EMAIL SPAM CLASSIFIER USING SUPPORT VECTOR MACHINE

Support Vector Machines to build our own spam classifiers.

We shall be training a classifier to classify the given email, x, is a spam(y=1) or not spam(y=0).

For this, we need to convert our email into a 'feature vector' .

The data set included for this is based on the subset of the SpamAssassin Public Corpus.

“<http://spamassassin.apache.org/old/publiccorpus/>”

PREPROCESSING THE EMAIL

Before starting a Machine Learning task, let us look at the sample email:

> Anyone knows how much it costs to host a web portal?

>

Well, it depends on how many visitors youre expecting. This can be

anywhere from less than 10 bucks a month to a couple of $100. You

should checkout http://www.rackspace.com/ or perhaps Amazon EC2 if

youre running something big..

To unsubscribe yourself from this mailing list, send an email to:

[groupname-unsubscribe@egroups.com](mailto:groupname-unsubscribe@egroups.com)

The sample email contains URLs, an email address, numbers, dollar amounts.

We have to “Normalize” these emails so that all URLs, email addresses, numbers, etc., are considered the same.

For example, we could replace URLs with a unique string “httpaddr” to indicate that a URL is present.

This has the effect of letting the spam classifier make a classification decision based on whether any URL was present, rather than whether a specific URL was present. This typically improves the performance of a spam classifier, since spammers often randomize the URLs.

NORMALIZATION STEPS:

1. Lower-casing: The entire email is converted into lower case, so that capitalization is ignored (e.g., aPPlE is treated the same as apple).
2. Stripping HTML: All email tags are removed from the emails.
3. Normalizing URLs: URLs are replaced with the string “httpaddr”.
4. Normalizing email addresses: Email addresses are replaced with the string “emailaddr”.
5. Normalizing numbers: All numbers are replaced with the string “number”.
6. Normalizing special characters: All special characters are replaced with the string “dollar”.
7. Word Stemming: Words are reduced to their stemmed form. For example, “discount”, “discounts”, “discounting” etc., are considered and replaced by a single word “discount”.
8. Removal of non-words: Non-words and punctuation are removed. Tabs, spaces, newlines, etc., are trimmed to a single space character.

After normalization, our preprocessed email looks like this,

anyon know how much it cost to host a web portal well it depend on how

mani visitor your expect thi can be anywher from less than number buck

a month to a coupl of dollarnumb you should checkout httpaddr or perhap

amazon ecnumb if your run someth big to unsubscrib yourself from thi

mail list send an email to emailaddr

After preprocessing the emails, we have a list of words for each email. The next step is to choose which words we would like to use in our classifier and which we would want to leave out.

For this, we have chosen only the most frequently occurring words as our set of words considered (**the vocabulary list**).

The complete vocabulary list is in the file “*vocab.txt”*

Our vocabulary list was selected by choosing all words which occur at least a 100 times in the spam corpus, resulting in a list of 1899 words.

1 aa

2 ab

3 abil

...

86 anyon

...

916 know

...

1898 zero

1899 zip

Given the vocabulary list, we can now map each word in the preprocessed emails into a list of word indices that contains the index of the word in the vocabulary list.

The figure shows the mapping for the sample email. Specifically, in the sample email, the word “anyone" was first normalized to “anyon" and then mapped onto the index “86” in the vocabulary list.

Word indices for the sample email:

86 916 794 1077 883

370 1699 790 1822

1831 883 431 1171

794 1002 1893 1364

592 1676 238 162 89

688 945 1663 1120

1062 1699 375 1162

479 1893 1510 799

1182 1237 810 1895

1440 1547 181 1699

1758 1896 688 1676

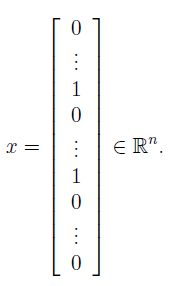
992 961 1477 71 530

1699 531

Extracting Features:

The feature extraction converts each email into a vector in . n = 1899 (from vocabulary list).

xi = 1 if the i-th word is in the email and xi = 0 if the i-th word is not present in the email. It looks like this:



After feature extraction, it is the training of Support Vector Machines. (Refer svmTrain.m)

Now that you have trained a spam classifier, you can start trying it out on your emails. In the starter code, I have included two email examples (emailSample1.txt and emailSample2.txt) and two spam examples (spamSample1.txt and spamSample2.txt).

You can also try your own emails by replacing the examples (plain text files) with your own emails!

Output:

