Content-Aware Email Multiclass Classifier

Based on Categorized Emails

Peeyush Agarwal, Aditya Modi, Lucky Sahani, Hardik Bansal

People nowadays are overwhelmed by tons of coming emails every day at work or in their daily life. The large quantities of emails keep causing confusions. Email classification can be applied to several different applications, including filtering messages based on priority, assigning messages to user-created folders, or identifying SPAM. Not only spam emails are considered to be ‘junk’, but also unwanted emails (e.g. advertisements) cause people to waste time on reading them. Therefore, it becomes urgent to develop reliable automatic categorization of emails to save the trouble.

OUR AIM:

Supervised classification of emails into user created folders according to email content, thereby creating a classification scheme based on previously categorised emails (create a Gmail web app for the same).

INTRODUCTION:

Email classification falls into text categorization section of machine learning. A successful example of its implementation over the years is spam filtering which has reached a great level of accuracy. Email foldering or automatic categorization of emails can also be implemented by using various techniques.

Email Representation:

Emails can be represented as distributions over features, where features can be words, sequences of words, part-of-speech tags, word clusters, etc. Therefore all possible features come in the bag of words approach. Document representations usually undergo a transformation of dimensionality reduction.

Classifier selection:

Various classifiers can be applied to the text categorization. We choose the one which suits
our data best and gives best results.

**DESIGN:**

**Dataset:**

To study and observe various methods and classifiers for email foldering we use a large corpus of real world email messages from Enron employees. The Enron corpus was made public during the legal investigation concerning the Enron Corporation. We used the cleaned and refined corpus made available by Carnegie Mellon University. It consists of email directories of seven former Enron employees with each of them having more than thousand messages and a good number of folders. Auto-generated folders such as “inbox”, “sent_items”, “trash” and so on have been removed from it. It consists of around 8000 mails.

**Workflow:**

We firstly grouped the emails from the corpus into five folders for general evaluation. We put all messages in these five categories:

- Personal
- Business
- Colleagues
- Advertisements
- Organisations

After extracting data from “To”, “From”, “Subject” and “Body” of the emails we applied stemming. Now various strategies can be used for feature selection now using the words list. It may vary from using complete word counts, counts of phrases, part of speech tag counts. We firstly used an open source python code to do stemming. This would highly reduce the total word counts obtained after feature extraction and change different forms of words into same form. Then word counts were stored in two different dictionaries corresponding to body and non-body text. The result of this step gives a sorted list of all tokens appeared in the dataset and along with
their count (i.e. number of times of presence in all the email documents). In summary, there are around 7500 emails in all, and almost 30000 tokens.

**Modifying token list:**
There can be an approach to treat part of speech tags as features and take their counts separately but we opted of removing the obvious and less important part of speech tags such as articles, prepositions, pronouns and all. The *natural language toolkit* available in python was used to remove these unnecessary words. This resulted in a greatly needed decrease in number of tokens from around 30000 to around 20000. It was so because we also opted to remove foreign words and one or two character occurrences as they increased the list of features way high without providing the desired benefit. Even after this step, there is a whole large number of tokens having only a small number of count, so that we can discard them as they rarely appear in other email documents. On the other hand, there are also some tokens having too many counts, which are not informative in terms of the classification task. Eventually, the list of tokens got reduced to the size of 1209 tokens in a total of 7562 emails. We also added 4 other special token: NUMBER, DOLLAR, HTTP and EMAIL. Therefore we have a total of 1213 attributes.

**Creating feature vector matrix:**
The main task of this step is to turn each example into a feature vector and then put them together to form an entire matrix.

1. Each email is parsed and firstly its label is identified.
2. Then, counts for respective elements in the token list are stored.
3. Special words, expressions and characters are replaced by special tokens NUMBER, DOLLAR, HTTP and EMAIL.
4. After creating the matrix by including the feature vector of each email we train and test the examples with an 80% split.

**Classification:**
We classify the emails by using Naive Bayes, SVM and Random forest techniques. SVM deploys one-vs-all method for multiclass classification.

