Web information retrieval and data mining - 2IMM15

Email Filtering and Querying

Group 35
Alex Anthis – Boolean IR, Naive Bayes Classifier and Java App

Eindhoven, 1 April 2019

Abstract

This report is about how we as a group implemented the course project on email filtering and querying. Including the literature surveys we have studied, from choosing the best suited methods to the final product of our work, and also the obstacles we faced as well as improvements made during the process.

Introduction

Choosing an efficient way to query email data, distinguish spam mail from a users mailbox, clustering emails into subject categories are all tasks and challenges an email provider is responsible for in order to serve the end user successfully. Using Information Retrieval and Data Mining techniques we worked on building such functionality of an email provider on our project and testing it on a large email dataset. For the implementation Java was used as the codebase as well as SQLite DBMS for the database. For the specific project, my personal part is to tackle the following challenges which will be explained in detail in this report:

1. Create Boolean IR database to query through the set of emails
2. Classify emails as spam or non-spam using Naive Bayes

Motivation and goal:

The overall goal of the project is to provide insight on the way an email provider handles tasks such as querying through a body of emails, the distinction of spam emails from the legitimate ones or classifying emails to different categories regarding their content.

Dataset used:

The dataset contains a body of emails in preprocessed format. Each message exists in a separate text file. The total size of the data is approximately 2700 emails. Dataset was created by AUEB professor I. Androutsopoulos.

Architecture:

As an overview of the project components, the Boolean IR implementation supplies the database with a boolean table and an inverted index table which can be used for querying the dataset of emails. The Naive Bayes implementation focuses on the classification of each email as spam or non-spam assembling the functionality of an email filter.

