Script

Good day esteemed jurors, thank you for having me. I will now present my Master´s Final Project called NLP Analysis of Email Interactions to Find Automation Opportunities.

The structure of the presentation will be as follows: I’ll start with the introduction, then go around quickly with the background, then go to the methodology, Next, I’ll describe the results analysis and lastly will address the discussions and conclusions.

First off, I want you think about how frustrating is to reply to every Monday morning email, or to ignore tons of forwarded emails just to find out days later that you ignored a student query or an important meeting invitation. Well, this is quite a common problem.

According to research, up to 65% of the employees interact with each other through social technologies, including email messages. About 28% of their time is consumed by responding, reading, and writing emails. Even more, companies can lose a significant amount of productivity due to spam.

In this sense, Automation is one of the ways to solve this problem, and it offers plenty of benefits, such as savings costs, improving organizational efficiency, and even improving employee’s quality of life. Hence, we look forward for automation prospects.

The general objective is to propose a system based on NLP and Unsupervised Machine Learning to look for opportunities from email patterns in email messages.

The Secondary objectives are to compare three clustering algorithms, evaluate cluster quality without ground-truth assumptions and explore feasibility of find interactions with chains.

Now, our departing background started from Email Mining. So, what is email-mining?

It is a sub-field of data mining that involves techniques applied on email data. There are six main tasks:

Spam Detection from either content or senders, Email categorization, contact analysis, email network property analysis, email visualization and Automatic Email Answering. We will try to find opportunities in any of these tasks.

Now, in the context of email mining, an email has two parts: the header and the body. In the header are located the fields “from”, “to”, copies to other users, subject, and date. In the body we will find unstructured text data, that may contain images, links, markup tags or attachments.

Email data can be represented in two ways: Feature based approach, which analyses emails through their header features, and Social Structure based approach, which is creating email representations through senders and recipients.

The data we used was the Enron Dataset. It is a dataset that was made public by the US Government after the Enron Fraud Scandal.

Since it was published has suffered several modifications due to personal requests of email removal, but the final version from 2015 to date contains around 562 thousand messages distributed along 150 user folders.

It has become a common practice to ignore the folders discussion threads and all documents folders since they contain duplicate emails or are computer generated.

The main challenges that everyone who uses the Enron emails faces is that there is incomplete information, inconsistent chain format, and a typical workflow for pre-processing and tokenization might not be enough.

The structure of the emails is as follows, in number one we have the header, with number two as the body.

Number three contains the features of the header that we used for the project.

Number four is the computer-generated metadata that most of the time we ignore.

Number Five is the most recent message.

Six is the chain inside message and seven the attached documents.

Throughout the dataset, we realized that this kind of chain inside the body has inconsistent metadata, and most of the emails that makes reference to are not included in the dataset, which makes it really difficult to work with. In fact, we defined our own chain which will be explained later.

Here are some examples of reviewed literature that used the Enron dataset, but the common characteristic is that they all used a ground-truth assumption for their analysis, either by defining their own labels or using external data to check if the user was employee or not.

In our case, we went straight to the unsupervised approach. We did not assume any ground truth.

Now, this is the pipeline that we created. We will briefly mention all steps and then go into details later. Starting with our Enron emails datasets we detect chains according to our own steps. Next, we pre-process the emails by cleaning them, tokenize them and remove empty messages. Once we are done with that, we convert emails into text representation to doc2vec. Next, we split our data in manageable chunks according to chain lengths.

Once we finished our data preparation stage, we go to the modeling stage. All our data is transformed into distance matrices. Which are the Euclidean Distance, Cosine Similarity, L2-Norm and Word Mover’s Distance, which is an special case for Earth Mover’s Distance.

This distance calculates how many words needs to be moved from one document to another to change word probability distribution. It has the advantage of considering semantic meaning, but it is expensive to compute since its runtime complexity is cubic.

Then we compute the clustering algorithms that we will cover briefly later. Next, we assess the quality of the obtained clusters with four metrics: the Silhouette score, the Calinski-Harabasz Score, Davies Bouldin Score and Entropy score. All these scores have the advantage of not requiring ground-truth.

Considering the tuning of the parameters, we run thousands of instances to get optimal results, and after getting them, we made a human evaluation to see if any of the obtained clusters were able to capture any interaction process.

Let’s go to the data preparation.

To detect a chain we did the following. First, for each email we allocate in a single array all unique users and sort them alphabetically.

Second, sort emails by a timestamp.

Third, create a string by concatenating subject and sorted users. For each string, we remove patterns from subject such as replies and forward tags. However, the original subjects are kept for later steps.

Fourth, we create a chain in a key-value pair with a unique id as the key and as the value we setup a tuple with the string and a boolean to indicate if the array is a reply.

