



Qatar Content Classification

Classifying Arabic texts using machine learning

Abstract

This is a course project for the CS6604 – Digital libraries course (Spring 2014). The project has been conducted under the supervision of Prof. Ed Fox and Mr. Tarek Kanan. The goal is to develop an Arabic newspapers classifier. We have built a collection of 700 Arabic newspaper articles and 1700 Arabic full-newspaper PDF files. A stemmer, named “P-Stemmer”, is proposed. Evaluation have shown that P-Stemmer outperform the widely used Larkey’s light stemmer. Several classification techniques were tested on Arabic data including SVM, Naïve Bayes and Random Forest. We built and tested 21 multiclass classifiers, 15 binary Classifiers, and 5 compound classifiers using the voting technique. Finally, we uploaded the classified instances to Apache Solr for searching and indexing purposes.

Mohamed Handosa

Handosa@vt.edu

Table of Contents

Figures	2
Tables.....	3
1 Introduction.....	4
2 Arabic Newspaper Taxonomy.....	4
3 Collecting the Training Set.....	5
3.1 The Al-Raya Crawler	6
3.2 The QNA HTML2TXT Tool	6
4 Collecting the Testing Set	6
4.1 The PDF2TXT Tool.....	7
5 Preprocessing the Dataset Instances	8
5.1 Extraction of Arabic Words.....	8
5.2 Normalization of Arabic Words	9
5.3 Stemming Arabic Words.....	9
5.4 P-Stemmer – A Proposed Stemmer.....	10
6 Text classification	11
6.1 Creating Feature Vectors.....	11
6.2 Multiclass Classifiers.....	15
6.3 Binary Classifiers.....	19
6.4 Compound Binary Classifiers	21
7 Classifying the Testing Set	24
8 Indexing and Searching Collections using Apache Solr	28
8.1 Installation Solr with Tomcat as a Web Container	28
8.1.1 Installing Apache Tomcat	28
8.1.2 Installing Apache Solr	28
8.2 Starting/Stopping the Apache Tomcat Service.....	28
8.3 Creating a Solr Core.....	29
8.4 Editing the Schema file	31
8.5 Working with Documents through the Dashboard	31
8.5.1 Adding Documents	32
8.6 Indexing Classified Collections.....	33
9 Conclusion and Future Work.....	34
10 References	35

Figures

Figure 2.1: Arabic newspaper taxonomy.....	5
Figure 3.1: A screenshot of the Raya Crawler tool.....	6
Figure 3.2: A screenshot of the QNA HTML2TXT tool.....	6
Figure 4.1: A screenshot of the PDF2TXT tool.....	8
Figure 5.1: Dataset preprocessing steps.....	8
Figure 5.2: A screenshot of the “Arabic Words Extractor” tool.....	8
Figure 5.3: A screenshot of the “Arabic Light Stemmer” tool.....	10
Figure 6.1: Creating ARFF files for the seven versions of the training set.....	12
Figure 6.2: Opening an ARFF file.....	13
Figure 6.3: Weka's filters.....	13
Figure 6.4: Opening the parameters of the selected filter.....	14
Figure 6.5: The parameters of the “StringToWordVector” filter.....	14
Figure 6.6: The parameters dialog of the “AttributeSelection” filter.....	15
Figure 6.7: Weka's classifiers tree.....	16
Figure 6.8: Opening the parameters of the selected classifier.....	16
Figure 6.9: The parameters of the “MultiClassClassifier” classifier.....	16
Figure 6.10: F-measure values for the three classification techniques.....	19
Figure 6.11: Creating ARFF files for the five training sets.....	19
Figure 6.12: The F-measure values for the three classification techniques.....	21
Figure 6.13: Weka's classifiers tree.....	22
Figure 6.14: Opening the parameters of Weka's voting compound classifier.....	22
Figure 6.15: Parameters dialog of the “voting” compound classifier.....	22
Figure 6.16: Voting members.....	23
Figure 6.17: The F-measure values for the three classifiers compared to the compound classifier.....	24
Figure 7.1: The “TXT2CSV” tool.....	24
Figure 7.2: Classification's test options.....	25
Figure 7.3: Saving a model.....	25
Figure 7.4: Specifying a testing set CSV file.....	26
Figure 7.5: Loading a saved model in Weka.....	26
Figure 7.6: Re-evaluating a model based on current test set.....	27
Figure 7.7: Results of classifying the testing set instances using the Art compound classifier.....	27
Figure 8.1: Starting the Apache Tomcat service.....	29
Figure 8.2: Stopping the Apache Tomcat service.....	29
Figure 8.3: Navigating to “~/Desktop/Solr”.....	29
Figure 8.4: The “conf” folder copied from “~/Desktop/Solr/collection1/”.....	29
Figure 8.5: Apache Solr Dashboard.....	30
Figure 8.6: Apache Solr, Core Admin screen.....	30
Figure 8.7: The location of the schema file for the “ar-collection” core.....	31
Figure 8.8: The “ar-collection” Documents screen.....	32
Figure 8.9: Document Builder screen.....	32
Figure 8.10: The “exampledocs” directory.....	33
Figure 8.11: Statistics of the ‘test’ core.....	33

Tables

Table 2.1: The first and second levels of the IPTC media topic taxonomy.....	4
Table 3.1: Articles collected from newspapers' categories corresponding to taxonomy subclasses.	5
Table 5.1: Normalization rules for Arabic words.....	9
Table 5.2: The versions of the Arabic light stemmer [7].	10
Table 6.1: The Seven versions of the training set.....	11
Table 6.2: Number of features for each training set version.	15
Table 6.3: The results of a 10-fold cross-validation using the <i>Words</i> version of the training set.	17
Table 6.4: The results of a 10-fold cross-validation using the <i>Stems1</i> version of the training set.	17
Table 6.5: The results of a 10-fold cross-validation using the <i>Stems2</i> version of the training set.	17
Table 6.6: The results of a 10-fold cross-validation using the <i>Stems3</i> version of the training set.	17
Table 6.7: The results of a 10-fold cross-validation using the <i>Stems8</i> version of the training set.	18
Table 6.8: The results of a 10-fold cross-validation using the <i>Stems10</i> version of the training set.	18
Table 6.9: The results of a 10-fold cross-validation using the <i>StemsP</i> version of the training set.....	18
Table 6.10: The F-measure values for the three classification techniques.	18
Table 6.11: Number of features for each training set version.	20
Table 6.12: The results of a 10-fold cross-validation of three the "Art" classifiers.....	20
Table 6.13: The results of a 10-fold cross-validation of the three "Economy" classifiers.....	20
Table 6.14: The results of a 10-fold cross-validation of the three "Politics" classifiers.	20
Table 6.15: The results of a 10-fold cross-validation of the three "Society" classifiers.	20
Table 6.16: The results of a 10-fold cross-validation of the three "Sport" classifiers.....	21
Table 6.17: The F-measure values for the three classification techniques.	21
Table 6.18: The results of a 10-fold cross-validation of the five compound classifiers.	23
Table 6.19: The F-measure values for the three classifiers compared to the compound classifier.....	23
Table 8.1: Solr command formats	31

1 Introduction

The goal of this project is to develop an Arabic newspapers classifier. We have built a collection of 700 Arabic newspaper articles and 1700 Arabic full-newspaper PDF files. A stemmer, named “P-Stemmer”, is proposed. Evaluation have shown that P-Stemmer outperform the widely used Larkey’s light stemmer. Several classification techniques were tested on Arabic data including SVM, Naïve Bayes and Random Forest. We built and tested 21 multiclass classifiers, 15 binary Classifiers, and 5 compound classifiers using the voting technique. Finally, we uploaded the classified instances to Apache Solr for searching and indexing purposes.

This report is organized in 8 main sections. Section 2 presents the taxonomy used in classification. Section 3 and section 4 illustrates the data sources and developed tools used to collect for the training set and the testing set, respectively. Section 5 provides a detained description of the preprocessing steps applied to the datasets. In section 6, we provide the results of applying different classifiers using different machine learning techniques. Section 7, shows how to classify the testing set using the trained classifiers. Finally, section 8 shows how to install and use Apache Solr for indexing and searching Arabic collections.

2 Arabic Newspaper Taxonomy

The IPTC Media Topic taxonomy [1], developed by the International Press Telecommunication Council (IPTC), is a five levels taxonomy of 1100 terms. Table 2.1 shows the first two levels of IPTC’s taxonomy.

Table 2.1: The first and second levels of the IPTC media topic taxonomy.

1 st level	2 nd level
Art, culture and entertainment	Arts and entertainment – Culture – Mass media
Crime, law and justice	Crime – Judiciary – Justice and rights – Law – Law enforcement
Disaster and accident	Accident – Disaster – Emergency incident – Emergency planning – Emergency response
Economy, business and finance	Business information – Economic sector – Economy – Market and exchange
Education	Parent organization – Religious education – School – Social learning – Teaching and learning
Environment	Climate change – Conservation – Environmental politics – Environmental pollution Natural resource – Nature
Health	Diseases and conditions – Health facility – Health organizations – Health treatment – Healthcare policy – Medical profession – Non-human diseases
Human interest	Accomplishment – Animal – Ceremony – People – Plant
Labor	Employment – Employment legislation – Labor market – Labor relations – Retirement – Unemployment – Unions
Lifestyle and leisure	Leisure – Lifestyle
Politics	Election – Fundamental rights – Government – Government policy – International relations – Non-governmental – Political crisis – Political dissent – Political process
Religion and belief	Belief – Interreligious dialog – Religious conflict – Religious event – Religious facilities – Religious institutions and state relations – Religious leader – Religious text
Science and technology	Biomedical science – Mathematics – Mechanical engineering – Natural science – Research – Scientific institutions – Social sciences – Standards – Technology and engineering

1 st level	2 nd level
Society	Communities – Demographics – Discrimination – Family – Mankind – Social condition – Social problem – Values – Welfare
Sport	Competition discipline – Disciplinary action in sport – Drug use in sport – Sport event – Sport industry – Sport organization – Sport venue – Transfer
Conflicts, war and peace	Act of terror – Armed conflict – Civil unrest – Coup – Massacre – Peace process – Post-war reconstruction – Prisoners and detainees
Weather	Weather forecast – Weather phenomena – Weather statistics – Weather warning

Inspired by the IPTC's taxonomy, the developed Arabic newspaper taxonomy reflects the categorization hierarchy, which is common in Arabic newspapers as shown in Figure 2.1.

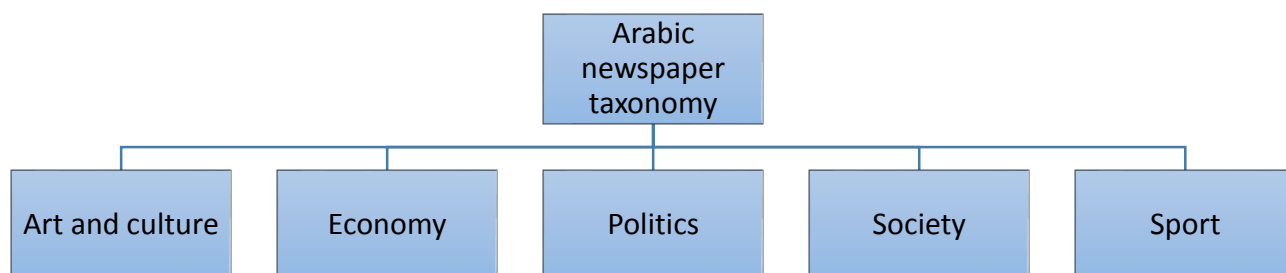


Figure 2.1: Arabic newspaper taxonomy.

3 Collecting the Training Set

The goal of this project is to build a text classifier that can classify Arabic articles of Qatar newspapers according to the proposed Arabic taxonomy shown in Figure 2.1. Training the classifier using supervised machine learning requires a training set of pre-classified articles. We have collected a set of articles from two Qatar newspapers, Al-Raya¹ and Qatar News Agency (QNA)², as shown in Table 2.1.

Table 3.1: Articles collected from newspapers' categories corresponding to taxonomy subclasses.

Class	Subclass	Newspaper	Newspaper's category	Retrieved articles
Art and culture	Art	Al-Raya	منوعات – أفاق وفنون	44
	Culture	Al-Raya	منوعات – ثقافة وأدب	52
	Mass media	Al-Raya	منوعات – إذاعة وتلفزيون	54
Economy	Local economy	QNA	أخبار محلية – اقتصاد	75
	International economy	QNA	أخبار دولية – اقتصاد	75
Politics	Local politics	QNA	أخبار محلية – سياسة	75
	International politics	QNA	أخبار دولية – سياسة	75
Society	Death notices	Al-Raya	مجتمع – وفيات	89
	Wedding announcements	Al-Raya	مجتمع – أفراس	61
Sport	Local sports	QNA	أخبار محلية – رياضة	75
	International sports	QNA	أخبار دولية – رياضة	75

¹ The website of Al-Raya newspaper is available at <http://www.raya.com/>

² The website of Qatar News Agency newspaper is available at <http://www.qna.org.qa/>

3.1 The Al-Raya Crawler

The website for the Al-Raya’s newspaper provides a categorized set of articles. In order to retrieve the articles from Al-Raya’s website, we developed a Java tool named “Al-Raya Crawler”. The tool takes the URL of an Al-Raya’s category page and a path to a destination directory on local machine as shown in Figure 2.1.



Figure 3.1: A screenshot of the Raya Crawler tool.

The tool retrieves the HTML page at the specified URL, extracts the links to all articles listed in that page, and filters out any links to non-article content (e.g. ads). Afterwards, for each extracted link, the tool retrieves the corresponding article’s HTML page and extract both the article’s header and the article’s text content. Finally, for each article, the tool saves the extracted content to a text file at the specified destination directory and uses the article header as a file name. We used the tool to retrieve instances from five categories of Al-Raya’s articles corresponding to five subclasses in the taxonomy as shown in Table 2.1.

3.2 The QNA HTML2TXT Tool

To obtain instances for the remaining six subclasses, we have retrieved 75 article in HTML format for each remaining subclass from the QNA website. In order to extract the header and the content of each article from the HTML files, we developed a Java tool named “QNA HTML2TXT”. The tool takes the path of the directory containing the HTML files and a destination directory as input as shown in Figure 3.2Figure 2.1. For each HTML file, the tool extracts the header and the content of the article and saves the extracted content to a text file at the specified destination directory using the article header as a file name.

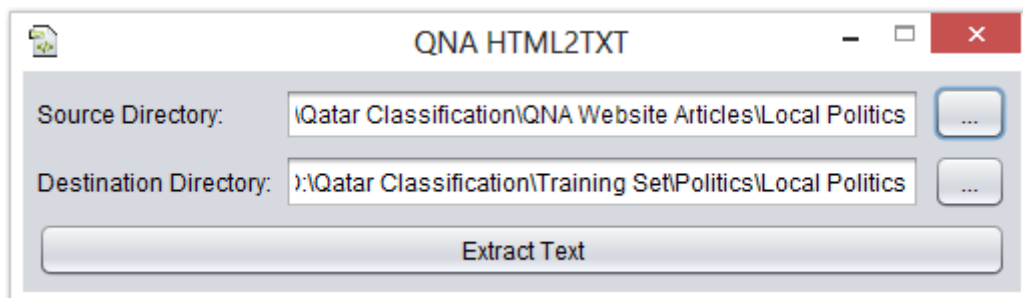


Figure 3.2: A screenshot of the QNA HTML2TXT tool.

