DETECTING SPAM EMAILS

STUDENT DETAILS

NAME : TAMIZHSELVAN V
NM ID : au513521105018
DEPT : EEE
COLLEGE : AMCET

ABSTRACT

Electronic spamming is the use of electronic messaging systems to send an unsolicited message (spam), especially advertising, as well as sending messages repeatedly on the same site. While the most widely recognized form of spam is email spam, the term is applied to similar abuses in other media: instant messaging spam, Usenet newsgroup spam, Web search engine spam, spam in blogs, wiki spam, online classified ads spam, mobile phone messaging spam, Internet forum spam, junk fax transmissions, social spam, spam mobile apps, television advertising and file sharing spam,[6] The source and identity of the sender is anonymous and there is no option to cease receiving future e-mails. Span e-mail is usually sent by spam bot, which is program that continually sends out email. Often spammers will create a virus that install a spam bot into unsuspecting users computers and will use there internet connection and computer to send spam.

Spam e-mail are message randomly sent to multiple addressees by all sorts of groups, but mostly lazy advertisers and criminals who wish to lead you to phishing sites. The sites attempt to steal your personal, electronic, and financial information.

Bayesian Spam Detection/ Filtering is used to detect spam in an email. A Bayesian network is a representation of probabilistic relationships. This paper will show that Bayesian filtering can be simply implemented for a reasonably accurate text classifier and that it can be modified to make a significant impact on the accuracy of the filter.

List of Figures

List of Abbreviations

CSS – Cascading Style Sheet

HTML – Hyper Text Markup Language

IDF – Inverse Document Frequency

ISP – Internet Service Provider

IT – Information Technology

JS – Java Script

PHP – Hypertext Preprocessor

TF – Term Frequency

URL – Uniform Research Locator

DFD – Data Flow Diagram

Table of Contents

Chapter: 1

Introduction

1.1 Background

Major approaches adopted towards spam filtering include text analysis, white and black lists of domain names and community based approaches. Text analysis of contents of mails is a widely used approach towards the spams. Many solutions deployable on server and client sides are available. Naive Bayes is one of the most popular algorithms used in these approaches. Spam Bayes and Mozilla Mail spam filter are examples of such solutions. But rejecting mails based on text analysis can be serious problem in case of false positives. Normally users and organizations would not want any genuine e-mails to be lost. Black list approach has been one of the earliest approaches tried for the filtering of spams. The strategy is to accept all the mails except the ones from the domain/e-mail ids. Explicitly blacklisted. With newer domains entering the category of spamming domains this strategy tends to not work so well. White list approach is the strategy of accepting the mails from the domains/addresses explicitly white listed and put others in a less priority queue, which is delivered only after sender responds to a confirmation request sent by the spam filtering system.

1.2 Problem Statement

Spamming is one of the major attacks that accumulate the large number of compromised machines by sending unwanted messages, viruses and phishing through emails. We have chosen this project because now days there are lot of people trying to fool you just by sending you fake e-mails like you have won 1000 dollars, this much amount is deposited in your account once you open this link then they will track you and try to hack your information. Sometimes relevant e-mails are considered as spam emails[4].

- Unwanted email irritating Internet consumers.
- Critical email messages are missed and/or delayed.
- Consumers change ISP's all the time looking for consistent email delivery.
- Loss of Internet performance and bandwidth.

- Millions of compromised computers.
- Billions of dollars lost worldwide.
- Identity Theft.
- Increase in Worms and Trojan Horses.
- Spam can crash mail servers and fill up hard drives.

1.3 Objective

The objective of classification of Spam e-mails are:

- To classify the email into spam and non spam.

1.4 Scope of the project

It considers a complete message instead of single words with respect to its organization. It can be referred as the intelligent approach due to its message examining criteria. It provides sensitivity to the client and adapts well to the future spam techniques. Even if the spam word is slightly modified, this algorithm still succeeds and notices the spam content.

Even though this frame work is accessible for the URL groups embedded in email, it lacks in action taking for IT security groups. The degree of business in the real time is one more major drawback that is faced by this frame work. It does not provide any clarity about how well it is performed in a real time for the spam campaigns. Spammers can easily develop techniques to meet the preventive measures of Auto RE framework like making legitimate domains fall into the list of illegitimate emails. The results that are obtained do not provide any aggregate view of the large groups of emails. Moreover, it does not let the network administrator to have an online monitoring system across the network

1.5 Applications

- It also reduce Network Resource Costs.
- It reduces IT Administration Costs.
- It reduces Legal Liability Risk.
- It increases Security and Control.

Chapter: 2

Literature review

Spam prevention is often neglected, although some simple measures can dramatically reduce the amount of spam that reaches your mailbox. Before they are able to send you spam, spammer's obviously first need to obtain your email address, which they can do through different routes. Our project is design to reduce spam or unwanted e-mail.

2.1 Document Preprocessing

2.1.1 Tokenization

Tokenization is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens. The list of tokens becomes input for further processing such as parsing or text mining. Tokenization is useful both in linguistics (where it is a form of text segmentation), and in computer science, where it forms part of lexical analysis.

