EURECOM
Sophia Antipolis

Building "betterMail"

Semester Project Report

Alessandro Artoni

Spring 2013

Supervisors:
Raphaël Troncy
Giuseppe Rizzo

EURECOM
Multimedia Department

Contents

Chapter 1

Introduction

Since its creation in 1993, email did not change much in the way we use it. It's basic idea is simple: a plain exchange of textual (html) messages through the internet. Thanks to its simplicity on these days most of us recive a huge amount of emails in a single day which makes almost impossible to focus on what's important to us and leave for later what's secondary.

On the other end this huge amount of informations is not exploited as we could: *Entity Recogition*, *Clustering* and *Semantic Web* are only examples of the technologies that can be applied to take advantage of this giant datasets.

Along with extracting informations from data comes the evolution of the web: from a collection of linked documents or *Web Pages* to full fledged *Web Applications*. Thanks to both the evolution of the developement environment (HTML5, CSS3, ECMAScript 5) and the rise of mobile devices the web is fastly becoming the universal platform.

Motivation for the project

The need for a new approach to electronic mail is getting bigger and bigger along with the amount of messages we recive daily. Moreover we think the potential of building a clustering algorithm based on the enitity recognition system provided by the NERD[1] framework developed in the Multimedia research group in Eurecom is very high and valuable for improving emails systems.

[1]Named Entities Recognition and Disambiguation http://nerd.eurecom.fr

Chapter 2

Similar projects

There is a lot of excitement about building and innovating the world of emails, both in universities and in IT companies. We studied some existing project in order to understand what tecniques were already been used, what are the major approaches used by todays systems and how we can build emails applications better.

These are the projects we studied in particular for mining data:

- Manu Aery and Sharma Chakravarthy developed eMailSift [1]: an email classification system based on structure and content. They propose a supervised data mining technique based on the exploitation of the structure informations present in email messages.

- In [2] is described an approach to determine if a message shuold be “replied to” or if it “requires attachments”. Altough the focus is very specific it’s interesting to see that their approach used a global training set to try to make guess without a specific training set for each user. This idea lend us to think that clustering was to be preferred with respect to classification.

- The CNR developed a system described in [3] for *Mining Categories for emails via clustering and pattern discovery.* It’s based on a three step approach: k-means for clustering, FP-Growth to extract cluster description and a custom technique to update cluster description.

Considering the developement of a Web Application as important as the clustering feature, we studied some of the commercial offer (or proposal) available on the market this day:

- GMail (`http://mail.google.com`) It’s by far the leader in the market and it’s considered to be the first *AJAX* web app to be developed.

While it offer some “intelligent” service (attachment prediction, “important” email labeling) it should be credited for building a great UI along with a reliable system.’

- Alto Mail (`https://login.altomail.com/login/signin`) while still in closed beta and not available for the public (we didn’t manage to try it out) aims to provide better understanding of email subscriptions: they state that Alto Mail understands that products like Facebook, Twitter, LinkedIn, Amazon, (and more?) regularly send notifications and receipts and make it easy to classify or group them.

- As a UI starting point we studied MyMail (`http://dennyshess.ch/mymail/`) which is a mocked up email web client. It’s interface allow to manage a lot of differents email folder at the same time, which is why we chose to it.

Many others papers and articles where red when starting this projects, those selected rappresent the most significant for the developement.