4 Collecting the Testing Set

The Al-Raya’s website has an archive of published newspapers in PDF format. Using the Heritrix crawler¹, we have collected 1700 PDF files. Each PDF file has a set of pages and each page contains one article or more. Some pages may contain no articles (e.g. a full-page image).

¹ Heritrix is a free license web crawling software written in Java by the Internet Archive.

4.1 The PDF2TXT Tool

Since the preparation of a testing set by manually extracting text articles from 1700 PDF files is not feasible, we have developed a Java tool named PDF2TXT to extract the text from PDF files automatically. A PDF document stores its content as a set of objects (root object, page objects, font objects, etc.) Each object stores a specific type of information. Text is stored in chunks of one or more characters and each chunk is located at a given X, Y coordinate. The text chunks can be stored in any order in the file, which makes the processing of text in PDF documents more challenging. Fortunately, there are several of-the-shelf tools and libraries for handling PDF documents including PDFTextStream¹, iTextSharp² and Apache PDFBox³. These tools provides many capabilities including text extraction.

Although there are several libraries that support text extraction from PDF documents, there is little support for extracting text written in right-to-left languages like Arabic. The problem with extracting Arabic text is due to the difference between logical and presentation order [2]. Logical order refers the order in which text characters are stored (i.e. first character stored is first character read or written) while presentation order is based on screen layout. Logical and presentation order are the same for left-to-right languages, but opposite for right-to-left languages.

Recalling that PDF documents stores text as chunks, each with x and y coordinates. The text extraction process tries to reconstruct the text by concatenating these chunks given there coordinates. That is, a text extraction algorithm uses presentation order to extract text. This works fine for left-to-right languages since presentation and logical order are the same. However, for right-to-left languages, where the logical and presentation order are opposite, the algorithm will concatenate the chunks in presentation order (i.e. left-to-right) which is the opposite of logical order (i.e. right-to-left). Consequently, the characters of the extracted text are in reverse order.

Generally, a multilingual text can have both left-to-right and right-to-left characters and each should go in the correct direction. The Unicode Bi-directional Text (BiDi) algorithm defines how to order characters in a paragraph (i.e. converts from logical to presentation order). Since the available text extraction tools use presentation order to extract text, then a reasonable solution is to apply a reverse BiDi algorithm to convert the extracted text from presentation order to logical order.

The first version of the PDF2TXT tool, developed in C#, used the PDFTextStream library to extract text from Arabic PDF documents. Since the PDFTextStream library does not support Arabic, it extracts the text in presentation order. Thus, the tool processes the extracted text line by line and reverses the order of characters to obtain the logical order. There are two problems with the first version of the PDF2TXT tool. First, the PDFTextStream library seems to have an encoding problem with processing Arabic text that is for a considerable subset of the PDF files the extracted text was miscoded and useless. Second, even for Arabic text extracted with a correct encoding the PDFTextStream library fails sometimes to extract the Arabic characters in the same order as their order of presentation. Therefore, when the tool reverses the extracted text is to obtain the logical orders, the letters of some Arabic words are disordered.

The second version of the PDF2TXT tool, developed in Java, uses the PDFBox library to extract text from multilingual PDF documents. PDFBox uses the ICU4J library from the International Components for Unicode (ICU) project to support bidirectional text. Since PDFBox provides full support for right-to-left languages like Arabic, the second PDF2TXT version avoids the problems of the first version. The PDF2TXT tool takes the path of the directory containing the PDF files and a destination path as shown in Figure 4.1. For each PDF file, the

¹ The PDFTextStream library is available at <http://www.snowtide.com/>

² The iTextSharp library is available at <http://sourceforge.net/projects/itextsharp/>

³ The PDFBox library is available at <http://pdfbox.apache.org/>

tool extracts the text as saves it to a text file at the destination directory. The tool allows a user to optionally split the pages of each PDF file and extract the text from each PDF page to a distinct text file.

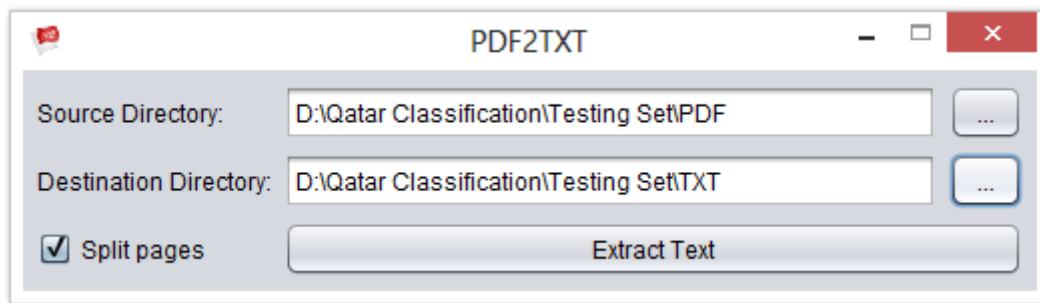


Figure 4.1: A screenshot of the PDF2TXT tool.

Given the 1700 newspaper PDF files, we used the PDF2TXT tool to extract the text content from each PDF page to a distinct text file. From the 1700 PDF files, we obtained 58,938 text files. Each text file represents an instance of the testing set. Ideally, the testing set instances should be articles rather than pages. However, we argue that a newspaper page usually contains articles belonging to the same category and hence we assumed that the text content of a PDF page is typically a concatenation of a set of articles that belongs to the same class.

5 Preprocessing the Dataset Instances

Most text classifiers use the bag-of-words model to represent documents. The bag-of-words model is a straightforward representation approach and the produced representation is essentially independent of the sequence of words in the document [3]. The goal of the preprocessing phase is to extract a set of Arabic words from each instance in the collected dataset (i.e. training and testing sets) in order to represent that instance using the bag-of-words model. The preprocessing phase has three steps as shown in Figure 5.1.

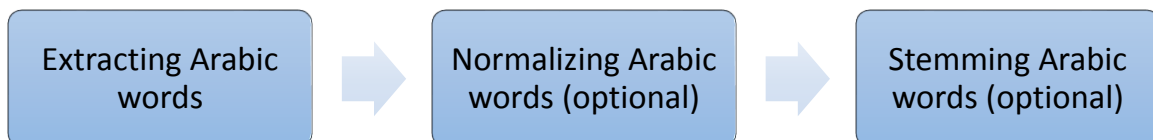


Figure 5.1: Dataset preprocessing steps.

5.1 Extraction of Arabic Words

The raw text files of the collected dataset might contain non-Arabic words and punctuation marks. In order to perform text cleaning, we have developed a Java tool named “Arabic Words Extractor”. As shown in Figure 5.2, the tool takes the path to the directory containing the raw text files and a destination directory as input.

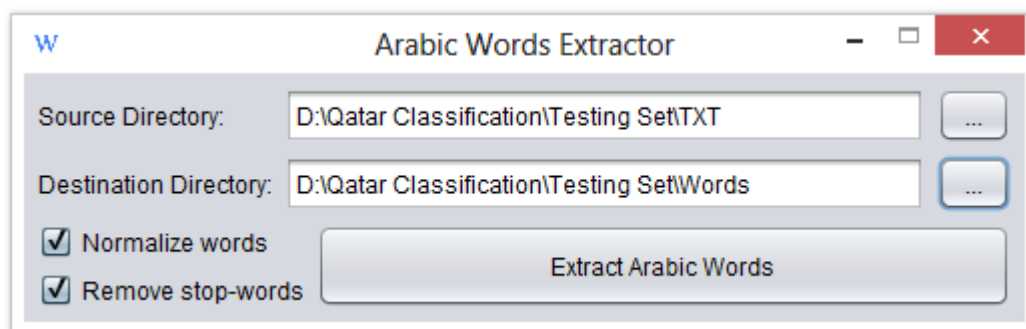


Figure 5.2: A screenshot of the “Arabic Words Extractor” tool.

For each raw text file, the tool extracts the Arabic words from the file content, filters out any non-Arabic letters, and saves the extracted Arabic words as space-delimited list to a text file at the destination directory.

Arabic is a very rich and complex language that has 28 characters and written from right to left. An Arabic word may contain primarily weak vowels, called “diacritics”, that determines the pronunciation of words. For example, the word “مدرسة” with the diacritic “ُ” on the letter “م” becomes “مُدرسة” (i.e. “a female teacher”). If the same word “مدرسة” has the diacritic “َ” on the letter “م”, it becomes “مدرسة” (i.e. “a school”). Arabic has eight different diacritics, which are َ, ُ, ِ, َ, ِ, ِ, ِ, and ِ. Although the diacritics can significantly change the meaning of an Arabic word, they are rarely used and the pronunciation of the word, which determines its meaning, is deduced from the word context. Moreover, for text formatting purposes, it is common to stretch an Arabic word using the “-” character, which does not change the meaning of the word. For example, it is possible to write the word “مدرسة” as “مدرسة”, “مدرسة” or “مدرسة”.

The developed “Arabic Words Extractor” tool removes the “-” character as well as the diacritics from extracted Arabic words. The tool can remove Arabic stop-words as well by comparing the extracted words against a list of 1,630 Arabic stop-words developed by Abu El-Khair [4].

5.2 Normalization of Arabic Words

Usually, Arabic information retrieval systems normalize Arabic words to increase retrieval effectiveness. Normalization of an Arabic word means replacing specific letters within the word with other letters according to a predefined set of rules as shown in Table 5.1.

Table 5.1: Normalization rules for Arabic words.

Rule		Example	
Letter	Replacement	Word	Normalized word
أ	ا	أحمد	احمد
إ	ا	إنشاء	انشاء
آ	ا	آلات	الات
ة	ه	مدرسة	مدرسه
ى	ي	على	علي

Although these replacements can result in misspelled words, these misspellings are common in Arabic text and the normalization of Arabic words helps avoiding the side effects of such misspellings on the performance of information retrieval. For example, the normalization of the word “مدرسة” and its misspelled version “مدرسه” results in the same normalized word “مدرسه”. Hence, the system recognizes the two words as being the same. It is worth mentioning that such misspellings occur rarely in official documents and newspapers. Consequently, the normalization of Arabic words might be unnecessary when working with newspapers or official documents (e.g. thesis and dissertations). The “Arabic Words Extraction” tool allows optional word normalization. Although we are working with newspapers, we chose to normalize words to capture even rare misspellings.

5.3 Stemming Arabic Words

The main goal of a stemmer is to map different forms of the same word to a common representation called “stem”. Stemming can significantly improve the performance of text classification systems by reducing the dimensionality of word vectors. Generally, there are two main categories of Arabic stemmers, root extraction stemmers and light stemmers [5]. The two most widely used stemmers are the root extraction stemmer developed by Khoja et al [6] and the light stemmer developed by Larkey et al [7].

In Arabic, each Arabic word has a root, which is its basic form. We can obtain several words including nouns, verbs and adjectives by adding certain letters at the beginning, end or within the root letters. For example, from the root “قصد”, we can derive the words “يقصد”, “مقاصد”, “اقتصادية”, “الاقتصادي”, etc. The goal of a root-

based stemmer it to extract the basic form for any given word. The problem with extracting the root is that the root is far more abstract than a stem. Different words with completely different meaning can originate from the same root. For example, the words “مقاصد” (i.e. “purposes”) and the word “الاقتصادي” (i.e. “The economic”) both originate from the root “قصد”. Consequently, using root stemmers can result in a very poor classification effectiveness.

The goal of a light stemmer is to find the representative form of an Arabic word by removing prefixes and suffixes. Thus, the meaning of the word remains intact, which results in improving the classification effectiveness. For example, the stem for the words “اقتصادي” (i.e. “economic”) and “والاقتصاد” (i.e. “and the economy”) is “اقتصاد” (i.e. “economy”) rather than the root “قصد” (i.e. “intended”).

We have developed a Java tool, named “Arabic Light Stemmer”, to stem Arabic words. As shown in Figure 5.3, the tool takes the path to the directory containing the raw text files and a destination directory as input.

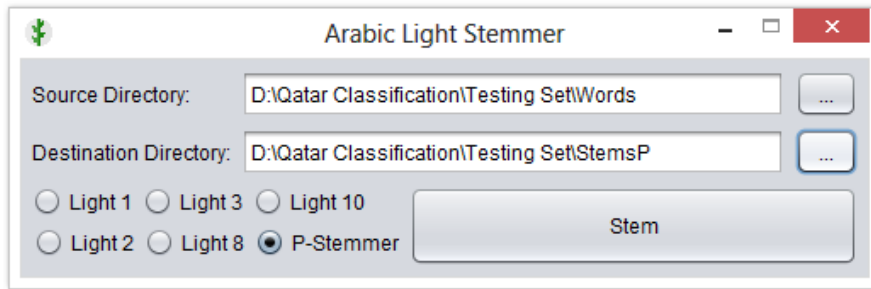


Figure 5.3: A screenshot of the “Arabic Light Stemmer” tool.

The “Arabic Light Stemmer” tool implements the five versions of the light stemming algorithm introduced by Larkey [7]. Each version of the algorithm strips off certain prefixes and suffixes as shown in Table 5.2. Although the Light10 version is the most widely used version of the light stemmer, we have implemented the other versions for evaluation and comparison purposes.

Table 5.2: The versions of the Arabic light stemmer [7].

Version	Prefixes to remove	Suffixes to remove
Light 1	“ال”, “وال”, “بال”, “كال”, “فال”	None
Light 2		
Light 3	“و”, “ال”, “وال”, “بال”, “كال”, “فال”	“ه”, “ة”
Light 8		“ها”, “ان”, “ات”, “ون”, “ين”, “يه”, “ية”, “ه”, “ة”, “ي”
Light 10	“و”, “ال”, “وال”, “بال”, “كال”, “فال”, “لل”	

5.4 P-Stemmer – A Proposed Stemmer

Light stemmers define a set of rules to remove word prefixes and suffixes, while preserving the meaning of the words. For example, the *Light10* stemmer stems the word “المدرسون” (i.e. “the teachers”) to the stem “مدرس” (i.e. “teacher”) by removing the “ال” (i.e. “the”) prefix and the “ون” (which indicates a male plural) suffix. However, we argue that removing word prefixes only can give better results than stemming and hence improves the effectiveness of text classifiers. For example, the *Light10* stemmer stems the word “المباحثات” (i.e. “the talks”) to the stem “مباحث” (i.e. “Investigation”) by removing the “ال” (i.e. “the”) prefix and the “ات” (which indicates a female plural) suffix. It is clear that the two words have completely different meaning and hence we argue that light stemmers that remove word suffixes can still suffers from the same abstraction problem found in root stemmers. To prove our argument we have developed *P-Stemmer*, a customized version of the *Light10* stemmer that removes word prefixes only. The “Arabic Light Stemmer” tool implements the *P-Stemmer* as well as the five versions of Larkey’s light stemmer.

6 Text classification

The problem of text-based classification has been widely studied in the data mining, machine learning, database, and information retrieval communities. The goal of a text classifier is to classify documents into a fixed number of predefined classes. A text classifier can be either a binary classifier or a multiclass classifier. In binary classification, a document can be in exactly one of the two classes. In multiclass classification, a document can be in multiple, exactly one, or no class at all. Using supervised machine learning, classifiers can learn from examples and perform the class assignments automatically. Several text classification algorithms have been proposed. We have chosen to use three of the most widely used text classification approaches, which are Support Vector Machines (SVM), Naïve Bayes, and Random Forest.

Cortes et al [8] have proposed Support Vector Machines (SVM) as a learning method for numerical data. The main principle of SVMs is to determine linear or non-linear separators in the data space, which can best separate the different classes. Joachims [9] has shown that text classification can benefit from SVM by transforming each document, which typically is a string of characters, into a quantitative feature vector, where each distinct word corresponds to a feature whose value is the number of times the word occurs in the document. In order to avoid unnecessary large feature vectors, word stems are used and stop-words are removed. This representation scheme can still lead to very high-dimensional feature spaces. However, one advantage of SVM, which makes it ideal for text classification, is its robustness to high dimensionality.

A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions. It assumes that the value of a particular feature is unrelated to the presence or absence of any other feature, given the class variable. Although this theory violates the fact that features are dependent on each other, its performance is feasible [3]. The naive Bayes classifier models the distribution of the documents in each class using a probabilistic model. It requires only a small set of training data to estimate the model parameters. The Naïve Bayes classifiers can handle text classification as well as other classification problems. In text classification, the model uses the bag-of-words approach to represent a document.

The goal is to develop a text classifier that can classify a given document under one of the five classes at the first level of the Arabic newspapers taxonomy shown in Figure 2.1. In order to train and text different classifiers using different machine learning techniques, we used Weka (Waikato Environment for Knowledge Analysis), which is a popular suite of machine learning software written in Java, developed at the University of Waikato, New Zealand.

6.1 Creating Feature Vectors

As mentioned previously in section 3, we have prepared a training set of 750 instances in the form of text files, 150 per class. We used the “Arabic Words Extractor” tool to clean the text and remove stop-words. Afterwards, we used the “Arabic Light Stemmer tool” to produce six different versions of the text files corresponding to the five versions of the light stemmer and the proposed *P-Stemmer*. Thus, we obtained seven versions of the training set as shown in Table 6.1.

Table 6.1: The Seven versions of the training set.

Training set	Description
Words	Obtained from the raw text files by cleaning text and removing stop-words.
Stems1	Obtained from the <i>Words</i> set by applying version 1 of the light stemmer.
Stems2	Obtained from the <i>Words</i> set by applying version 2 of the light stemmer.
Stems3	Obtained from the <i>Words</i> set by applying version 3 of the light stemmer.
Stems8	Obtained from the <i>Words</i> set by applying version 8 of the light stemmer.
Stems10	Obtained from the <i>Words</i> set by applying version 10 of the light stemmer.
StemsP	Obtained from the <i>Words</i> set by applying the proposed <i>P-Stemmer</i> .

In order to use Weka, the training set must be converted into a single ARFF file. An ARFF¹ (Attribute-Relation File Format) file is a text file that describes a list of instances sharing a set of attributes. Weka provides a Java tool, named TextDirectoryLoader, which can convert a set of text files into an ARFF file. TextDirectoryLoader takes two parameters, a directory and the output file name. It assumes that there are subdirectories within the supplied directory, each corresponding to a given class and contains the text files representing the instances of that class. TextDirectoryLoader produces a single ARFF file that contains all instances with two attributes per instance, text and class. For a given instance, the value of the text attribute is the contents of the text file corresponding to that instance and the value of the class attribute is the name of the subdirectory that contains this instance.

In order to convert the 750 training set instances into an ARFF file, we created five directories corresponding to the five classes (i.e. Art, economy, politics, society, and sport) with 150 text files per directory corresponding to the class instances. To create the file, we used the following command

```
java -cp [weka.jar] weka.core.converters.TextDirectoryLoader -dir [main directory] > output.arff
```

where [main directory] refers to the directory containing the five directories corresponding to the classes and [weka.jar] refers to the bath at which the “weka.jar” file exists. We have created seven ARFF files corresponding to the seven versions of the training set (i.e. *Words*, *Stems1*, *Stems2*, *Stems3*, *Stems8*, and *Stems10*) using the command above as shown in Figure 6.1.



Figure 6.1: Creating ARFF files for the seven versions of the training set.

In order to open an ARFF file in Weka, simply click on the “Open file ...” button and select the ARFF file. Weka will load the file as shown in Figure 6.2. The ARFF file contains 750 instances, 150 per class, and 2 attributes per instance, “text” and “@@class@@”.

¹ A description of the ARFF format available at <http://weka.wikispaces.com/ARFF+%28book+version%29>

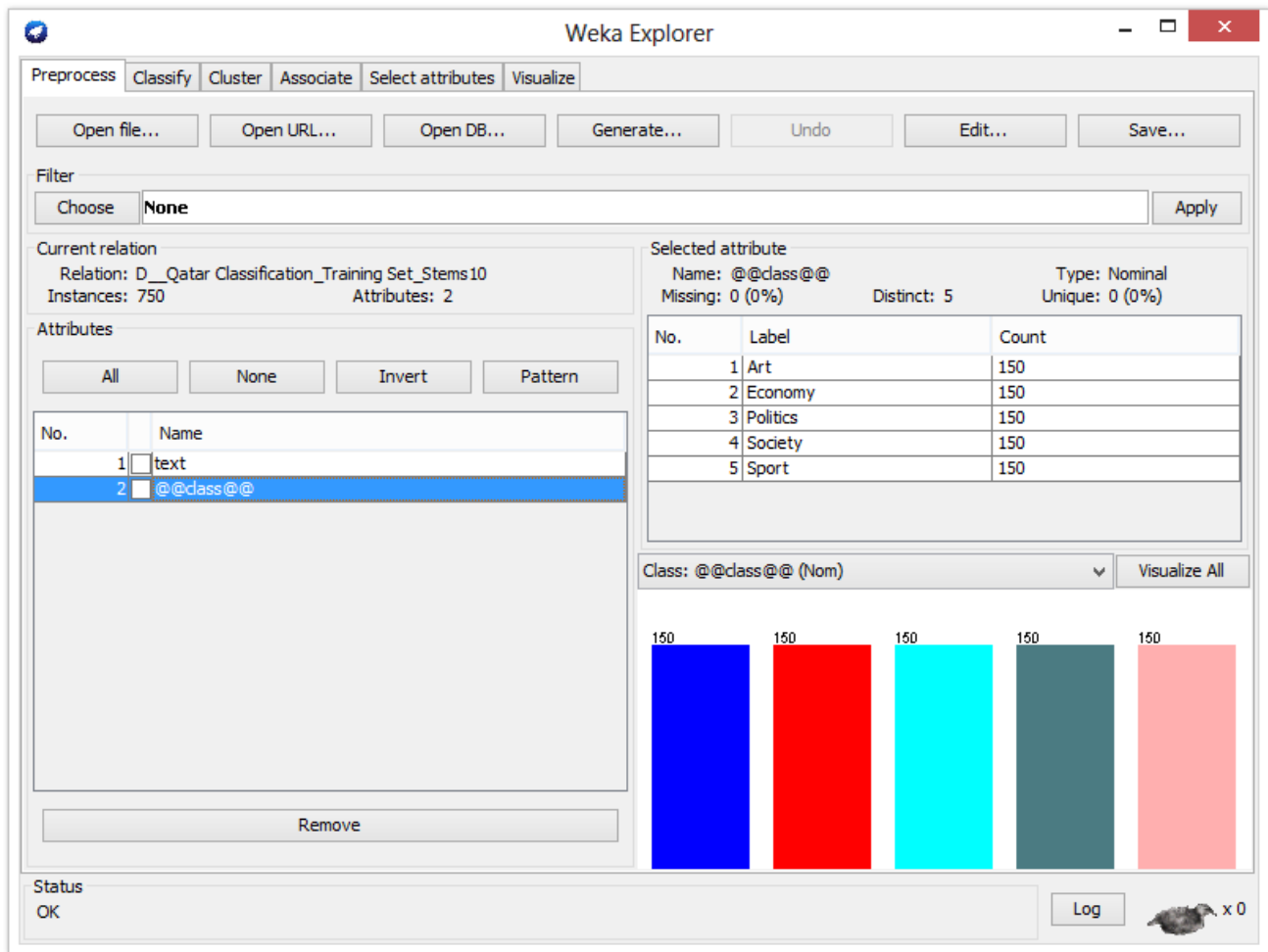


Figure 6.2: Opening an ARFF file.

Text classifiers cannot handle the “text” attribute as a single attribute. Therefore, we must first convert the “text” attribute to a word vector. To perform this, simply click on the “Choose” button under the “Filter” group box. Weka will display a tree of available filters as shown in Figure 6.3. To convert the “text” attribute into a word vector, select the “StringToWordVector”, which is found under “filters → unsupervised → attribute”.

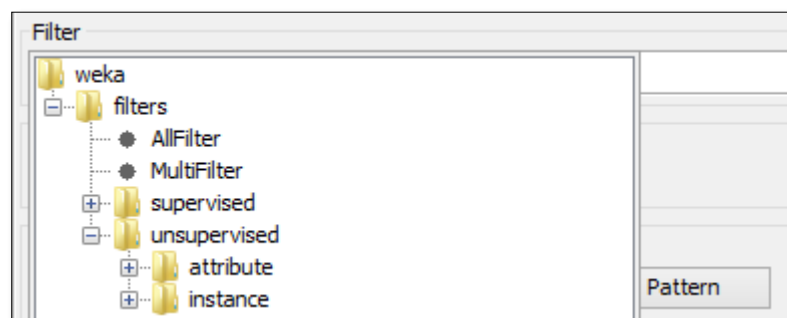


Figure 6.3: Weka's filters.

The “StringToWordVector” filter explores the value of the “text” attribute for each instance and creates a word vector. Afterwards, it uses the word vector to replace the “text” attribute with a set of numerical attributes, each corresponding to a word from the word vector. By default, the value of a numerical attribute

is a binary value. If a word appears in the text for a given instance then the value of its corresponding attribute is 1; otherwise the value is 0.

The “StringToWordVector” provides several options to specify how to compute the values of the numerical attributes. To display the parameters of the “StringToWordVector” filter click on the text box under the “Filter” group box as shown in Figure 6.4. This will display the parameters dialog shown in Figure 6.5.

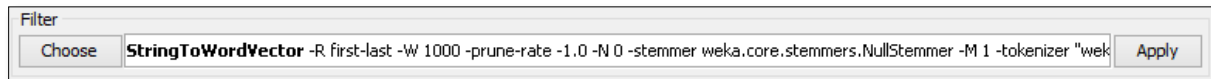


Figure 6.4: Opening the parameters of the selected filter.

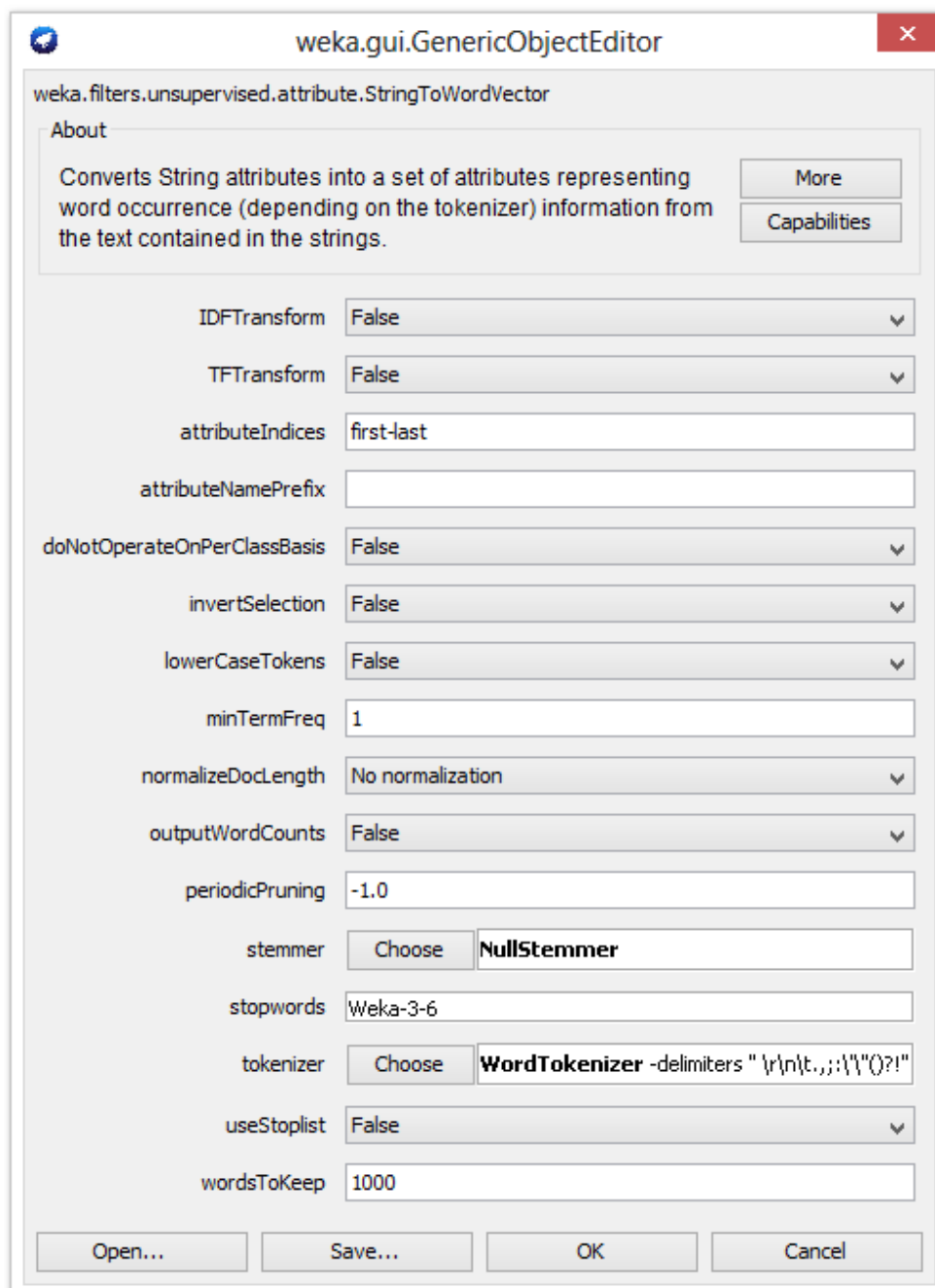


Figure 6.5: The parameters of the “StringToWordVector” filter.

