Classifying Unsolicited Bulk Email (UBE) using Python Machine Learning Techniques

Sabah Mohammed[1], Osama Mohammed[1], Jinan Fiaidhi[1],
Simon Fong[2] and Tai hoon Kim[3]

[1]*Lakehead University, Ontario, Canada*
[2]*University of Macau, Macau, China*
[3]*Konkuk University, Korea*
E-mail {mohammed, omohamme,jfiaidhi}@lakeheadu.ca, ccfong@umac.mo, taihoonn@kku.ac.kr

Abstract

Email has become one of the fastest and most economical forms of communication. However, the increase of email users has resulted in the dramatic increase of spam emails during the past few years. As spammers always try to find a way to evade existing filters, new filters need to be developed to catch spam. Generally, the main tool for email filtering is based on text classification. A classifier then is a system that classifies incoming messages as spam or legitimate (ham) using classification methods. The most important methods of classification utilize machine learning techniques. There are a plethora of options when it comes to deciding how to add a machine learning component to a python email classification. This article describes an approach for spam filtering using Python where the interesting spam or ham words (spam-ham lexicon) are filtered first from the training dataset and then this lexicon is used to generate the training and testing tables that are used by variety of data mining algorithms. Our experimentation using one dataset reveals the affectivity of the Naïve Bayes and the SVM classifiers for spam filtering.

Keywords: *Spam Filtering; Machine Learning; Python*

1. Introduction

Spam is the popular term for unsolicited commercial e-mail (UCE) that's sent in bulk. One subset of UBE is UCE. The opposite of "spam", email which one wants, is called "ham". E-mail spam has steadily, even exponentially grown since the early 1990s. Not only is it a major annoyance but it has now become a real security threat as spam has become increasingly dangerous, delivering worms, viruses and spyware to unsuspecting users and scaring them into revealing personal information that can be used to commit fraud[1]. Spammers attempt to send emails in a few different ways including purchasing large lists of email addresses and using software to randomly generate an email address. The practice of spamming, and in particular the way in which e-mail addresses are collected or sold, raises a number of additional concerns. Techniques such as phishing have become increasingly sophisticated. Many anti-spam solutions have been proposed and a few have been implemented (e.g. filters, reverse lookups and cryptography) [1]. Unfortunately, these solutions do not prevent spam as much as they interfere with every-day email communications. Nonetheless, spam filters are still the most viable solution, even though this solution will cost Internet processing money, and take up CPU and processing time. Spam filtering is a method that eliminates unwanted

[1] http://www.zerospam.ca/uploads/ZEROSPAM:ViaRail_NewsRelease-1.pdf

emails before they reach your inbox. Spam filters remain popular because they can do a lot of the heavy lifting associated with removing bad emails. The sensitivity of the spam filters can be adjusted, giving the user a lot of control over what gets removed. Spam filtering can also be done based on attachments, data types, 'bad words,' and a whole lot more. Truly comprehensive solutions combine the best of multiple scanning concepts in order to offer a robust filtering solution to keep email safe. There are many different types of traditional filter systems including [2]:

- **Word lists**. Simple and complex lists of words that are known to be associated with spam. For example, "viagra".
- **Black lists and White lists**. These lists contain known IP addresses of spam and non-spam senders, respectively.
- **Hash-tables**. These systems summarize emails into pseudo-unique values. Repeated sightings of hash values are symptomatic of a bulk mailing.

However, new generations of spam filtering goes even further than traditional means, where they rely on statistical features of spam. These types of filters scan and analyze the entire email before making a decision on whether the email is spam or not. They compare the new email against a database/dataset of known spam emails and can match up whether the message is spam or not. This type of method generally pushes spam detection rates to 99% or higher. This type of filtering is very effective, even against phishing attempts [3]. However using some supervised machine learning mechanisms, a complete ranking of examples based on a training set of examples is not enough. What is needed in addition is a mechanism to calibrate the probability that each test example is a member of the class of interest [4]. The idea is that the scores output by a classifier need to be calibrated so that they can be understood. And, specifically, if you want to understand them as a probability that a document falls into a particular category then calibration gives you a way to estimate the probability from the scores.