Chapter 3

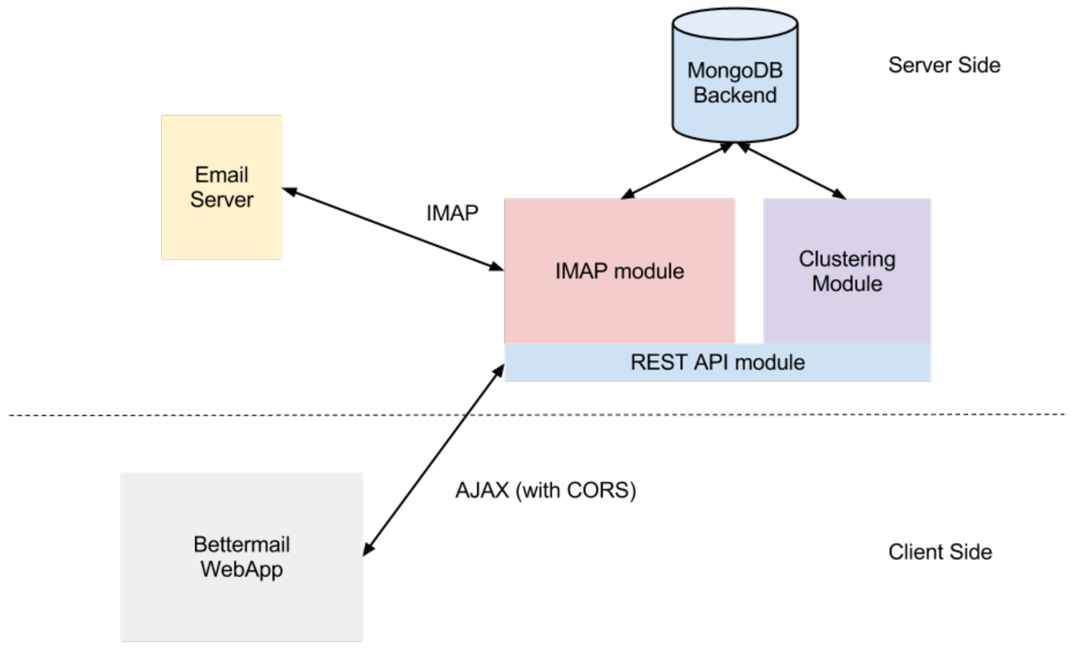
The Approach

In the next chapters we are going to analyze the choice made while designing the system along with some techical description and specification about the implementation. This chapter is divided in three section which reflects the main parts of this project itself: the first part describe the overall architecture of the system, the second section details the implementation of the server-side part of the system together with the specifications of how the clustering algorithm works. In the last part we will describe the prototyping of the client side of the application.

Altought this document tries to separate as much as possible this three aspect we must point out that both the design and the development of the applications was much more parallell than sequential and so are some technical considerations regarding the overall architecture.

3.1 Architectural overview

The application "bettermail" was designed with this goal: provide an intelligent system capable of extracting semantic information from the emails body and use them to automatically classify (label) emails. We tried to investigate how a named entities based clustering approach could prove itself in make the email experience better by automating the email labelling process. Moreover the project aims to give the user a simple and intuitive user interface to access those information.



As described in the picture above the application is strongly separated in two parts: client and server side. Because the main idea was to build the client as a SPA[1] and at the same time to make the server functionalities available to potentially more then one client-side application we decided to use REST[2] to build the main comunication layer between them. We first choose GMail as the targeted email service (mostly because of its wide diffusion) but the final choice was to use a generic IMAP connection to offer better integration with various email provider (altough the application was developed and tested with GMail).

[1]Single Page Application - http://en.wikipedia.org/wiki/Single-page_application

[2]Representational State Transfer - a style of software architecture for distributed system

Chapter 4

The Server

The design and development of the server was one of the major activities of this project. Because our goal was to build an application prototype from the ground up we first need to get the integration with an existing email provider, thus we build the IMAP module. Next step was to expose this IMAP capabilities and, as discussed previously, we build a RESTfull web service to access it, which allowed us to start working on the client too. After that we were able to finally start working on clustering and labelling.

4.1 About general tecnical choices

The whole server is built using Node.js. It is an event-driven system, built on top of Google's V8 javascript engine which provides a very fast and scalable infrastructure to build server side applications. Because of it's event-driven nature it proved to be very suitable for I/O intensive application while being comparable to multi-threaded system under a CPU intensive workload. Among performances one of the main idea behind this choice was the developing paradigm. While building an application with a strongly typed language like Java requires a very long a precise designing phase to avoid the need of big refactoring, javascript is generally considered better for fast prototyping and Agile software development [1].

REST has been chosen as the main API access point because of two main reasons: first of all it requires nothing but a client able to use basic HTTP verbs - in constrast with other system like SOAP that requires additional layers or CORBA which is specific to a platform - and because it's generally

[1]Altought the personal opinion of the author is that javascript can be better even in a more "classical" waterfall design end development cycle this matter is too long to be discussed here

simpler to reason about and define: once the URIs are defined the operations on them follows by the definition of RESTfull web service.

