Eindhoven University of Technology

Email Filtering and Querying

2IMM15 Web Information Retrieval and Data Mining

Alex Anthis, Chun Li, Kostantinos Messanakis, Tong Wu

April 3, 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, detailed step by step estimation approach, 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.

Contents

1. 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 and Python was used as the codebase as well as SQLite library for the creation of the database.

1.1. Motivation and Goals

For the specific project, our group needs 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
3. Building a clustering algorithm for the dataset
4. Create a Word to Vector database
5. Create a RNN for spam-ham classification utilizing the Word to Vector database

2. Architecture

As an overview of the project components (Figure 2.1), 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 e-mails. The Naive Bayes implementation focuses on the classification of each e-mail as spam or non-spam assembling the functionality of an e-mail filter. The clustering component performs K-means algorithm to obtain clusters of e-mails with similar context and provides the results to the Java Application to implement one of its functionalities. For an alternative approach, Word2Vec is used to vectorize the input text, to enable the application of RNN classification model and detect spam e-mails.

3. Prior Work

The dataset is a preprocessed subset of the Ling-Spam Dataset, provided by AUEB professor Ion Androutsopoulos. The legitimate emails are extracted from public archives of mailing lists and more topic specific resources. It is based on 960 real email messages from a linguistics mailing list. Each message exists in a separate text file. The total size of the data is approximately 2700 emails. The dataset contains approximately 2300 non-spam and 400 spam emails.

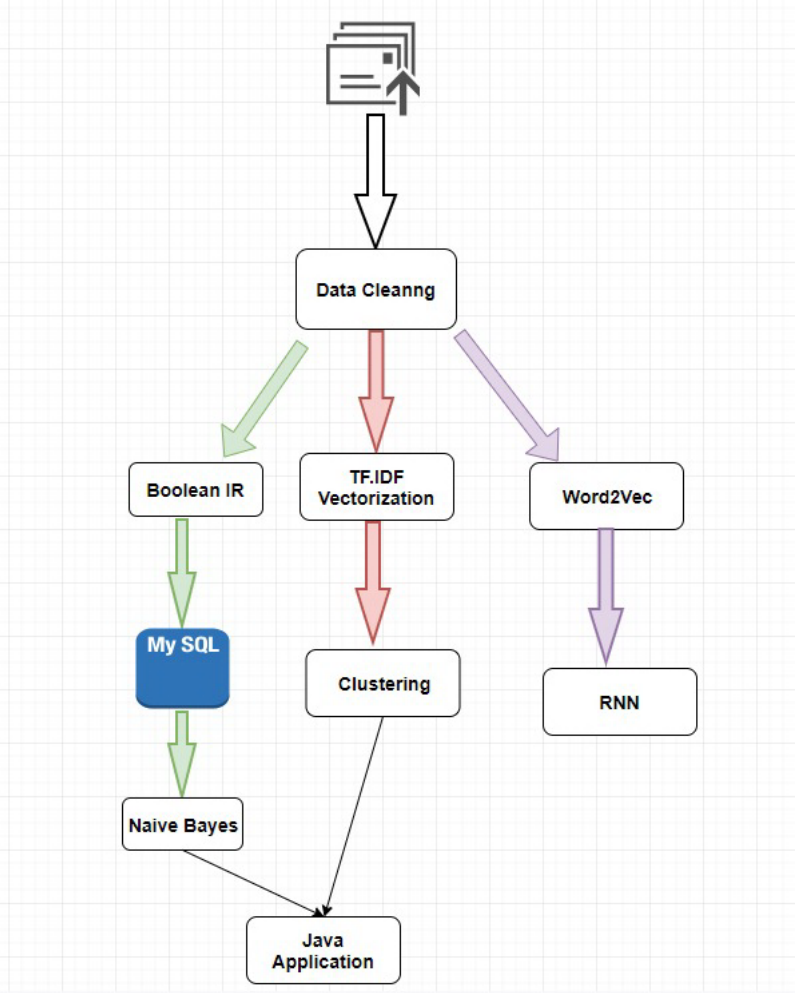

Figure 2.1: Project Architecture

On the Boolean IR and Naive Bayes components the implementation was hardcoded in Java without the use of external libraries. However, when implementing the Naive Bayes filter, the following pages were referenced:

- `https://en.wikipedia.org/wiki/Naive_Bayes_spam_filtering`
- An example of an information retrieval problem, 2008 Cambridge University Press, 2009-04-07

On the clustering component the implementation was also coded from scratch without the use of a library or framework. The following links are referenced:

- `https://en.wikipedia.org/wiki/K-means_clustering`
- `https://en.wikipedia.org/wiki/Dunn_index`
- `https://www.researchgate.net/post/How_can_we_say_that_a_clustering_quality_`

`measure_is_good`

The implementation of Word2Vec is a work based on:

- Chris Moody, Introducing our Hybrid lda2vec Algorithm, San Francisco, CA, United States, May 27, 2016, `https://multithreaded.stitchfix.com/blog/2016/05/27/lda2vec/`
- Rong, X. (2014). word2vec parameter learning explained. arXiv preprint arXiv:1411.2738.

The libraries or packages used are NLTK, NumPy and Gensim.

The RNN implementation makes use of TensorFlow, Keras, and scikit-learn.

4. Components

4.1. Boolean IR

Author: Alex Anthis

4.1.1. Component Motivation

The motivation behind our team implementing a Boolean IR system is to be able to query the data set and retrieve relevant email containing specific terms. Considering the methods of retrieving large sets of documents with their own draw-backs as well their advantages, we choose to build a Boolean IR model as it serves the reasons of simplicity and accuracy on the queries we will subject it in order to retrieve desired documents.

4.1.2. Problem Formulation

In order to process queries on the data set of emails we have to be able to store and retrieve 425 thousand terms contained in 2700 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 data set 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.

4.1.3. Component Approach

The creation of such a system manifests through parsing the data set and building the required tables in the database. In our case, the model consists of the document names (emails) as well as the terms contained in those. 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 exists in the specific email d. Otherwise it is 0.

Table: boolean

	word	mail	frequency
	Filter	Filter	Filter
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 4.1: Small Sample of Boolean Table

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. The index table contains the term as a primary key and the documents it is exists in, as the 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 4.2: Small sample of the Index table

For querying the database we have constructed a method to retrieve emails that contain or exclude specific terms. A query for e-mails containing the term “utrecht” but not the term disclosing is executed as Figure 4.3.

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

Figure 4.3: Querying the E-mail Database

4.2. Naive Bayes Classifier

Author: Alex Anthis

4.2.1. Component Motivation

An important aspect of a successful mail provider is the ability to distinguish and block spam mail arriving at a users mailbox. By building a classifier for that reason we are able to offer insight on whether an email is legitimate or not by running a query to the application providing the txt file.

4.2.2. Problem Formulation

The Naive Bayes model takes into account the probabilities of particular terms occurring in spam and legitimate mail. The model uses 90% of the data set in order to train and then verifies its accuracy testing the rest 10% of mails. 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 (Equation 4.1). Comparing the probability of the mail being spam with the probability of it being legitimate we can claim for the legitimacy of the email.

$$Pr(S|W) = \frac{Pr(W|S) \cdot Pr(S)}{Pr(W|S) \cdot Pr(S) + Pr(W|H) \cdot Pr(H)} \tag{4.1}$$

Equation 4.2 is the probability of an e-mail being legitimate using the probabilities of the terms it contains.

$$p = \frac{p_1 p_2 \dots p_N}{p_1 p_2 \dots p_N + (1 - p_1)(1 - p_2) \dots (1 - p_N)} \tag{4.2}$$

4.2.3. Component Validation

The results of the model were tested and improved by limiting the number of terms used to calculate the overall probability. With this method we use a top performing set of words rather than a more exhaustive list. This also prevents the implementation from Bayesian poisoning, a technique used by spammers in an attempt to degrade the effectiveness of spam filters 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.

Table 4.1 is the confusion matrix 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.

The results can be improved by the use of different frequency and occurrence models like the n-gram model in order to take into account whole phrases instead of just single words. To eliminate the 2% of legitimate mail targeted as spam, weighing terms according to their contribution in the distinction between spam non spam could be used.