Python has a good reputation when it comes to text processing and string manipulation where it is the case in spam filtering. The reasons behind choosing Python for email spam filtering and classification can simply be summarized as follows:

1. Python is an open source, cross-platform programming language and has a vibrant community as evidenced by the number of books and websites dedicated to it.
2. It is a popular web scripting language with very powerful text processing and visualization libraries (e.g. NLTK[2], Genism[3], Plex[4], Dictionary[5], Pattern[6], Scrapy[7], NetworkX[8]).
3. It has many mature machine learning libraries (e.g. Orange[9], PyML[10], Shogun[11], MDP[12], SciKit-Learn[13], MlPy[14], MILK[15], PyBrain[16], Elefant[17]).

[2] http://nltk.org/
[3] http://radimrehurek.com/gensim/
[4] http://packages.python.org/plex/index.html
[5] http://www.laurentluce.com/posts/python-dictionary-implementation/comment-page-1/
[6] http://www.clips.ua.ac.be/pages/pattern
[7] http://scrapy.org/
[8] http://networkx.lanl.gov/
[9] http://orange.biolab.si/
[10] http://pyml.sourceforge.net/
[11] http://www.shogun-toolbox.org/
[12] http://mdp-toolkit.sourceforge.net/
[13] http://scikit-learn.org/stable/

Python comes in two flavors: Python 2.x and Python 3.x. Python 3 is the way forward but Python 2 has some established projects that have not been updated to Python 3 yet. Python 2 programs can run in Python 3 interpreter with some modification but Python 2 and Python 3 are not compatible out of the box.

2. Building a Dictionary-Based Spam Classifier

The dictionary approach is usually the first thing programmers try when they write spam filters. There are many of such dictionaries available online that contains words and phrases that trigger spam (e.g. E-Commerce Spam Triggers List[18]). Python enable developers to easily create lexicons and dictionaries as it has its own dedicated data structure for building and retrieving dictionaries using key:value pairs. The following example illustrates how a dictionary can be created and retrieved in Python:

```
my_dict = {
      'key1': 'value1',
      'key2': 'value2',
      'key3': 'value3'
      }
my_dict['key1']
    # Out: 'value
  for item in my_dict:
  print item
#key3
#key2
     #key1
```

There are many Python built-in dictionary operations (e.g. delete, update, get, has_key)[19]. However, it is important to note that using a static lexicon for spam classification is a weak idea as the spammers language constantly changes. According to Kaspersky, [20] a spam lexicon/database that is not updated for a week, will cause the quality of spam detection to fall from around 90–95% to as low as 40–60%. Thus the practical goal is to update the spam lexicon frequently so to adapt and calibrate to the spammers language changes.

3. Calibrating the Spam Dictionary

The task of generating a flexible and adaptive spam lexicon may simply be tackled via two basic techniques involving stemming and calibration. Stemming requires a preprocessing of emails to convert all the words to their stems before testing their availability at the spam lexicon that contains only the common linguistic base form of the words. In Python, stemming is a straightforward process that can be achieved via importing one of the stemming libraries like the PyStemmer[21]. For English, PyStemmer uses the classic Porter stemming algorithm. However, stemming enable us to find an email containing "beneficiary" given the

[14] http://mlpy.sourceforge.net/
[15] http://packages.python.org/milk/
[16] http://pybrain.org/
[17] http://elefant.developer.nicta.com.au/
[18] http://www.responsemagic.com/pdf/triggers.pdf
[19] http://www.tutorialspoint.com/python/python_dictionary.htm
[20] http://www.kaspersky.com/images/whitepaper_kas_3_en.pdf
[21] http://pypi.python.org/pypi/PyStemmer/1.0.1

"benefit" stem word stored at the spam lexicon. Actually, spam emails often contain word derivatives and special characters to try to fool spam filters. These word derivatives and variations cannot be detected via stemming. For instance instead of using the word Viagra in a solicitation the offending email may use word variations such as V.I.A.G.R.A, VIAGRA!!, $VIAGRA$, and VlAGRAV to fool spam filtering software. Luckily, capturing word variations is a simple process when we use *regular expressions*[22] that are not available in the regular words lexicons. Today most of spam filters use a two-layer defense model[23] where regular expression is the first layer before using the spam lexicon. The use of regular expressions provides a technique for calibrating the spam lexicon to recognize words variations. Using such two layers spam filters one can catch obfuscations, such as "V. I. A. G. R. A" "viaaggggra", "via gra", "via.gra", "\/|AGRA, \/I/\GR/\", "v1aqra" and "vi@gr@", by using one representative regular expression for all these variations (e.g. "(?i)[v\\]+/?.?[i:1!\|]+.?[a@/]+\\?.?[gq]+.?r+.?[a@/]+\\?"). Certainly, you can create regular expressions to capture variations of tokens using many toolkits like the RegEx Coach[24]. Examples of some regular expressions are:

bulgary	(?i)\b[b8].?[uv].?[li17|].?[gq].?[a@/][\\]?r.?[yi1!:|]?
cialis	(?i)\bc.?[i1!:|].?[a@/][\\]?.?[li17|].?[i1!:|].?[s235$]
credit	(?i)\bc.?r.?[e3€].?d.?[i1!:|].?[t\+]
discount	(?i)\bd.?[i1!:|].?[s235$].?c.?[oQ0].?[uv].?n.?[t\+]?
money	(?i)\bm.?[oQ0].?n.?[e3€].?y
omega	\b[OoQ0].?[Mm].?[Ee3€].?[Ggq].?[Aa@/][\\]?
rolex	\b[Rr].?[OoQ0].?[LlIi17|].?[Ee3€].?[Xx]
valium	(?i)[v\\]+/?.?[a@/]+\\?.?[li17|]+.?[il1!:|]+.?[uv]+.?m+\b

Basically, Python deals with regular expressions using the " re" package[25] which provides excellent support for regular expressions, with a modern and complete regex flavor. The re package provides many useful operations[26] such as *search*, *match*, *finditer*, *replace*, *split and group*. Here is a simple example which demonstrates the use of *finditer* to find all the occurrences of the word "the" and prints "the" and the following word. It also prints the character position of each match using the *MatchObject's start()* method:

```
html = urllib2.urlopen('http://www.smartpassiveincome.com/my-first-online-business/
').read()
regularexpression_pattern = r'\b(the\s+\w+)\s+'
regex = re.compile(regularexpression_pattern, re.IGNORECASE)
for match in regex.finditer(html):
   print "%s: %s" % (match.start(), match.group(1))
```

However, if we want to use the "re" package for matching multiple regular expressions, one need to join them all together into one big regular expression, which is very complicated task to achieve. For this purpose, developers use the Plex package. Plex is a Python module for constructing lexical analyzers, or scanners. Tokens are defined by regular expressions, and each token has an associated action, which may be to return a literal value, or to call an arbitrary function. The following is an example of creating a simple lexicon with Plex:

[22] http://www.policypatrol.com/
[23] http://www.itsecurity.com/whitepaper/five-essential-steps-to-safer-email-ironport/
[24] http://weitz.de/regex-coach/
[25] http://docs.python.org/library/re.html
[26] http://www.regular-expressions.info/python.html

```
from plex import *
lexicon = Lexicon([
   (Str("free"),      "SPAM"),
   (Str("Credit"),        "SPAM"),
   (Str("offer"),       "SPAM"),
   (Rep1(Any(" \t\n")), IGNORE)
(Repl(?i)\b[b8].?[uv].?[li17\|].?[gq].?[a@/][\\]?r.?[yi1!:\|]?])
(Repl (?i)\bc.?[i1!:\|].?[a@/][\\]?.?[li17\|].?[i1!:\|].?[s235$]) )]
```

It was found that non-lexicon words that can be identified by regular expressions to constitute around 66% of the total number of lexis in the body of the corpus [8].

4. Identifying Spam Trigger Words from a Training Corpus

Identifying spam trigger words requires the use of a spam training corpus[27] where emails are already categorized as either spam or ham. Each message in the training corpus must then be tokenized and stemmed. The resulting tokens then are used to update the frequency counts in a database (usually called HistoricDataset). The HistoricalDataset consists of a hash table that keeps track of the number of times a particular token has appeared in spam and ham messages. The database must also keep track of the total number of spam and ham messages. Once the token counting has been completed, the frequency counts within the HistoricalDataset can be used to classify additional messages. Paul Graham [7] suggested a technique for classifying tokens as spam or ham in any given new email based on the frequencies of tokens at the training ham or spam corpus using a probability measure. The probability (P) values range from 0.0, for a word that is a very strong indicator of ham, to 1.0, for a word that is a very strong indicator of spam. To compute the probability, let SH and IH be the total number of appearances in spam and ham emails for a given token. Morover, let TS and TI be the total number of spam and ham messages in the HistoricalDataset. Then P, the value of the word being ham or spam can be calculated as follows:

$$P = (SH /TS)/ (SH /TS + IH /TI)$$

If the token has no spam occurrences or no ham occurrences, it is given a value of 0.1 or 0.9 respectively. When new email arrives, it is scanned into tokens, and the most interesting number of tokens (e.g. fifteen tokens), where interesting is measured by how far these words probabilities are from the neutral .5. However, Graham used a simplified statistical approach to classify a given email based on the interesting words probabilities using a combined probability measure:

Graham Combined Probability =

$$(abc...n)/(abc...n + (1- a)(1- b)(1-c)...(1- n))$$

Where a,b,…,n are the probabilities of the interesting words.