All this services relies on a MongoDB powered backend database which is the natural choice for working in accordance with Agile development and together with Node.js.

4.2 Building the IMAP module

The module responsible for handling the comunication with the IMAP server has passed various iterations. First we tried to build the basic functions to access the IMAP server, query for emails range, add and remove labels and so on[2].

Subsequently, following the guidelines described in RFC4549[3] we implemented a basic IMAP client (on the server) with support for incremental syncronization. All these feature where then exposed via RESTfull APIs described in the next chapter.

One of the major problems encountered whithin this phase resides in the sequential nature of the IMAP protocol itself. The first approach was to create a new connection with the IMAP server for each requested resource. While this approach seems fine when tested in a manual request sequence flow it turned out that GMail, like many other providers, impose a very low limit on the maximum number of concurrent request for the same account (it is not specified in the documentation but we experihenced it to be around 10). This behaviour force us to a fixed, sequential approach, which solved the problem.

4.3 The REST server

For symplifing the process of building our APIs we decided to use the *express* Node.js framework. Its architecture allow to define "routes" on which you can define the behaviour for each separate HTTP verb. In addition it provides a system for plug middelware in the system before routes are handled (for example autenticating user before serving the request). We used this feature to implement a CORS[4] compliant web service.

[2]node-imap (https://github.com/mscdex/node-imap) and mailparser (https://github.com/andris9/mailparser) had been used as a starting point

[3]Synchronization Operations for Disconnected IMAP4 Client (http://www.faqs.org/rfcs/rfc4549.html)

[4]Cross-origin resource sharing, a new standard to allow web applications to safely bypass the same origin policy

These are the main resources defined with the associated URI.Except for labels the service support the GET verb only because the first goal of the application was regarding email reading and retriving. Anyway the architecture would allow to easily add feature like deleting or sending new emails provided to use an SMTP module which at the moment we didn't yet wrote. Each method exchange infomations with the JSON format. The structure of the returned item is listed after each resource.

- `/boxes`: The list of all the mailbox available.

```
[
  {
    "attribs": [
      "HASNOCHILDREN"
    ],
    "delimiter": "/",
    "name": "SomeLabelName"
  },
  //...
]
```

- `/boxes/:boxId`: The details of the mailbox specified by the boxId

```
{
  "uidnext": 293,
  "readOnly": true,
  "flags": [
    "Answered",
    "Flagged",
    "Draft",
    //...
  ],
  "newKeywords": false,
  "uidvalidity": 30,
  "keywords": [
    ""
  ],
  "permFlags": [],
  "name": "SomeLabelName",
  "messages": {
    "total": 279,
    "new": 0
  }
}
```

- `/boxes/:boxId/:mailUid`: The full body of the specified email (identified by `boxId` AND `mailUid`)

```
{
  "text": "someText",
  "html": "<html>someHtml</html>",
  "headers": {
    "delivered-to": "artoale@gmail.com",
    "received": [...]
    //...
  },
  "subject": "aSubject",
  "references": [
    // Related email messageId
  ],
  "messageId": "aMessageId",
  "inReplyTo": [
    "anotherMsgId"
  ],
  "priority": "normal",
  "from": [
    {
      "address": "someOne@gmail.com",
      "name": "Some One"
    }
  ],
  "to": [
    {
      "address": "me@gmail.com",
      "name": "Alessandro Artoni"
    }
  ],
  "seqno": 146,
  "uid": 153,
  "flags": [
    "\\Seen"
  ],
  "date": "24-Oct-2010 18:10:53 +0000",
  "_events": {},
  "x-gm-thrid": "1350194035284457856",
  "x-gm-msgid": "1350507014051968106",
  "x-gm-labels": [
    "\\\\Inbox",
    "\\\\Important"
  ]
}
```

- `/boxes/:boxId/unseen`: The headers and mime structure of all the unseen emails in the specified mailbox

```
[
  {
    //an object like the previous example but
       without text and html
  },
  //more objects...
]
```

- `/boxes/:boxId/:mailUid/labels`: The list of labels assigned to the specified email

```
[
  "\\\\Inbox",
  "\\\\Important",
  //...
]
```

- `/boxes/:boxId/from/:startUid`: The structure and headers of all emails in a specific mailbox with an UID strictly bigger than `startUid`.
- `/boxes/:boxId/from/:startUid/to/:endUid`: The structure and headers of all emails in a specific mailbox with an UID ranging from `startUid` (excluded) to `endUid`(included). If `endUid` is bigger then the maxium UID available this URI is equivalent to the previous one.
- `/sync/:boxId`: Start a new syncronization of the specified mailbox. This URI fires the syncronization mechanism described in the previous section.