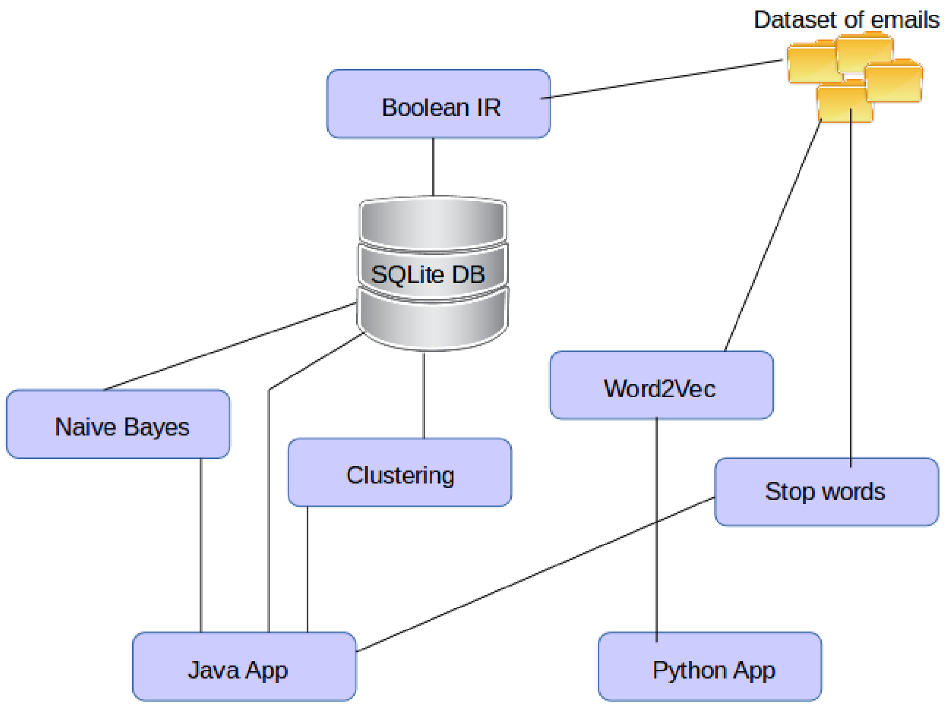


Figure 1: Project Structure Overview

Boolean IR component

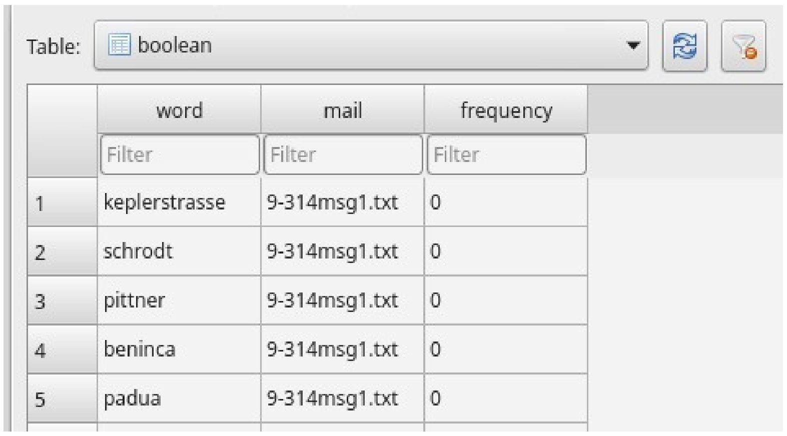
I. Component Description

Considering the methods of retrieving large sets of documents with their own drawbacks as well their advantages, we choose to build a Boolean IR system for the reason of simplicity.

The creation of such a system manifests through parsing the dataset and building the required tables of the database. In our case, the model consists of the document names (emails) as well as the terms contained in those. We have to be able to store and retrieve 425 thousand terms contained in 2.700 emails which makes a total of approximately 1.1 billion records.

Therefore, one has to be acknowledged that the creation of the complete database storing all the required terms for the dataset will demand a great amount of time and resources. On our part we handled a small portion of the data and tested our model on those as an outcome of limited workforce on our machines.

The database we built consists of a boolean table and an index table. The boolean table works as a term-document incidence matrix in which element (t,d) is 1 if the term t exists in the specific email d[2]. Otherwise it is 0. In order to represent the table in SQL we use the element id (t,d) as a composite primary key and the frequency as the value for a record.

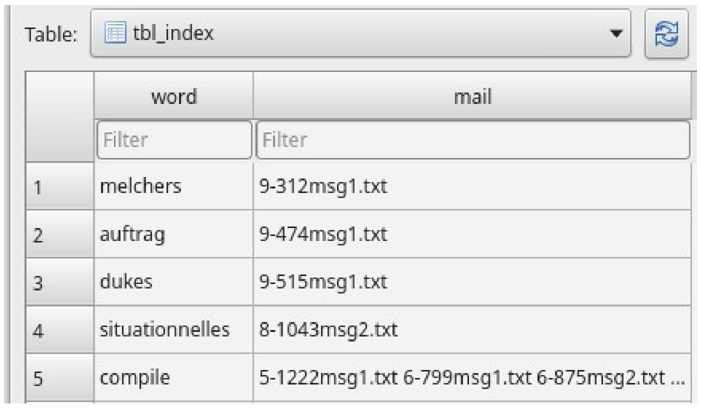


	word	mail	frequency
1	keplerstrasse	9-314msg1.txt	0
2	schrodt	9-314msg1.txt	0
3	pittner	9-314msg1.txt	0
4	beninca	9-314msg1.txt	0
5	padua	9-314msg1.txt	0