		Predicted	
		Non-spam	**Spam**
Actual	**Non-spam**	0.98	0.113
	Spam	0.02	0.887

Table 4.1: Confusion Matrix

4.3. Clustering

Author: Konstantinos Messanakis

4.3.1. Component Motivation

Considering the fact that the goal of this project is to analyze and handle the data of an e-mail dataset we decided to put the data through a clustering analysis in order to create groups of e-mails (clusters). Emails belonging to the same clusters are expected to be more similar (in some sense) to each other than to those in other groups(clusters). More specifically, the results of the clustering are used in our Application, where the user can:

1. Give an e-mail as an input and retrieve all (or can specify the number) the e-mails that are similar in some sense to the input e-mail(e-mails in the same cluster)
2. Retrieve a set containing all the most significant words that makes his e-mail be similar to other e-mails

4.3.2. Component Problem Formulation

The problem that this component is trying to solve is the following.

Given a sample of e-mails:

1. Preprocess data to obtain them in a way that the clustering algorithm can work with them efiiciently
2. perform a clustering algorithm in a way that all the e-mails in the same cluster share some similarities
3. Get the clustering results and use them in the Application
4. Try to evaluate the results of the clustering algorithm

4.3.3. Component Approach

The first thing to do, was to obtain a suitable representation of our data, in order to perform our clustering algorithm. To achieve that, we implemented a TF-IDF Calculator in order to obtain our e-mails as mutually comparable TF-IDF Vectors.

After the TF-IDF vectorization we implemented (from scratch) the K-Means clustering algorithm ($K = 5$) and obtained the e-mails clusters (Equation 4.3).

$$W_{j,i} = TF_{j,i} \times \log \frac{N}{DF_j} \tag{4.3}$$

After the TF-IDF vectorization we implemented (from scratch) the K-Means clustering algorithm ($K = 5$) and obtained the e-mails clusters

4.3.4. Justification of Choosing TF-IDF and K-Means

TF-IDF We decided to implement the TF-IDF algorithm because of its usefulness to figure out the most important words in a big context, and thus spot the similarities between the emails.Another important factor was that when using TF-IDF we didn't have to worry about some stopwords that every email could contain

K-Means Algorithm We decided to implement the K-Means algorithm instead of another algorithm because we wanted to code the algorithms from scatch and it seemed the simpler choice among the other clustering algorithms.Furthermore, K-Means performs really well with big loads of data, and is not really exensive in terms of speed and memory

4.3.5. Component Evaluation

In order to evaluate the results of the clustering we calculated the Dunn Index metric. The result of this metric is base on the clustered data itself. The aim is to identify if the clusters are compact and well separated

$$DI_m = \frac{\min_{1 \leq i < j \leq m} \delta C_i, C_j}{\max_{1 \leq k \leq m} \Delta_k}$$

The Dunn Index of our algorithm gives a score of: 0.6907953435513402(69%)

The performance of the algorithm is not optimal. This is mainly because when testing the clustering results, we observed that K-Means doesn't work really good with multidimensional data(Our data were TF.IDF Vectors).We tried to perform some techniques in order to reduce the dimension of the Vectors, for example using Python's PCA(Principal Component analysis) but this approach was never completed due to time limitations

4.3.6. Remarks about the Component

Clustering was implemented on clean data after stop word removal.

The clustering was performed using a data sample of 300 e-mails, mainly because the load of the e-mails was big we were facing a heap out of Memory exception when executing the clustering.

4.4. Data Cleaning

Author: Chun Li

4.4.1. Motivation

There are two reasons for reprocessing the data. First reason is that prepossessed data will improve the efficiency of the model training. The other reason is making model training more effective.

4.4.2. Problem Formulation

In the data context, there are some words existing frequently but not important for analyzing, like: "I", "is", "we" etc. Also there are some word they looks different but they have same meaning, for example: "includes" and "include". To solve these problems, we need to reprocess these words.

4.4.3. Component Approach

Python package NLTK is used here for implementation. In the end, all the data is stored as lists of the list in JSON file. Each list represents one Email. The final data has the following characteristics:

- No stopping words
- No symbols
- No Numbers
- Length of each word is longer than 2
- In each mail (list) only contains the unique words
- All the words are through the process of determining the lemma

4.5. Word2Vec

Author: Chun Li

4.5.1. Motivation

We would like to learn what is the relationship among the words, and transform the text into featured for a classifier. In the end, the results should be compared to finding

out the best model for mail filtering. Therefore, word embedding is one option for the information retrieval of this project.