Rather than using binary values for the attributes that indicates the appearance or absence of a word in a given instance, we chose to use the Term Frequency-Inverse Documents Frequency (TF-IDF), which is a numerical statistic often used as a weighting factor to reflect how important a word is to a document in a collection. The TF-IDF is the product of two statistics, term frequency and inverse document frequency. The term frequency, in its simplest form, is the number of times the term appears in a document while the inverse document frequency is a measure of whether the term is common or rare across all documents. To use TF-IDF in Weka, set the “IDFTransform”, “TFTransform”, and “outputWordCounts” parameters shown in Figure 6.5 to true. If you want to use all words appearing in all documents, you may set the “wordsToKeep” parameters to some large value.

After applying the “StringToWordVector”, the obtained feature vectors tends to have very large dimensionality, which can affect the robustness of the text classifier. In order to avoid unnecessary large feature vectors, feature selection can be used. Weka provide a filter named “AttributeSelection”, which can select the most relevant attributes based on a given criteria. The “AttributeSelection” filter is available from the filters tree at the path “filters”→“supervised”→“attribute”. We used the “AttributeSelection” filter with “InfoGainAttributeEval” as evaluator and a “Ranker” with a threshold of value 0 as shown in Figure 6.6.

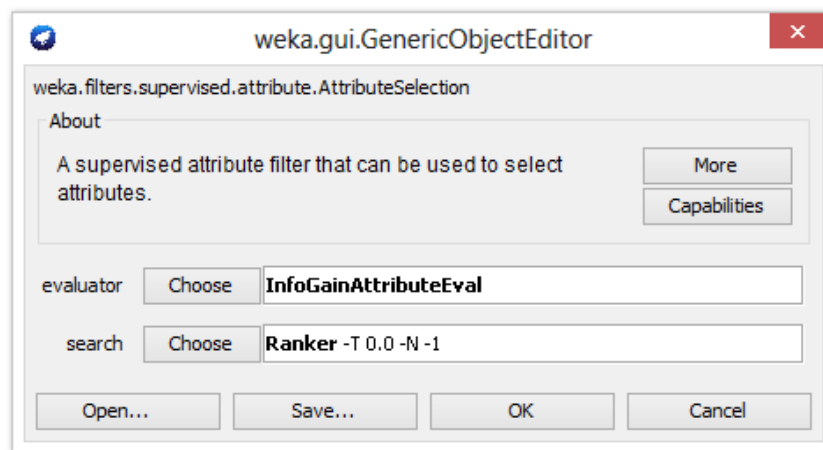


Figure 6.6: The parameters dialog of the “AttributeSelection” filter.

Table 6.2 shows the number of features for each of the seven training set versions. In the table, “Distinct words” refers to the number of features after applying the “StringToWordVector” filter and “Selected features” refers to the number of features after applying the “StringToWordVector” filter followed by the “AttributeSelection” filter.

Table 6.2: Number of features for each training set version.

	Words	Stems1	Stems2	Stems3	Stems8	Stems10	StemsP
Distinct words	28,704	23,703	21,283	19,282	15,899	15,124	20,457
Selected features	1,933	1,755	1,738	1,644	1,465	1,427	1,700

Now, we have the selected the feature vectors and are ready to start the training of the classifiers.

6.2 Multiclass Classifiers

As mentioned previously we chose to train and test three of the most widely used classification approaches, which are Support Vector Machines (SVM), Naïve Bayes, and Random Forest. Weka provides a variety of text classifiers, available from the classifiers tree. In order to show the classifiers tree, go to the “Classify” tab and click the “Choose” button under the “Classifier” group box. This shows the classifiers tree as shown in Figure 6.7.

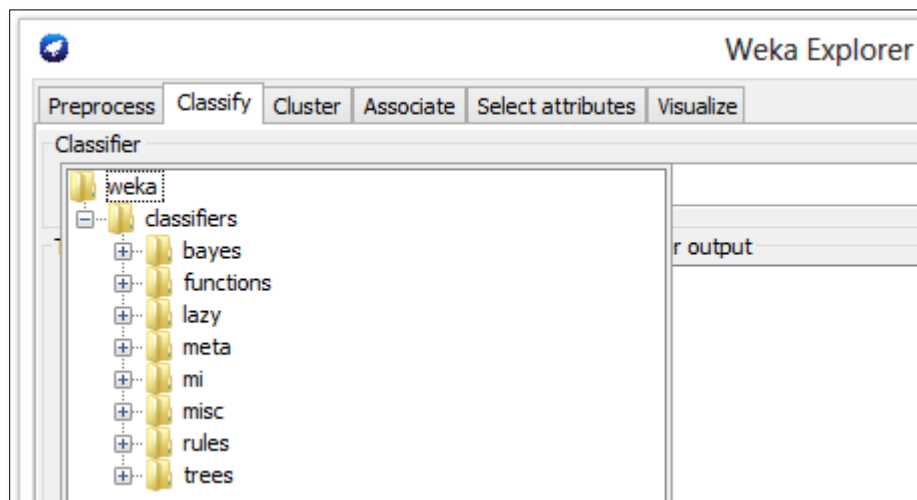


Figure 6.7: Weka's classifiers tree.

In order to build a multiclass classifier, choose the “MultiClassClassifier” classifier from the classifiers tree at “weka→classifiers→meta”. To determine the classification technique to use, click on the text box under the “Classifier” group box as shown in Figure 6.8Figure 6.4. This will display the parameters dialog shown in Figure 6.9Figure 6.5. To select the classification approach click on “Choose” and select the desired classification technique from the displayed tree.

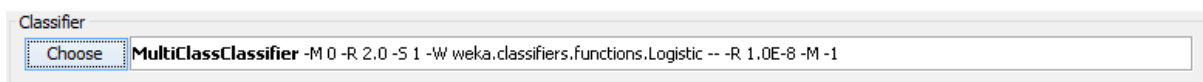


Figure 6.8: Opening the parameters of the selected classifier.

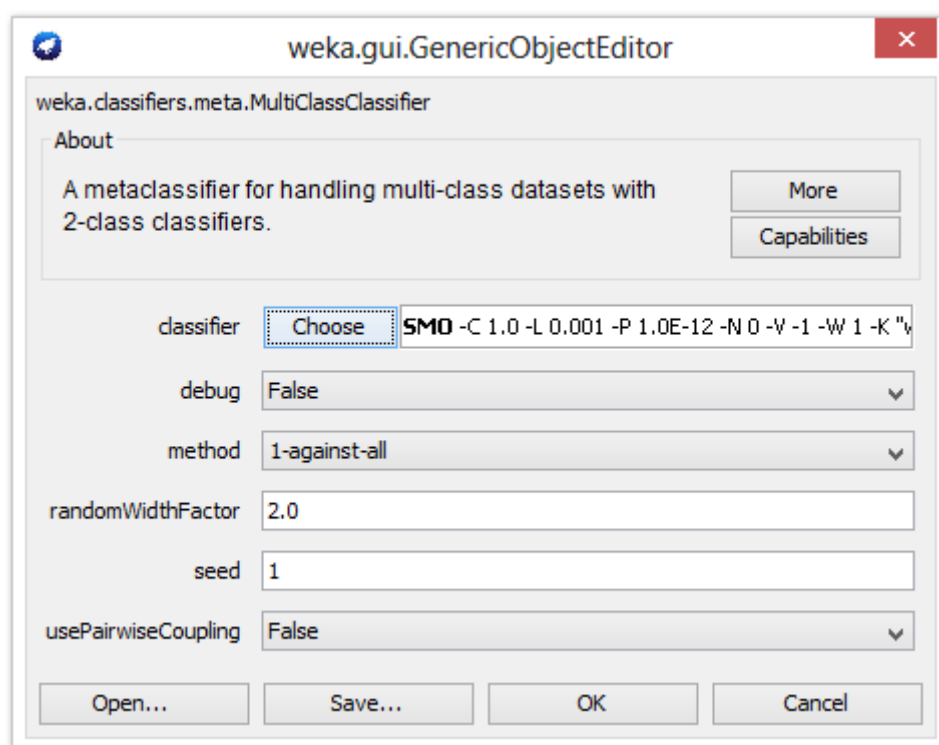


Figure 6.9: The parameters of the “MultiClassClassifier” classifier.

We have built three text classifiers corresponding to the three classification techniques for each of the seven training set versions. We used 10-fold cross-validation to evaluate each of the 21 classifiers. The results for the training sets *Words*, *Stems1*, *Stems2*, *Stems3*, *Stems8*, *Stems10*, and *StemsP* are shown in Table 6.3, Table 6.4, Table 6.5, Table 6.6, Table 6.7, Table 6.8, and Table 6.9, respectively.

Table 6.3: The results of a 10-fold cross-validation using the *Words* version of the training set.

	SVM (SMO)			Naïve Bayes			Random Forest		
	Precision	Recall	F	Precision	Recall	F	Precision	Recall	F
Art	0.794	1	0.885	0.886	0.987	0.934	0.94	0.947	0.944
Economy	0.938	0.813	0.871	0.83	0.973	0.896	0.862	0.873	0.868
Politics	0.896	0.86	0.878	0.93	0.707	0.803	0.833	0.833	0.833
Society	1	0.987	0.993	1	0.973	0.986	0.98	1	0.99
Sport	0.986	0.913	0.948	0.973	0.953	0.963	0.979	0.94	0.959
Average	0.923	0.915	0.915	0.924	0.919	0.916	0.919	0.919	0.919

Table 6.4: The results of a 10-fold cross-validation using the *Stems1* version of the training set.

	SVM (SMO)			Naïve Bayes			Random Forest		
	Precision	Recall	F	Precision	Recall	F	Precision	Recall	F
Art	0.852	1	0.92	0.893	1	0.943	0.941	0.953	0.947
Economy	0.964	0.887	0.924	0.827	0.953	0.885	0.89	0.867	0.878
Politics	0.925	0.907	0.916	0.949	0.74	0.831	0.846	0.84	0.843
Society	1	0.987	0.993	1	0.98	0.99	0.98	1	0.99
Sport	0.993	0.933	0.962	0.986	0.953	0.969	0.967	0.967	0.967
Average	0.947	0.943	0.943	0.931	0.925	0.924	0.925	0.925	0.925

Table 6.5: The results of a 10-fold cross-validation using the *Stems2* version of the training set.

	SVM (SMO)			Naïve Bayes			Random Forest		
	Precision	Recall	F	Precision	Recall	F	Precision	Recall	F
Art	0.847	1	0.917	0.893	1	0.943	0.959	0.927	0.942
Economy	0.957	0.9	0.928	0.833	0.967	0.895	0.878	0.86	0.869
Politics	0.018	0.893	0.908	0.966	0.753	0.846	0.834	0.873	0.853
Society	0.924	0.987	0.993	1	0.973	0.986	0.968	1	0.984
Sport	1	0.92	0.955	0.986	0.953	0.969	0.979	0.953	0.966
Average	0.944	0.94	0.94	0.936	0.929	0.928	0.924	0.923	0.923

Table 6.6: The results of a 10-fold cross-validation using the *Stems3* version of the training set.

	SVM (SMO)			Naïve Bayes			Random Forest		
	Precision	Recall	F	Precision	Recall	F	Precision	Recall	F
Art	0.838	1	0.912	0.886	0.98	0.93	0.946	0.94	0.943
Economy	0.936	0.873	0.903	0.847	0.96	0.9	0.851	0.8	0.825
Politics	0.882	0.847	0.864	0.921	0.773	0.841	0.795	0.853	0.823
Society	1	0.987	0.993	1	0.973	0.986	0.98	1	0.99
Sport	0.993	0.92	0.955	0.986	0.933	0.959	0.979	0.953	0.966
Average	0.93	0.925	0.926	0.928	0.924	0.923	0.91	0.909	0.909

Table 6.7: The results of a 10-fold cross-validation using the *Stems8* version of the training set.

	SVM (SMO)			Naïve Bayes			Random Forest		
	Precision	Recall	F	Precision	Recall	F	Precision	Recall	F
Art	0.856	0.993	0.92	0.855	0.987	0.916	0.935	0.96	0.947
Economy	0.949	0.873	0.91	0.88	0.973	0.924	0.842	0.82	0.831
Politics	0.898	0.88	0.889	0.935	0.767	0.842	0.814	0.847	0.83
Society	1	0.987	0.993	1	0.973	0.986	1	1	1
Sport	0.986	0.94	0.962	0.979	0.927	0.952	0.979	0.94	0.959
Average	0.938	0.935	0.935	0.93	0.925	0.924	0.914	0.913	0.914

Table 6.8: The results of a 10-fold cross-validation using the *Stems10* version of the training set.

	SVM (SMO)			Naïve Bayes			Random Forest		
	Precision	Recall	F	Precision	Recall	F	Precision	Recall	F
Art	0.856	0.993	0.92	0.86	0.987	0.919	0.928	0.947	0.937
Economy	0.935	0.867	0.9	0.878	0.96	0.917	0.849	0.86	0.854
Politics	0.905	0.887	0.896	0.921	0.773	0.841	0.842	0.82	0.831
Society	1	0.987	0.993	1	0.973	0.986	0.987	1	0.993
Sport	0.986	0.933	0.959	0.979	0.927	0.952	0.973	0.953	0.963
Average	0.936	0.933	0.933	0.928	0.924	0.923	0.916	0.916	0.916

Table 6.9: The results of a 10-fold cross-validation using the *StemsP* version of the training set.

	SVM (SMO)			Naïve Bayes			Random Forest		
	Precision	Recall	F	Precision	Recall	F	Precision	Recall	F
Art	0.838	1	0.912	0.92	0.993	0.955	0.973	0.953	0.963
Economy	0.965	0.907	0.935	0.829	0.967	0.892	0.891	0.873	0.882
Politics	0.937	0.893	0.915	0.95	0.767	0.849	0.855	0.867	0.861
Society	1	0.987	0.993	1	0.98	0.99	0.987	1	0.993
Sport	0.993	0.92	0.955	0.979	0.947	0.963	0.961	0.973	0.967
Average	0.946	0.941	0.942	0.936	0.931	0.93	0.933	0.933	0.933

As shown in Table 6.10 and Figure 6.10, the *StemsP* training set gives the highest average F-measure across the three text classifiers. Recalling that *StemsP* was obtained by the proposed *P-Stemmer*, this supports our argument mentioned in section 5.4, which states that the removal of word prefixes only can improve the effectiveness of text classifiers compared to a light stemmer which removes both word prefixes and suffixes. In addition, the results shows that SVM outperforms both Naïve Bayes and Random Forest. One reason for that is the robustness of SVM against high-dimensional feature spaces.

Table 6.10: The F-measure values for the three classification techniques.