Figure 2: Small sample of boolean table

[2] – An example of an information retrieval problem, 2008 Cambridge University Press, 2009-04-07

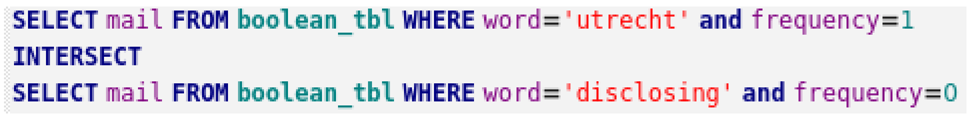
The index table contains the term as a primary key and the documents it is exists in as value of the record. The documents in the value column are separated with whitespaces.



	word	mail
	Filter	Filter
1	melchers	9-312msg1.txt
2	auftrag	9-474msg1.txt
3	dukes	9-515msg1.txt
4	situationnelles	8-1043msg2.txt
5	compile	5-1222msg1.txt 6-799msg1.txt 6-875msg2.txt ...

Figure 3: Small sample of index table

For querying the database we have constructed a method to retrieve emails that contain or dont contain specific terms. A query for mails containing the term utrecht but not the term disclosing is executed as such:



```
SELECT mail FROM boolean_tbl WHERE word='utrecht' and frequency=1
INTERSECT
SELECT mail FROM boolean_tbl WHERE word='disclosing' and frequency=0
```

Figure 4: Querying the email database

Now that we have run the query into a small part of the dataset due to computational workforce limitations which justifies the limited number of emails we get as a result to our query.

Naive Bayes Classifier component

I. Component Description

An important aspect of a successful mail provider is the ability to distinguish and block spam mail arriving at a users mailbox. For this we decided to implement the Naive Bayes classifier due to its robust results and simple implementation. The model uses 90% of the dataset in order to train and then verifies its accuracy testing the rest 10% of mails.