Typically, tokenization occurs at the word level. However, it is sometimes difficult to define what is meant by a "word". Often a tokenizer relies on simple heuristics, for example:

- All contiguous strings of alphabetic characters are part of one token; likewise with numbers.
- Tokens are separated by whitespace characters, such as a space or line break, or by punctuation characters.
- Punctuation and whitespace may or may not be included in the resulting list of tokens.

In languages such as English (and most programming languages) where words are delimited by whitespace, this approach is straightforward. However, tokenization is more difficult for languages such as Chinese which have no word boundaries. Simple whitespace-delimited tokenization also presents difficulties when word collocations such as New York should be treated as one token. Some ways to address this problem are by developing more complex heuristics, querying a table of common collocations, or fitting the tokens to a language model that identifies collocations in a later processing step.

For example:

Input: “Friends, Romans and Countrymen”

- Output: Tokens
- Friends
- Romans
- Countrymen

A token is an instance of a sequence of characters

2.1.2 Lemmatization

Lemmatization in linguistics, is the process of grouping together the different inflected forms of a word so they can be analysed as a single item.

In computational linguistics, lemmatisation is the algorithmic process of determining the lemma for a given word. Since the process may involve complex tasks such as understanding context and determining the part of speech of a word in a sentence (requiring, for example, knowledge of the grammar of a language) it can be a hard task to implement a lemmatizer for a new language.

Lemmatization is also used in natural language processing and many other fields that deal with linguistics in general. It also provides a productive way to generate generic keywords for search engines or labels for concept maps.

Lemmatisation is closely related to stemming. The difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech. However, stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications.

For instance:

1. The word "better" has "good" as its lemma. This link is missed by stemming, as it requires a dictionary look-up.
2. The word "walk" is the base form for word "walking", and hence this is matched in both stemming and lemmatisation.

3.The word "meeting" can be either the base form of a noun or a form of a verb ("to meet") depending on the context, e.g., "in our last meeting" or "We are meeting again tomorrow". Unlike stemming, lemmatisation can in principle select the appropriate lemma depending on the context.

2.1.3 Removal of Stop Word

Sometimes, the extremely common word which would appear to be of very little value in helping select documents matching user need are excluded from the vocabulary entirely. These words are called stop words and the technique is called stop removal.

The general strategy for determining a stop list is to sort the terms by collection frequency and then to make the most frequently terms, as a stop list, the members of which are discarded during indexing.

Some of the examples of stop-word are: a, an, the, and, are, as, at, be, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with etc.

2.1.4 Document Representation

In order to reduce the complexity of the documents and make them easier to handle, the document have to be transformed from the full text version to a document vector which describes the contents of the document. A definition of a document is that it is made of a joint membership of terms which have various patterns of occurrence.

Vector space model or term vector model is the popular algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, such as, for example, index terms. It is used in information filtering, information retrieval, indexing and relevancy rankings. Using vector space model documents are represented using term frequency (tf), inverse document frequency (idf) or tf- idf weighting scheme.

Chapter: 3

Requirement Analysis and Feasibility Study

3.1 Requirement Collection and Specification:

The Primary data were collected through our supervisor from the related corpus. The secondary data were collected from research papers and some test data were sampled data made by ourselves to check for the expected output.

The Primary data were collected through our supervisor from the related corpus. The secondary data were collected from research papers and some test data were sampled data made by ourselves to check for the expected output.

- Corpus is maintained
- The user should be able to receive the genuine mails.

3.2 Functional Requirements

The main function of this project is to classify the e-mails which is done by first taking out the feature vector extraction which involves first taking out whether the word is a spam or not by representing in the form of matrix.

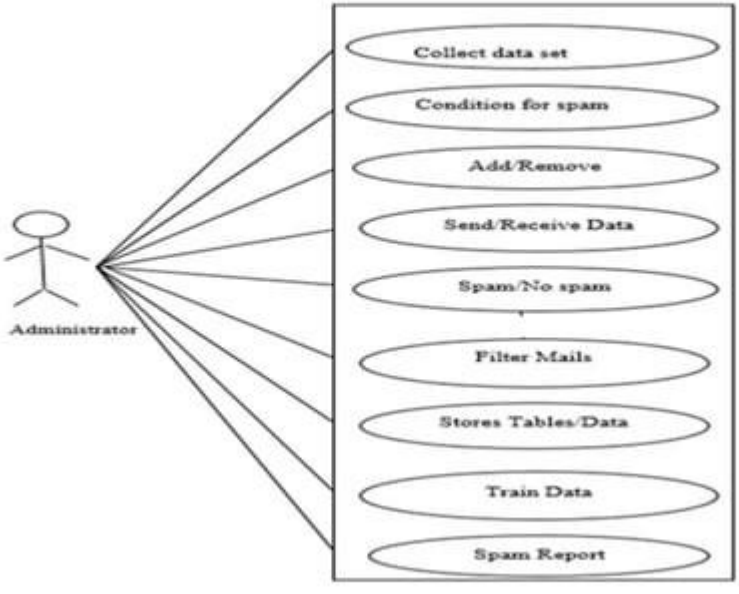


Fig 3.3.1: Use case diagram

3.3 Non Functional Requirements

- Ensures high availability of email data here datasets.
- User should get the result as fast as possible.
- It should be easy to use i.e., user is just required to type the words and click then the result is displayed or user is just required to enter a pair of reasonable sentence.

