Project Description and Work Plan

Repositories

Core
https://gitlab.eeecs.qub.ac.uk/40180175/distributed_nlp_emails

Frontend
https://gitlab.eeecs.qub.ac.uk/40180175/distributed_nlp_emails_frontend

Infrastructure
https://gitlab.eeecs.qub.ac.uk/40180175/distributed_nlp_emails_infrastructure

Problem Statement

Email management can take up a significant amount of times for individuals in the workplace, who often find themselves bombarded with them constantly. Attempting to identify which emails are worth opening and reading can be cumbersome and slow.

This project describes a number of machine learning models which can be used to aid the management of emails and improve general efficiency of a task many of us are required to perform numerous times each day, effectively assisting email triaging.

References

[1] The social economy: Unlocking value and productivity through social technologies

Solution Description

Creation of a pipeline to process bulk email files (.eml).

These emails will be partially provided by Proofpoint, an email security company with whom I've been working for over the last year. Additional emails will be gathered using a web-scraping tool to crawl the web for public mailing lists (ethically of course).

All emails will be scrubbed of any personally identifiable information (PII) before being processed, in order to comply with regulations such as GDPR. All PII identified will either be removed or replaced with generated alternatives.

The processing of said emails must be stateless, in that emails are streamed in, processed entirely in data, and then the trained models are the only artifacts from the build.

This machine learning aspect of the pipeline will consist of:

- Clustering and topic modelling of emails into categories such as 'Announcements', 'Business', 'E-Commerce', 'Social Media', etc.
- Identification of emails requiring a response, while identifying the degree of urgency required, such as high, normal or low urgency.
- Summarization of emails to an optimal length while maintaining relevance to matter at hand.

An API will serve the trained model, so the application can be offered in real-time at the email scanning stage. The model may also be served directory in the browser.

A basic email client GUI will be created to showcase the change in workflow when managing emails with this mined information.

De-Identification Tool

Using non-domain specific pre-trained models, create a pipeline to consume raw email files, identify PII and replace this information with alternative 'fake' data.

The emails will be sourced from a distributed data store (likely AWS S3 or Google Cloud Storage). In terms of the amount of emails, my aim is to work on the scale of multi-millions.

For entity recognition, I'll likely be using spaCy and one of its pre-trained models, specifically the en_core_web_lg model, which has been trained using the OntoNotes corpus.

By no means is this a perfect solution, as spaCy will miss some personal data. But with the use of Faker, we can obscure the data with generated alternatives, making it far harder to identify an individual with this email. If the quality of the fakes were enriched, this effect would be amplified. Faker seems to only concatenate names together and call this a valid company name, which isn't fantastic.

As part of the email de-identification, the emails components will be parsed, and elements extracted. The output of the process will can be an Apache Parquet file but should be integrated into the remaining pipeline as a whole to remain stateless.

These are the emails which will be used to train the forthcoming models.

Acceptance Criteria

- Approval by Proofpoint's data controller on the legality side of things.

References

- [1] spaCy Named Entity Recognition
- [2] Faker Providers

Clustering and Topic Modelling of Emails

Standard clustering and topic modelling of unstructured text into categories. Since we won't know the categories ahead of time, this will encompass topic detection as well. Overall a popular topic within NLP, specifically around medical text and social media.

Using the Spark ML library, we can vectorize the parsed string email components, specifically the body and subject, to result in a dictionary of terms and processed text. These will be in the format of NumPy arrays, which we will split into training and testing sets, with the dictionary being a Python dict. This process may be optimized by using the TextVectorizer very recently added to TensorFlow 2.1, which remains in alpha at the time of writing.

From here, the vectorized text will enter our TensorFlow code. The Estimator class of model will be used as the starting point, with heavy usage of the TensorFlow Probability sub-library expected.

References

- [1] Online Learning for Latent Dirichlet Allocation
- [2] Email Classification with Machine Learning and Word Embeddings for Improved Customer Support

Identification of Actionable Emails

More domain specific challenge and likely to be the trickiest.
Work done on this topic for emails specifically, intertwined with intent understanding in text. Once identified, attempt to sort by priority/urgency (low, normal, high). Perhaps the number of requests in text could be useful.

Identifying sentences which include imperatives will help find sentences with requests, along with simple questions. From this limited set of data, we can attempt to classify by urgency by leveraging a sentiment analysis style approach.

