SPAM DETECTION REPORT

CSE 4403

SOFT COMPUTING

SUBMITTED BY

Nimrat Virk	208712572
Kingsley Okeke	209333402

TABLE OF CONTENT

Introduction ………………………………………………………. 3

Learning Data ……………………………………………….. 3

Data Pre-processing …………………………………………… 3

Stop Words …………………………………………... 3

Stemming …………………………………………….. 4

Inverted Index File …………………………………… 4

Attribute Selection ……………………………….... 4

Attribute Representation Value …………………………5

Arff Files ……………………………………………………6

Test Data ………………………………………………………6

Process Flow Diagram ……………………………………………..6

WEKA ……………………………………………………………..7

Testing Statistics ……………………………………………….. 7

Experimental Results ……………………………………………9

Conclusion ………………………………………………………11

Programs ………………………………………………………...11

INTRODUCTION

In today's globalized world, email is a primary source of communication. This communication can vary from personal, business, corporate to government. With the rapid increase in email usage, there has also been increase in the SPAM emails. SPAM emails, also known as junk email involves nearly identical messages sent to numerous recipients by email. Apart from being annoying, spam emails can also pose a security threat to computer system. It is estimated that spam cost businesses on the order of $100 billion in 2007. In this project, we use text mining to perform automatic spam filtering to use emails effectively. We try to identify patterns using Data-mining classification algorithms to enable us classify the emails as HAM or SPAM.

LEARNING DATA

The data used for this project was taken from the Spam Assassin public corpus website. It consists of two data sets: train and test. Each dataset contains a randomly selected collection of emails in plain text format, which have been labelled as HAM or SPAM. The training data is used to build a model for classifying emails into HAM and SPAM. The test data is used to check the accuracy of the model built with the training data. The training data set contains 400 emails with 283 ham and 117 spam emails. The test data contains 200 emails with 139 ham and 61 spam emails.

DATA PREPROCESSING

The emails in the learning data are in plain text format. We need to convert the plain text into features that can represent the emails. Using these features we can then use a learning algorithm on the emails. A number of pre-processing steps are first performed.

We convert the plain text files to files with one word per line. In this project, we look at emails just as a collection of words. So, to make it easier we convert each file into a list of words using Bourne Shell Scripts (*extractmultfiles.sh* and *extractwords.sh*).The output files are named as '*filename.words*.'

STOP WORDS

There are some English words which appear very frequently in all documents and so have no worth in representing the documents. These are called STOP WORDS and there is no harm in deleting them. Example: the, a, for etc. There are also some domain specific (in this case email) stop words such as mon, tue, email, sender, from etc. So, we delete these words from all the files using a Bourne Shell Script. These words are put in a file 'words.txt'. The shell script takes multiple files as an argument and then deletes all the stop words mentioned in the words.txt file.

STEMMING

The next step to be performed is stemming. Stemming is used to find a root of a word and thus replacing all words to their stem which reduces the number of words to be considered for representing a document. Example: sings, singing, sing have sing as their stem. In the project, we use JAVA implementation of Porter stemming algorithm which is slightly modified to meet our needs. The resultant files are named with an extension '*words_stemmed'*.

INVERTED INDEX FILE

In the next step, we create an inverted index file. This file has 3 columns – word, filename and frequency of word in the file. The file is sorted in alphabetical order of words. For this, we first create an inverted file for each individual file and then append them all together to build one inverted index file. The snapshot of the index file looks as:

```
abl spam_4_train 1
abl spam_55_train 1
abl spam_8_train 1
abli spam_47_train 1
abmv spam_111_train 3
abo ham_94_train 1
abound ham_6_train 1
abovement ham_173_train 1
abr ham_3_train 2
abreau ham_277_train 8
abreauj ham_277_train 2
abroad ham_193_train 2
absbottom ham_31_train 1
absenc ham_273_train 1
```

For this task, we create two Bourne script files. The script 'filename.sh' creates files with an extension '.out' which is an inverted index file for a single file. Then the script 'append.sh' appends all the '.out' files together and sorts them in alphabetical order of words.

ATTRIBUTE SELECTION

In the next step, we chose words to represent all the emails from the inverted index file. We use "Bag of words" method to select attributes, i.e. we use a set of words as attributes. We have to select some n specific words as all the documents contain thousands of unique words altogether and we cannot use all of these words for learning algorithm.