3.4 Data Flow Diagram

It is a directed graph where nodes represents processing activity and arc represent data items transmitted between processing nodes.

Level 0

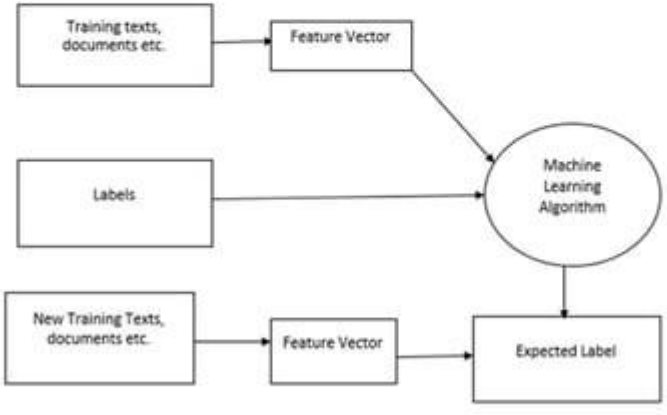


Fig 3.4.1: DFD- level 0

Level 1

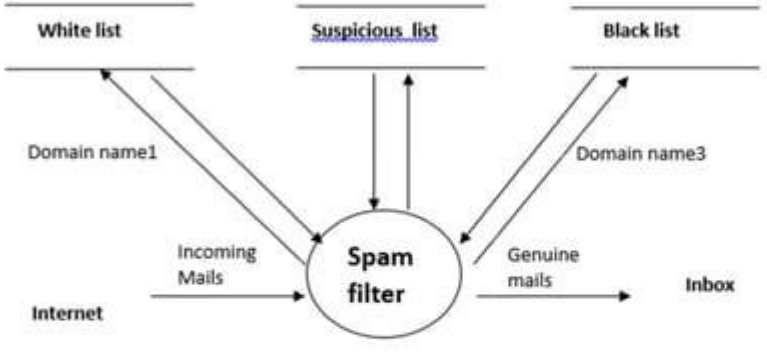


Fig 3.4.2: DFD-level 1

Level-2

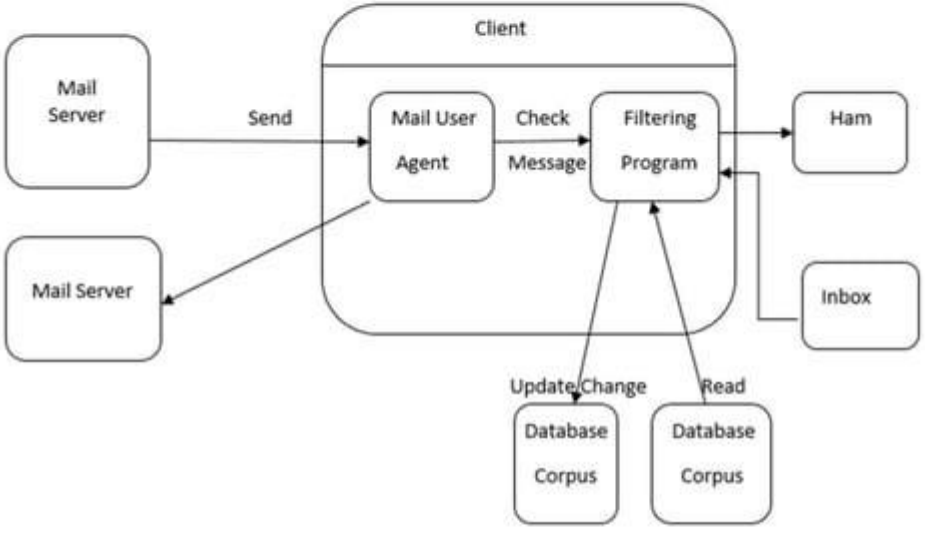


Fig 3.4.3: DFD-level 2

3.5 Feasibility Study

The feasibility of the system has been studied from the various aspects like whether the system is feasible technically, operationally and economically. The present technology is found to be sufficient to meet the requirements of the system. This system is believed to work well when it is developed and installed. Hence, operational feasibility is achieved. Since the requirements for the project are easily available, ware headed with the intention to use the available resources to fulfill the system requirement. The detail feasibility study we conducted is mentioned below:

3.6.1 Technical Feasibility

The technology needed for the proposed system that we are going to develop is available. We can work for the project is done with current equipment existing tools like python .We can develop our system still using this technology if needed to upgrade. In future, if we want to use new technology like android app of our system it is possible. Hence, the system that we are going to develop will successfully satisfy the needs of the system for technical feasibility

3.6.2 Economic Feasibility

Since the system is developed as a part of project work, there is no manual cost to spend for the proposed system. Also all the resources are already available, it gives an indication that the system is economically possible for development. Economic justification is generally the "Bottom Line" consideration for most systems. The cost to conduct a full system investigation is negotiable because required information is collected from internet.We can run our system in our normal hardware like desktop, laptop mobiles and so on. This system won't require extra specific software to use it. Hence, the project that we are going to develop won't require enormous amount of Money to be developed so it will be economically feasible.