4.4 The clustering algorithm

The basic idea behind the approach to labeling emails is diveded in three step: named entity recognition, starting centroids definition and clustering. This technique is inspired by that described by [3] but it tries to exploit as much as possible the structured informations from the email.

4.4.1 Named Entity Recognition

Each email content is first parsed by the NERD framework thanks to the node module provided by the team[5]. The email body is then annotated with a set of informations; for our work we considered two of them: `NerdType` which rappresent the "high level" class of the part of the text considered (e.g. Person, Thing, Date ...) and `label`, the specific entity recognised by NERD (e.g. Barack Obama, Personal Computer, 2012 ...). Those informations are then stored permanently on the database for subsequent elaboration.

4.4.2 Defining Centroids

Instead of using a pure k-means approach with random starting centroids we choose to firt use an Agglomerative Hibrid Clustering (AHC) implementation on a randomly selected subsets of emails. This algorithm outputs a three known as a *dendogram* in which each node correspond to a partition of the clusters and it's child as progressive subpartitions. We traverse this three selecting all the node smaller then a parameter L as basic clusters. For each of this base clusters (if they're bigger than L/k with k parameter) we pick a random element in it as one of the starting centroids for the next step.

4.4.3 K-Means

The last step in the clustering algorithm is to apply a slightly modificated version of k-means with *linkage* criterion. As described above: instead of starting from randomly selected centroids we tried to exploit some knowledge to improve the clusters quality. While the AHC was executed on a subset of emails for performances reason, we run k-means over the whole dataset.

4.4.4 Metrics

We gradually added features (F_i) to the definitions of our distances functions (one for each algorithm):

[5]https://github.com/giusepperizzo/nerd4node

- F_1: NERD type extracted (only if text field available, no HTML)
- F_2: Email address (in both *from* and *to* fields)
- F_3: Datetime
- F_4: Named entity label

This features are then used to computed a coefficient C_i, one per feature, with this relationship:

$$C_i = F_i^3/3$$

Each coefficient is then normalized and multiplied by a wheight w_i, depending on the clustering technique used as follow:

$$distance = 1 - \sum_{i=1}^{4} g(C_i) * w_i$$

where g is a function for normalizing values between 0 and 1 and w is $[0.7, 0, 0, 0.3]$ for AHC and $[0.4, 0, 0.1, 0.5]$ for k-means.

Chapter 5

Building the client

Multiple different approaches where investigated on which type of client to develop. Among other ideas the two main possible approaches where:

- adding feature to an existing web client (GMail, Hotmail, etc...) by using either public APIs provided or by hacking the basic website by introducing script with browser extensions (Greasemonkey, Tampermonkey, ...)
- creating a basic email web client from scratch

Because the first approach seems more reasonable in terms of time and efforts we first decided to go on with it, specifically targeting GMail. After few tries with prototipation and reading a lot of documentation[1] two probelms arised: first of all the main APIs for adding functionalities to gmail is depracated and won't be supported after July 2013. On the other end as the term itself suggests, hacking the website is full of problems: programming it's tedious, it requires a lot of DOM manipulation and traversing, done without a clear specification of how the HTML is structured and thus, how to manipulate it and there is no guarantee for the page structur not to change withoute notice.

For this reason we decided to start over implementing a basic, web based, email client.

5.1 Architecture

As previously noted we decided to implement the client as a SPA. That is because interaction obtained thanks to the intensive use of XHR[2] it's widely

[1]https://developers.google.com/gadgets/

[2]the main object used for sending asyncronous HTTP requests

considered ways better whith respect to the classical multi-page model. In fact most web applications[3] today prefer this model to offer a faster and more engaging UI to its users.