	SVM (SMO)	Naïve Bayes	Random Forest	Average
Words	0.915	0.916	0.919	0.917
Stems1	0.943	0.924	0.925	0.931
Stems2	0.94	0.928	0.923	0.93
Stems3	0.926	0.923	0.909	0.919
Stems8	0.935	0.924	0.914	0.924
Stems10	0.933	0.923	0.916	0.924
StemsP	0.942	0.93	0.933	0.935

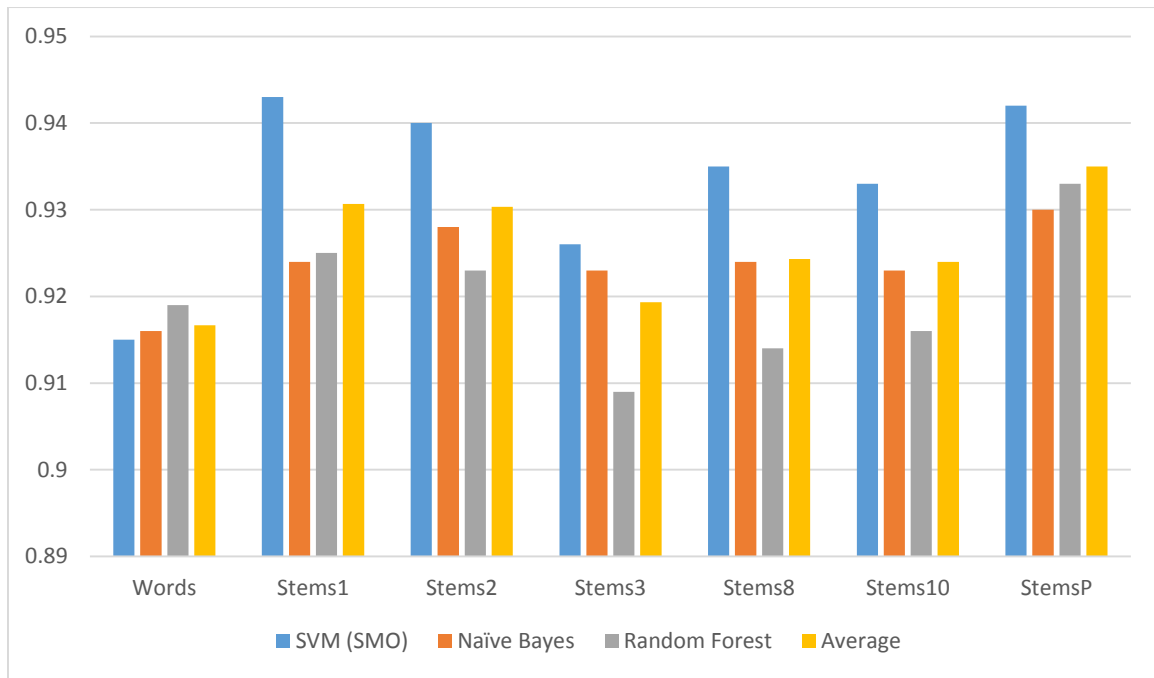


Figure 6.10: F-measure values for the three classification techniques.

6.3 Binary Classifiers

Using the instances stemmed by the proposed *P-Stemmer*, we have created five training sets corresponding to the five classes (i.e. Art, Economy, Politics, Society, and Sport). Each training set has 150 positive instances and 600 negative instances (i.e. 150 from every other class). We used the Weka’s “TextDirectoryLoader” tool to create the ARFF files for the five training sets as shown in Figure 6.11.

```

C:\>
D:\Qatar Classification\Training Set\Binary Training Set>java -cp weka.jar weka.
core.converters.TextDirectoryLoader -dir Art > Art.arff

D:\Qatar Classification\Training Set\Binary Training Set>java -cp weka.jar weka.
core.converters.TextDirectoryLoader -dir Economy > Economy.arff

D:\Qatar Classification\Training Set\Binary Training Set>java -cp weka.jar weka.
core.converters.TextDirectoryLoader -dir Politics > Politics.arff

D:\Qatar Classification\Training Set\Binary Training Set>java -cp weka.jar weka.
core.converters.TextDirectoryLoader -dir Society > Society.arff

D:\Qatar Classification\Training Set\Binary Training Set>java -cp weka.jar weka.
core.converters.TextDirectoryLoader -dir Sport > Sport.arff

D:\Qatar Classification\Training Set\Binary Training Set>

```

Figure 6.11: Creating ARFF files for the five training sets.

Again, we used the “StringToWordVector” filter followed by the “AttributeSelection” filter with “InfoGainAttributeEval” as evaluator and a “Ranker” with a threshold of value 0. Table 6.11 shows the number of features for each of the five training sets. In the table, “Distinct words” refers to the number of features after applying the “StringToWordVector” filter and “Selected features” refers to the number of features after applying the “StringToWordVector” filter followed by the “AttributeSelection” filter.

Table 6.11: Number of features for each training set version.

	Art	Economy	Politics	Society	Sport
Distinct words	2,0457	2,0457	2,0457	2,0457	2,0457
Selected features	2,058	1,214	883	1,065	1,529

Afterwards, we used Weka to build three text classifiers using the three classification techniques for each of the five training sets. The results of a 10-fold cross-validation of the three classifiers for the training sets *Art*, *Economy*, *Politics*, *Society*, and *Sport* are shown in Table 6.12, Table 6.13, Table 6.14, Table 6.15, and Table 6.16, respectively.

Table 6.12: The results of a 10-fold cross-validation of three the “Art” classifiers.

	SVM (SMO)			Naïve Bayes			Random Forest		
	Precision	Recall	F	Precision	Recall	F	Precision	Recall	F
Positive	0.982	0.998	0.99	0.998	0.958	0.978	0.92	0.998	0.958
Negative	0.993	0.927	0.959	0.856	0.993	0.92	0.99	0.653	0.787
Weighted Average	0.984	0.984	0.984	0.97	0.965	0.966	0.934	0.929	0.924

Table 6.13: The results of a 10-fold cross-validation of the three “Economy” classifiers.

	SVM (SMO)			Naïve Bayes			Random Forest		
	Precision	Recall	F	Precision	Recall	F	Precision	Recall	F
Positive	0.955	0.983	0.969	0.995	0.915	0.953	0.925	0.983	0.953
Negative	0.924	0.813	0.865	0.742	0.98	0.845	0.911	0.68	0.779
Weighted Average	0.949	0.949	0.948	0.944	0.928	0.931	0.922	0.923	0.918

Table 6.14: The results of a 10-fold cross-validation of the three “Politics” classifiers.

	SVM (SMO)			Naïve Bayes			Random Forest		
	Precision	Recall	F	Precision	Recall	F	Precision	Recall	F
Positive	0.953	0.982	0.967	0.982	0.91	0.945	0.899	0.983	0.939
Negative	0.917	0.807	0.858	0.722	0.933	0.814	0.894	0.56	0.689
Weighted Average	0.946	0.947	0.945	0.93	0.915	0.919	0.898	0.899	0.889

Table 6.15: The results of a 10-fold cross-validation of the three “Society” classifiers.

	SVM (SMO)			Naïve Bayes			Random Forest		
	Precision	Recall	F	Precision	Recall	F	Precision	Recall	F
Positive	0.997	1	0.998	0.997	1	0.998	0.998	0.998	0.998
Negative	1	0.987	0.993	1	0.987	0.993	0.993	0.993	0.993
Weighted Average	0.997	0.997	0.997	0.997	0.997	0.997	0.997	0.997	0.997

Table 6.16: The results of a 10-fold cross-validation of the three “Sport” classifiers.

	SVM (SMO)			Naïve Bayes			Random Forest		
	Precision	Recall	F	Precision	Recall	F	Precision	Recall	F
Positive	0.985	0.998	0.992	0.997	0.995	0.996	0.958	0.998	0.978
Negative	0.993	0.94	0.966	0.98	0.987	0.983	0.992	0.827	0.902
Weighted Average	0.987	0.987	0.987	0.993	0.993	0.993	0.965	0.964	0.963

As shown in Table 6.17 and Figure 6.12, the society classifiers have the highest F-measure values. Again, SVM outperforms both Naïve Bayes and Random Forest.

Table 6.17: The F-measure values for the three classification techniques.

	SVM (SMO)	Naïve Bayes	Random Forest	Average
Art	0.984	0.966	0.924	0.958
Economy	0.948	0.931	0.918	0.932
Politics	0.945	0.919	0.889	0.918
Society	0.997	0.997	0.997	0.997
Sport	0.987	0.993	0.963	0.981

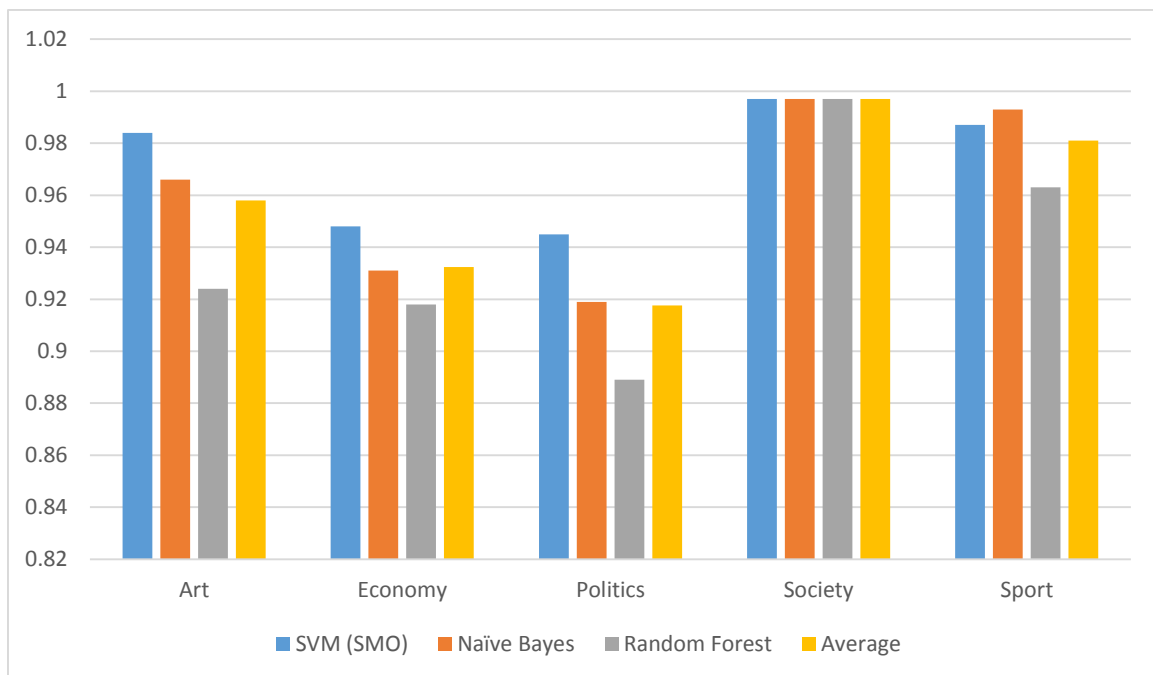


Figure 6.12: The F-measure values for the three classification techniques.

6.4 Compound Binary Classifiers

The results of multiclass classifiers (see section 6.2) and binary classifiers (see section 6.3) shows that the effectiveness of multiple binary classifiers is far much better than the effectiveness of multiclass classifiers. Moreover, we have used the binary classifiers to build compound classifiers using the voting approach.

In Weka, to combine multiple classifiers, click on the “Choose” button under the “Classifiers” group box. This will display a tree of classifiers as shown in Figure 6.13 from which you can select “vote” which is under “meta” in the classifiers tree.

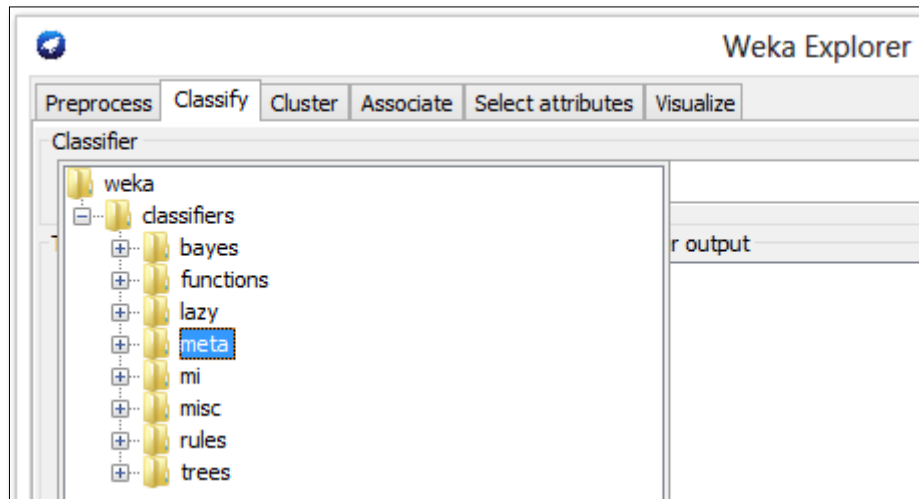


Figure 6.13: Weka's classifiers tree.

To display the parameters of “voting”, click on the textbox under the “Classifier” group box as shown in Figure 6.14. This will display the parameters dialog shown in Figure 6.15.

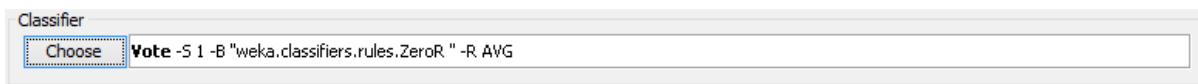


Figure 6.14: Opening the parameters of Weka's voting compound classifier.

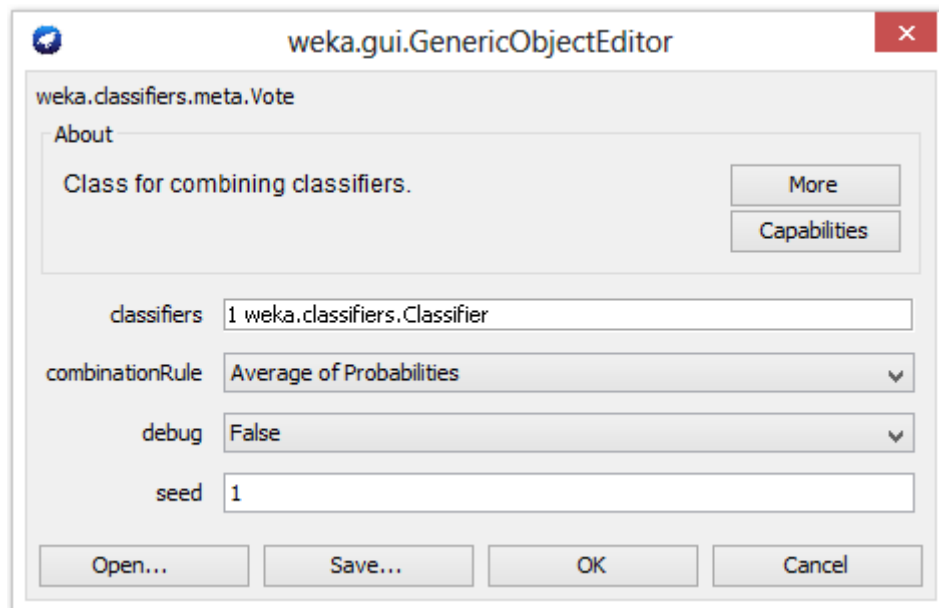


Figure 6.15: Parameters dialog of the “voting” compound classifier.