3.6.3 Operational Feasibility

The user interface will be user friendly and no training will be required to use the application. The solution proposed for our project is operationally workable and most likely convenient to solve the irrelevant document and fraud e-mail [2].

3.6 Working Schedule of Project

We had the following schedule of activities during our project preparation.

Weeks **Activities**	1	2	3	4	5	6	7	8	9	10	11	12	13
Study and Analysis													
Data Collection													
Implementation													
Testing													
Documentation													
Review													

Fig 3.7.1: Gantt chart showing activities and time duration of project

Chapter: 4

System Design

4.2 Flowchart

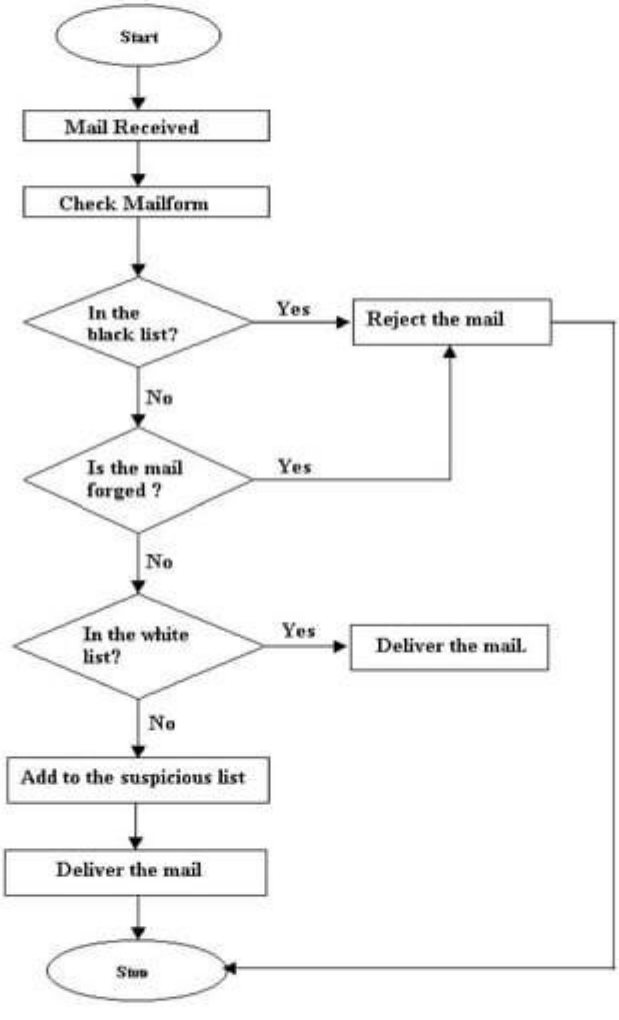


Fig 4.2.1: Flowchart of spam email filtering

4.3 Spam Filter Algorithm Steps

- **Handle Data**: Load the corpus file and split it into training and test datasets.
- **Summarize Data**: summarize the properties in the training dataset so that we can calculate probabilities and make predictions.
- **Make a Prediction**: Use the summaries of the dataset to generate a single prediction.
- **Make Predictions**: Generate predictions given a test dataset and a summarized training dataset.
- **Evaluate Accuracy**: Evaluate the accuracy of predictions made for a test dataset as the percentage correct out of all predictions made.
- **Tie it together**: Use all of the code elements to present a complete and standalone implementation of the Naive Bayes algorithm.

4.4 Naïve Bayes Classifier

The Naive Bayes algorithm is a simple probabilistic classifier that calculates a set of probabilities by counting the frequency and combination of values in a given dataset [4]. In this research, Naive Bayes classifier use bag of words features to identify spam e-mail and a text is representing as the bag of its word. The bag of words is always used in methods of document classification, where the frequency of occurrence of each word is used as a feature for training classifier. This bag of words features are included in the chosen datasets.

Naive Bayes technique used Bayes theorem to determine that probabilities spam e-mail. Some words have particular probabilities of occurring in spam e-mail or non-spam e-mail. Example, suppose that we know exactly, that the word Free could never occur in a non-spam e-mail. Then, when we saw a message containing this word, we could tell for sure that were spam email. Bayesian spam filters have learned a very high spam probability for the words such as Free and Viagra, but a very low spam probability for words seen in non-spam e-mail, such as the names of friend and family member. So, to calculate the probability that e-mail is spam or non-spam Naive Bayes technique used Bayes theorem as shown in formula below.

Where:

(i) P(spamword) is probability that an e-mail has particular word given the e-mail is spam.

(ii) P(spam) is probability that any given message is spam.

(iii)P(wordspam) is probability that the particular word appears in spam message.