The Naive Bayes model takes into account the probabilities of particular terms occurring in spam and legitimate mail. After training we can link the probability for each term occurring in a legitimate or a spam email. Viewing the mail as a bag of words we combine the probabilities of the terms to calculate the posterior probability computed using Bayes' theorem. Comparing the probability of the mail being spam with the probability of it being legitimate we can claim for the legitimacy of the email.

$$\Pr(S|W) = \frac{\Pr(W|S)\cdot\Pr(S)}{\Pr(W|S)\cdot\Pr(S) + \Pr(W|H)\cdot\Pr(H)}$$

The following is the probability of a mail being legitimate using the probabilities of the terms it contains.

$$p = \frac{p_1 p_2 \cdots p_N}{p_1 p_2 \cdots p_N + (1-p_1)(1-p_2)\cdots(1-p_N)}$$

II. Component Validation

The results of the model were tested and improved by limiting the terms used to calculate the overall probability. This also prevents the implementation from Bayesian poisoning, a technique used by spammers in an attempt to degrade the effectiveness of spam filters[3] including terms used in legitimate mails to manipulate the overall probability of a mail being spam. Additionally this leads to noise reduction and ultimately gave better results to our model.

[3] Naive Bayes Spam Filtering - https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering

Below are the graphs presenting the overall accuracy of our mail filter and on the testing data of 290 emails. The left graph features a 98% accuracy on legitimate mail and the right one a 88.7% accuracy on spam mail.

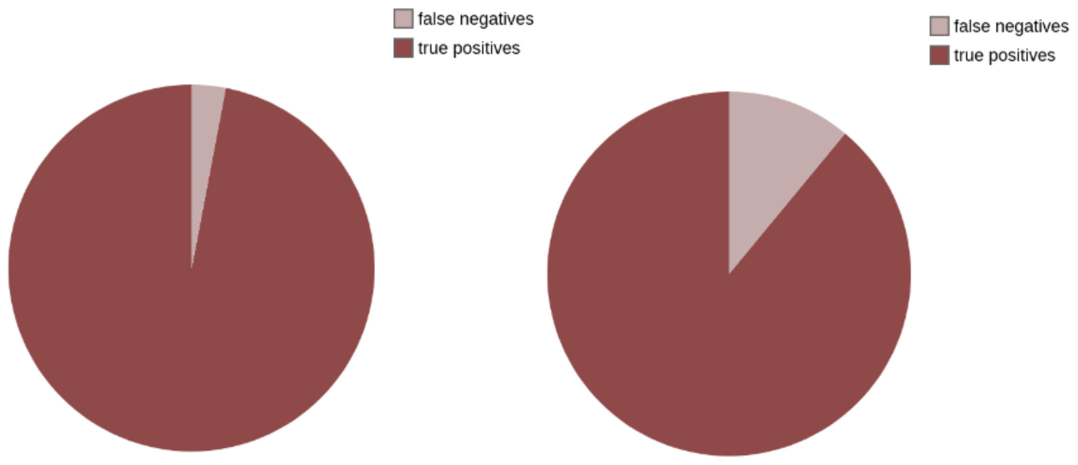
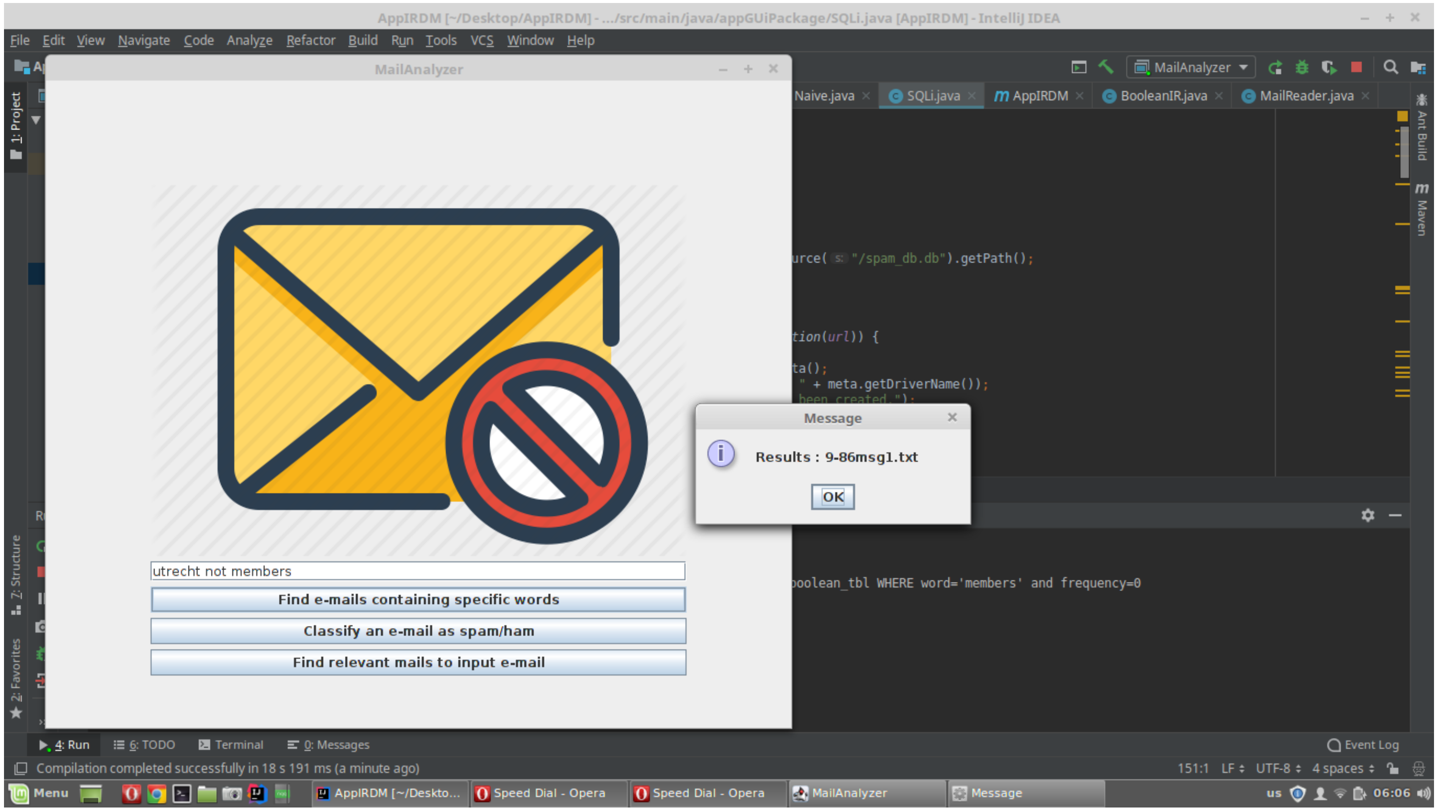