Within the voting parameters dialog, click of the “classifiers” textbox to display the dialog shown in Figure 6.16, from which you can choose the classifiers to be combined.

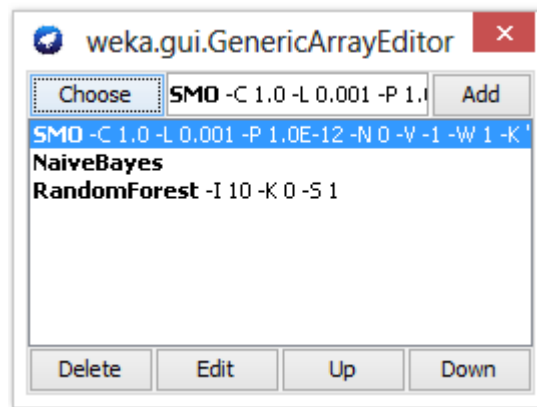


Figure 6.16: Voting members

Using the voting approach, we have combined the three binary classifiers (i.e. SVM, Naïve Bayes, and Random Forest) of each of the five classes to form five compound classifiers. The results of a 10-fold cross-validation of the five compound classifier with the corresponding five training sets (i.e. *Art*, *Economy*, *Politics*, *Society*, and *Sport*) are shown in Table 6.18.

Table 6.18: The results of a 10-fold cross-validation of the five compound classifiers.

		Positive	Negative	Weighted Average
Art	Precision	0.982	0.993	0.984
	Recall	0.998	0.927	0.984
	F-Measure	0.99	0.959	0.984
Economy	Precision	0.964	0.908	0.953
	Recall	0.978	0.853	0.953
	F-Measure	0.971	0.88	0.953
Politics	Precision	0.965	0.884	0.949
	Recall	0.972	0.86	0.949
	F-Measure	0.968	0.872	0.949
Society	Precision	0.998	1	0.999
	Recall	1	0.993	0.999
	F-Measure	0.999	0.997	0.999
Sport	Precision	0.987	0.993	0.988
	Recall	0.998	0.947	0.988
	F-Measure	0.993	0.969	0.988

As shown in Table 6.19 and Figure 6.17, the compound classifier have the highest F-measure values compared to each of the three binary classifiers with minor exceptions.

Table 6.19: The F-measure values for the three classifiers compared to the compound classifier.

	SVM (SMO)	Naïve Bayes	Random Forest	Voting
Art	0.984	0.966	0.924	0.984
Economy	0.948	0.931	0.918	0.953
Politics	0.945	0.919	0.889	0.949
Society	0.997	0.997	0.997	0.999
Sport	0.987	0.993	0.963	0.988

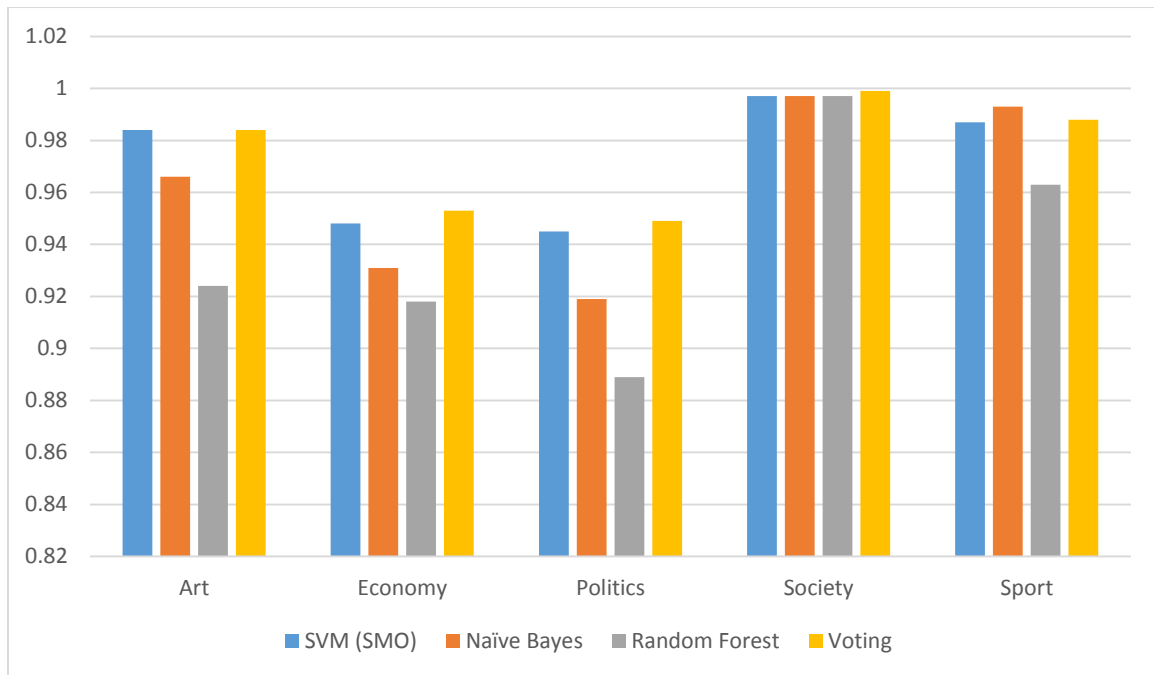


Figure 6.17: The F-measure values for the three classifiers compared to the compound classifier.

7 Classifying the Testing Set

Given the results of evaluating different classifiers using 10-fold cross-validation over the training set, we chose to use five compound classifiers as illustrated in section 6.4 to classify the test set. For each compound classifier, the classifier is trained using one of the five training sets (i.e. *Art*, *Economy*, *Politics*, *Society*, and *Sport*). Afterwards, the trained classifier takes the unlabeled testing set and labels each instance as either positive or negative. The training and the testing sets must be compatible, that is they must have the same attributes stored in the same order. Therefore, we started by extracting the list of attributes from the ARFF files used by each of the compound classifiers in section 6.4. Having a list of attributes (i.e. word vector) for each of the five classes, we developed a tool, named “TXT2CSV”. The tool takes the path to a directory containing the text files representing instances, the filename of the file containing the attributes of interest, and the class label to be assigned to the instances. The tool creates a CSV file as an output file containing the attributes of interest together with their values for each instance as shown in Figure 7.1.

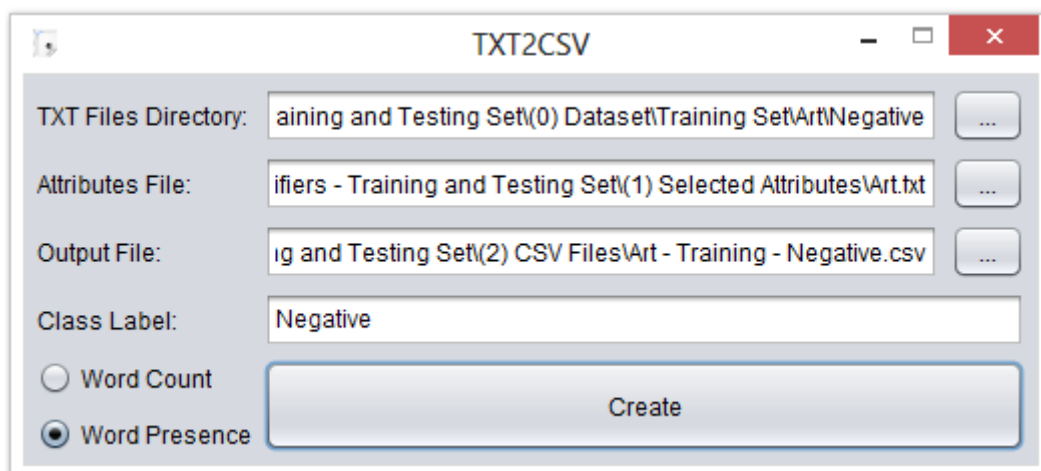


Figure 7.1: The “TXT2CSV” tool.

Having five classes, we used the “TXT2CSV” tool to produce two CSV files per class, one for positive instances and the other for negative instances. Afterwards, for each class the two CSV files representing positive and negative instances were merged into a single CSV file, which represents the training set. For the testing set, we used the “TXT2CSV” tool to generate a CSV file using the same attributes file and “Negative” as a “Class Label” (The value “Negative” is just a place holder). Now, for each of the five classes we have two CSV files representing compatible training and testing sets.

In order to train a classifier using the training set, open the training CSV file in Weka. Then go to the “Classify” tab and select the “Use training set” radio button under the “Test options” group box as shown in Figure 7.2.

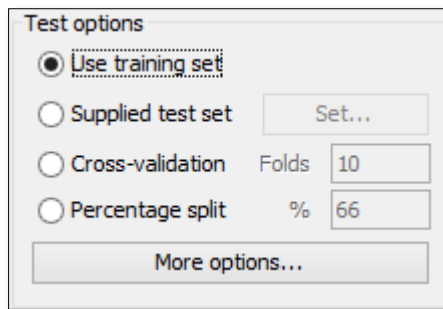


Figure 7.2: Classification's test options.

After training the five compound classifiers, we have saved the models for future use by right clicking on a model and selecting “Save model” as shown on Figure 7.3.

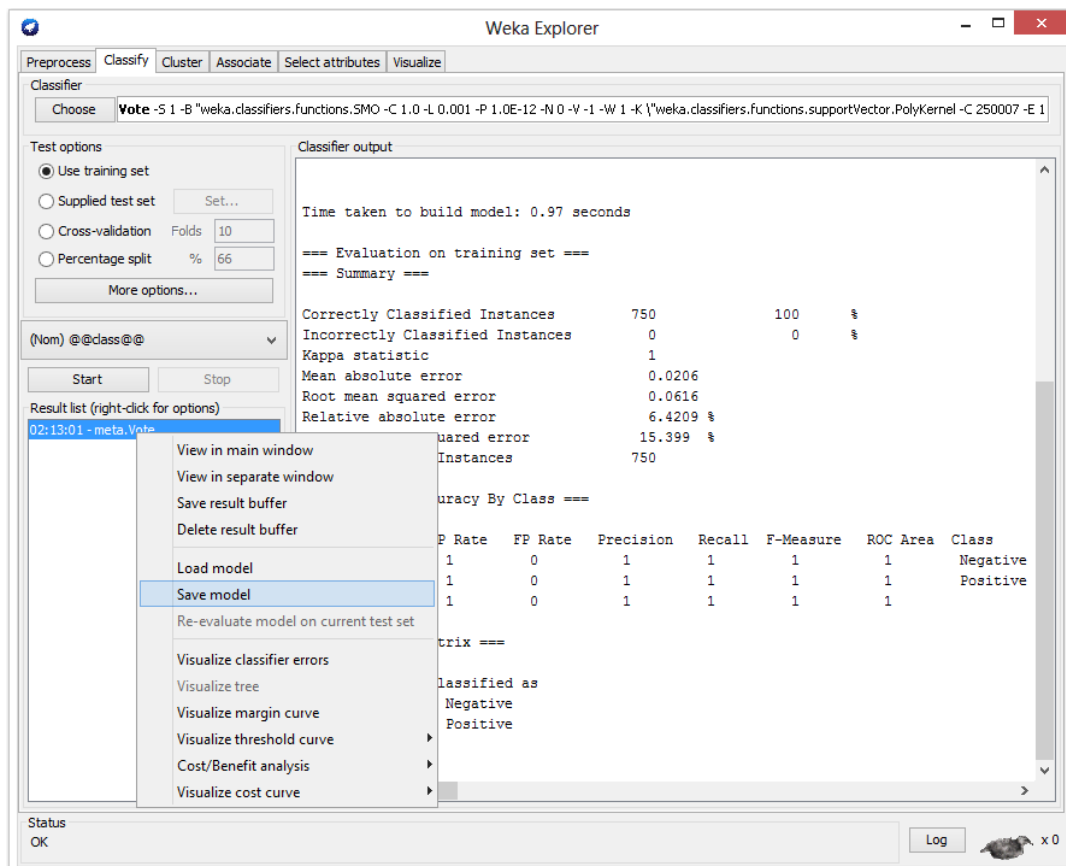


Figure 7.3: Saving a model.

As mentioned in section 4, the testing set in its raw form is a set of 1700 newspaper PDF files. First, we used the developed PDF2TXT tool to extract the text from the PDF files, such that the text from each page is extracted to a distinct text file. Thus, we obtained a set 58,938 text files, each file corresponding to a page. Second, we used the “Arabic Words Extractor” tool for cleaning and normalizing the text as well as removing stop-words. Third, we used the “Arabic Light Stemmer” tool to stem the words in each of the text files using the proposed P-Stemmer. Since some of the PDF Pages of the collected newspapers were only full-page images, some of the text files were empty. Moreover, some pages contained too little text. We have removed the text files with a size less than 2 kilobytes. Thus, out of the 58,938 text files we still have 38,017 text files. To reduce computation overhead and memory requirements, we decided to split the testing set into chunks of 2000 instance each. For each chunk, we have created 5 CSV files, one for each class using the attributes selected for that class.

In use the testing set, open the training CSV file using Weka, then go to the classify tab, select “Supplied test set” under the “Test options” group box, and click on the “Set...” button to specify the testing set CSV file as shown in Figure 7.4. You may click on “More options...” to display the options dialog from which you can select “Output predictions” to list the testing set instances together with their corresponding assigned class.

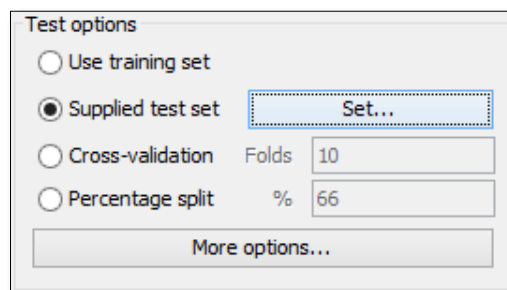


Figure 7.4: Specifying a testing set CSV file.

Then, right click on the “Results list” and select “Load model” to load the model you saved before as shown in Figure 7.5. Finally, right click on the loaded model and select “Re-evaluate model on current test set” as shown in Figure 7.6.

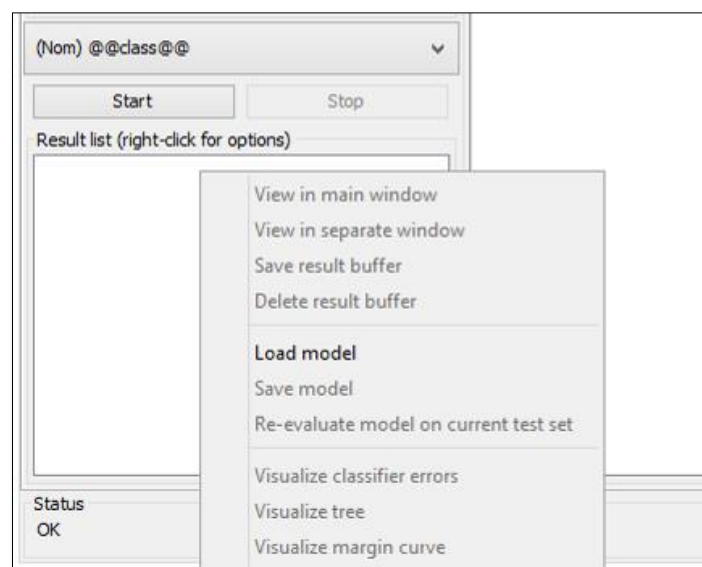


Figure 7.5: Loading a saved model in Weka.