(iv)P(non − spam) is the probability that any particular word is not spam.

(v) P(wordnon − spam) is the probability that the particular word appears in non-spam message.

To achieve the objective, the research and procedure is conducted in three phases. The phases involved are as follows:

1. Phase 1: Pre-processing
2. Phase 2: Feature Selection
3. Phase 3: Naive Bayes Classifier

The following sections will explain the activities that involve in each phases in order to develop this project. Figure 2 shows the process for e-mail spam filtering based on Naive Bayes algorithm.

4.5 Pre-processing

Today, most of the data in the real world are incomplete containing aggregate, noisy and missing values. Pre-processing of e-mails in next step of training filter, some words like conjunction words, articles are removed from email body because those words are not useful in classification[i].

4.6 Feature Selection

After the pre-processing step, we apply the feature selection algorithm, the algorithm which deploy here is Best First Feature Selection algorithm.\

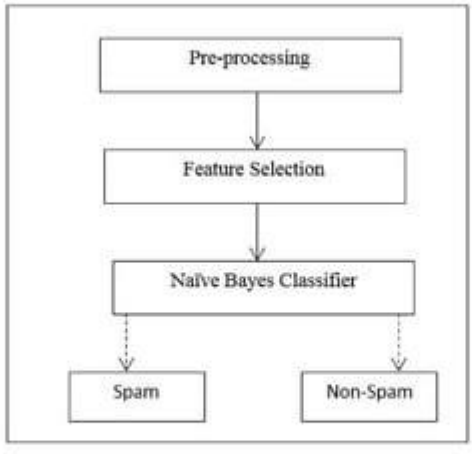


Fig 4.6.1: Process of E-mail spam filtering based on Naive Bayes Algorithm

4.7 The Graphical Model

The naive Bayes classifier is a simple probabilistic model with strong independence assumptions. In particular, the underlying Bayesian network consists of a single node *Y* (whether the email is spam, for instance) that determines a set of features *F*1,...,*Fn* independently (with *Fi* corresponding to the word at position *i*, for instance). For example, if the first word is "Free", the email is more likely to be spam than if it is "Hey" (Or more accurately, spam is more likely to begin with "Free" than Hey) . The graphical model is below:

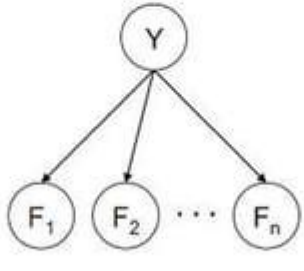


Fig:.4.7.1 :Graphical model

Based on this assumption, we know that the distribution of Y given the features (words at various positions) is $\Pr(Y|F1,...,Fn) \propto \Pr(Y)\prod iP(Fi|Y)$

(Recall that $\Pr(Y|F1,...,Fn)=\Pr(Y,F1,...,Fn)\Pr(F1,...,Fn)$ by Bayes rule). In other words, to determine the distribution, you simply compute $\Pr(Y)\prod iP(Fi|Y)$ and normalize the final distribution. Remember, $\Pr(Y|F1,...,Fn)$ is a distribution where, in the case of spam identification, the two possible values of Y are "Spam" or "Ham".

Chapter: 5

Implementation

5.1 Feature Vector Extraction

We have calculated the Feature Vector by using a dictionary and extracted the probabilities of word that appear to be spam or non-spam. The code below shows the extraction of feature vector extraction

```
def createFeautes(self, text):
    words = text.split()
    feature = np.zeros(len(self.dict))
    for word in words:
        try:                                        /// Handling exception in feature vector exception
            d = list(self.dict.keys()).index(word)
            feature[d] = 1
        except Exception as e:
            pass
    return feature
```

5.2 Testing

In order to create algorithm for this, we need to teach our program what a spam email looks like and what non-spam emails look like. . We also need a way to test the accuracy of our spam filter. One idea would be to test it on the same data that we used for training. However, this can lead to a major problem in ML called overfitting which means that our model is too biased towards the training data and may not work as well on elements outside of that training set. One common way to avoid this is to split our labeled data for training/testing. This ensures we test on different data than we trained on. It's important to note that we need a mix of both spam and non-spam data in our data sets, not just the spam ones. We really want our training data to be as similar to real email data as possible. The sample data for training of non-spam data is shown as below:

5.3 Dataset

Dataset is a collection of data or related information that is composed for separate elements. A collection of dataset for e-mail spam contains spam and non-spam messages. In this research, two datasets are be used to evaluate the performance of Naive Bayes algorithm to filter e-mail spam.

Dataset 1: Spam Data

Spam Data is used in order to test the performance of spam filter based on Naive Bayes algorithm [7].

Subject: re :

2 . 882 s - > np np> deat : sun , 15 dec 91 2 : 25 : 2 est > : michael < mmorse @ vm1 . yorku . ca > > subject : re : 2 . 864 query > > wlodek zadrozny ask " anything interest " > construction " s > np np " . . . second , > much relate : consider construction form > discuss list late reduplication ? > logical sense " john mcnamara name " tautologous thus , > level , indistinguishable " , , here ?" . ' john mcnamara name ' tautologous support those logic-base semantics irrelevant natural language . sense tautologous ? supplies value attribute follow attribute value . fact value name-attribute relevant entity ' chaim shmendrik ' , ' john mcnamara name ' false . tautology , . (reduplication , either .)