Such approach assumes all emails as having a structured pattern of interesting tokens where the combined probability will be sufficient for classification. However, in reality emails do not have a structured pattern of interesting tokens. In the next section, we are presenting an approach that computes the probability of each token from an email corpus of training data, then to generate a spam-ham lexicon based on the highest probabilities (e.g. the highest sixty five interesting tokens). Finally to use any number of testing emails and set of

[27] http://plg1.cs.uwaterloo.ca/cgi-bin/cgiwrap/gvcormac/foo

data mining algorithms to classify the testing emails as spam or ham according to the best matching patterns from the training data. Figure 1 provides a block diagram involving the major operations used in our approach.

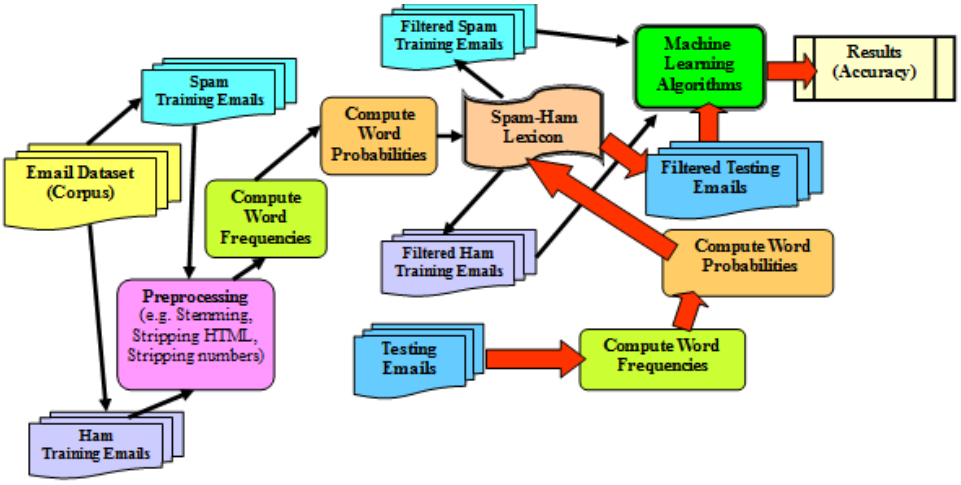

Figure 1. Machine Learning Approach for Emails Classification

Basic to all operations used in Figure 1 is the frequency-count function. The following is a code snapshot for this function:

```
def  spam_fequency (dictionary_file_path, email_file_path):
        dictionary = open(dictionary_file_path)
        dictionary_words = dictionary.read()
        email = open(email_file_path)
        email_words = email.read()
        spam_word_count = 0
        for each word in dictionary_words:
                if(email_words.contain(word)):
                        spam_word_count++
        return spam_word_count
```

However, it is important to note that in our approach we are generating a small lexicon form each provided training dataset using words frequencies and word probabilities measures. Words with highest probabilities (either near 1 for Spam or near 0 for Ham) will be included at this lexicon that we call spam-ham lexicon. In our lexicon, we choose to have 32 highest probability words for the ham as well as 32 highest probability words for the spam. Using this lexicon the training dataset and the testing dataset are scanned for the availability of words at the spam-ham lexicon. If a word is available it is assigned the value of 1 otherwise 0 at the filtered training or testing emails file. The filtered emails files will be at end representing two tables that can be feed to the data mining programs. Figures 2 and 3 illustrate the CVS format of the training and testing filtered emails files. Note these figures only show a snapshot of the filtered data (column and row wise).