To implement this, the pre-processing step with spaCy can extract relevant sentences, and then a text classifier can be created using TensorFlow, perhaps with the Keras Sequential model.

References

- [1] Detecting Emails Containing Requests for Action
- [2] Classifying Action Items for Semantic Email
- [3] Extracting Tasks from Emails: first challenges
- [4] Context-Aware Intent Identification in Email Conversations
- [5] Characterizing and Predicting Enterprise Email Reply Behavior

Summarization of Emails

Another more general goal of text modelling. Unlikely to cause too much trouble.
The problem at hand can be viewed as unstructured text once again, and hence is fitting for unsupervised learning. Will likely be phrase based, as opposed to word based for the clustering. As we will be dealing with single documents, this task will be made easier.

Implementing this should be trivial, minimal pre-processing will be required to maintain coherent sentence structure. The most common approach involves Seq2Seq models, which will likely be my starting point for this component.

References

- [1] A Survey of Unstructured Text Summarization Techniques
- [2] Email Summarization-Extracting Main Content from the Mail
- [3] Text Summarization Techniques: A Brief Survey

Model Serving

TensorFlow has fairly extensive guides on serving their trained models. Ideally model will be served as an API (likely created using Python) and hosted as part of Kubernetes cluster. An alternative approach offered by TensorFlow is serving models through JavaScript, which may integrate well with the email client component.

Currently I have been persisting my models using the checkpointing system, but I will need to convert this to using the SavedModel export in order to benefit through model serving. This file type may also lead to benefits in serialization the model between Spark workers, as the checkpoint type is not as easily compressed.

References

- [1] Serving Models
- [2] TensorFlow.js

Email Harvesting

A web crawling tool will be created to scrape publicly accessible email, such as the mailing list archives for Apache projects, to download and store raw emails files in a distributed data store. These can then be used to benchmark the de-identification tool by processing mails with and without the tool in use.

The tools likely to be used are Scrapy, a web scraping framework for Python. Alternatively, a Selenium headless browser could be created to imitate actual web traffic to certain sites.

The sites which are chosen for email harvesting must legally approve of the downloading of emails from their site in a programmatic fashion and will likely limit the amount of emails that can be downloaded at once. As such, we will preferably have a large variety of email deposits to harvest.

References

- [1] Scrapy
- [2] Selenium
- [3] Mailing List ARChives

Email Client

Creation of a progressive web app (PWA) / single page application (SPA) using React with TypeScript to view emails while displaying the capabilities of the above models and showcasing the change in workflow (hopefully being efficient). The CSS framework that will be used is Tailwind, which allows for a simple utility-first approach in the site design.

By utilizing PWA capabilities, the finished web client will be able to be stored locally, leading to a more seamless replacement of traditional email client options (such as Outlook).

Example capabilities:
- Automatically creating folders per topic.
- Allow searching per discovered topic.
- Have flag for urgency - can view actionable mails only
- Instead of showing a truncated version of the email body, show the summarized version when showcasing all mails.

Attempt to emulate Outlook viewing capabilities, but worth noting this will only be used to view mail and will not be a fully-fledged email client (or at least will not set out to be initially).

Infrastructure

The infrastructure will be a major component of this project. Elastic cloud computing will be the backbone of the infrastructure, where we can automatically generate infrastructure using infrastructure as code (IaC) and Kubernetes.

Using Terraform, we can create a Kubernetes cluster automatically of a given number of instances. This number can scale automatically based on the current number of emails being processed. Once the cluster has been created, it will be provisioned using Helm, where we will specify the Spark configuration we need.
Likely we will be using the Google Cloud Engine, as we can access the Tensor Processing Units (TPUs) which Google provides specifically for TensorFlow workloads.

From this infrastructure, we will be able to run Spark jobs manually. However, the preference would be the ability to automatically run these jobs in a batch format and streaming format. As a batch system, a scheduling tool such as Cron or Airflow will initiate a build, then the cluster will pull the new data from a distributed data store and process them. In streaming mode however, the cluster will exist over a longer period of time, and shrink and scale based on the current flow of emails into the system.

Requirements Met

Currently, I have completed the de-identification tool, which in turn involves the processing of raw email files into a data frame or Parquet format.

The topic modelling TensorFlow model is near completion and is currently fully functional. Additional changes may be made due to recent releases in TensorFlow 2.1. However, a large chunk of the code will need to be made TensorFlow 2 native in order to benefit from the serving advancements in TensorFlow 2.

