Bonus Project: Email Spam Detector

E4525 Spring 2020,
IEOR, Columbia University

Due: May 1st, 2020

1 Introduction

Email spam defined as unsolicited email messages by a sender with no previous relationship to the receiver represents between 50% and 70% of all email traffic over the last few years (see email statics).

Given the significant waste of network bandwidth, computer resources and recipients time a significant amount of effort has been invested developing automated methods to detect and filter spam messages.

Spam email filtering techniques can be based on explicit hand written rules detecting patterns in messages (i.e if message contains `cheap VIAGRA` it is spam); based in white and/or black lists of email addresses from, respectively, known relationship of the recepient and well know spam senders, finally, statistical approaches based on machine learning have beem also attempted.

Statistical (machine Learning) based spam detectors have a long history (see [1],[2]) due to their ability to handle large quantities of unstructured text and the simplicity of their implementation.

Some well known spam detectors (See [3], [4]) use a combination of machine learning and black list filtering rules to provide state of the art spam detection [5].

In this project you will build a spam detector based in simple Naive Bayes classifier techniques. Our focus will not be to built an state of the art spam detector, rather, we will focus in illustrating how the key machine learning concepts we have learned this semester can be applied in the context of spam detection.

2 Email

An email message contains a set of *headers* providing information like the message's subject, the sender, the intended recipients etc.. It also has a body consisting of one, or more parts. The parts usually contain the body of the message in one or more formats like text and or html, but also any message attachments. Different sections of the body are separated by part boundaries as illustrated in Figure 1.

The format of email messages is complex (see RFC5322 [6] and the documents referenced by it) and parsing email messages is made more difficult by email software not always following the specifications as defined by [6]. We will rely the `python` standard library's `email` module for extracting information from email messages.

Further complicating matters a message's body is frequency encoded with `html` markup and must be further processed to extract the textual information from the html tags. For that we can use `python` beautifulsoup library.

2.0.1 A Simplified Model of Email Structure

For this project we will consider three sources of information present in an email:

Structural Information describing whether the email contains attachments, whether it is html encoded or not, whether it contains hyper-links, etc.

Subject Textual information provided solely by the message subject.

Body Textual information contained in the message body.

2.1 TREC Email Spam Corpus

We will build a classifier based in the TREC 2005 email spam corpus [7]. TREC 2005 is a collection of email messages that was used for an spam detection competition in the year 2005.

You must read carefully and accept the user agreement in `https://plg.uwaterloo.ca/~gvcormac/treccorpus/` and then download and uncompress the corpus from `https://plg.uwaterloo.ca/cgi-bin/cgiwrap/gvcormac/trec05p-1.tgz`.

Many of emails in the TREC 2005 corpus are malicious. The might contain computer viruses as attachments and links to dangerous web pages. **Do not open** emails contained in the TREC 2005 corpus using a regular email client (Outlook, Gmail, etc). We will safely manipulate the email contents using `python`'s `email` module.

The TREC 2005 corpus is contained into subdirectories of the `data` directory as described in Figure (2). Each one of the files `data/xxx/yyy` contains a single email message with the structure described in section (2).

There are different subsets of the corpus defined by the proportion of spam, but we will only work with the full corpus. The index file `full/index` is a space separated file containing two columns: the first column contains the sample label, one of `ham`,`spam`, where `ham` indicates regular email, and `spam` indicates that the email is spam. The second column in the file is the file path (relative to the directory `full` of the email classified (i.e. `../data/020/015`).

```
Received: from nahou-mscnx06p.corp.enron.com ([192.168.110.237]) by nahou-ms
          Fri, 8 Feb 2002 09:01:46 -0600
Received: from NAHOU-MSMSW04P.corp.enron.com ([192.168.110.224]) by nahou-ms
          Fri, 8 Feb 2002 09:01:45 -0600
         id 1CH7LSQH; Fri, 8 Feb 2002 06:52:22 -0800
⋮
Message-ID: <E53BB6A9D4B1D411B6EA00508B6FCE510251C03F@ex-sdge-m04.sdge.com>
From: "Fawcett, Jeffery" <JFawcett@Sempra.com>
To: "'lorraine.lindberg@enron.com'" <lorraine.lindberg@enron.com>,
   "'susan.lindberg@duke-energy.com'" <susan.lindberg@duke-energy.com>,
   "'julie.armstrong@enron.com'" <julie.armstrong@enron.com>,
   "'kevin.hyatt@enron.com'" <kevin.hyatt@enron.com>
Subject: SMILE....!!!
Date: Fri, 8 Feb 2002 07:00:03 -0800
MIME-Version: 1.0
X-Mailer: Internet Mail Service (5.5.2653.19)
Content-Type: multipart/mixed;
         boundary="----_=_NextPart_000_01C1B0B1.49F35BB0"
Return-Path: JFawcett@sempra.com

------_=_NextPart_000_01C1B0B1.49F35BB0
Content-Type: multipart/alternative;
         boundary="----_=_NextPart_001_01C1B0B1.49F35BB0"

------_=_NextPart_001_01C1B0B1.49F35BB0
Content-Type: text/plain;
         charset="iso-8859-1"

You won't believe your eyes....

------_=_NextPart_001_01C1B0B1.49F35BB0
Content-Type: text/html;
         charset="iso-8859-1"

You won't believe your eyes....

------_=_NextPart_001_01C1B0B1.49F35BB0--
------_=_NextPart_000_01C1B0B1.49F35BB0
Content-Type: application/octet-stream;
         name="SONY_W~1.PPS"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
         filename="SONY_W~1.PPS"
------_=_NextPart_000_01C1B0B1.49F35BB0--
```

Figure 1: Sample Email

```
data/000/000
        /001
        /002
          ⋮
        /299
data/001/000
        /001
        /002
          ⋮
        /299
    ⋮
data/307/000
        /001
        /002
          ⋮
        /090
```

Figure 2: File system organization of the TREC 2005 corpus of email messages.

3 Data Pre-processing

We need to extract from the TREC 2005 corpus described in section (2.1) information suitable to train a machine learning model.

Create a `DataTable` with the following columns:

`email_id` This is a message identifier with format `xxx/yyy` where `xxx` is the subdirectory and `yyy` us message number within that subdirectory as detailed in Figure (2).

`parts` this value should be zero if the email is not multi-part (`message.is_multipart()` returns False), otherwise, should contain the number of parts the message as returned by `message.walk()`.

`attachments` The number of attachments contained by a message defined as parts of a multi-part message with `'Content-Disposition'` header equal to `attachment`.

`html` set to 1 if the email contains a part with content type (as returned by `part.get_content_type()` is `text/html`, zero otherwise.

`subject` contains the contents of the `Subject` message header.

`body` contains the payload of the message (if multi-part, the payload of a part with content type `text/plain` or `text/html`) processed by `python`'s `BeautifulShop` package to extract the text from html tags if needed.

`links` number of `html` links contained in a message's body as identified by `soup.find_all("a")`.

To get started processing multi-part messages a good guide is the answer to the following `StackOverflow` question `https://stackoverflow.com/questions/17874360/python-how-to-parse-the-body-from-a-raw-email-given-that-raw-email-does-not` (look at the answer with the highest score). You can modify the code found there to suit your needs.

Once you have a data table with the processed information for each message you must join this table with the target label's as found in the file `full/index` to generate a final `DataFrame` with, at least the columns: `email_id`, `parts`, `attachments`, `html`,`subject`,`body`,`links` andm `spam`. The last column is the target label that we must predict based on the information contained by the other columns.

You can add additional data columns with information about the messages you think may be useful to predict spam.

Save the resulting `DataFrame` into a csv file named `emails_05.csv` so that you can re-used later.

3.1 Train/Test Split

For analysis you should split the email data into 15% test set and 85% training set.

4 Naive Bayes Spam Detection

We will build a number of Naive Bayes models using different types of information to investigate how to best predict spam.

4.1 Structural Information Model

The simplest spam detection model we will build will use only structural information about email messages.

Define the following variables

`multipart` 1 if message is multi-part, 0 otherwise.

`html` 1 if message contains a `text/html` part, zero otherwise.

`links` 1 if message body contains hyperlinks, zero otherwise.

attachments 1 if message contains attachments, zero otherwise.

Using those variables only, train a Naive Bayes model predicting if a particular message is spam or ham.

What is the 5-Fold cross-validated AUC (ROC area under the curve) for this model?

4.2 Subject Model

Train a multinomial Naive-Bayes model on the *bag-of-word* representation of the message subject.

What is the 5-Fold cross-validated AUC (ROC area under the curve) for this model?

4.3 Message Body Model

Train a multinomial Naive-Bayes model on the *bag-of-word* representation of the message body.

What is the 5-Fold cross-validated AUC (ROC area under the curve) for this model?

4.4 Your Own Model

Feel free to try your own Naive-Bayes model using additional data about the email messages that you collected in section (2). As usual, report the AUC for your own model.

4.5 Combined Model

Now build a Naive-Bayes model that combines the structural information, subject information and message body information.

There are a number of different ways you could do this:

- You can implement your own Naive Bayes model that takes all the information into account. This method is the most work, but it will give you more control and flexiblity.
- You can train three separate `sklearn` Naive Bayes models and then combine their probability estimates to find $p(y|x)$. If you do it this way you must take care not to triple count the marginal probability $p(y)$, you can use the argument `fit_prior` to the `sklearn` models to control this.
- You could join structural, subject and message body into one larger X array on inputs and train a single model. In this case you must take care of the fact that the subject and body bag-or-word counts will be encoded as sparse arrays. The functions `hstack` and `vstack` in `scipy.sparse` module could be useful.

5 Analyze Best Model

5.1 Best Model Selection

Using Area under the curve as a selection criteria, and 5-Fold cross validation select the best of the models you trained before.

5.2 Analysis of Best Model

Retrain the best model using all the training data. For this model report the accuracy and AUC using the test set data you set aside before.

Plot the ROC curve (estimated using the test set data) for all the models you have considered. Make sure you train each model using all the training data.

If you wanted to block no more than 1% of all real (ham) email, what percentage of spam would the classifier be able to detect? What if you wanted to block no more than 0.2% percent?

6 TREC 2006 Analysis

The TREC 2006 [7] is a different corpus of email messages. It was used for a spam detection competition in the year 2006.

6.1 TREC 2006 Data Pre-processing

Download the TREC 2006 corpus from the TREC website [7]. This corpus has the same format as the TREC 2005 data set so you can re-use the pre-processing steps discussed in section (2.1)

6.2 Performance Analysis

Without re-training the models, estimate, using the test data, the accuracy, area under the curve, and plot the ROC curve for each one of the models you have considered.

6.3 Discussion

Write a few sentences describing your observation about the different models you have trained n this project and what you have learned.

Can you comment on the difference in performance between the original test set and the TREC 2006 data?

Can you explain why most anti-spam systems provide methods for the user to notify the spam software that some messages have been miss-classified (missed spam, or regular messages that were flagged as spam).

If you trained any of the additional models in section (7) can you comment on the improvements on performance (if any) on the TREC 2005 test set and in the TREC 2006 dataset? What are the trade offs between Naive Bayes and the more sophisticated models you considered?

7 Further Work

If you want to investigate further you can repeat the analysis we have done in this project with any of the other machine learning techniques we have studied this semester.

Try, for example:

- Logistic Regression.
- Gradient Boosted Trees.
- Dense Neural Network
- Convolutional Neural Network.

You should not attempt any of these methods until you have completed the analysis of the Naive Bayes model, as described in the previous sections. Naive Bayes should be your baseline to decide if the extra complexity of these models is providing any benefits, and by how much.

8 Bibliography

References

[1] Paul Graham. A plan for spam. `http://www.paulgraham.com/spam.html`, 2003.

[2] Paul Graham. Better bayesian filtering. `http://www.paulgraham.com/better.html`, 2003.

[3] The Spam Bayes Project. Spam bayes. http://spambayes.sourceforge.net/index.html, 2002–2007.

[4] The Apache Software Foundation. Apache spamassassin. http://spambayes.sourceforge.net/index.html, 2003–2018.

[5] Gordon Cormack and Thomas Lynam. On-line supervised spam filter evaluation. *CM Transactions on Information Systems*, July 2007. `https://plg.uwaterloo.ca/~gvcormac/spamcormack06.pdf`.

[6] IETF Nework Working Group. RFC5322: Internet message format, 2008. `https://tools.ietf.org/html/rfc5322`.

[7] NIST. Trec spam track. https://trec.nist.gov/data/spam.html, 2005–2007.