4.5.2. Problem Formulation

Word embedding is a common way to process natural language. It is processing natural language to computers. There are two models considered in the beginning, namely, word2vec and Ida2vec. Word2vec is predicting the relationship between the word and its neighboring words. While lda2vec was a model combing LDA and word2vec. LDA (Latent Dirichlet Allocation) is classifying text in the document. Therefore, compare to word2vec, lda2vec is not only predicting local words but also the whole document.

Obviously, lda2vec is a very powerful model, but from Chris Moody's *Introducing our hybrid lda2vec algorithm*, he suggested do not use lda2vec for practical reasons, for example, it needs a GPU in your desktop and so on. Also, limited information can be searched about lda2vec. Therefore, word2vec was used in this project.

4.5.3. Component Approach

Word2Vec is a three layer neutral network. First layer is Input Layer, the second one is hidden layer, and the last one is output layer.

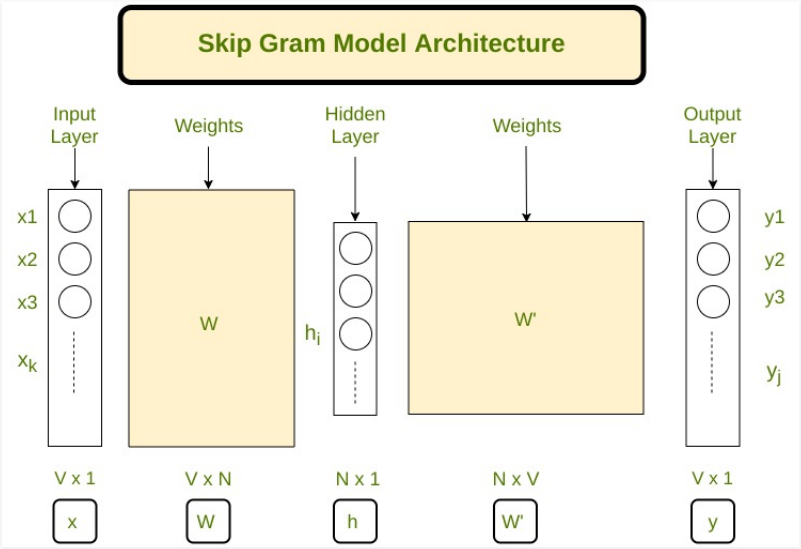

Figure 4.4: Word2Vec CBOW and Skip-Gram Network Architectures

In Word2Vec, there are two models: bag of words (CBOW) and Skip-gram model. CBOW model is using neighboring words to predict central word, as to Skip-gram model is using central word to predict neighboring words. In this project, skip-gram model is

used. Because, although CBOW model has better performance on speed, but not on rare words prediction.

The concept of central word and context words can be though as a sliding window that cross the text, for example, the sentence "I attended the web information retrieval and data mining course". If "web" is central word and window size is 2, then "attended", "the", "information", "retrieval" are the context words.

Like we introduced above, the natural word cannot be understood by computers, so we need to convert them as numbers. Here we employ one-hot encoding. One-hot encoding is a vector, length is the total number of unique words in the text. Every word in this vector has a corresponding position which value is 1, and all the rest will be 0, see Figure 4.5 below. The one-hot encoding vector will be the input, and it will gain the weights and go to the hidden layer, the output of the hidden layer will get another weights then feed into the output layer.

Figure 4.5: Word2Vec One-Hot Encoding Example

Writing from Scratch Using NumPy Word2Vec is implemented in Python. In the beginning, it is implemented from scratch, only Numpy package used, but during the training process, I met some difficulties, like long training period; results are not satisfied, etc. Therefore, I also used package Gensim to train the model.

The functions are built for implementation are: `generate_one_hot`, `forward_pass` and `back_pass`. The function `generate_one_hot` is used to generate one-hot vector; `forward_pass` get a vector; and `back_pass` is used to adjust the the weights W and W . For training the model, we tried to get the window size from range 2 to 15, and initialize weights is randomly get by `np.random.uniform()`. As to the hidden node is by root square of the input node plus a, and a could be from range 1 to 10.

Implemented word2vec from scratch helps me to understanding word embedding better and also for word2vec itself. But in piratical, it needs long training period and also big data set to get the better feature of the words. So for the whole project, I used Gensim package later.