For this process, we use information-gain method. We use this method as it's a class dependent method, so on average it gives better accuracy. We use a JAVA program to calculate the gain value for each word using the formula-

$$Gain(w) = -\sum_{i=1}^{k} P(C_i)\log P(C_i) + P(w)\sum_{i=1}^{k} P(C_i \mid w)\log P(C_i \mid w) + P(\overline{w})\sum_{i=1}^{k} P(C_i \mid \overline{w})\log P(C_i \mid \overline{w})$$

In this case, P(C) refers to the probability of ham and spam class which can be calculated easily as we already have a predefined number of ham and spam emails. Next is P(w) which is the probability of the given word. It is given by the number of documents containing the word / total number of documents. P(C|w) ,means the number of documents which are labelled as spam or ham and which also contain the word w divided by the probability of the word w. So, looking at the inverted index file we can easily calculate all the values.

The JAVA program generates a text file with all the words and their respective gain values. For this project, we manually select top 10, 15 and 20 words with highest gain values and put them in separate files.

ATTRIBUTE VALUE REPRESENTATION

Once we have selected words, the next step is to represent the values for the selected attributes. We assign numerical values to them using 2 different methods –

a. Term Frequency

   Definition: TF = t (i,j)

   This gives the frequency of a word i in jth document.

b. TF-IDF (Term Frequency - Inverted Document Frequency)

   Definition: TF×IDF = t(i,j) × log(N/n)

   Here N = total number documents and n = number of documents that contain the respective word.

For each of this method, we use a JAVA program which creates a text file in a tabular format with the document name, attribute name and attribute value with the class value specifying it's a spam or ham email. A snapshot of this file looks as:

File name	listinfo	beenther	subscrib	remov	mailman	Error	Keyword	Bulk	Preced	archiv	class
ham_120_train	0	5	0	4	4	2	3	3	7	2	ham
spam_120_train	3	1	2	6	7	4	5	5	3	5	spam
ham_225_train	1	3	3	1	5	2	5	4	2	7	ham
ham_82_train	4	0	2	3	3	6	3	6	5	0	ham

ARFF FILES

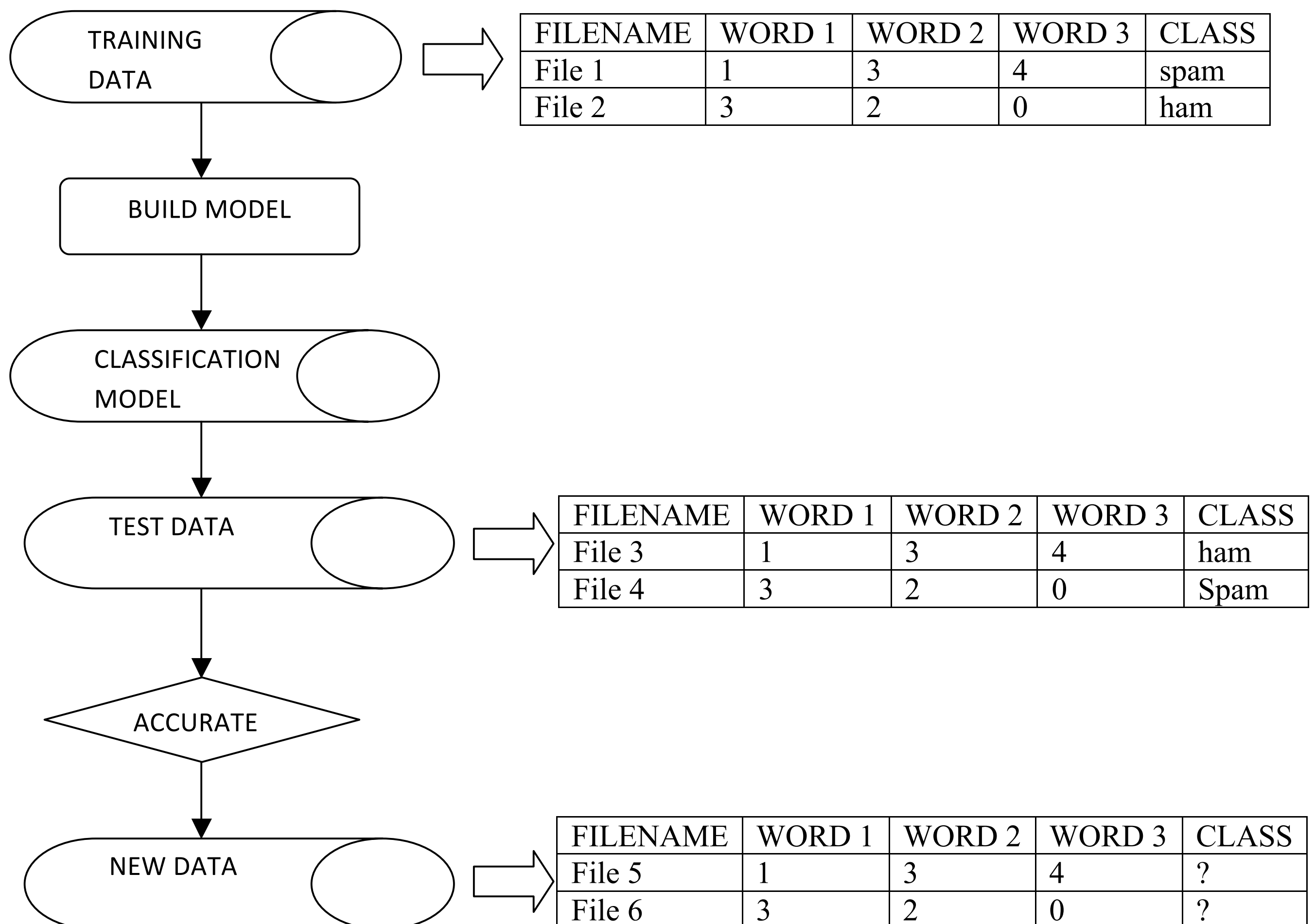
Once we have the file in the above mentioned format, we convert this file into .arff format to process it in WEKA. We use Excel to convert the file to CSV format and thereby adding the headers.