To better structure our work we decided to use an MVC javascript framework choosen between Backbone.js, Ember and AngularJS. The first is the oldest and has many interesting feature, but it force users to a specif type of data access, which is pretty far from plain javascript, and doesn't provide by itself a clear, structured way of developing the application. While Ember may be a good choice we finally choose the latter among those. Because the choice of a framework strongly effects the overall architecture of the system, on the next session we will explain why we choose AngularJS as our library and how this choice effected the client structure.

5.1.1 About AngularJS

The reason behind our choice is to be found in the three main features of this framework, often denotated as *3D* (Data Binding, Directives and Dependency Injection).

Two ways data binding

Most client-side frameworks offer some kind of templating system which allow to define how objects will be rappresented when attached to the DOM. These libraries offer a one-way data binding: if the object changes, the framework is notified and the DOM is altered accordingly. AngularJS on the other end improve this approach in two ways: it uses plain HTML (even W3C compliant, if required) as the templating system and offer data binding in both directions: not only objects changes are reflected in the DOM, but the other way round too. If an input box is binded with a javascript variable, each time the input box is changed the variable itself is modified accordingly.

Directives

Frontend developer usually create or use widget to encapsulate specific behaviour in a single piece of code. Thanks to *directives* it is possible to extend the native HTML markup lanugage by

"teaching the browser new tricks".

[3]the terms doesn't refer to general web sites but to complete applications built on top of the web platform like gmail, Cloud9IDE, google drive...

Pavel Kozlowsky, Codemotion, Rome - 23 March 2013[4]

In other words it is possible to define your own tag and then use them like they were plain HTML; AngularJS compile them in an internal rapresentation and attach the real HTML on the fly. This feature provides two big advantages: you can build higly reusable components and the code looks much more cleaner and readable thanks to the markup being embedded in the directives.

Dependency Injection

One of the main feature required by a industrial-ready system is to provide an easy way to unit test your components. AngularJS encourage developers to write separate modules; each module specify the list of other modules it depends on, without explicitly allocating the resource (process handled by the framework). "requiring" something instead of directly "creating" it make the components of an application much more decoupled (and thus, easier to mantain). Moreover thanks to the ability to use different implementation of a specific component depending on the environment it become much easier to test this component independently: it is sufficient to provide a "moked" version of a dependencies to avoid the need of having an instance of the real dependency up and running (this can be very usefull if the dependency rely on some external service, it's slow or requires the system to change state, which is bad for unit testing).

As AngularJS was our framework of choice, the application follow the structure described by the framework itself:

- a set of *service*s is resposible for creating an abstraction layer to access data. In our case we have services for restrieving data in the IndexedDB storage system of the browser (described later) and a service for accessing the REST API of our server.
- a set of *directive*s to encapsulate the behaviour behind the graphical components of our applications
- some *controller*s, resposible for binding data to the right HTML fragment (including event handlers).

5.1.2 Tooling, scaffolding and external libraries

In the last few years a lot of efforts have been put in improving and automating the workflow of front-end web development. A lot of tools and systems

[4]http://www.youtube.com/watch?v=wmhPfx0s40o

have been created in order to reduce developers efforts. Moreover, together with AngularJS we used other external resources to speed up and simplify the process. Here's a list of tools and external libraries of invaluable worth:

- Yeoman (yo - `http://yeoman.io`): a tool for project scaffolding. It creates a template web application including HTML5 boilerplate, EscmaScript 5 polifyll and many optionals libraries. Thanks to it's excellent integration with AngularJS it can be invoked each time we need to create a new angular component (e.g to create a directive called tab `yo angular:directive tab`). Moreover it creates a complete and customized configuration file for *Grunt*, described below

- Grunt (`http://gruntjs.com`): used to build, preview and test the project, thanks to help from tasks created using Yeoma it's very well integrated in the echosystem. Specifically, it allows to run a static web server to serve file and try the systems, automatically compiles coffescript into javascript and SASS/SCSS into plain CSS every time a file changes, add support for in-browser livereload (again, on file changes). It integrates with Karma-runner, the javascript test runner developed by the AngularJS team. It allows to build the application by minifing and eventually optimizing javascript and css, compressing images and much more.