Gensim Package Gensim pacakage is much more efficient and more practical. Model word2vec is used here, and after training, window size is 10. We are going to show the graph of visualization first 200 word in the data set in the figure 6.The top 5 similarity

words for “first” in our data set are “slapping” (0.391) , “permentiers” (0.336), “before”’ (0.334), “foreignborn” (0.328), and “casestudies” (0.317) .

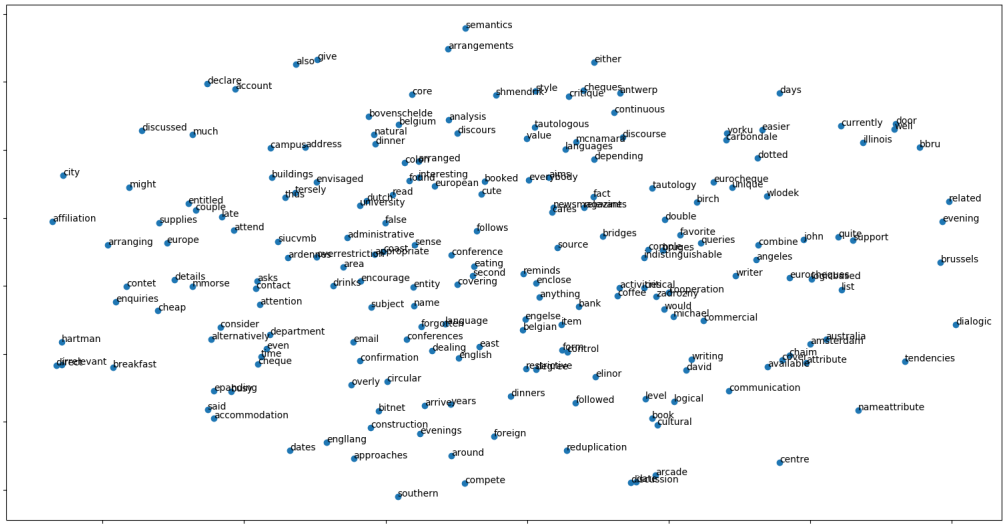

Figure 4.6: Visualization of Closest 200 Words According to Word2Vec

4.5.4. Evaluations

Writing from Scratch In order to get the optimal window size and weights, in the implementation from scratch, calculating the loss value by loss formula. The table below shows part of the result based on the window size.

Window Size	**Loss Percentage**	W	W'
2	0.3%	[[0.43056057 ...]]	[[-0.7859653 ...]]
2	0.6%	[[0.41652962 ...]]	[[-0.785966 ...]]
...	...	...	...
2	58.6%	[[0.41652962 ...]]	[[-0.7456865 ...]]
...	...	...	...

4.6. RNN for Classification

Author: Tong Wu

4.6.1. Component Motivation

One of the goal of this project is categorizing e-mail (ham or spam), given its textual representation. Word embedding transforms textual information into N-dimensional (N-D) vectors. A sentence is a sequence of words, therefore it can be mapped to a sequence

of N-D vectors. Then the problem transforms into how to utilize this sequence, to reveal implicit characteristics of sentences. For example, by categorization or prediction.

Neural networks (NN) are suitable to solve this kind of problems. With cells and layers, a NN is fundamentally an implicit function with many hidden inputs. In this project, the materials are represented by text. From the view of each word, it has a context, and an order in the sentence, just as human usually process words one by one. Using a NN with consideration of sequence order should be able to make good evaluation of the sentences. Recurrent neural network (RNN) is a suitable choice.

4.6.2. Problem Formulation

The problem here is another way of classification, using NN. The final result is a trained RNN model, which accepts text (e-mail body) as input, and classifies whether the e-mail is a spam or not.

4.6.3. Model Construction

RNN has recurrent structure that allows it to generate output based on current input and previous input(s). This can have interesting applications, for example, evaluating a word based on its immutable characteristic (e.g. spelling) and previous words (i.e. context).

