Flo

An Email Management System

The Problem

As a large public institution, UC Berkeley finds itself facing a substantial number of bureaucratic challenges. Whether it be adequately addressing student enrollment, financial aid, or research support, administrative officials can become overwhelmed with their workload. In an attempt to make these processes more efficient, efforts have been put forth to implement easy-to-use, integrated systems. With that sentiment in mind our team created a streamlined solution to a very relevant issue within this domain -- email management.

Teachers and TA's at Berkeley can find themselves sifting through hundreds of emails per week. These emails typically contain very predictable inquiries, rendering the response process a bit tedious and unnecessary. Furthermore these emails can pile up quickly, leaving some of the larger courses with up to twenty thousand emails in one semester alone. Such an estimate does not include professors, further attesting to the severity of the situation at hand. Upon receiving testimony from several instructional support specialists, our team was convinced that this issue needed to be addressed.

Alternatives

Based on our research, we have not been able to identify alternative solutions that effectively address the Berkeley email problem. Google Inbox appears to be the most active in this field, particularly in regards to their email classification and response strategies. Inbox, however, is not specifically geared towards meeting the needs of educators -- or niche markets, for that matter. Given that Google places priority on universality of their product, our management system has the potential to make a reasonable impact at Cal.

Our Solution

Our product uses a Google Chrome extension to accomplish the classification task. It is compatible with all Gmail accounts, and our classification process does not destructively edit messages. Upon opening the reply view to an email, users may choose to automatically respond with a simple click of an icon. This autoresponse is generated from the category corresponding to an email's label. For example, 'Scheduling Conflict' emails will have a default response containing specific course policies on rescheduling. Moreover our extension has created the infrastructure necessary for relaying email analytics. This option is provided in a sidebar, where account holders can investigate frequent inquiries sent by students, amongst other things. Based on our estimates, this project can have a considerable impact on the Berkeley community. If Flo saves 20 Seconds per email, instructors are estimated to aggregately save 6.7 Months in one semester.

Methodology

Our first task was downloading all of our emails in the form of an mbox file. Considerable time was subsequently spent anonymizing, cleaning and preprocessing our dataset. Our team created a series of notebooks dedicated to removing all course-identifying words/email ID's, unwanted thread attachments, and any additional clutter associated with forwarded messages. Preprocessing then consisted of removing punctuation, lemmatizing words, converting everything to lower case, and removing stop words.

The second task involved becoming more acquainted with our dataset. This was an important step, as it is necessary for both understanding the real needs of educators and drafting a model accordingly. After much deliberation, our team came to a consensus on the most useful classification categories. The categories are listed below.

1. Miscl. : Miscellaneous emails that don't belong to one of the categories above (anything that we can't generate a generic response to).
2. Conflicts : Midterm/final scheduling conflicts.
3. Attendance : Attendance of labs, discussion, or lecture.
4. Assignments : Homework/lab assignments (e.g. submissions, excuses, regrade requests).
5. Enrollment : Anything related to CalCentral/course enrollment issues.
6. Internal : Course logistics, hiring (interviews), or other internal administrative issues.
7. DSP : DSP letters or accommodation requests
8. Regrades : Anything related to homework or exam regrades

Once these 1520 emails were labeled, our group moved on to the classification task at hand. Our team experimented with a wide array of classification models during this phase. We first trained a LDA (Latent Dirichlet Allocation) model using num_topics=7 topics. LDA is a generative statistical model that allows sets of observational data to be explained by unobserved groups. This method can potentially create useful explanations about how parts of the input data can be similar. In our case, the LDA model transforms each document into a vector of shape (1, 7). Then the documents in the training set are used to train :

- A cosine classifier (accuracy = 38.6%): this classifier takes the cosine of vectorized input data (along with the vector representing the label) and outputs a prediction associated with the higher cosine value.
- A Random Forest classifier (accuracy = 61.4%) : this classifier takes as an input an array of shape (num_mails_in_train_set, num_topics) where each row corresponds to the vector representing the document in the LDA model

We also attempted to train a cosine classifier using Word2Vec vectors trained by Google (accuracy = 59.8%) : each document is represented as a 300 dimensional vector -- a sum of all vectorized words in a document document. Our team finally decided to settle on a Recurrent Neural Network (RNN) that implemented bidirectional Long Short-term Memory (LSTM) architecture. Although LSTM's were invented in the 90's, they are still remarkably helpful for ML tasks that require storing relational information between input data. Furthermore RNN's themselves are quite useful for dealing with variable amounts of input, an arguable shortcoming of our previous classification models. Bidirectional LSTM's

are the same as ‘vanilla’ LSTM’s with one main modification: In addition to taking previous input data into consideration, bidirectional LSTM’s consider future data as well. In the context of analyzing messages, this can be particularly helpful when the relevant meaning in phrases/words can only be extracted given future context. The final product rendered a validation accuracy rating of approximately 75 percent, a drastic improvement upon previous classification methods. This was a considerable accomplishment for our team, given such a small dataset.

Chrome Extension

While the classification models were being built, another portion of the team focused on chrome extension development. The process began by drawing out plans for the user interface, crafting visually appealing ways to organize its functions. Afterward we studied chrome extension development, working with sample extensions found online. The very first API we had to implement was the Gmail REST API. This implementation proved to be incredibly challenging -- the documentations were out of date, and there wasn’t one complete and functional example online. A very pertinent challenge we faced was receiving authentication from gmail to connect with a user’s account. While seeking more promising alternatives, our group discovered Gmail.js, a javascript API for Gmail developed by Kartik Talwar. However, given its affiliation with Google, this API had limited capabilities. Using Gmail.js our team managed to scrape email information and manipulate HTML content. Unfortunately we were unable to get access to any of Gmail’s basic functions, such as creating a draft or labeling an email. We eventually discovered an alternative to Gmail.js, InboxSDK. Although it still had limited functions, it was much easier to use and implement.

Once we managed to label emails by injecting snippets of HTML code, we turned towards integrating the model with the extension. We initially attempted to use AWS for the computational backend, but we eventually resorted to a simpler implementation on Heroku. While implementing the connection with the model, one of the first challenges we ran into was circumventing a Cross-Origin Resource Sharing restriction which prevented requests made to a different origin. After resolving that issue using a different request function, we ran into another issue of not being able to make multiple requests to the Heroku server at once. Unfortunately this issue was unresolved, and the most straightforward solution may be transferring the service to AWS and have the model hosted there.

Another challenge was figuring out the architecture of the extension design. Specific scripts had unique access to certains parts of the browser, requiring us to pass messages and information between scripts. For example, the content script managed the HTML portion of the gmail and is able to add email labels and inject text. However, it cannot use the Gmail.js function to extract emails. This function can only be run in main.js and in order for information to be passed between the two. We attempted several methods to overcome this obstacle. We initially loaded the other script and ran its function directly. We also attempted to use triggers to set up message passing between the two scripts. Lastly, we attempted to store information in Chrome’s local storage in order for another script to gain access. The last method was the only method that gained relative success, but it is still not fully-implemented.

Despite these challenges, our team managed to achieve significant success within all domains of our project. Additionally, the multitude of challenges we faced throughout the semester have enabled us to develop a clear vision for the future. Given our intimate understanding within each aspect of development, we can efficiently polish our final product. We are thankful to have worked on such a high-impact project this semester, and we look forward to its release in the near future.