This step has three cases.

If the original subject does not contain a reply tag, and there is no email with a prior timestamp with the same subject and users, we consider it as the head of the chain and we assign it a unique ID and then set the boolean to false.

If the original subject DO contain a reply tag, and there is a previous email, the same chain id of the previous email will be assigned to it and set the reply boolean to true.

However, if the email contains the same string as the previous emails but the original subject does not contain a reply tag, we consider it as a new chain.

We observed that this last case happened mostly for recurrent automated messages, like weekly reports.

The fifth and last step is that once we iterate over all emails, we count the number of emails with the same id and this will be the length of the chain.

Now, for the text processing over the body of the email, we removed all content below the tag original message with this syntax, since that original chain does not provide us with the information we need consistently.

Also, we removed unwanted content such as html tags, and other kind of content noise with this regular expression. Once done that, we lower case all text, tokenize it and remove non alpha-numeric characters, and lastly we removed the empty tokens.

The resulting data set contains a little more than 250 thousand messages, which is around the 45% of the original messages, with little less than 120 thousand users and almost 20 thousand unique senders.

And here we also show the distribution of chains length. The number of chains with a single message is the largest group by far, being more than 200 thousand chains.

The number of chains with two emails are little more than 12 thousand. Chains with three emails about 3 thousand, and then the frequency of larger email chains gets lower.

However, notice that for chains longer or equal to 10 are 192, but here are included chains with 10 emails to the largest chain with about 7 hundred emails.

After we obtained the chains and pre-processed our text, we convert the messages to paragraph vectors, commonly known as doc2vec or document embeddings.

These parameters were selected according to literature, but we make a special mention to the creation of two vectors, one with 50 dimensions, and the other with 300 dimensions.

The largest one is the most used, but we wanted to choose 50 because in some reports they find that it should be enough to find some solutions and are less expensive to compute.

Notice that we also decided to use a Distributed Bag of Words approach, which is also cheaper to compute but less precise, inversely to its counterpart, the Distributed Memory version.

Then, we split the data into five chunks. Three groups with 1, 2, and 3 chains length, respectively.

Another group for chains with length between 4 and 9 emails each, and last one with chains greater or equal to 10. These partitions were done for two reasons. One, because of computational limits.

The resulting matrices became too large by using more than 25 thousand emails. And the other reason was to see if the algorithms can detect clusters with similar approaches or interactions.

We would expect that chains of emails would be put together since they talk a about the same topic. So, by this logic, we left out from our analysis chains with length 1 and carried on with the rest.

Now, for the modeling part, let’s quickly review the clustering algorithms that we used. The first is K-Means, for which is our baseline because is widely used, is cheap to compute and easy to implement but the problem with this method is that we have to set the number of clusters and it is sensible to noise and outliers.

We also used DBSCAN, which uses density instead of partitions. Here I show how the algorithm detects core points, border points and noise points, to the end determining an Epsilon Neighborhood. We use two parameters, epsilon and minimum points. The epsilon is the radius of these points, and the minimum points are the number of them that should be inside the radius to be considered a core point.

Getting Epsilon is difficult, so the authors that created the algorithm provided an heuristic to ease the problem by finding the Nearest Neighbors, sorting them in ascending order and then finding the slope closest to zero.

But it is also suggested to look for small values of epsilon. So we tested for both cases.

The last method was the HDBSCAN, which uses both hierarchy and density to find clusters. It has five stages.

First, it locates all densities. Second, it finds a minimum spanning tree. Third, it creates a clustered tree through Robust Single Linkage Hierarchy. Fourth, finds clusters’ stability with a condensed tree, and fifth, it maps the most stable clusters.

For their implementation, we used the Python library RAPIDS.AI, which uses GPU to perform said algorithms. The same applies to all distance matrices and cluster evaluation scores. However, there are some special cases. For the Word Mover’s Distance, Gensim’s implementation and Scipy’s Library were tried but were terribly slow.

But we found another implementation that used Library Numba and enable GPU calculations.

The other exception was HDBSCAN.

Currently, RAPIDS.AI’s implementation only allows Euclidean Distance and not pre-computed matrix distances, so the original HDBSCAN library was used for the rest of the distances, which is CPU based and can use several cores at once.

And now we start describing the results. Starting with the case of Chains with Length 2, we obtained that for all scores, the best number of clusters suggested was 2.

We also noted that DBSCAN was the best method for Silhouette score, Calinski Harabasz and Davies-Bouldin scores. And for Entropy, K-Means provided the most stable clusters.

After DBSCAN, HDBSCAN usually came second in best performance most of the time.

In terms of distances, the pre-computed Euclidean distance matrices provided the best results across all methods and scores, except for the Word Mover’s distance which was the best metric with DBSCAN in the Davies-Bouldin Score.

Also, all best results were obtained with doc2vec of 300 dimensions.

However, when we did the visual inspection, we observed that most clustered emails were short answers. But could not detect a clear interaction pattern. And indeed, in the table at the right we can see that most of the clusters detected by DBSCAN were almost 99 % of noise.

For the case of Chains of Length 3, we got that best number of clusters also was two.

And also DBSCAN was the best method overall scores.

However, in this subset we have a variety of distances.

With the Silhouette scores and Davies-Bouldin scores the Word Mover’s Distance obtains the best results,

with Calinski Harabasz we get it with the Euclidean distance and as for the Entropy we have that Cosine similarity is the one that provides the most stable clusters.

In terms of doc2vec dimensions, in contrast with the previous subset, here we obtained the best results were with 50 dimensions in all cases.

However, when we checked visually the results, we found the same pattern of short answers and was not possible to detect a clear interaction process.

And indeed, we also see here in this table that almost all emails are drawn together.

It is worth noting that DBSCAN was able to detect emails written in Spanish and Russian and allocate their respective emails in the same cluster.

Moving on to the next subset, for the Chains of Length between 4 and 9, we get again two clusters as the best number in all cases.

And again, DBSCAN takes the crown as the best method.

Same with the Euclidean Distance as the best one in Silhouette, Calinski Harabasz and Davies-Bouldin scores. For the entropy it was the cosine similarity.

And in terms of dimensions, again all cases were better with 50 dimensions.

Unfortunately, after visual inspection we also were unable to find a clear interaction in the clusters.

And lastly, we focus on the subset with Chains of Length equal or greater than 10. Here contains a greater variety of chains, as we noted earlier, since we can go from 10 emails in a chain, to the largest with more than 7 hundred emails. Now, in all cases, also got that the best number of clusters was two.

In case of clustering methods, the Silhouette Score, Calinski-Harabasz and Davies-Bouldin Scores detected that DBSCAN provided the best cluster quality. And for Entropy, K-Means provided the most stable clusters.

Also, the Euclidean Distance again was the best overall.

As in the case with the first subset of Chains with Length 2, here we get the best results with paragraph vectors of 300 dimensions. Nonetheless, we couldn’t detect any interactions or process with the obtained clusters.

Once we reviewed the results, now we go to their discussion. In general, we found that DBSCAN dominated almost every metric in all the subsets.

We also found that clustering methods with Euclidean distance tend to perform better than the other metrics.

There are some exceptions where the Word Mover’s Distance is better according to some scores, and sometimes Cosine Similarity provide stable clusters.

It is also noticeable that there is a consensus in our results that the best number of clusters was two.

We also noticed that aside from Entropy, Cosine Similarity did perform worse for almost every metric.

As to the Word Mover Distance, it had a mixed performance, since sometimes, as we previously saw, gave the best results, but most of the time had a mediocre performance.

Regarding the number of dimensions of doc2vec, it is difficult to assess which one should we choose, since for two cases we had that best result were obtained with 50 dimensions and the other with 300 dimensions.

Another thing that came to our attention is that the heuristic to find the epsilon value for the DBSCAN did perform poorly consistently

We found also that L2-Norm usually came close to the Euclidean Distance in terms of quality in all scores (specially in the case of HDBSCAN), but never outperformed it.

Also, despite Word Mover’s Distances mixed performance, we notice that is capable to detect context of words, since the language clustering that we mentioned earlier was not present in any other method with other distances.

Now, in terms of interactions, unfortunately most of the allocated clusters were short answers. But even those short answers, there was no clear pattern if they were in the context of a formal email, private conversations, or acknowledgement of a repeated task.

We can also say that working with Enron emails presented several challenges since we had to review several times what was removed and what not. We reached a point where it was impossible to know the pattern of strings that made the noise.

To conclude, our results suggest that our proposed system is not capable of detecting a clear process from the analyzed emails.

However, since most of the clustered emails seemed to be short answers to another incoming message, we can guess that there might be an opportunity In Automatic Email Answering tasks, applying, of course, the corresponding adjustments.

Another thing that we noticed is that maybe these results were due to the approach we took. It would be nice do a future work with a Social Network base approach.

Lastly, we think that there should be Pre-processing Golden Standard for the Enron Emails since it is a messy dataset and the steps taken in the literature review was not homogeneous, and none of the papers addressed the inconsistencies or opportunities of information within the body of the emails. Having a Golden Standard would provide better results for reproducibility and replicability.

That is all from my part. Thank you for your attention and please, we can go head to the questions part.