Veritas Email Filter Final Paper

Andrew Lu, Avi Gulati

Professor Melissa Dell, TF Tom Bryan

Abstract

In this paper, we propose an email classification system pipeline and deployment in the form of a Chrome extension tailored to Harvard students. Through the Gmail RESTful API, we preprocess an email inbox by extracting the author, subject, and importance score of each email. We then use supervised learning to estimate an importance label for each email. Training occurs on a classification head. The classification head is fed embeddings generated by LLMs that process the author and the subject line. We use multiple models and find that GPT Ada-embedding-002 is the best. The MSE loss on the test data, which comprises 20% of the inbox data, is 0.0291. Model predictions are skewed toward generating low importance scores (shown in Table 3), since most of the training data is made up of low importance scores. Although predictions are far for the highest labels (average of 0.36 instead of 1), relativity is still maintained: important emails are predicted to be more important than unimportant emails—even if the difference in prediction isn't as high as anticipated. Results suggest that our pipeline is capable of accurately classifying emails according to their importance.

---

1 Introduction

1.1 Dataset

The dataset we use is a set of email metadata and text from aggregated Harvard students' email inboxes. This includes but is not limited to message content—subject line, text body, attachments; Gmail labels—starred, sent, unread, archived, muted, snoozed, scheduled sent; and user settings.

Collectively, we have 60,000 emails for the dataset across a diverse set of purposes. Since we are Harvard students and the target audience is similar, we expect our combined set of emails to be approximately relevant even though it may not fully generalize.

1.2 Deep-Learning Task and Motivation

Fundamentally, the deep learning task is the estimation of two key labels for every email: importance on a scale of 0 to 1. The scale is more extensive in our preprocessing stage, described in the Methods. We use supervised training for the deep learning pipeline.

This work can create important insights into how students use email, and the importance and urgency of emails across different groups of students should such research be an area of interest. Whereas Gmail as it exists today has importance labels, it only provides binary classification, using chronological sorting otherwise. It does not sort the inbox by urgency or order of importance. This becomes especially problematic when there is a large influx of emails, as the chronological order pushes important emails down the page. Softmax probabilities generated for the importance of each email enables us to sort emails in inboxes by descending order of importance, as opposed to simply flagging which emails are Important (which is the current state of Gmail).

2 Model/Background

2.1 Pipeline and Training

There are multiple approaches to similar problems. Naive Bayes is used frequently for spam filtering. While training and evaluation can happen quickly, the downside is the assumption of independence of words and no learning about the context in which words appear. Getting an email about a 10% salary promotion at work may have a similar probability of importance as getting an email about a 10% discount promotion at Staples.

Whereas Naive Bayes requires training, a common technique without training is keyword filtering—which is less flexible than Naive Bayes. Words like "deadline," "urgent," and "ASAP" would be stored in a predefined importance list. If an email contains these words, it is flagged as important.

Another technique is sentiment analysis. Whether rule-based or lexicon-based, an email is given a sentiment score that then renders it important or unimportant. Topic modeling is also used, though labeling the data can be lengthy and difficult at scale.

For our research, during training, for each email, we feed the contextualized embeddings from an LLM (BERT base, RoBERTa, GPT, and XLNet) into a linear classification layer.

Use of the Gmail API helps provide the associated labels for each email. For Andrew's 30K emails, he stars emails himself when they are important. The API can extract when an email has been starred, read, or labeled important. For Avi's 30K emails, it's not so simple. While in Andrew's case, the labels are easily identifiable, in Avi's case, there is no filtering mechanism every time he reads an email like starring it. Avi's dataset is significantly noisy because there he deletes and archives every email. He responds to all emails that are important, so emails that warrant a response can be considered urgent. This requires more complicated pre-processing. A method we considered is to classify the importance of emails based on whether or not the email received a reply, but this is a challenging task with the API.

There are many labels we can use. The binary classification "Priority" vs "Non-Priority" is interesting, but we could also complicate these labels by generating classes on a spectrum like "Reply within 12 hours, Reply within 12-72 hours, Unimportant." Such a spectrum would work well if our data is labeled based on if and when the email received a reply, which necessitates significant familiarity with the API to extract time features from emails and threads. Other class options and spectra may require manual labeling. Our methods section elaborates further on our importance representation.