- Bower (`http://bower.io`): it offers a solution to the front-end package management problem, as it allows to mantain and manage package installations and dependencies for generic front-end development resources (tipically javascript and css libraries).

- Twitter Bootstrap (`http://twitter.github.io/bootstrap/`): a stylesheet framework which simplifies the developement of user interfaces. It provides a grid system for layouting and a wide set of components (buttons, tables, tooltip, etc...)

5.2 Design

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Sequi, distinctio, voluptatum ratione recusandae perferendis laboriosam illum earum dicta consequatur nam.

5.2.1 Sketches

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Amet, pariatur, assumenda, labore ullam libero atque doloribus asperiores quasi est incidunt officia numquam doloremque ut temporibus quis nisi totam cum laudantium tenetur nihil et! Saepe, impedit maxime commodi ab minus provident sunt cupiditate voluptates veritatis eveniet ex expedita dolorum dolorem non minima perspiciatis beatae porro soluta officiis nihil eligendi aliquid explicabo eum. Deserunt, commodi voluptatem eligendi nobis nihil libero ducimus sunt distinctio asperiores ratione porro quam quae vel modi sit necessitatibus explicabo magnam ipsa ad suscipit blanditiis cupiditate ipsam error illum voluptates dicta cum consectetur quibusdam eius quas beatae reiciendis laudantium possimus. Officiis, ab voluptatibus corporis quam assumenda harum facilis illo ex dolorem sunt doloremque hic unde repellat facere dolores accusantium soluta! Architecto, magnam nobis maiores beatae fuga sed voluptatibus cum nisi reiciendis iure id eum rem nihil velit non veritatis aliquam. Officiis, incidunt, itaque, perspiciatis reiciendis at ex consequatur tempora vel dolor commodi beatae molestias labore aut porro numquam architecto illum quae maiores blanditiis eaque quis reprehenderit enim amet cumque adipisci? Reprehenderit, animi, qui, quia repudiandae nam esse pariatur facere blanditiis aperiam inventore consequuntur voluptates dignissimos provident vel praesentium! Vero, nihil, consectetur magni possimus ratione ullam neque totam accusantium porro.

5.2.2 Prototypes

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Tempore, sapiente, odio, provident animi veniam incidunt vel blanditiis atque voluptatem laborum eveniet eligendi dignissimos libero amet natus ullam odit sint voluptatum saepe officiis iusto obcaecati itaque illo praesentium porro iste nulla quibusdam commodi aliquam. Qui, doloribus, quasi, alias cum neque nisi laborum quis porro hic quaerat inventore blanditiis voluptas nihil unde rerum repellendus quae. Molestias, numquam, expedita odio adipisci tenetur quasi animi sint veritatis placeat voluptatum dignissimos earum impedit maxime nulla accusamus doloribus atque magnam provident consequuntur quia quibusdam hic obcaecati inventore et ipsum quo similique odit itaque aperiam assumenda laborum sed dolorem quidem vitae corporis excepturi fugit! Illum, vel numquam esse corrupti facilis eaque mollitia praesentium harum assumenda a doloribus et eveniet iste asperiores optio doloremque ratione explicabo ut natus necessitatibus voluptate corporis sit vero sint saepe quos aliquam nobis cumque aspernatur non repellendus perspiciatis eum quas. Quo, iure, suscipit, aspernatur ipsum dignissimos assumenda enim aliquid velit impedit voluptatum omnis eligendi cumque incidunt quaerat quidem maiores inventore ratione officia facilis fuga dolorem quia voluptate quam consequuntur animi repudiandae natus nostrum! Modi, perspiciatis, eveniet, nostrum dolores ab fugit eum tempora eos iusto vel illum pariatur quibusdam temporibus officia quam enim necessitatibus ut eius libero nulla repellat. Quaerat, aut, voluptates, hic, commodi officia rerum nam consectetur necessitatibus fugiat ea vero atque voluptatem harum magnam expedita quam magni ab incidunt. At, magnam, fuga corporis rem vitae laborum veniam quos error aut minima ea odit itaque explicabo ducimus harum sint impedit neque recusandae ratione dignissimos quam maiores nulla natus sequi molestias libero necessitatibus amet perferendis pariatur eos saepe qui odio deleniti nemo commodi dolores ipsam consequuntur in autem magni. Saepe, quisquam ex odio dicta optio aut provident autem reprehenderit veritatis expedita illo eveniet ab rerum quas facilis aliquam quo corporis vero laudantium eos dolore!