1	family	cash	warranty	news	law	Spam/Ham
2	discrete	discrete	discrete	discrete	discrete	discrete
3	0	0	0	0	0	Spam
4	0	0	0	0	0	Spam
5	0	0	0	0	0	Spam
6	0	0	0	0	0	Spam
7	0	0	1	0	0	Spam
8	0	0	0	1	0	Spam
9	1	0	1	0	0	Spam
10	0	0	0	0	0	Spam
40	0	0	0	1	0	Regular
41	1	0	0	0	0	Regular
42	0	0	0	0	0	Regular
43	0	0	0	0	0	Regular
44	0	0	0	0	0	Regular
45	0	0	0	0	0	Regular
46	0	0	0	0	0	Regular
47	0	0	0	0	0	Regular

Figure 2. The Structure of Spam-Ham Training Filtered Emails

1	family	cash	warranty	news	business	priority	law	Spam/Ham
2	discrete	discrete	discrete	discrete	discrete	discrete	discrete	discrete
3	0	1	0	1	1	0	0	
4	0	1	0	1	1	0	0	
5	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	
9	0	0	0	1	0	0	0	
10	0	0	0	0	0	0	0	
80	0	0	0	1	1	0	0	
81	0	0	0	1	0	0	0	
82	0	0	0	0	1	0	0	
83	0	0	0	0	0	1	0	
84	0	0	0	1	1	0	0	
85	0	0	0	0	0	1	0	

Figure 3. The Structure of Spam-Ham Testing Filtered Emails

In this article we used one widely used dataset called Email-1431created by F.A. Nielsen[28] to illustrate and test our machine learning approach. Email-1431 consists of texts from 1431 emails in three categories: jobs (272), conferences (370), and spams (789). The conferences and jobs emails are regular (non spam) emails but they contain several spam trigger words which can easily be filtered by several spam filters as spam. Since conferences and jobs emails are regular emails we decided to take jobs as a representative sample of ham emails. However, in Email-1431 the number of spam emails is way higher than the emails in jobs and this will affect the training of spam classifiers. For this purpose we decided to take the first 272 emails from the spam set so to provide equal opportunity to train the classifiers. For the training purposes we divided the jobs and the first 272 spam into two halves, one for training and the other for testing.

[28] http://www.imm.dtu.dk/~rem

5. Machine Learning Techniques for Email Clasification

There are several general purpose machine learning packages in Python (e.g. Orange, scikits.learn), however, the orange package seems to take the lead over the other packages [8]. Orange is a library of C++ core objects and routines that includes a large variety of standard and not-so-standard machine learning and data mining algorithms, plus routines for data input and manipulation. Orange is also a scriptable environment for fast prototyping of new algorithms and testing schemes. It is a collection of Python-based modules that sit over the core library and implement some functionality for which execution time is not crucial and which is easier done in Python than in C++. This includes a variety of tasks such as pretty-print of decision trees, attribute subset, bagging and boosting, and alike. Orange is also a set of graphical widgets that use methods from core library and Orange modules and provide a nice user's interface. Widgets support signal-based communication and can be assembled together into an application by a visual programming tool called Orange Canvas.

In Orange There are different methods for data classification such as decision trees (Tree), rule based methods (Rules), Naïve-Bayes (NB), Support Vector Machine (SVM), k-nearest neighbor (KNN) and so forth. The comparison of the classifiers and using the most predictive classifier is very important. For this purpose we tested these classifiers using the following Python code:

```
import Orange
from Orange.classification import svm
test_data_size = 25 + 67
def print_classification(classifier, test_data):
    for i in range(test_data_size):
        c = classifier(test_data[i])
        print "Email ", i, " classified as ", c
# set up the classifiers
train_data = Orange.data.Table("spamTrainingTable.csv")
test_data = Orange.data.Table("spamTestingTable.csv")
bayes_learner = Orange.classification.bayes.NaiveLearner()
bayes_classifier = bayes_learner(train_data)
cn2_rule_learner = Orange.classification.rules.CN2Learner()
cn2_rule_classifier = cn2_rule_learner(train_data)
knnLearner = Orange.classification.knn.kNNLearner()
knnClassifier = knnLearner(train_data)
svm_learner = svm.SVMLearner()
svm_classifier = svm_learner(train_data)
#c45 = Orange.classification.tree.C45Learner(train_data)
tree_learner = Orange.classification.tree.TreeLearner()
tree_classifier = tree_learner(train_data)
print_classification(tree_classifier, test_data)
```

Figure 4 illustrate the results of comparing five notable classifiers for accurately detecting spam or ham emails for the dataset provided by the Email-1431 dataset. The comparison counts the accuracy of detecting correctly the ham or spam emails for intervals of ten emails.