TEST DATA

At this point, we have .arff files for training data. We also need test files in the same format as training data so they are compatible. We repeat the all the above steps except 5 where we calculate Information Gain, and convert the test files to arff format also.

PROCESS FLOW DIAGRAM

The whole process of classification is depicted in the following diagram:



FILENAME	WORD 1	WORD 2	WORD 3	CLASS
File 1	1	3	4	spam
File 2	3	2	0	ham

FILENAME	WORD 1	WORD 2	WORD 3	CLASS
File 3	1	3	4	ham
File 4	3	2	0	Spam

FILENAME	WORD 1	WORD 2	WORD 3	CLASS
File 5	1	3	4	?
File 6	3	2	0	?

WEKA

The training data set arff files are given as an input to WEKA and different classifying algorithms such as Naïve Bayes, Bayes Net, Neural Network, k-NN and Decision Table were used to build models. Then, the test data set arff files were tested to find the classification accuracy of each model. The k-NN method gives the highest classification accuracy using k=3, cross-validation with 10 folds and 20 attributes. The accuracy came up to 94.5 %.

TESTING STATISTICS

The following data shows the testing statistics of different models on the test data. The data shown here is for the frequency method used for attribute value representation with 20 attributes and cross validation with 10 folds.

Naïve Bayes

Classification Accuracy: 81%

```
== Confusion Matrix ===
a   b   <-- classified as
  56   5 |   a = spam
  33 106 |   b = ham
```

Bayesian Network

Classification Accuracy: 80%

```
=== Confusion Matrix ===
 a   b   <-- classified as
  58   3 |   a = spam
  37 102 |   b = ham
```

Neural Network

Classification Accuracy: 93.5 %

```
=== Confusion Matrix ===
   a   b   <-- classified as
  56   5 |   a = spam
   8 131 |   b = ham
```

SMO

Classification Accuracy: 88.5 %

```
=== Confusion Matrix ===
  a   b   <-- classified as
  47  14 |   a = spam
   9 130 |   b = ham
```

k-NN (k=1)

Classification Accuracy: 91.25 %

```
=== Confusion Matrix ===
 a   b   <-- classified as
 102  15 |   a = spam
  20 263 |   b = ham
```

k-NN (k=3)

Classification Accuracy: 94.5 %

```
=== Confusion Matrix ===
 a   b   <-- classified as
  58   3 |   a = spam
   8 131 |   b = ham
```

Decision Table

Classification Accuracy: 88 %

```
=== Confusion Matrix ===
a   b   <-- classified as
  45  16 |   a = spam
   8 131 |   b = ham
```

