
onto
or
other
others
otherwi
se
our
ours
ourselv
es
out
over
own
part
per
perhaps
please
put
rather
re
same
see
seem
seemed
seeming
seems
serious

several
she
should
show
side
since
sincere
six
sixty
so
some
somehow
someone
somethi
ng
sometim
e
sometim
es
somewhe
re
still
such
system
take
ten
than
that
the
their
them
themsel
ves
then
thence
there
thereaf
ter
thereby
therefo
re
therein
thereup
on
these
they
thick
thin
third
this
those
though
three
through
through
out
thru
thus
to

togethe
r
too
top
toward
towards
twelve
twenty
two
un
under
until
up
upon
us
very
via
was
we
well
were
what
whateve
r
when
whence
wheneve
r
where
whereaf
ter
whereas
whereby
wherein
whereup
on
whereve
r
whether
which
while
whither
who
whoever
whole
whom
whose
why
will
with
within
without
would
yet
you
your
yours
yoursel
f
yourse