Figure 5: Spam prediction results on test data

Java Application component

I. Component Description

This section of the report offers some basic documentation on the Java application and its functionality. Due to limited time we implemented a basic UI featuring the main functions of our project such as mail filtering and mail querying. The application was created using Javax Swing.


AppIRDM [~/Desktop/AppIRDM] - .../src/main/java/appGUiPackage/SQLi.java [AppIRDM] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MailAnalyzer
Naive.java
SQLi.java
AppIRDM
BooleanIR.java
MailReader.java
Message
OK
Find e-mails containing specific words
Classify an e-mail as spam/ham
Find relevant mails to input e-mail
4: Run
6: TODO
Terminal
0: Messages
Event Log
151:1 LF UTF-8 4 spaces
Menu
AppIRDM [~/Deskto...
Speed Dial - Opera
Speed Dial - Opera
MailAnalyzer
Message
us 06:06
Ant Build
Maven
1: Project
Z: Structure
2: Favorites

```
urce( "/spam_db.db").getPath();

tion(url)) {

ta();
" + meta.getDriverName());
been created.");
```

Results : 9-86msg1.txt

utrecht not members

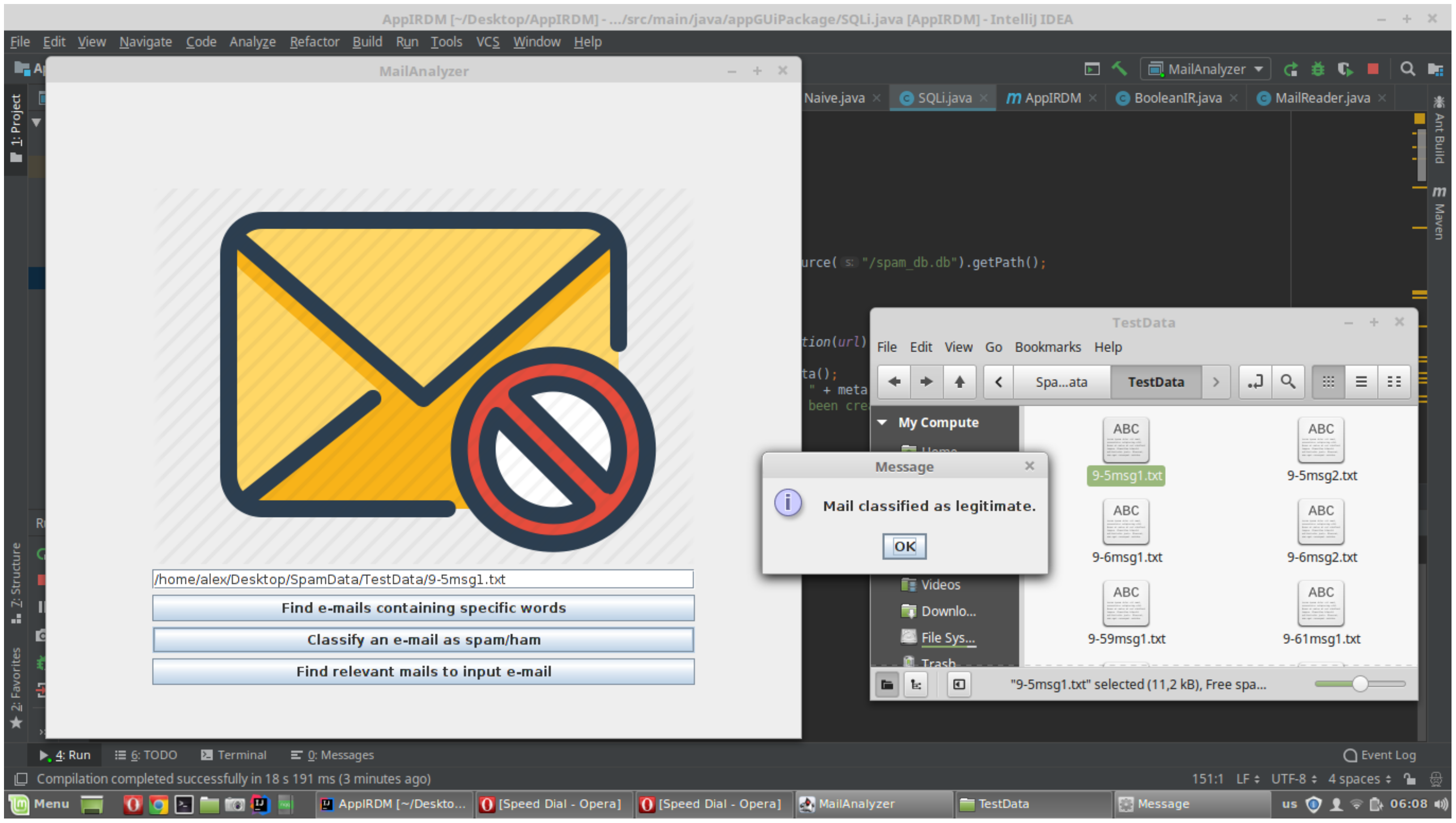
```
boolean_tbl WHERE word='members' and frequency=0
```

Compilation completed successfully in 18 s 191 ms (a minute ago)

Figure 6: Querying data through the application

The above image presents the functionality of the Boolean IR component where a query is given as input to the database and a result is received. The query “utrecht not members” returns those emails which contain the term “utrecht” but not the term “members”. Due to limitations on our end machines the boolean table in the SQLite database only contains a small fraction of the 1.1 billion records it was estimated to store in order to serve the complete mail dataset of 2.700 mails. For this, specific queries are suggested to be executed in order to retrieve results. The implementation suggests that the terms entered in the query are contained in the same mail (intersect sql command) thus the limited number of results.

Below, the functionality of the email filter is presented. The absolute path of the email is used as input in the application and a message is handed as an output declaring the legitimacy of the specific email. Note that on the first time the user requests the classification of an email, the application will execute the machine learning process of training the 90% of the dataset, compute its accuracy on the rest 10% test data and finally use the term frequencies in order to estimate the class of the given email.



```
urce( "/spam_db.db").getPath();
tion(url)
ta();
" + meta
been cre
```

/home/alex/Desktop/SpamData/TestData/9-5msg1.txt

Mail classified as legitimate.

"9-5msg1.txt" selected (11,2 kB), Free spa...

Compilation completed successfully in 18 s 191 ms (3 minutes ago)

151:1 LF ÷ UTF-8 ÷ 4 spaces ÷

Figure 7: Spam mail detection on our application

The above functionality can be used to test any email in txt format giving the absolute path of file to the search bar of our application.