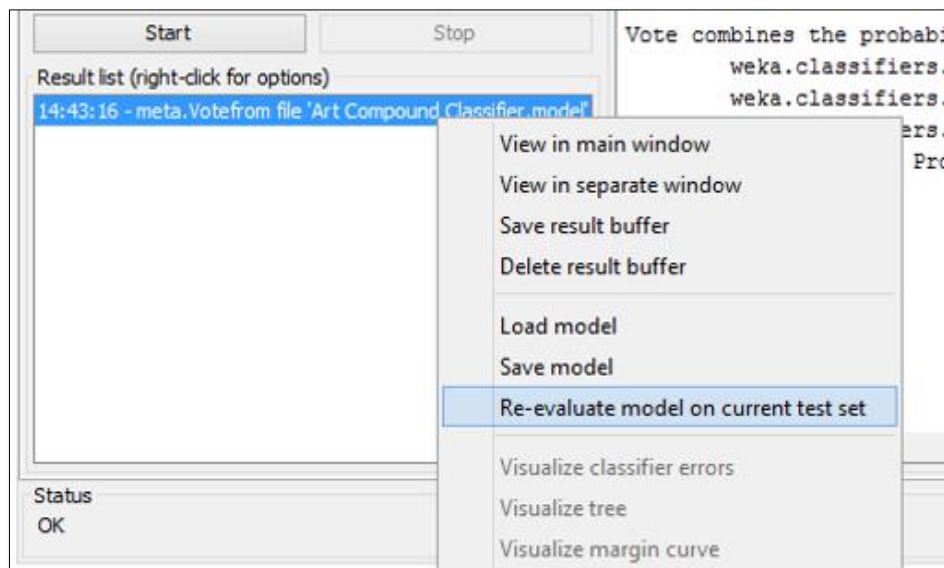


Figure 7.6: Re-evaluating a model based on current test set.

For instance, the results of classifying the testing set using the Art compound classifier are shown in Figure 7.7. for each instance we chose to print its filename as well as the assigned label. All instances, considered to be positive by the Art classifier will be uploaded to the Art core on Solr. The same applies to the remaining 4 classifiers and their corresponding 4 Solr cores. Creating Solr cores and uploading documents to then is illustrated in section 8.

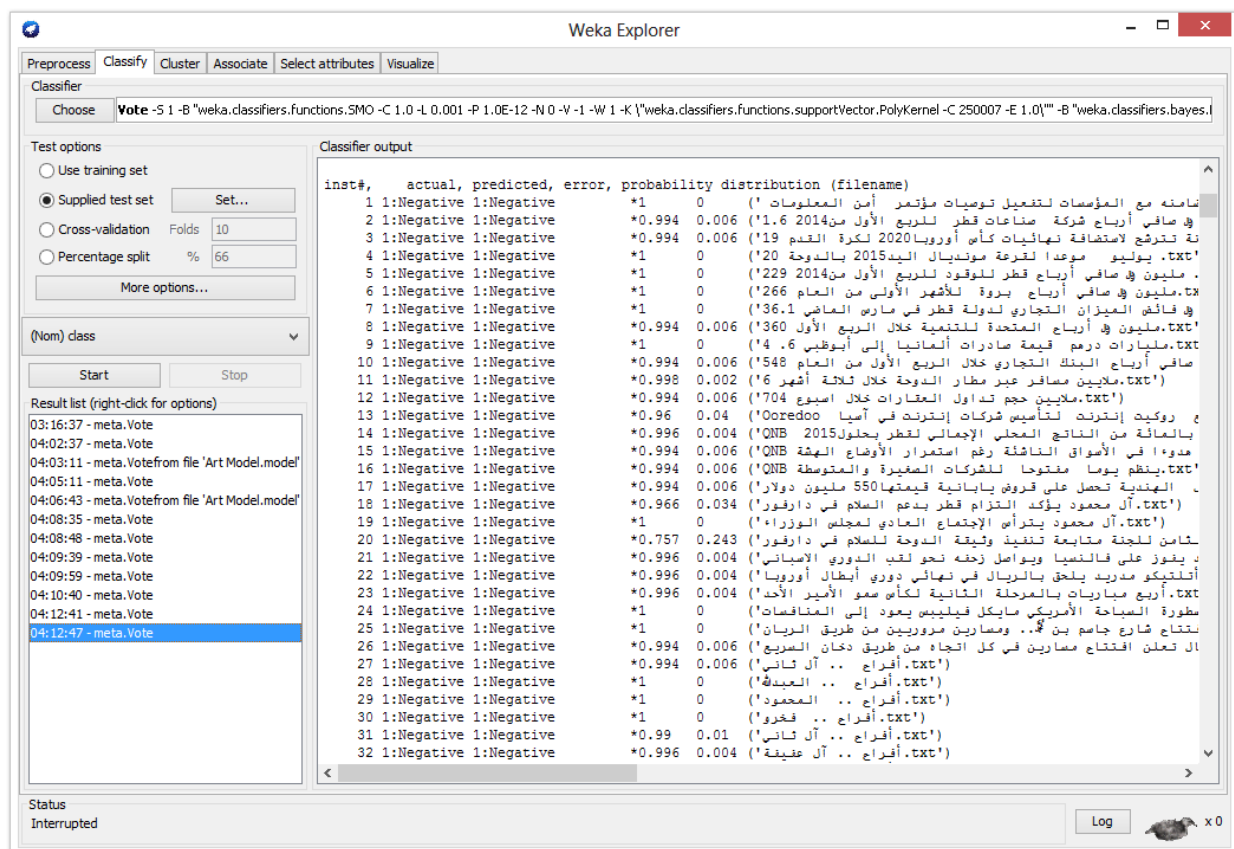


Figure 7.7: Results of classifying the testing set instances using the Art compound classifier.

8 Indexing and Searching Collections using Apache Solr

Solr is an open source enterprise search platform. Its major features include full-text search, hit highlighting, faceted search, dynamic clustering, database integration, and rich document (e.g., Word, PDF) handling. Solr supports the Arabic language and uses the Larkey's light stemmer to stems the Arabic words before indexing [10]. Solr is written in Java and runs as a standalone full-text search server within a servlet container such as Apache Tomcat. Apache Tomcat is an open source web server and servlet container developed by the Apache Software Foundation (ASF).

8.1 Installation Solr with Tomcat as a Web Container

This section describes briefly how to install Apache Solr on Ubuntu with Apache Tomcat as a web container.

8.1.1 Installing Apache Tomcat

1. Download Apache Tomcat from Apache Software Foundation. You can download it from the location <http://tomcat.apache.org/download-80.cgi>. I downloaded Tomcat 8.0.3 [tar.gz](#).
2. Extract the *apache-tomcat-8.0.3.tar.gz* file to *~/Desktop/Tomcat8/*
3. Edit the *~/.bashrc* file and add the line *export CATALINA_HOME=~/Desktop/Tomcat8* to its end.
4. Open a terminal and execute the command *~/.bashrc* to apply the changes you made to the file.
5. Created a file named *setenv.sh* in *~/Desktop/Tomcat8/bin/*
6. Edit the *setenv.sh* file and write *JAVA_HOME=/usr/lib/jvm/default-java*
7. Open a terminal and execute the command *\$CATALINA_HOME/bin/startup.sh* to start Tomcat.
8. Tested the installation by browsing to <http://localhost:8080>. You should see the Tomcat home page.

8.1.2 Installing Apache Solr

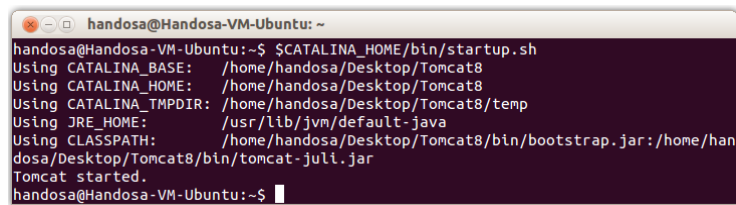
1. Download Apache Solr from Apache Software Foundation. You can download it from the location <https://lucene.apache.org/solr/>. I downloaded [solr-4.6.1.tgz](#).
2. Extract the *solr-4.6.1.tgz* file to *~/Desktop/solr-4.6.1*
3. Copy all the contents in *~/Desktop/solr-4.6.1/example/solr/* to *~/Desktop/Solr/*
4. Copy all the jar-files in *~/Desktop/solr-4.6.1/example/lib/ext* to *~/Desktop/Tomcat8/lib/*
5. Copy *solr.war* from *~/Desktop/solr-4.6.1/example/webapps* to *~/Desktop/Tomcat8/webapps/*
6. Edited the *~/Desktop/Tomcat8/webapps/solr/WEB-INF/web.xml* file, uncommented the *env-entry* section and replaced */put/your/solr/home/here* with */home/[user-name]/Desktop/Solr*
7. Add the line *JAVA_OPTS="-Dsolr.solr.home=/home/[user-name]/Desktop/Solr* to the *setenv.sh* file.
8. Tested the installation by browsing to <http://localhost:8080/solr/>. You should see the Solr home page.

8.2 Starting/Stopping the Apache Tomcat Service

To start the Apache Tomcat service, open a terminal and run the command

```
$CATALINA_HOME/bin/startup.sh
```

You should see something like what is shown in Figure 8.1.



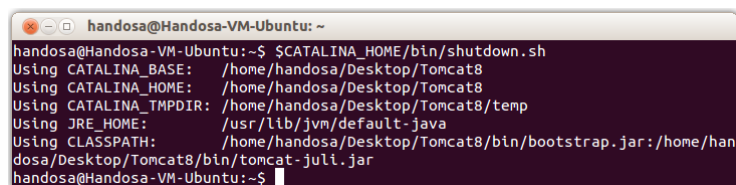
```
handosa@Handosa-VM-Ubuntu: ~  
handosa@Handosa-VM-Ubuntu:~$ $CATALINA_HOME/bin/startup.sh  
Using CATALINA_BASE: /home/handosa/Desktop/Tomcat8  
Using CATALINA_HOME: /home/handosa/Desktop/Tomcat8  
Using CATALINA_TMPDIR: /home/handosa/Desktop/Tomcat8/temp  
Using JRE_HOME: /usr/lib/jvm/default-java  
Using CLASSPATH: /home/handosa/Desktop/Tomcat8/bin/bootstrap.jar:/home/handosa/Desktop/Tomcat8/bin/tomcat-juli.jar  
Tomcat started.  
handosa@Handosa-VM-Ubuntu:~$
```

Figure 8.1: Starting the Apache Tomcat service.

To stop the Apache Tomcat service, open a terminal and run the command

```
$CATALINA_HOME/bin/shutdown.sh
```

You should see something like what is shown in Figure 8.2.



```
handosa@Handosa-VM-Ubuntu: ~  
handosa@Handosa-VM-Ubuntu:~$ $CATALINA_HOME/bin/shutdown.sh  
Using CATALINA_BASE: /home/handosa/Desktop/Tomcat8  
Using CATALINA_HOME: /home/handosa/Desktop/Tomcat8  
Using CATALINA_TMPDIR: /home/handosa/Desktop/Tomcat8/temp  
Using JRE_HOME: /usr/lib/jvm/default-java  
Using CLASSPATH: /home/handosa/Desktop/Tomcat8/bin/bootstrap.jar:/home/handosa/Desktop/Tomcat8/bin/tomcat-juli.jar  
handosa@Handosa-VM-Ubuntu:~$
```

Figure 8.2: Stopping the Apache Tomcat service.

8.3 Creating a Solr Core

Navigate to `~/Desktop/Solr` as shown in Figure 8.3 and create a folder named `ar-collection`.

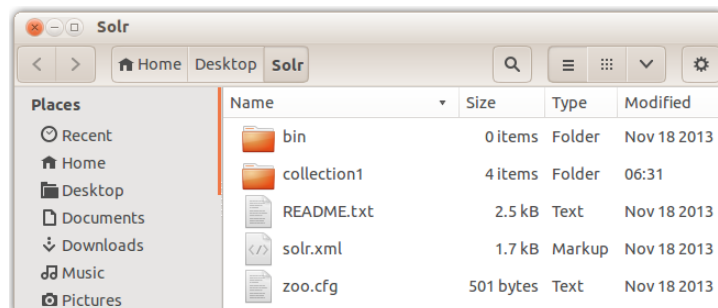


Figure 8.3: Navigating to “~/Desktop/Solr”.

Copy the directory `~/Desktop/Solr/collection1/conf` to `~/Desktop/Solr/ar-collection` as shown in Figure 8.4.

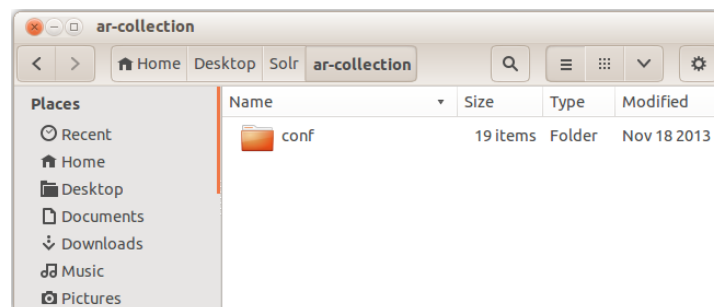


Figure 8.4: The “conf” folder copied from “~/Desktop/Solr/collection1”.

Start the Apache Tomcat web service then open a browser and navigate to <http://localhost:8080/solr/>. You should see the Apache Solr home page as shown in Figure 8.5.