The email client is in very early stages, with the web components having been selected and a skeleton of the project created.

The infrastructure currently has basic Terraform code to create the required Kubernetes cluster, and the default Helm chart for Spark is being utilized to create the Spark cluster.

This allows us to run jobs manually, but as the distributed data store has been set up, this hasn’t been attempted yet.

In respect to the original deadlines created previously, I have met all but one deadline. I expect the pace of the project to increase drastically from January, as I no longer have to manage three modules at once.

Quality of System/Results

Here is a short overview of the current systems.

De-Identification Tool

The de-identification tool works as expected. The only critique I have is that the generated fake data is subpar, and if timing is available, I will manually create my own Faker data sources. Here is a before and after comparison of a raw email body from the Enron dataset.

Before

```
As you are aware, Enron utilizes
temporary staffing services to satisfy
staffing requirements throughout the
company.
For the past several months, a project
team, representing Enron's temporary
staffing users, have researched
and evaluated alternative Managed
Services programs to determine which
source
would best meet our current and future
needs in terms of quality, performance
and cost containment objectives.  The
Business Unit Implementation Project
Team members are:

Laurie Koenig, Operations Management,
EES
Carolyn Vigne, Administration, EE&CC
Linda Martin, Accounting & Accounts
Payable, Corporate
Beverly Stephens, Administration, ENA
Norma Hasenjager, Human Resources, ET&S
Peggy McCurley, Administration,
Networks
Jane Ellen Weaver, Enron Broadband
Services
Paulette Obrecht, Legal, Corporate
George Weber, GSS

In addition, Eric Merten (EBS), Kathy
Cook (EE&CC), Carolyn Gilley (ENA),
Larry Dallman (Corp/AP), and Diane
Eckels (GSS) were active members of the
Selection Project Team.
```

After

```
As you are aware, ORG utilizes
temporary staffing services to satisfy
staffing requirements throughout the
company.
For DATE, a project team, representing
ORG's temporary staffing users, have
researched
and evaluated alternative ORG programs
to determine which source
would best meet our current and future
needs in terms of quality, performance
and cost containment objectives.  The
Business Unit Implementation Project
Team members are:

PERSON, ORG, ORG
PERSON, Administration, EE&CC
PERSON, ORG Payable, Corporate
PERSON, Administration, ENA
PERSON, ORG, ORG
PERSON, Administration, ORG
PERSON, ORG
PERSON, Legal, Corporate
PERSON, ORG

In addition, PERSON (ORG), PERSON
(EE&CC), PERSON (ENA),
PERSON (ORG), and PERSON (ORG) were
active members of ORG.
```

As you can see, very little entities were missed by the tool, and as such, this has been approved for usage by Proofpoint. However, it may be some time before Proofpoint can provide emails in bulk, but as I am not reliant on these emails, I can continue working.

Clustering and Topic Modelling of Emails

This currently has been used on a section of the Enron dataset (20,000 emails). We perform 120,000 steps of processing, with batch sizes of 32. Here are the top ten categories from the full run over one and a half hours.

Final Iteration

```
b'index = 38, alpha = 0.15, meeting thank need know work let review time
issue report'

b'index = 15, alpha = 0.07, know time let work think like want thank good
look'

b'index = 40, alpha = 0.04, email request click message site send receive
new web link'

b'index = 10, alpha = 0.03, business team new group report management
role continue risk join'

b'index = 49, alpha = 0.03, market price information company stock option
credit trade reserve service'

b'index = 35, alpha = 0.02, game play say good think start win team right
like'

b'index = 23, alpha = 0.02, dollar bid cost price change expense pay
total limit pound'

b'index = 7, alpha = 0.02, click email receive offer free gift special
online include time'

b'index = 1, alpha = 0.02, program interview candidate thank need analyst
know attend time meeting'

b'index = 12, alpha = 0.02, permit issue facility emission pipeline
environmental agency require state activity'
```

The resulting output shows the identified topics within the Enron dataset.

Infrastructure

Currently, there is Terraform and Helm scripts to manually create a Kubernetes Spark cluster. There has not been any attempt to create the Airflow schedular, however this should be a trivial task.

Spark jobs have been tested on the cluster, however until we have gathered enough data inside our data store, a real job will not be run. The Horovod framework has also not been tested, so the distribution capability of our TensorFlow code has not been tested either, however this also should not cause much trouble, as it will be a small wrapper around the existing code which runs locally.