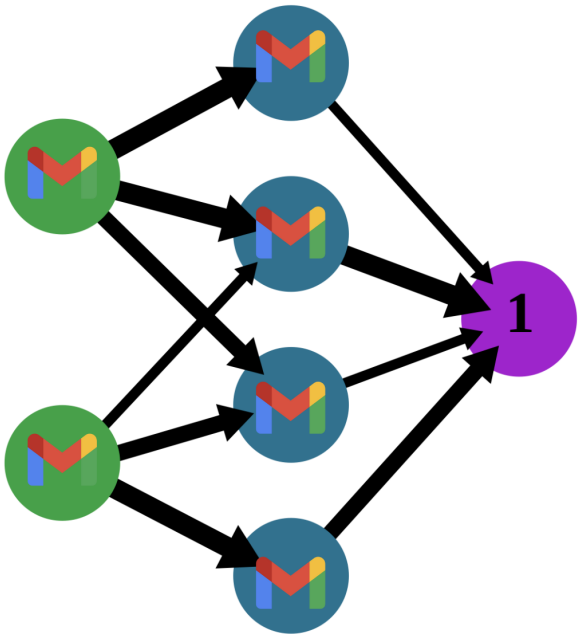

Figure 1: Original image built on SNN image

3 Methods

We developed this deep learning pipeline for email importance classification in a few steps. First, we preprocess the data to extract the relevant format for training. We then generate the importance labels through an importance calculation we perform. Next, we generate the dataset for training, validation, and testing, and train various models to evaluate their performance on new emails. Finally, we deploy this in a version of the chrome extension.

1. **Preprocessing**: This project specifically applies to Gmail, as it is the most commonly used form of email and has easy-to-use APIs to retrieve information about emails. We extract email threads by ID, which provides thread data and metadata. This was done on an account by account basis because of the security features in the API.

Preprocessing requires the use of an authenticated Gmail API Object that first permits access to a user's account. The user's account is specified with the `get_email_address(service)` from the API's helper methods, which prompts the user to sign into their email address on Chrome. We tested this function on multiple email addresses and found that enhanced security features on particular accounts may require users to log in every time data is preprocessed. For example, on the normal personal account "avipratap9@gmail.com," the API requires a single sign-in, whereas on "avigulati@college.harvard.edu", the API requires a sign-in any time data is evaluated or trained (which of course, requires preprocessing).

An optional `limit` parameter in our preprocessing section allows users to specify how many emails threads they would like to include in the dataset. If the limit is not specified, the API extracts all emails in the inbox. As noted above, the API provides thread data and metadata. To train and test the dataset, for each email, we created a headline `msg_headline`. This headline is a format string that started with the author of the email, the subject of the email, and, optionally, the email preview, which is the first 10 words of the email body.

At the end of preprocessing, we have a NumPy array that contains `limit` number of tuples, each tuple containing the `msg_headline` with the associated importance score, which is elaborated on in the subsequent section. Preprocessing was a particularly tedious task due to formatting and trying to circumvent usage throttles/limits.

2. **Importance labels**: To calculate the importance labels, we generate a custom importance metric with which to apply to the emails as their true importance labels. This is because the signal to noise for Gmail's importance engine itself may not be the most accurate. As a result, our importance label considers factors such as whether an email is read, marked as starred, and/or marked as important. The importance label for a thread contributes 0.55 to an email thread's importance. The starred label contributes 0.3. And the read label contributes 0.15. This results in a spectrum of 0 to 1 based on the three binary labels and the sum of their contributions to the importance score of an email. This is generated for all emails based on the metadata labels included in the thread retrieval. An importance label is generated with each `msg_headline` and stored in the preprocessed dataset.

Email Attribute	Weighting Value
Important	+0.55
Starred	+0.30
Read	+0.15

Table 1: Weighting values for email attributes.

3. **Dataset**: To generate the dataset for this deep learning pipeline, we perform a random split 0.6/0.2/0.2 between training, validation, and test datasets.

4. **Training**: To train an accurate deep learning pipeline for predicting email thread importance scores, we add classification heads to multiple pre-trained language models. We perform this training operation for BERT, GPT Ada-embedding-002, and ULMFiT. We train on 50 epochs with a learning rate of 1e-4 and evaluate the performance both qualitatively and quantitatively. While we initially attempted binary cross-entropy, we soon realized the simple mistake of using this loss function on multiple classes (namely > 2). We ultimately used mean squared error as the loss due to the non-binary nature of the importance score.

5. **Deployment**: Putting the model into practice, we created a Chrome extension that is able to manipulate a

Gmail inbox's HTML DOM and rearrange the order of the emails on the main page of the inbox. This is done by retrieving the thread IDs from the HTML elements and calling the importance score model to retrieve importance scores for the emails, ranking them relatively.

4 Results

From the evaluation of various models, we determined that the GPT ada-embedding-002 performs the best. The table below shows the results of the model with a classification head on the test dataset. While the results for important emails are not close to their actual importance scores, due to the inherent noise in the dataset, it is unlikely to predict high scores and thus these results are expected. The MSE loss is included in Table 2. Further, qualitative results demonstrate strong predictive ability and initial testing on other email accounts with different emails and no training yields very strong results (i.e., important emails float to the top, read emails before unread emails). No training means we use the classification head trained on another inbox which generalizes well to external inboxes. We found that the other models, such as the BERT and ULMFit models performed substantially worse, potentially due to smaller model sizes, as scaling can significantly impact accuracy.

loss	val_loss	test_loss
0.0277	0.0296	0.0291

Table 2: MSE Loss on GPT ADA-Embedding-002

	Label	Average
None	0.00	0.1385
Read	0.15	0.1926
Read, starred	0.45	0.1946
Important	0.55	0.2749
Important, read	0.70	0.3337
Important, read, starred	1.00	0.3607

Table 3: Predicted Average Score Per Label

To deploy our models, we built a rudimentary Chrome extension as described in the methods section. The figure below shows the extension pop-up mini HTML page with buttons and text fields as the UI elements.