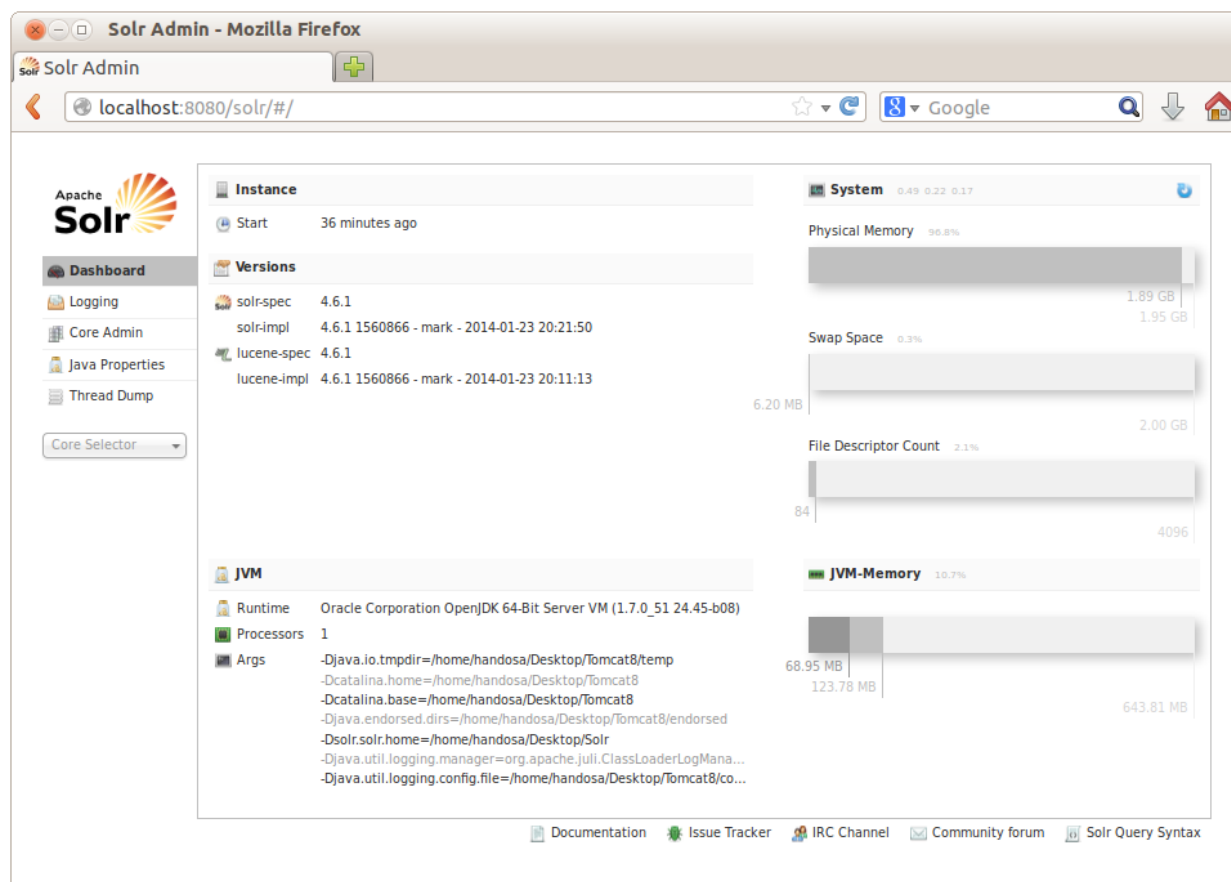


Figure 8.5: Apache Solr Dashboard.

From the left panel, select *Core Admin* then click on the *Add Core* button at the top as shown in Figure 8.6. Enter *ar-collection* in both the *name* and the *instanceDir* fields and click the *Add Core* button below.

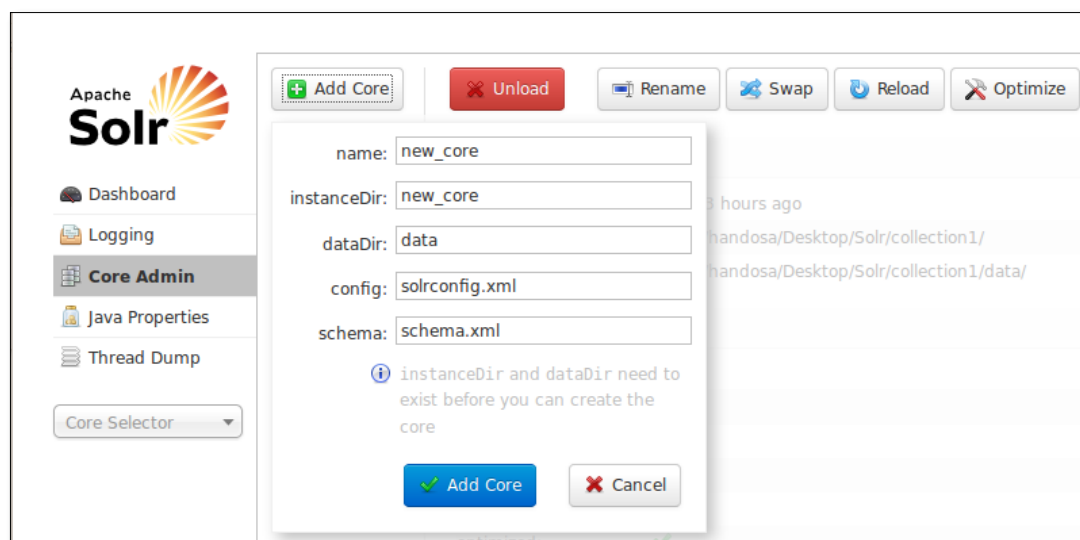


Figure 8.6: Apache Solr, Core Admin screen.

8.4 Editing the Schema file

The schema for the *ar-collection* core is located at `~/Desktop/Solr/ar-collection/conf/` (see Figure 8.7).

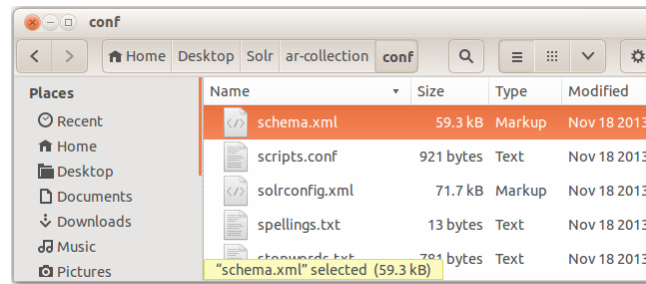


Figure 8.7: The location of the schema file for the “ar-collection” core.

The *schema.xml* file contains all of the details about which fields your documents can contain, and how those fields should be dealt with when adding documents to the index, or when querying those fields. Edit the *schema.xml* file and replace its contents with the following

```
<?xml version="1.0" encoding="utf-8" ?>
<schema name="ar-schema" version="1.5">
  <fields>
    <field name="_version_" type="long" indexed="true" stored="true"/> <!--Reserved-->
    <field name="_root_" type="string" indexed="true" stored="false"/> <!--Reserved-->
    <field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false"/>
    <field name="content" type="text_ar" indexed="true" stored="true" multiValued="true"/>
  </fields>
  <uniqueKey>id</uniqueKey>

  <types>
    <fieldType name="string" class="solr.StrField" sortMissingLast="true" />
    <fieldType name="long" class="solr.TrieLongField" precisionStep="0" positionIncrementGap="0"/>

    <fieldType name="text_ar" class="solr.TextField" positionIncrementGap="100">
      <analyzer>
        <tokenizer class="solr.StandardTokenizerFactory"/>
        <filter class="solr.LowerCaseFilterFactory"/> <!-- for any non-arabic -->
        <filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords_ar.txt" />
        <filter class="solr.ArabicNormalizationFilterFactory"/> <!-- normalizes ى to ي, etc -->
        <filter class="solr.ArabicStemFilterFactory"/>
      </analyzer>
    </fieldType>
  </types>
</schema>
```

You might need to restart the Tomcat service for changes to take effect and reflect in Solr UI.

8.5 Working with Documents through the Dashboard

A Solr index can be modified by POSTing commands to Solr to add (or update) documents, delete documents, and commit pending addition and deletion operations. These commands can be in a variety of formats including XML, JSON, CSV and JAVABIN as shown in Table 8.1.

Table 8.1: Solr command formats

JSON	<code>{"id":"###","content":"\$\$\$"}</code>
XML	<code><doc><field name="id">###</field><field name="content">\$\$\$</field></doc></code>
CSV	<code>id,title ###,\$\$\$</code>

8.5.1 Adding Documents

From the left panel, click on *Core Selector* and choose *ar-collection* then from the bottom panel select Documents. This will display the screen shown in Figure 8.8.

Apache Solr

Dashboard
Logging
Core Admin
Java Properties
Thread Dump
ar-collection
Overview
Analysis
Config
Dataimport
Documents
Ping
Plugins / Stats
Query
Replication
Schema
Schema Browser

Request-Handler (qt)
/update

Document Type
JSON

Document(s)
{\"id\":\"change.me\",\"title\":\"change.me\"}

Commit Within
1000

Overwrite
true

Boost
1.0

Submit Document

Documentation Issue Tracker IRC Channel Community forum Solr Query Syntax

Figure 8.8: The “ar-collection” Documents screen.

From the *Document Type* dropdown list, select *Document Builder*. This will display the document builder screen as shown in Figure 8.9. From the *Field* dropdown list select a field and enter the corresponding text in the *Field Data* textbox then click on *Add Field*. After finishing document building, click on *Submit Document*.

Apache Solr

Dashboard
Logging
Core Admin
Java Properties
Thread Dump
ar-collection
Overview
Analysis
Config
Dataimport
Documents
Ping
Plugins / Stats
Query
Replication
Schema
Schema Browser

Request-Handler (qt)
/update

Document Type
Document Builder

Field: id

Field Data: Enter your field text here and then click "Add Field" to add the field to the document.

Add Field

Document(s)

Commit Within
1000

Overwrite
true

Submit Document

Documentation Issue Tracker IRC Channel Community forum Solr Query Syntax

Figure 8.9: Document Builder screen.

8.6 Indexing Classified Collections

It is possible to modify the Solr index by POSTing XML documents containing instructions to add (or update) documents, delete documents, commit pending additions and deletions, and optimize the index [11]. The *exampledocs* directory, within the Solr distribution, contains samples of the types of instructions Solr expects, as well as a java utility for posting them from the command line as shown in Figure 8.10.

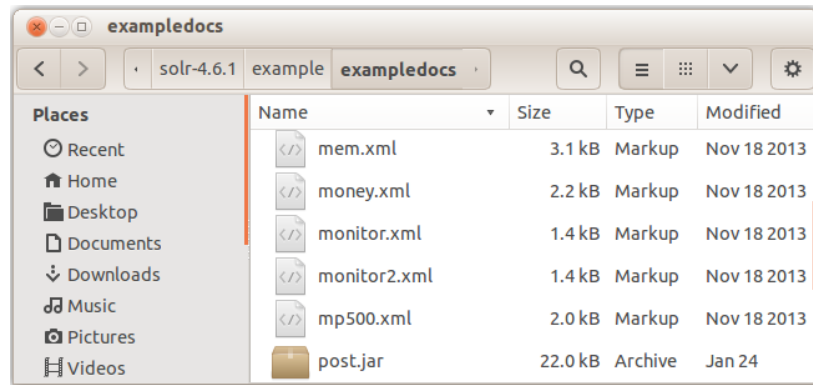


Figure 8.10: The “exampledocs” directory.

For example, to add the *monitor.xml* file to a Solr core named *test*, open a terminal window, enter the *exampledocs* directory, and run the following command

```
java -Durl=http://localhost:8080/solr/test/update -jar post.jar monitor.xml
```

From the Solr web interface, the statistics of the *test* core shows that the number of indexed documents is 1 as shown in Figure 8.11.

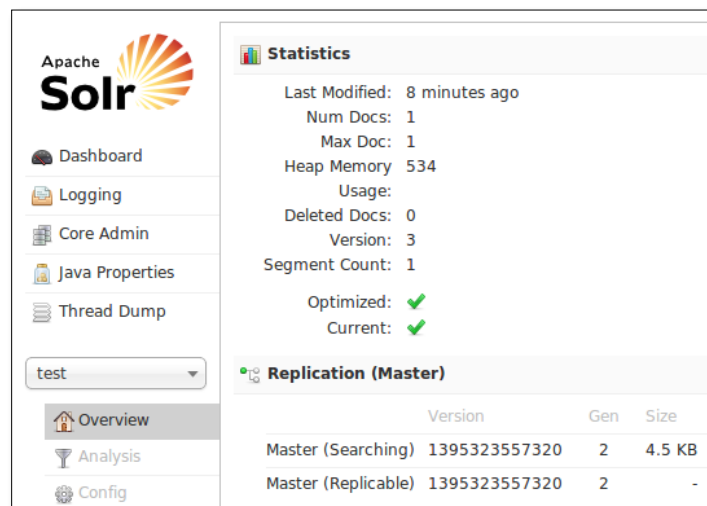


Figure 8.11: Statistics of the ‘test’ core.

Similarly, to index all XML documents within the *exampledocs* directory run the command

```
java -Durl=http://localhost:8080/solr/test/update -jar post.jar *.xml
```

The command above can be used to upload the testing set instances, each to its corresponding core, after transforming each test file into an XML file and add the necessary instructions to add the document to the index.

9 Conclusion and Future Work

We have shown that our proposed stemmer, P-Stemmer, outperforms the widely used Larkey's stemmer. We have built different classifiers using different machine learning techniques and shown that a compound binary classifier that uses the voting approach to combine SVM, Naïve Bayes and Random Forest outperforms both multiclass classifiers and binary classifiers based on a single learning technique (i.e. SVM, Naïve Bayes, or Random Forest). We also illustrated how to use Solr for indexing and searching Arabic collections.

Regarding the future work, we are planning to use Solr to assist in the evaluation of text classifiers on larger datasets, for which it is infeasible to classify manually. There are two proposed strategies. The first strategy is to use Solr to prepare the testing set by uploading all instances to a core and execute a query related to a given class. Afterwards, the output results are automatically labeled as belonging to that given class. The second strategy is to classify the instances using the classifier being evaluated and then upload the labeled instances to Solr. Afterwards, we execute carefully chosen queries and explore the search results with respect to their assigned labels. Hence, we can calculate the precision and recall values for the classifier under evaluation.

10 References

- [1] "NewsCodes," International Press Telecommunication Council (IPTC), December 2010. [Online]. Available: http://www.iptc.org/site/NewsCodes/View_NewsCodes. [Accessed 30 April 2014].
- [2] B. Carrier, "Extracting searchable text from Arabic PDFs," Basis Technology, 2009.
- [3] C. C. Aggarwal and C. Zhai, "A survey of text classification algorithms," in *Mining text data*, Springer US, 2012, pp. 163-222.
- [4] I. Abu El-Khair, "Effects of stop words elimination for Arabic information retrieval: a comparative study," *International Journal of Computing & Information Sciences*, vol. 4, no. 3, pp. 119-133, 2006.
- [5] M. Ababneh, R. Al-Shalabi, G. Kanaan and A. Al-Nobani, "Building an Effective Rule-Based Light Stemmer for Arabic Language to Improve Search Effectiveness," *International Arab Journal of Information Technology (IAJIT)*, vol. 9, no. 4, pp. 368-372, 2012.
- [6] S. Khoja and R. Garside, "Stemming arabic text," Computing Department, Lancaster University, Lancaster, UK, 1999.
- [7] L. S. Larkey, L. Ballesteros and M. E. Connell, "Light stemming for Arabic information retrieval," *Arabic computational morphology*, vol. 38, pp. 221-243, 2007.
- [8] C. Cortes and V. Vapnik, "Support vector machine," *Machine learning*, vol. 20, no. 3, pp. 273-297, 1995.
- [9] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *10th European Conference on Machine Learning*, Chemnitz, 1998.
- [10] "Language Analysis," The Apache Software Foundation, 27 March 2014. [Online]. Available: <https://wiki.apache.org/solr/LanguageAnalysis>. [Accessed 7 May 2014].
- [11] "Solr Tutorial," The Apache Software Foundation, 2012. [Online]. Available: http://lucene.apache.org/solr/3_6_2/doc-files/tutorial.html. [Accessed 20 March 2014].