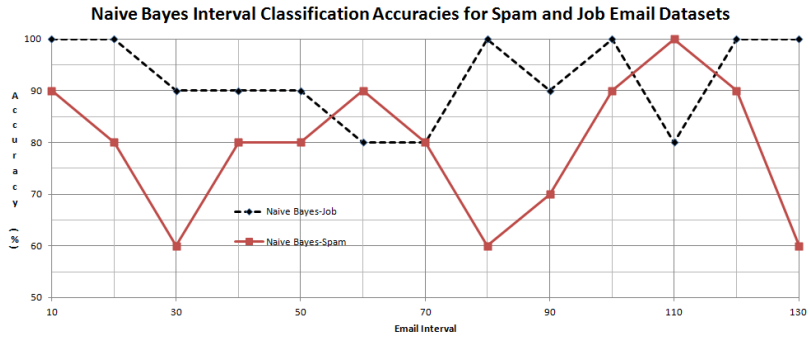

Figure 4-(a)

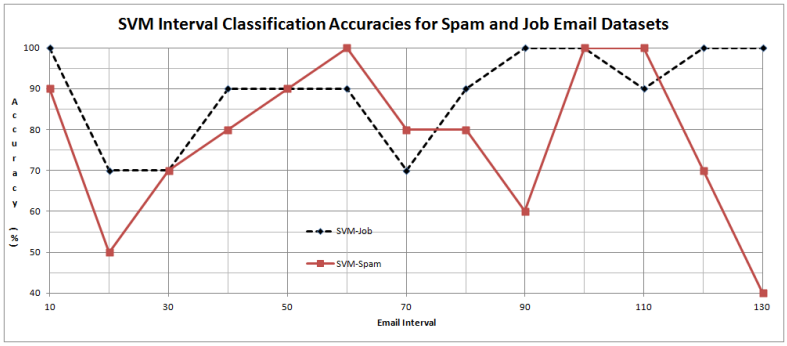

Figure 4-(b)

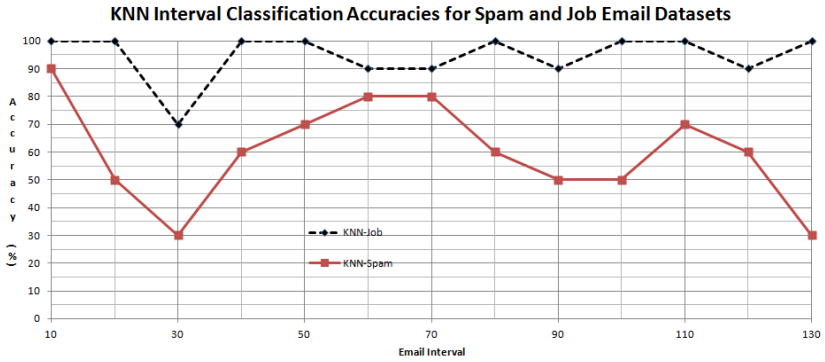

Figure 4- (c)

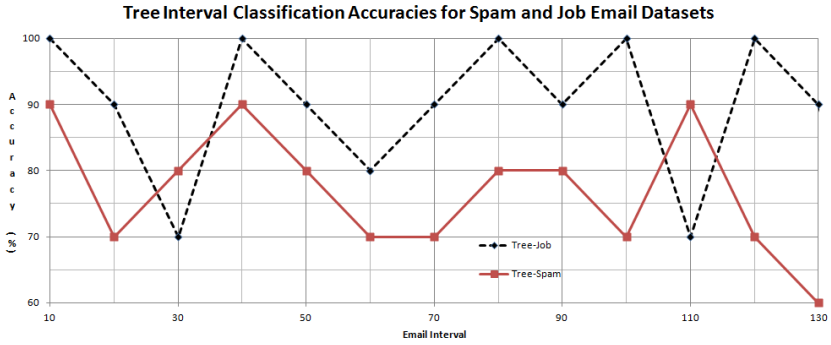

Figure 4- (d)

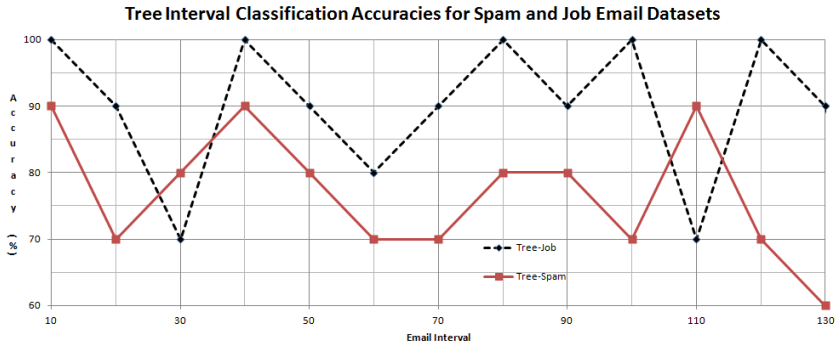

Figure 4- (e)

Figure 4. The Accuracy of Each Classifier in Detecting Spam and Ham Emails

Figures 5 and 6 provides a comparative overall view for detecting ham (i.e. job emails) and spam emails.

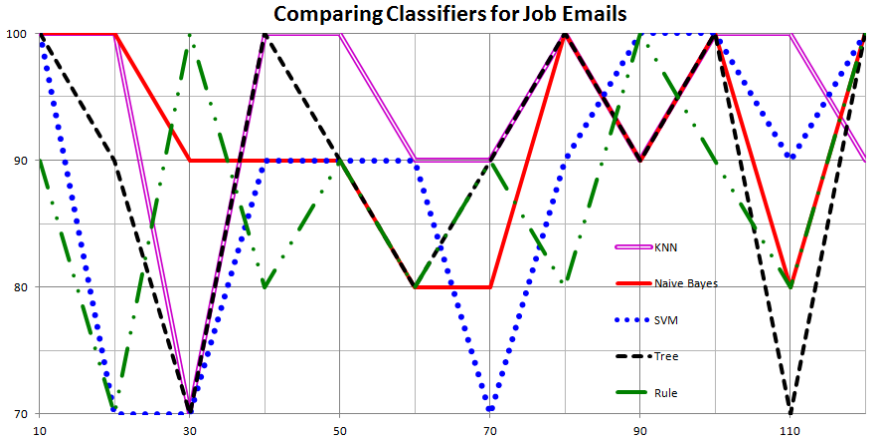

Figure 5. Comparing Classifiers for Detecting Ham Emails

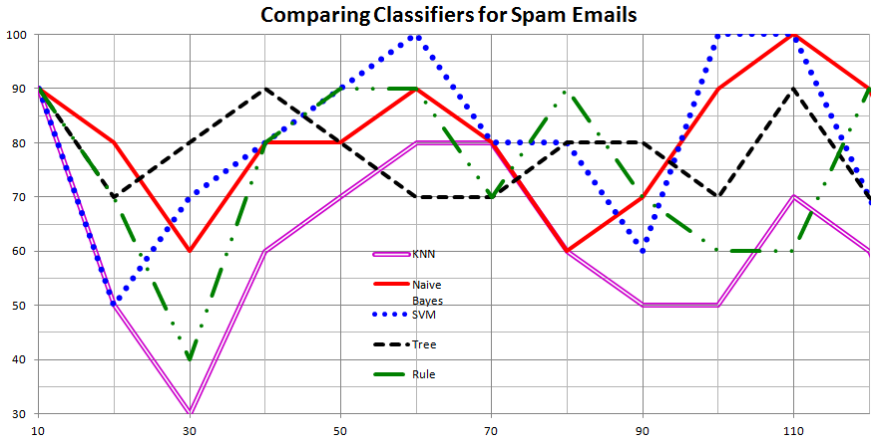

Figure 6. Comparing Classifiers for Detecting Spam Emails

However, Table 1 provides another comparative result between the five classifiers.

Table 1. The Overall Accuracy of the Five Classifiers

Machine Learning Algorithm	Accuracy Detecting Ham (job)	Accuracy Detecting Spam	Average Accuracy for both Ham and Spam
NB	90.71	78.57	85.96
SVM	87.86	77.14	82.96
KNN	92,86	66.71	77.04
Tree	87.86	75.71	82.59
Rules	86.43	75.00	81.11

Table 1 results reveal that the best performing classifiers in detecting both spam and ham emails are the Naïve Bayes and the SVM. However, all the five classifiers have high accuracy in detecting the job emails (i.e. ham) as regular emails. This result shows clearly the effect of the training dataset as well as the dynamic spam-ham dictionary generated from the training dataset.

6. Conclusions and Experimental Results

This article introduced an approach for spam filtering that starts by generating a spam-ham lexicon from a given training data and uses this lexicon to filter the training and testing tables that can be used by variety of data mining algorithms. Previous research of spam filtering that uses machine learning techniques utilize a static list of spam trigger words [9, 10] which may work well for group of emails and may not work for other groups. Using Python we demonstrated that it is a powerful language that can be used for emails text mining as it have very rich natural language and data mining packages. Using the Nielson Email-1431 dataset we found that the most effective spam classifiers to be the Naïve Bayes and the SVM. Although the Email-1431 dataset has been widely used by researchers studying the issue of spam filtering [11, 12, 13], we are intending to use wider standard dataset like the one provided by the NIST TREC[29].