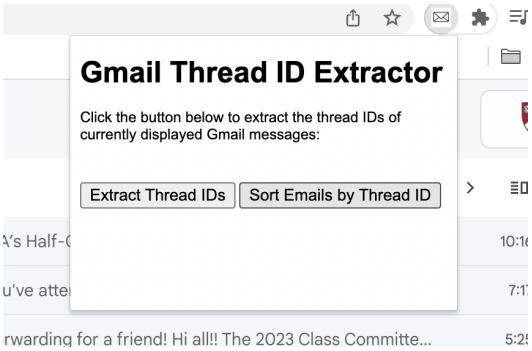

Figure 2: Image of Chrome Extension Pop-Up

Figure 3 shows an inbox prior to sorted emails. We observe spam emails at the top from Bing, Imgur, The Harker School. Figure 4 shows an inbox after sorting occurs. At the top of the inbox, which is now sorted by the predicted importance of threads, we see emails that are read are more likely to sit at top and emails that are unread are more likely to sit at the bottom. We perform this sorting on the 50 emails that are immediately in the inbox.

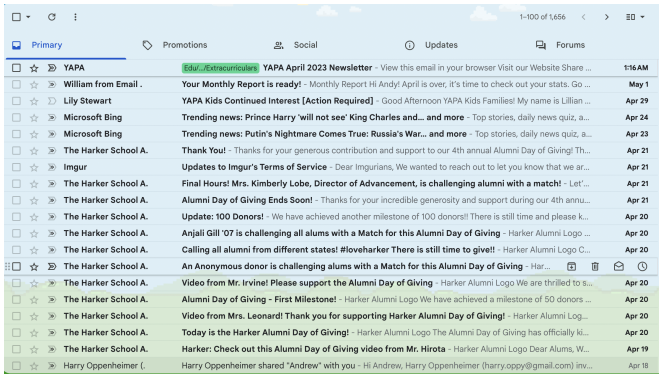

Figure 3: Image of Inbox Before Chrome Extension

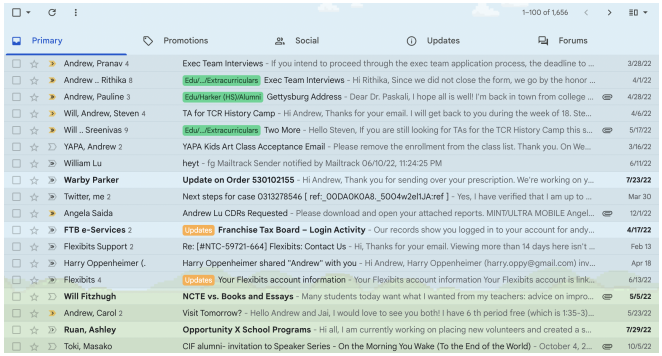

Figure 4: Image of Inbox After Chrome Extension

5 Discussion/Conclusion

Our representative sample was heavily weighted toward the non-urgent/junk category which, coupled with noise, resulted in low ratings for all the importance categories. Still, relativity was maintained. Important emails were predicted to be more important than unimportant emails (unstarred, unread, and unimportant)—even if the difference in prediction wasn't as high as anticipated. Indeed, results suggest that our pipeline is capable of accurately classifying emails according to their importance.

We were initially concerned that classifying emails may not be complex enough for a project, but the challenging preprocessing required for supervised training for the classification head allayed that concern. In addition to testing multiple models, the Chrome extension was a wonderful opportunity for us to make our project immediately relevant to our lives. The demo of the extension in front of the class was exciting, as the read emails floated to the top and the unread ones to the bottom. Further improvements to our extension would be having emails stay in their sorted position for a certain amount of time, and then emails that come into the inbox are immediately placed into the sorted order. During the class-wide demo, after a few seconds of the sorted inbox, the Gmail interface returns to the chronological inbox. Overriding this would be an important feature if this were to ever be a product.

Future applications of this project could help visualize what the most common patterns/tokens are for important emails vs. unimportant emails. A tool like `BertViz` can help answer questions like, if we filter for "Starred, Read, and Important" which tokens are most frequently showing the model that the subject line + author is urgent and requires a response. If we filter for completely unimportant emails, which tokens are most frequently showing the model that the subject line + author indicates an unimportant email. In our initial ideation of our research, we meant to visualize the attention, but that required using models from HuggingFace and we imported models from TF Hub. The BertViz tool, while possible to use with GPT-2, is not built for TF Hub models.

Another interesting nuance to this research could be which method is the most effective for Harvard students. Given a set of topics, or a predefined list for keyword filtering, is classifying subject line embeddings better?

As with any deep learning task, there are multiple solutions at hand, and it is critical to evaluate and compare each solution. We have explored one solution for this final project.