Email Analyzer

Roll number - 22M0783, 22M0797, 22M0787
Suman Saurav Panda, Akash Sahoo, Santosh Kavhar

An email scanner which will summarize email(past N days mail) and then club them together under predefined categories and visualize them in a html page through diagrams for various analysis purposes.

The email analysis will be done under various categories like key word count, email group and total messages received in that category. (optional - Ranking them as per their importance). Users can choose to filter data as per their requirement like which groups they want to focus. For data visualization we will take help of pie charts, histograms, word clouds and other visualization tools as and when required.

Project status1

Designing Phase

Rendering a HTTP webpage(HTML) -

1. How to handle http requests from a browser and send http responses for the data analysis part. We have explored the GET request and Post request formats and appropriate HTTP response for these requests
2. Building login
3. How to show the contents of summarized mail in webpage
4. How to render image in the webpage using python django library

Backend Application server logic(Python) -

1. extract mail from email server with given user id and credentials, using python imap module
2. filter out the mails which lies within the intended time period provided by user for analysis
3. Process and parse each content files received from imap module
    a. Parse the mails i.e. - find out the headers, sender of the mail, exact received time, original content of the mail
    b. Use tf-idf method to summarize the mail
    c. How to categorize the mail(yet to decide - ml part)
    d. After identifying the summarized mail and category
        i. Create a struct array of field(category, time, sender, attachment, content, summarized content)
        ii. Create hashmap for quick retrieval of mail depending upon the category it has classified into
    e. Create pie chart for email classification

    f. Create a histogram to indicate mail distribution over some predefined time slots(date/day)
    g. Create a word cloud on the summarized contents of the analyzed mails

Project final status

1. We have successfully implemented all of the above mentioned features for our current version of application.
2. For email extraction tasks a major decision change was needed. Instead of using python imap we are currently using google API for the extraction task. [*According to new security policy by Google, after June 30th, third party application access to email was prohibited*].
    a. Google Python API documentation was a bit exacting to comprehend.
    b. Handling http connection and fetching documents over the network with proper try-cache logic.
    c. Cleaning the mails content received as text
3. For the email summarization task we are using the tf-idf methods to rank words. Then a numpy library to fit the word weights to a normal distribution and then pick the top 2 sigma words as summary. [*All logic implemented by the team*].
    a. Sentence tokenization with our own simple regex rules were failing and we referenced online material to implement the logic as NLTK library was also not handling corner case properly for sentence tokenization
    b. Error with wikipedia definition for TF-IDF to avoid division by 0 case were resulting in negative log inverse document frequency value.
    c. Identifying pretrained models for email classification tasks. Currently we are using Zero Shot topic classification using a pre-trained model(Bart Large MNLI) of Hugging Face library.
    d. For classification we are picking sentences which contain only the top summarized words and then sending those sentences to the hugging face library. ***The library takes a lot of time to analyze sentences and then predict its category.***
4. For plotting tasks we have used the matplotlib library. As discussed earlier we are currently showing 3 plots
    a. Pie-chart for email classification
    b. Histogram of email received against number of days
    c. Word-cloud for frequently appearing words
5. Simple frontend to interact with the user and show him/her the analyzed plots.
    a. Simple python server to handle the get and post request
    b. Upon receival of input from user, running email analyzer, generating plots and then showing it back to user in the web page
    c. Rendering an image using python baseHTTPRequestHandler is really troublesome and needs quite a bit of exploration to get it fixed. Now we know how using sophisticated frameworks saves a lot of development time.

Algorithm

For summarization we have used the tf-idf method. Please refer this link to know more about it
https://en.wikipedia.org/wiki/Tf%E2%80%93idf

For classification we have used Zero Shot topic classification using a pre-trained model(Bart Large MNLI)
https://huggingface.co/facebook/bart-large-mnli?candidateLabels=urgent%2C+not+urgent%2C+phone%2C+tablet%2C+computer&multiClass=false&text=I+have+a+problem+with+my+iphone+that+needs+to+be+resolved+asap%21%21

High level overview of modules, classes and functions

Introduction

We run the program by passing 2 arguments: INBOX and DAYS FOR ANALYSIS.
Extraction of email happens using google api. Then we clean them [Get rid of symbols and non-characters] and send them further for summarization task

We get the top 2 sigma summarized words using the tf-idf algorithm and then fitting them on normal distribution using numpy library. We avoid commonly used words by looking at the NLTK library provided stopwords.