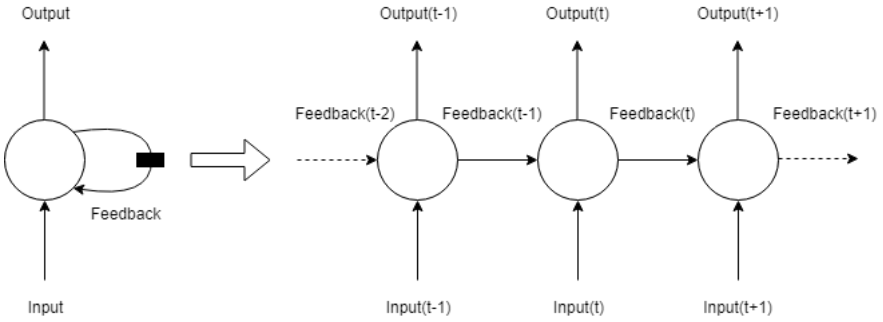

Figure 4.7: RNN Structure

The model structure is shown in Figure 4.8.

The embedding layer is used to lower the dimensions of input, and discover the implicit correlations between words. Then, the RNN is used to process word sequence (vector sequence), which plays a role of understanding the context, with sliding windows. Next the hidden and activation layers are used to form weights ("learning"), and the dropout layer is used to clean the learning result ("forgetting the errors"). In the end the output layer combining with an activation layer (which uses sigmoid function for activation) do the classification. The output is evaluated by a loss function to correct the hidden parameters inside the model.

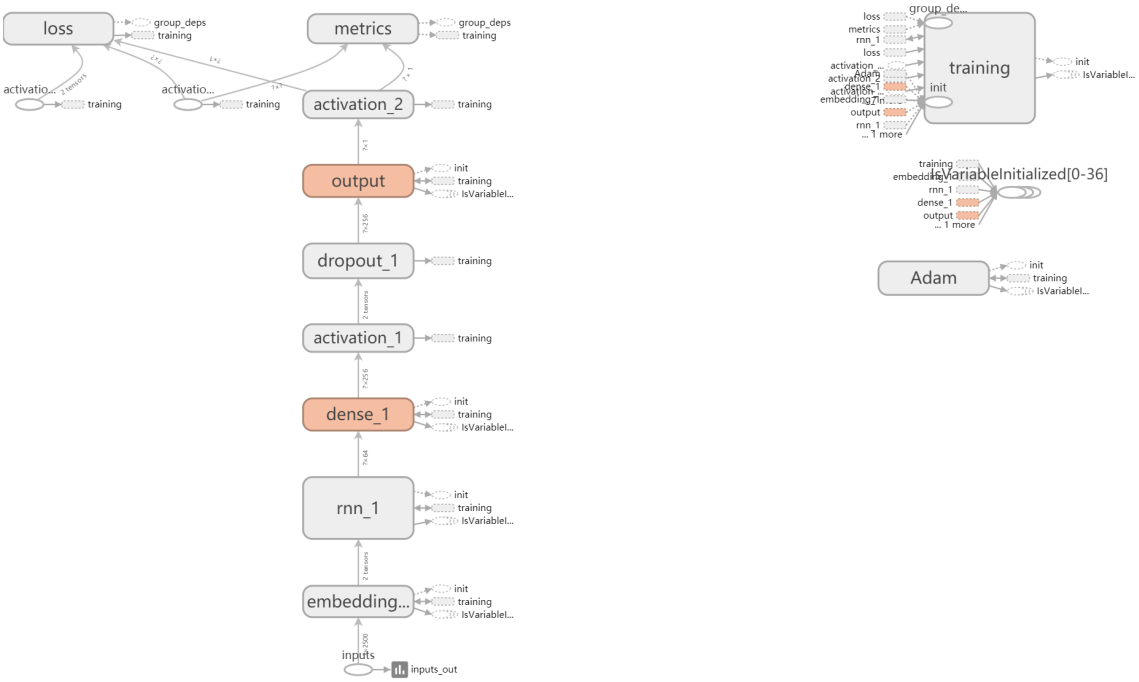

Figure 4.8: The RNN Model

4.6.4. Implementation

First, a trained Word2Vec model is loaded or generated. Then, a vector of N-D vectors is constructed using this method: for each email of all emails, retrieve the first M words of its body; if the length L is shorter than M, pad it to M by adding $M - L$ zero N-D vectors. During this process, label all the N-D vectors as 0 (non-spam) or 1 (spam). The N-D vectors are served as input and labels are expected output.