**RESULTS:**
Results on the seven Enron datasets are reported as the accuracy in the table below.

<table>
<thead>
<tr>
<th>USER</th>
<th>NO OF EMAILS</th>
<th>NO OF FOLDERS</th>
<th>NAIVE BAYES</th>
<th>SVM</th>
<th>RANDOM FOREST</th>
</tr>
</thead>
<tbody>
<tr>
<td>beck-s</td>
<td>1971</td>
<td>101</td>
<td>28.26</td>
<td>41.4</td>
<td>48.45</td>
</tr>
<tr>
<td>farmer-d</td>
<td>3672</td>
<td>25</td>
<td>47.79</td>
<td>61.98</td>
<td>69.66</td>
</tr>
<tr>
<td>kaminski-v</td>
<td>4477</td>
<td>41</td>
<td>38.44</td>
<td>47.78</td>
<td>56.42</td>
</tr>
<tr>
<td>kitchen-l</td>
<td>4015</td>
<td>47</td>
<td>35.12</td>
<td>48.97</td>
<td>57.36</td>
</tr>
<tr>
<td>lokay-m</td>
<td>2489</td>
<td>11</td>
<td>61.55</td>
<td>65.45</td>
<td>76.66</td>
</tr>
<tr>
<td>sanders-r</td>
<td>1188</td>
<td>30</td>
<td>53.37</td>
<td>61.36</td>
<td>65.66</td>
</tr>
<tr>
<td>williams-w3</td>
<td>2769</td>
<td>18</td>
<td>72.44</td>
<td>84.33</td>
<td>89.82</td>
</tr>
</tbody>
</table>

Accuracies are higher on datasets that have one or two dominant folders. This is actually the case with *lokay-m* and *williams-w3*. *williams-w3*, infact practically consists of two folders only. This highly facilitates the classification process and thus increases the accuracy. Obviously, classification with one most probable class is an easy task, which explains why the accuracies are so high. One might expect the classification performance improve when the training set size increases, but this rarely happens in practice. It can be explained by the observation that email is usually related to other recently received email, rather than to email received long ago. Thus, old email in the training set probably does not affect the classification procedure.

We also generated a combined dataset classified under five commonly used folders as stated in the design before. The results for balanced dataset and just grouped emails are shown separately.

<table>
<thead>
<tr>
<th>DATA TYPE</th>
<th>NO OF EMAILS</th>
<th>NO OF FOLDERS</th>
<th>NAIVE BAYES</th>
<th>SVM</th>
<th>RANDOM FOREST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unbalanced</td>
<td>9853</td>
<td>5</td>
<td>56.87</td>
<td>63.82</td>
<td>75.67</td>
</tr>
<tr>
<td>Balanced</td>
<td>7562</td>
<td>5</td>
<td>60.58</td>
<td>67.42</td>
<td>79.78</td>
</tr>
</tbody>
</table>

The classification results are surprisingly low. They show the complexity of the task of
categorizing email, as opposed to regular text classification.

IMPLEMENTATION:

We decided to convert our idea into a working model so we shortlisted the possible tools that we can use.

Possibilities:

- Google App Script.
- Gmail Gadgets.
- Chrome extension.

What we chose:

After reading a lot about different possibilities and their features we decided to use GAS. We rejected the idea of making a chrome extension because it does not provide support Gmail API and so we cannot assign labels to the incoming mails using chrome extension. Also we rejected the idea of using Gmail Gadgets because we found that GAS had more tools that were required to implement our idea.

We faced the following shortcomings in our design when we started implementing it with GAS:

- Until now our work was done in python, but now we had to look for libraries and sources in Javascript as GAS doesn't support Python.
- We tried to use pyjs to integrate python with Javascript but we found out that it is very tough to use a package under GAS.
- We then tried to run python on local server but again Google does not allow us to send data from app script to a local server.
- Finally we tried to make some cut-offs and make our model in Javascript only.

We found out that the libraries available in Javascript for implementing machine learning supports only binary classification.

So, we decided to make a prototype of the idea that we had thought that would classify the emails in two categories i.e. personal and work.

We decided to implement SVM and after searching we found a javascript code on github that could be used in our implementation. We wrote the code that will access the user’s Gmail account, scan the already classified emails, train a model using SVM and classify unclassified emails.
Now we decided to implement it as a standalone web app that can be accessed by typing its URL in the address bar.

**Further Improvements:**
- As we have seen above we need to improve accuracy much more. For that we must have a good dataset and we can improve different techniques we implement to classify data.
- We can generalise this implementation to any user.
- We can implement the idea to classify the mails into multiple categories, rather than just binary classification.
- Use various sampling and regression methods in addition to increase accuracy.
- Have greater use of natural language processing for easier distinction between categories.