The sample data for testing of non-spam data

Subject: book : phonetic / speech production

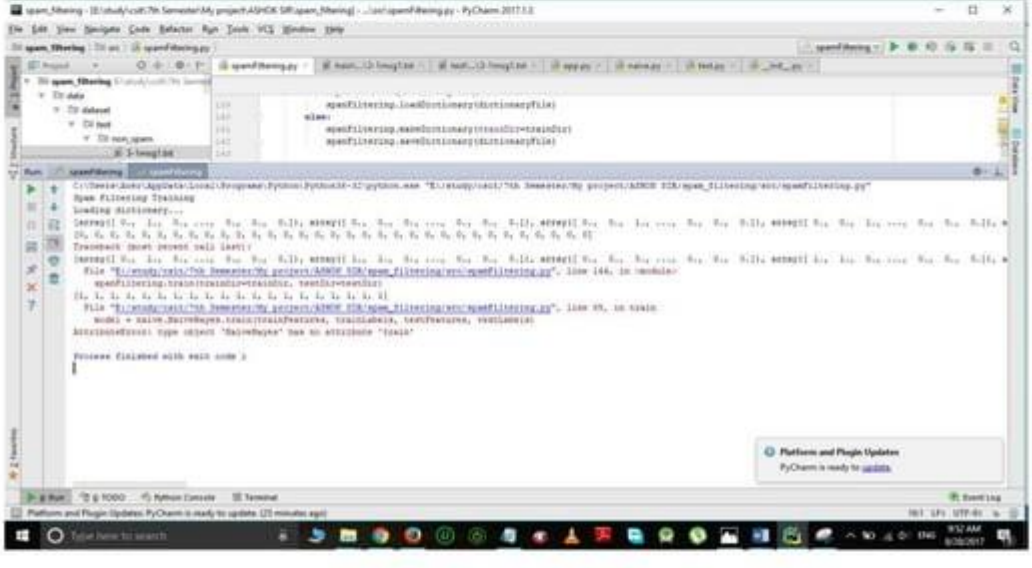
shigeru kiritanus , hajime hirose hiroya fujisakus (editor) speech production language honor osamu fujimura 1997 . 23 x 15 , 5 cm . x , 302
page . cloth dm 188 , - / approx . us $ 134 . 0 isbn 3-11 - 6847 - 0 speech research 13 mouton de gruyter * berlin * york osamu fujimura renown interest competence wide variety subject rang physics, physiology phonetics linguistics artificial intelligence . through fusion discipline show us human speech language relate physical physiological process phonetics abstract , higher-level linguistic structure. reflect osama fujimura 's long-stand interest , chapter volume provide wide perspective various aspect speech production (physical , physiological , syntactic , information theoretic) relationship structure speech language . content 1 background * manfr r . schroeder , speech : physicist

remember * 2 larygeal function speech * minoru hirano , kiminorus sato keiichiro yukizane , male - female difference anterior

* index _ _ _ _ _ _ _ _ _ _ _ _ _ _ mouton de gruyter walter de gruyter

, inc . postfach 30 34 21 200 saw mill river road d-10728 berlin hawthorne , ny 10532 germany usa fax : + 49 (0) 30 26005-351 fax : + 1 914 747-1326 email : 100064 . 2307 @ compuserve

com publication de gruyter order vium world wide web : http : / / www . degruyter . de



Screen shot: train and test data feature vector mode

Chapter: 6

Experimental Result

For any two documents that contains the words they are classified as spam and non-spam using Naïve Bayes theorem. The spam mails are labelled as 0 and non-spam mails are labelled as 1. We have to show that the classified mails help us to classify other mails and help us to calculate the total accuracy of the words that are relevant.

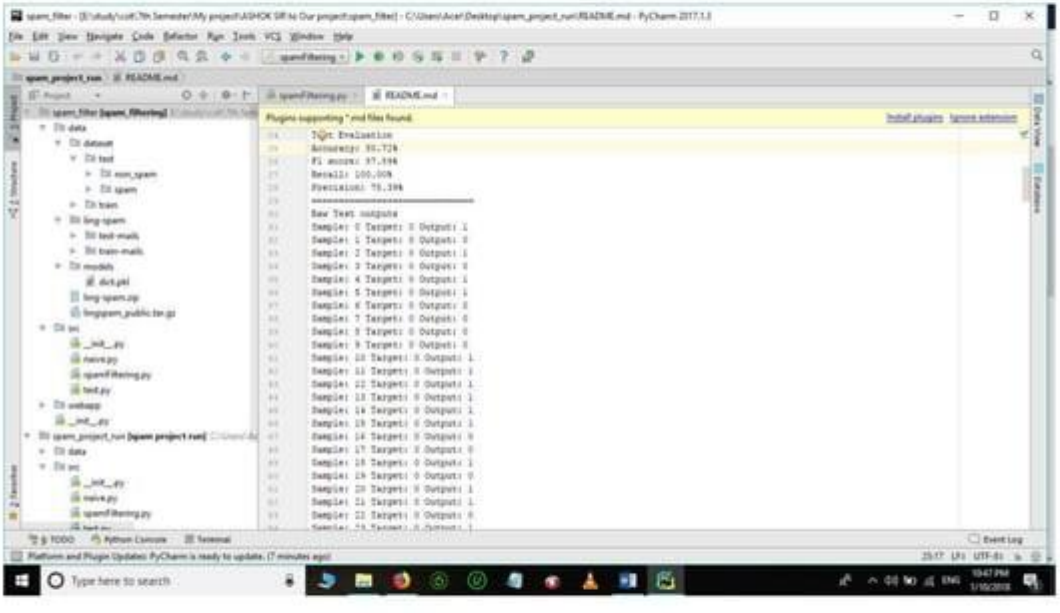


Fig: 6.1: Output of Experiment

6.1 Result And Analysis

The following table shows the [evaluation] measures for spam filters.

Evaluation Measure	Evaluation Function
Accuracy	Acc = TN+TP/(TP+FN+FP+TN)
Recall	R= TP/(TP+FN)
Precision	P= TP/TP+FP
F-Measure	F= 2PR/(P+R)

Table 6.1: Evaluation measures for spam filters

Where accuracy, recall, precision, F-measure, FP, FN, TP and TN are defined as follows:

(i) Accuracy: Percentage of correctly identified spam and not spam message

(ii) Recall: Percentage spam message manage to block

(iii) Precision: Percentage of correct message for spam e-mail

(iv) F-measure: Weighted average of precision and recall

(v) False Positive Rate (FP): The number of misclassified non spam emails

(vi) False Negative Rate (FN): The number of misclassified spam emails

(vii) True Positive (TP): The number of spam messages are correctly classified as spam

(viii) True Negative (TN): The number of non-spam e-mail that is correctly classified as non-spam.

The two datasets are compared based on the percentage of correctly identified spam and non-spam message, percentage of spam message manage to block, percentage of correct message for spam e-mail and weighted average of precision and recall. For each dataset, 10 run of experiments was conducted. The experiments have been done in two parts; (i) using random number of attribute, (ii) using same number of attribute.

Dictionary size: 3000

Train samples: Spam=100, Non-Spam=750, Total=850

Test samples: Spam=92, Non-Spam=214, Total=306

labels[0 = spam, 1 = non-spam]

Training Evaluation

Accuracy: 99.18%

F1 score: 99.53%

Recall: 99.07%

Precision: 100.00%

Test Evaluation

Classified result: Spam=33, Non- Spam= 274

Recall: (33) / (33+ 60) = 35.55 %

Accuracy: (214+33) / (32+ 60 +6 + 214) = 247/ 306 = 80.72%

F1 score: (2*78.39* 35.5)/(78.39+ 35.5)= 50%

Precision: 33/ (33+6) = 33/ 39 = 84.61%

Chapter: 7

Conclusion

We are able to classify the emails as spam or non-spam. With high number of emails lots if people using the system it will be difficult to handle all possible mails as our project deals with only limited amount of corpus.

7.1 Limitations of The Project

Following are the limitations of our project:

Our project, therefore spam filter is capable of filtering mails according to the domain names listed in black list only. Therefore it, at this stage is not able to filter the spams on the basis of the its contents or some other criteria

7.2 Future Enhancement

There is a wide scope of enhancement in our project. Following enhancements can be done: Filtering of spams can be done on the basis of its contents. The spam email classification is very important in classifying e-mails and to separate e-mails that are spam or non-spam. This method can be used by big organization to distinguish good mails that is only the mails they wish to receive.

References

[1]Clemmer, A. (2012). How Bayesian algorithm works. [online] Available at: https://www.quora.com/How-do-Bayesian-algorithms-work-for-the-identification-of-spam [Accessed 16 Aug. 2017].

[2]Mehetha, A., Jain, A., Dubey, K. and bhisee, M. (2009). Spam Filterer [online] https://www.slideshare.net/MaitreyeeBhise/spam-filter-51951717. Available at: https://www.slideshare.net/MaitreyeeBhise/spam-filter-51951717 [Accessed 19 Aug. 2017].

[3]What is Email Spam?. (2017). [Blog] comm100. Available at: https://emailmarketing.comm100.com/email-marketing-ebook/email-spam.aspx [Accessed 27 Aug. 2017].

[4]G. He, Spam Detection, 1st ed. 2007.

[5]sharma, a. and jain, D. (2014). A survey on spam detection.

[6] En.wikipedia.org. (2017). Spamming. [online] Available at: https://en.wikipedia.org/wiki/Spamming [Accessed 29 Aug. 2017].

[7] bot2, V. (2017). Email Spam Filtering : A python implementation with scikit-learn. [online] Machine Learning in Action. Available at: https://appliedmachinelearning.wordpress.com/2017/01/23/email-spam-filter-python-scikit-learn/ [Accessed 30 Aug. 2017].

Appendices

First we import some of the necessary files that we need which is shown by following code

```
import os
from collections import Counter  // Sets the counter
from six.moves import cPickle as pickle
import numpy as np // Imports numpy code for support
from src import naive


class SpamFiltering:
   def __init__(self):
```

```
        self.dict = None // We define class dictionary
        self.model = None


def makeDictionary(self, trainDir, size=3000):  //  We make dictionary with 3000 words
    print("Creating dictionary...")
    if self.dict is None:
        emails = [os.path.join(trainDir+"/spam", f) for f in os.listdir(trainDir+"/spam")]
        emails += [os.path.join(trainDir+"/non_spam", f) for f in os.listdir(trainDir+"/non_spam")]
        allWords = []
        for mail in emails:
            with open(mail) as m:
                for i, line in enumerate(m):
                    if i == 2:  // Reads the body of the email
                        words = line.split()
                        allWords += words

        dictionary = Counter(allWords)


for item in list_to_remove:
    if not item.isalpha():
        del dictionary[item]
    elif len(item) == 1:
        del dictionary[item]
dictionary = dictionary.most_common(size)
self.dict = {k:v for k,v in dictionary} #dictionary
# returns dictionary



def loadDictionary(self, filename):
    print("Loading dictionary...")
```

```
    f = open(filename, 'rb')
    self.dict = pickle.load(f)
    f.close()

def saveDictionary(self, filename):
    print("Saving dictionary...")
    f = open(filename, 'wb')
    pickle.dump(self.dict, f)
    f.close()


def train(self, trainDir, testDir):
        if self.dict is None:
            print("Error: Dictionary is not created!")
            return
        trainFeatures, trainLabels = self.createFeautesFromDir(trainDir)
        trainFeatures = []
        trainLabels = []
        testFeatures = []
        testLabels = []
        model = naive.NaiveBayes.train(trainFeatures, trainLabels, testFeatures, testLabels)
        self.model = model

    def eval(self, testDir):
        pass

    def predict(self, text):
        if self.dict is None or self.model is None:
            print("Error: Model is not loaded!")
            return -1
        # todo predict body
        features = []
```

```
    return naive.NaiveBayes.predict(features)

def createFeautesFromDir(self, directory):
    features = []
    labels = []

    spam = directory+"/spam"
    label = 0
    temp = self.computeFeatues(spam,label)
    features.append(temp[0])
    labels.append(temp[1])

    non_spam = directory + "/non_spam"
    label = 1
    temp = self.computeFeatues(non_spam, label)
    features.append(temp[0])
    labels.append(temp[1])

    return features, labels

def computeFeatues(self, directory, label):
    emails = [os.path.join(directory, f) for f in os.listdir(directory)]
    features = []
    labels = []
    for mail in emails:
        with open(mail) as m:
            for i, line in enumerate(m):
                if i == 2:
                    feature = self.createFeautes(line)
                    features.append(feature)
                    labels.append(label)
```

```
        print(features)
        print(labels)
        return features, labels

    def createFeautes(self, text):
        words = text.split()
        feature = np.zeros(len(self.dict))
        for word in words:
            try:
                d = list(self.dict.keys()).index(word)
                feature[d] = 1
            except Exception as e:
                pass
        return feature

if __name__ == '__main__':
    mode = "train"

    if mode == "train":
        print("Spam Filtering Training")
        dataDir = "data/dataset"
        trainDir = dataDir + "/train"
        testDir = dataDir + "/test"

        modelDir = "data/models"
        dictionaryFile = modelDir + "/dict.pkl"
        modelFile = modelDir + "/model.pkl"

        spanFiltering = SpamFiltering()

        if os.path.exists(dictionaryFile):
```

```
        spanFiltering.loadDictionary(dictionaryFile)
    else:
        spanFiltering.makeDictionary(trainDir=trainDir)
        spanFiltering.saveDictionary(dictionaryFile)

    spanFiltering.train(trainDir=trainDir, testDir=testDir)
    spanFiltering.saveModel(modelFile)
elif mode == "test":
    modelDir = "data/models"
    dictionaryFile = modelDir + "/dict.pkl"
    modelFile = modelDir + "/model.pkl"
    spanFiltering = SpamFiltering()
    spanFiltering.loadDictionary(dictionaryFile)
    spanFiltering.loadModel(modelFile)

    text = "Please forward me the class notes for Information Retrieval."
    result = spanFiltering.predict(text)
    print("Spam Filtering Prediction")
    print("Input: " + text)
    print("Predicted class: " + str(result))
else:
    print("Unknown mode of operation: " + mode)
```

CONCLUSION

Detecting spam emails is crucial in maintaining a secure and efficient communication environment. Through advanced algorithms and machine learning techniques, email providers can sift through vast amounts of data to identify patterns indicative of spam. By continuously updating detection methods and incorporating user feedback, email providers can stay ahead of evolving spam tactics. As a result, users can enjoy a more streamlined and secure email experience, minimizing the risk of falling victim to phishing scams or other malicious activities.