C 4.5

Classification Accuracy: 90.5 %

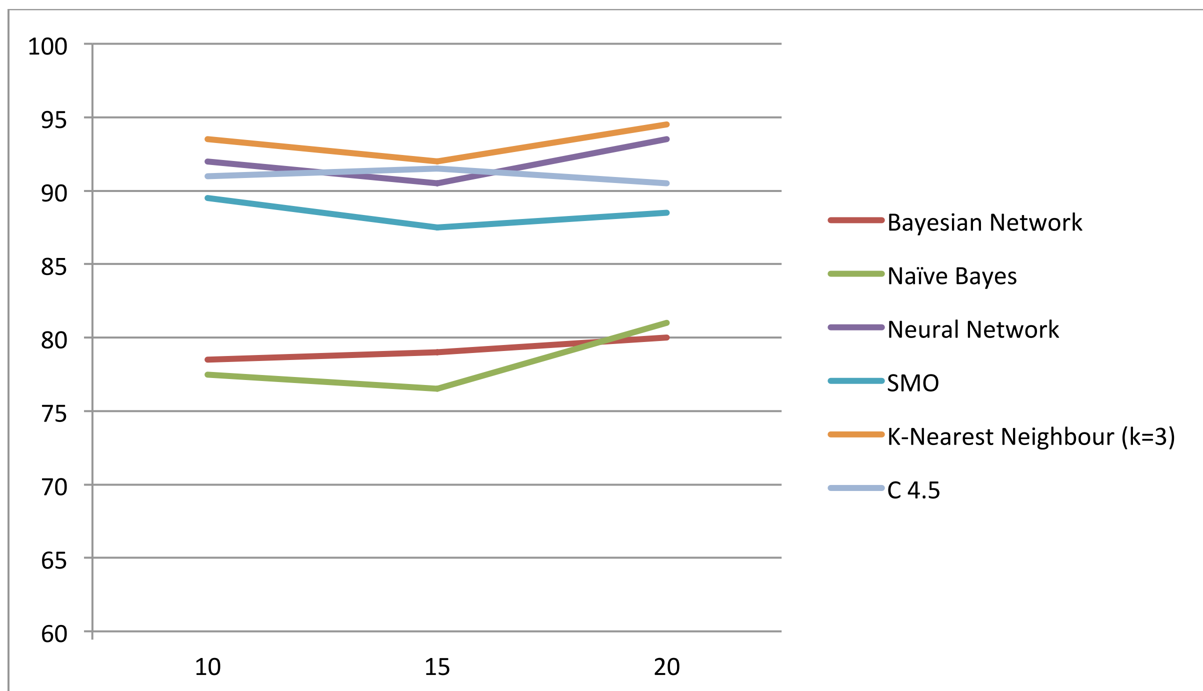
```
=== Confusion Matrix ===

   a   b   <-- classified as
  57   4 |   a = spam
  15 124 |   b = ham
```

EXPERIMENTAL RESULTS

From our experiments with different number of attributes and learning algorithms, we look at the classification accuracy for the built models on testing data and then compare the results -

