Build a spam classifier

- Download examples of spam and ham from Apache SpamAssassin’s public datasets: https://spamassassin.apache.org/old/publiccorpus/
- Unzip the datasets and familiarize yourself with the data format.
- Split the datasets into a training set and a test set.
- Write a data preparation pipeline to convert each email into a feature vector. Your preparation pipeline should transform an email into a (sparse) vector that indicates the presence or absence of each possible word. For example, if all emails only ever contain four words, “Hello,” “how,” “are,” “you,” then the email “Hello you Hello Hello you” would be converted into a vector [1, 0, 0, 1] (meaning [“Hello” is present, “how” is absent, “are” is absent, “you” is present]), or [3, 0, 0, 2] if you prefer to count the number of occurrences of each word.

  You may want to add hyperparameters to your preparation pipeline to control whether or not to strip off email headers, convert each email to lowercase, remove punctuation, replace all URLs with “URL,” replace all numbers with “NUMBER,” or even perform stemming (i.e., trim off word endings; there are Python libraries available to do this).

  Finally, try out several classifiers and see if you can build a great spam classifier, with both high recall and high precision.

Hints

1. Understand the Apache SpamAssassin’s public datasets (**easy_ham.tar.bz2** and **spam.tar.bz2**)
2. Fetch the data and load all the emails
3. Use Python's **email** module to parse these emails (this handles headers, encoding, and so on)
4. Look at one example of ham and one example of spam, to get a feel of what the data looks like
5. Some emails are actually multipart, with images and attachments (which can have their own attachments). Look at the various types of structures we have
   a. It seems that the ham emails are more often plain text, while spam has quite a lot of HTML. Moreover, quite a few ham emails are signed using PGP, while no spam is. In short, it seems that the email structure is useful information to have.
6. Take a look at the email headers
   a. There's probably a lot of useful information in there, such as the sender's email address (12a1mailbot1@web.de looks fishy), but we will just focus on the Subject header
7. Split it into a training set and a test set
8. Learn too much about the data
   a. Start writing the preprocessing functions. First, we will need a function to convert HTML to plain text. Arguably the best way to do this would be to use the great **BeautifulSoup** library, but I would like to avoid adding another dependency to this project, so let's hack a quick & dirty solution using regular expressions. The function first drops the <head> section, then converts all <a> tags to the word HYPERLINK, then it gets rid of all HTML tags, leaving only the plain text. For readability, it also replaces multiple newlines with single newlines, and finally it unescapes html entities (such as &gt; or &nbsp;)
   b. eg. def html_to_plain_text(html)

9. Write a function that takes an email as input and returns its content as plain text, whatever its format is
    a. eg. email_to_text(email)
10. Let's throw in some stemming! For this to work, you need to install the Natural Language Toolkit (NLTK). It's as simple as running the following command
    a. pip3 install nltk
11. We will also need a way to replace URLs with the word "URL". For this, we could use hard core regular expressions but we will just use the urlextract library. You can install it with the following command
    a. pip3 install urlextract
12. We are ready to put all this together into a transformer that we will use to convert emails to word counters. Note that we split sentences into words using Python's **split()** method, which uses whitespaces for word boundaries.
    a. eg. class EmailToWordCounterTransformer(BaseEstimator, TransformerMixin)
13. Now we have the word counts, and we need to convert them to vectors. For this, we will build another transformer whose **fit()** method will build the vocabulary (an ordered list of the most common words) and whose **transform()** method will use the vocabulary to convert word counts to vectors. The output is a sparse matrix.
    a. eg. class WordCounterToVectorTransformer(BaseEstimator, TransformerMixin)
14. Train our first spam classifier! Let's transform the whole dataset
15. Print out the precision/recall we get on the test set

How to submit assessment solution

1. Your GitHub/BitBucket/GitLab profile
    a. Create a repository for assessment solution
    b. Create a Readme file under solution repository which includes
        i. Setup instructions
        ii. Execution instructions
        iii. Any other supporting information
    c. Share repository link
2. Use of Python virtual environment is mandatory
    a. Requirement file
3. Email GitHub/BitBucket/GitLab assessment repo/project link and other relevant details to: vikrams@bizanalytix.com