We plot pie-chart for email classification, histogram for number of emails received against past few days and a word cloud of all the popular words based on their frequency in all documents.

Then we implement a simple frontend which acts as a user interaction medium for this data visualization task.

Server.py:

doGet : sends the required parameters(label, duration). It uses HTTP GET method to show the web page. It accepts input(label eg. [INBOX, OUTBOX etc], no.of days eg. [2,3,4 etc]) from the user.
doPost: receives these labels and duration and invokes the email extractor and email analyzer logic to generate the summary files and images(pie chart, histogram, word cloud etc)

Email_extractor.py:

driver: It uses gmail API to authenticate the user. And then extracts emails from his/her account. First we try to establish a connection between the gmail server.
establish_connection: It establishes a connection with Google API using Google OAuth 2.0 using the user credentials(gmail username and password). It also saves a token.json so that we don't have to authenticate the same user again and again.
get_message_ids: It fetches only those message ids, those are present in the user mentioned labels in the given time period.
get_message: Given a message id it extracts the raw message from google API.
Save_message_list: This takes a list of message ids and for each id it parses or extracts the message content using parse_message function. If a message is parsed successfully it writes it to a text file.
parse_message: This parses information like header, sender, time and content respectively from a given raw message. Parse message uses Beautiful soup library to parse the lxml type message received from google API.
text_cleaning: Removes the unwanted HTML headers, non-ASCII characters, blank lines and returns a filtered email content.
Write_to_file: It saves subject, sender, date and body to a file.

Email Analyzer.py:

It summarizes all the emails. Then it classifies the summarized email along with that it also generates metadata(csv files) required for data analysis.

driver: It creates an Analyzer class. This class takes a bunch of documents present inside a given directory and a threshold parameter for summarization and generates the summary. It also creates another class called classifier which tries to classify the emails into some specified categories.

time_diff(internal_time): finds the time difference between the given UTC time and the current time in days
get_filtered_words(sentence): tokenize a sentence and removes punctuations and filters stop words
split_into_sentences(text): given a text paragraph, it splits them into sentences
write_output(analyzer): takes the analyzer object and output all csv for further data analysis purpose

Class Analyzer: It computes inverse document frequency for each vocabulary across all documents. It also creates a list of document objects representing individual mails, present in the directory.
compute_word_frequency
compute_document_frequency

compute_inverse_document_frequency
compute_summary

Class Document: For a given document it computes term frequency and then after getting the inverse document frequency dictionary from analyzer class it computes the TF-IDF (Term Frequency- Inverse Document Frequency) weights of each word which are required for the summarization task. It also has all the information like sender, header, date and summarized content of emails.
process_document(document)
find_sentence_list(file_string)
tokenize
compute_word_frequency
compute_term_frequency
compute_tf_idf(inverse_document_frequency)
compute_nsigma_threshold(n)
find_top_nsigma_summary(n)

Class Classifier: It does Zero Shot topic classification using a pre-trained model(Bart Large MNLI) of Hugging Face library. Given the summarized content for an email, it tries to classify it to a set of predefined categories.
find_prediction(sequence)
Get_classification_list

Email_plt.py:

pie_chart(): draw a pie chart for the email categories
word_cloud(): most frequently used words are shown in word clouds
histogram(): email count vs the day on which email arrived

Compilation Instructions

Required library - http.server, matplotlib, numpy, nltk, tensorflow, pytorch, BeautifulSoup, google.auth, google.oauth2
Create a google api to extract emails. For more information refer the below link
https://developers.google.com/gmail/api/quickstart/python

Run server.py using command - python3 server.py
Open browser and try to connect localhost:8000
Provide input as mentioned in the webpage and your analysis will be completed and graphs would be shown on a next page.

Possible Future work

1. While fetching email from server, some emails are getting skipped because of empty content, need to explore why such scenarios occur
2. Splitting sentences written in plain English works well for our application but when it comes to email, full stops are not the sole criteria of splitting a text and we need to add more conditions to break a paragraph robustly so that we get more concise summarized statements. For now sometimes we have very long lines and sometime we see repeated lines in the summarized sentence files
3. ML model takes way too much time for email classification need to figure out a way to make it a real time application
4. A basic Front-end is implemented, we need to improvise this with more features for better user interaction.
5. Priority email identification