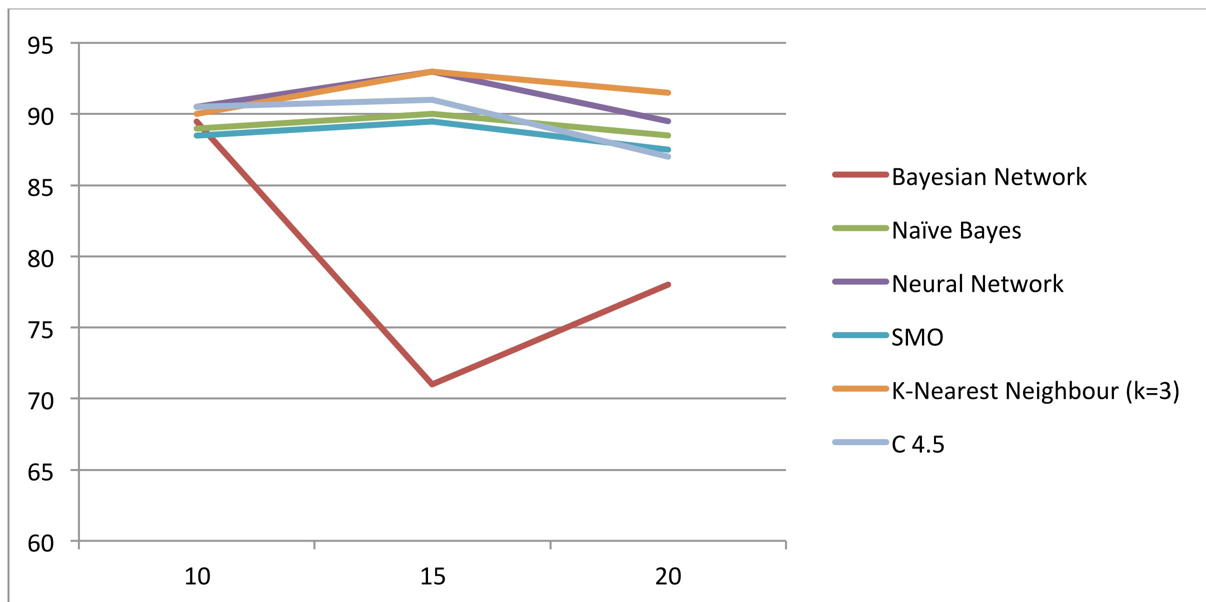
METHOD: FREQUENCY

No. of attributes	Bayesian Network	Naïve Bayes	Neural Network	SMO	K-Nearest Neighbour (k=3)	C 4.5	Decision Table
10	78.5	77.5	92	89.5	93.5	91	90
15	79	76.5	90.5	87.5	92	91.5	90
20	80	81	93.5	88.5	94.5	90.5	88



METHOD: TF-IDF

No. of attributes	Bayesian Network	Naïve Bayes	Neural Network	SMO	K-Nearest Neighbour (k=3)	C 4.5	Decision Table
10	89.5	89	90.5	88.5	90	90.5	88.5
15	71	90	93	89.5	93	91	88.5
20	78	88.5	89.5	87.5	91.5	87	83



The tables above show the classification accuracy for different classifiers using 10 -15 -20 attributes for both frequency and TF/IDF methods. We didn't experiment with more than 20 attributes as the data set contains only 400 documents, so according to us 20 is an optimal value.

CONCLUSION

Given a set of words, we used feature selection to obtain words which allo us to distinguish between spam and ham emails. We also compared the accuracy of various classifiers in predicting the class attribute. We see that k-NN method gives the highest classification accuracy no matter how many attributes are used and which method is used. Also, the value for k = 3

gives better result than k = 1. We also see that on average the accuracy improves as the number of attributes increase. It is possible that the accuracy may increase more than 94.5 % if we further increase the number of attributes.

PROGRAMS

For this project, we have written many Bourne Shell Scripts and JAVA programs. In this section we will explain how to compile and use these programs and the functionality of each of these.

stopwords.sh

This script takes multiple arguments and removes all the stop words from the file. We run this script by using following command:

stopwords.sh filename1 filename2

For our ease we provide the filenames at once using *'*.words'* parameter which takes all the words files in the current directory.

*stopwords.sh *.words*

Stemmer.java

This is Porter algorithm with some modifications. This program finds the root of the words and replace them with their root. In this program, we consider all the files in train and test data set and create new files with extension *'.stemmer'* in the respective folders. No arguments are provided for this program.

Compile Command: *javac Stemmer.java*

Run Command: *java Stemmer*

filename.sh & append.sh

After the stemmer program, we run filename.sh bourne shell script. This script takes multiple stemmed files created from the above program as argument and converts each file to an inverted file specifying all the words in the file with their frequency and filename. The script creates new files with extension *'.out'* in the current directory. We run the script using command:

*filename.sh *.stemmer*

After each file is used to create inverted file, we append all these '.out' files to create one inverted index file using the '*append.sh'* bourne shell script using the command-

*append.sh *.out*

The script generates a file called *'append.out'*.

InformationGain.java

This file is used to calculate the gain value of each unique word occurring in the *append.out* file created above for the train data. The program also uses 2 other java classes – Line.java and InvertNode.java. These programs need to be compiled first.

javac Line.java
javac InvertNode.java

Then we compile and run the InformationGain class.

javac InformationGain.java
java InformationGain

The program creates a file called 'InfoGain.txt' which outputs each word with their gain value in an increasing order of their gain values.

FileGenerator.java

This program generates a text file in a tabular format using the frequency method for attribute value representation. The files reads the manually created selected words file which has the top 10/15/20 words based on their gain values and creates a new file named 'a_frequency_b_attribute' which is used to create arff file later. Here a = {train,test} and b = {10,15,20}. Compile and run the program using commands:

javac FileGenerator.java
java FileGenerator

TIFDF.java

This program also generates a text file in the same format as above except using the TF-IDF method for attribute value representation. The new created files are named as 'a_tfidf_b_attributes' where a = {train,test} and b ={10,15,20}

Compile and run the program using commands:
javac TIFDF.java
java TIFDF

REFERENCES

- Porter, M (n.d). *The Porter Stemming Algorithm*. Retrieved March 2011, from http://tartarus.org/~martin/PorterStemmer

- Retrieved March 2011, from http://spamassassin.apache.org/publiccorpus/