Acknowledgements

The third author would like to thank NSERC for their partial support to this research.

References

[1] N. Krawetz, "Anti-Spam Solutions and Security", The Symantic Blog, **(2010)** November 2, http://www.symantec.com/connect/articles/anti-spam-solutions-and-security.
[2] D. Mertz, "Spam filtering techniquesSix approaches to eliminating unwanted e-mail", IBM Developer Work Megazine, **(2002)** September 1, http://www.ibm.com/developerworks/linux/library/l-spamf/index.html.

[29] http://plg.uwaterloo.ca/~gvcormac/spam/

[3] T. S. Guzella and W. M. Caminhas, "A review of machine learning approaches to Spam filtering", Int Journal of Expert Systems with Applications, vol. 36, **(2009)**, pp. 10206-10222.
[4] B. Zadrozny and C. Elkan, "Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers", ICML 2001 Proceedings of the Eighteenth International Conference on Machine Learning, **(2001)**, pp. 609-616.
[5] P. Pantel and D. Lin, "SpamCop: a spam classification and organization program", In Learning for Text Categorization – Papers from the AAAI Workshop, **(1998)**, pp. 95–98, Madison Wisconsin. AAAI Technical Report WS-98-05.
[6] M. Sahami, S. Dumais, D. Heckerman and E. Horvitz, "A Bayesian approach to filtering junk e-mail", In Learning for Text Categorization – Papers from the AAAI Workshop, **(1998)**, pp. 55–62, Madison Wisconsin. AAAI Technical Report WS-98-05.
[7] P. Graham, "A Plan for Spam", **(2002)** August, http://paulgraham.com/spam.html.
[8] J. Demsar and B. Zupan, "From Experimental Machine Learning to Interactive Data Mining", White Paper, **(2010)**, http://www.celta.paris-sorbonne.fr/anasem/papers/miscelanea/InteractiveDataMining.pdf.
[9] D. K. Renuka, *et al.*, "Spam Classification Based on Supervised Learning Using Machine Learning Techniques", IEEE 2011 International Conference on Process Automation, Control and Computing (PACC), **(2011)** July 20-22.
[10] M. W. Berry, J. Kogan and E. P. Jiang, "Content-Based Spam Email Classification using Machine-Learning Algorithms", Chapter 3, In Text Mining: Applications and Theory, Copyright © 2010 John Wiley & Sons, Ltd., **(2010)**.
[11] N. Kang C. Domeniconi and D. Barbar á, "Categorization and Keyword Identification of Unlabeled Documents", Fifth IEEE International Conference on Data Mining, **(2005)** November 27-30.
[12] R. E. Madsen, S. Sigurdsson and L. K. Hansen, "Enhanced Context Recognition by Sensitivity Pruned Vocabularies", Proceedings of 17th International Conference on Pattern Recognition (ICPR 2004), vol. 2, **(2004)**, pp. 483-486.
[13] C. Domeniconi and M. Al-Razgan, "Weighted Cluster Ensembles: Methods and Analysis", Technical Report ISE-TR-07-06, Department of Computer Science, George Mason University, **(2007)** December, http://cs.gmu.edu/~tr-admin/papers/ISE-TR-07-06.pdf.

Authors

Sabah Mohammed

Professor of Computer Science at Lakehead University of Canada. Professional Engineer of Ontario and Adjunct research Professor with University of Western Ontario. Research is on Web Intelligence and Medical Informatics.

Osama Mohammed

Masters in Software Engineering from Lakehead University of Canada. Engineer in Training with Professional Engineers Ontario. Research on medical diagnosis recommenders and data mining.

Jinan Fiaidhi

Professor of Computer Science and Graduate Coordinators at Lakehead University of Canad. Professional Engineer of Ontario and Adjunct research Professor with University of Western Ontario. Research is on Collaborative Learning and Machine Learning.

Simon Fong

Professor with the Department of Computer and Information Science at Macau University of China. Research is on Data Analytics, E-Commerce technology, Business Intelligence and Data-mining.

Tai hoon Kim

Professor of Computer Science, Konkuk University, Korea. Also with GVSA and UTAS, Australia. Vice President of SERSC. Research is on Computer Security.