5.2.3 Final result

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Voluptate, amet repudiandae eaque saepe molestias inventore perspiciatis iste natus doloremque quo soluta temporibus ea minima veniam ipsam incidunt rem ut non accusantium corrupti veritatis autem a culpa suscipit laborum? Quam in tempora at sequi nobis eaque culpa. Ab, cupiditate, distinctio, rem, fugit sunt porro quae optio eveniet molestias reiciendis deleniti dolorum. Repudiandae, praesentium, placeat, ab, ipsum culpa sed illum sapiente quis eaque porro dolore facere quae nam ullam beatae laborum impedit amet officia asperiores accusantium laboriosam non deserunt aspernatur voluptate enim accusamus doloremque. Eaque quod officiis sed unde dolores repellendus quae!

Chapter 6

Use cases (if any) - Should i put it?

Chapter 7

Evaluation

7.1 Analysis on a personal dataset

7.2 Metric measures on “enron” dataset

Chapter 8

Conclusion/Future work

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quas, debitis accusantium ratione maxime veritatis eveniet nisi laborum cum labore libero modi amet odio inventore ipsum facere non voluptatum velit natus hic beatae quaerat fuga accusamus fugit excepturi perspiciatis aperiam neque aut culpa autem porro ea minus repudiandae aspernatur quam ducimus molestias eaque. Praesentium, odit, est, eos distinctio debitis repudiandae vero deleniti nihil molestiae sit ab reprehenderit tempore! Quibusdam, saepe, minima, libero, iure temporibus voluptate doloremque aliquid maxime nam deserunt quam odit illo voluptatibus ducimus non recusandae nesciunt ratione dicta consequuntur cumque. Esse, molestias, inventore labore incidunt quas distinctio perferendis animi odio ab ratione cupiditate qui eum aliquam? Deleniti, delectus nesciunt at quas eligendi fuga provident mollitia perferendis magnam laboriosam officia eaque harum animi culpa reprehenderit voluptate repellat iure quasi temporibus autem illo nemo sed quo eveniet laborum a similique nostrum rem adipisci rerum sint voluptatum accusantium ea. Asperiores, incidunt, non atque cupiditate fuga quasi perspiciatis. Deserunt, debitis nemo laudantium vero nisi qui enim voluptatum minus animi asperiores eius maiores voluptates atque voluptate dolores. Porro, voluptas, minus, rem asperiores fugiat eligendi commodi hic debitis eaque voluptates optio esse laudantium odit ab sequi harum recusandae amet iusto tempora expedita nisi reiciendis accusantium quisquam veritatis ipsam. Placeat, magni, rerum ipsam est a explicabo cupiditate repudiandae odit praesentium temporibus sequi voluptatem harum officia consequuntur laborum necessitatibus facilis! Numquam, expedita, iusto, animi laborum nihil omnis culpa impedit quisquam tempora adipisci iste odio nisi ducimus assumenda sint aut molestias. Cumque, doloremque, nemo, magni pariatur maxime adipisci nihil quibusdam ipsa earum perferendis alias libero eligendi excepturi voluptatum voluptate quas ullam consequatur voluptates distinctio tempora explicabo consequuntur id nostrum ad numquam maiores architecto fugiat repellendus quod beatae incidunt blanditiis necessitatibus aut quisquam quasi rem culpa. Id, maxime, autem, perspiciatis debitis fugit vero quia modi voluptatem temporibus vel similique.

Bibliography

[1] Manu Aery and Sharma Chakravarthy. *eMailSift: Email Classification Based on Structure and Content.* IEEE, 2005.

[2] Mark Dredze, Tova Brooks, Josh Carroll Joshua Magarick, John Blitzer, and Fernando Pereira. *Intelligent Email: Reply and Attachment Prediction.* ACM, 2008.

[3] Giuseppe Manco, Elio Masciari, and Andrea Tagarelli. *Mining categories for emails via clustering and pattern discovery.* Springer Science, 2008.