Next, split the generated input/output set into training set (70%), validation set (15%) and testing set (15%). Training set and validation set are provided as a whole but they are shuffled (i.e. computing a new set and its complement from the partition ratio) in each run.

Finally, fit the model and evaluate. The RNN is implemented using Keras, with backend of Tensorflow.

4.6.5. Evaluation

From multiple runs of the model (Figures 4.9 to 4.12).

The general performance is not as good as expected. Test set accuracy falls between 0.83 and 0.85, which is lower than common tasks using neural networks (> 0.9), even

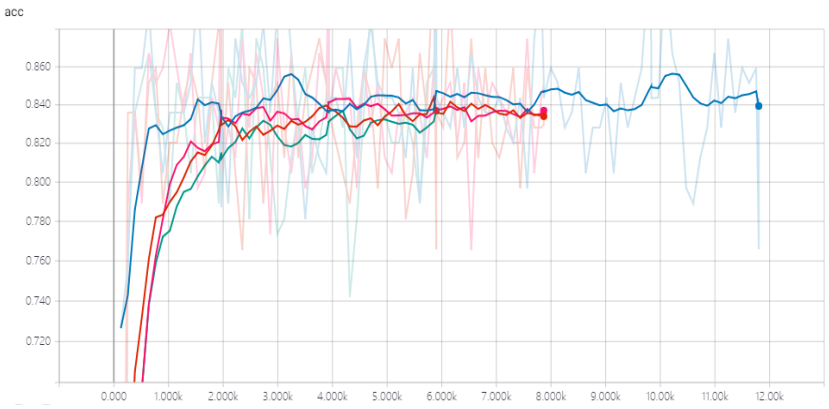

Figure 4.9: Training Accuracy

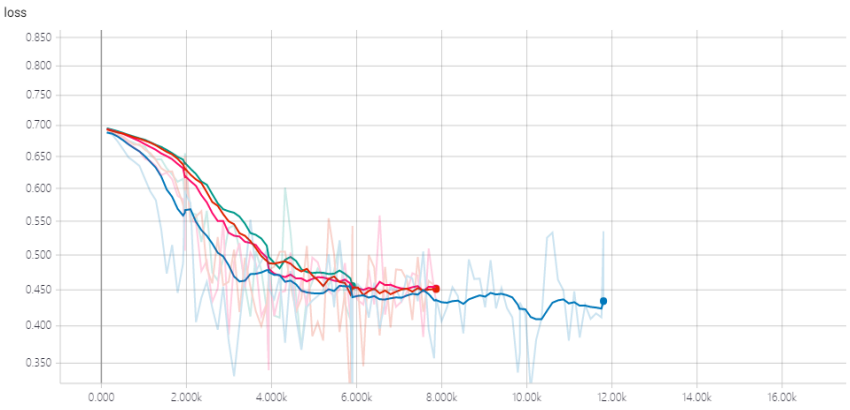

Figure 4.10: Training Loss

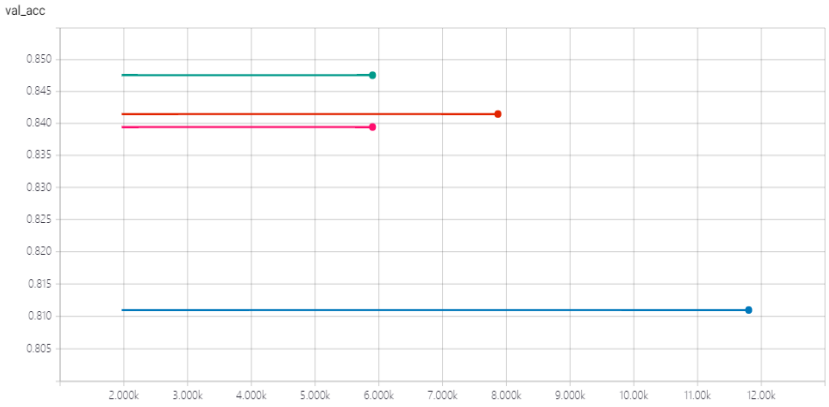

Figure 4.11: Validation Accuracy

worse than the Naive Bayes filter (see section 4.2). Also, the test set loss is too high. Using binary cross entropy as loss function, test set loss varies from 0.43 to 0.45. Considering

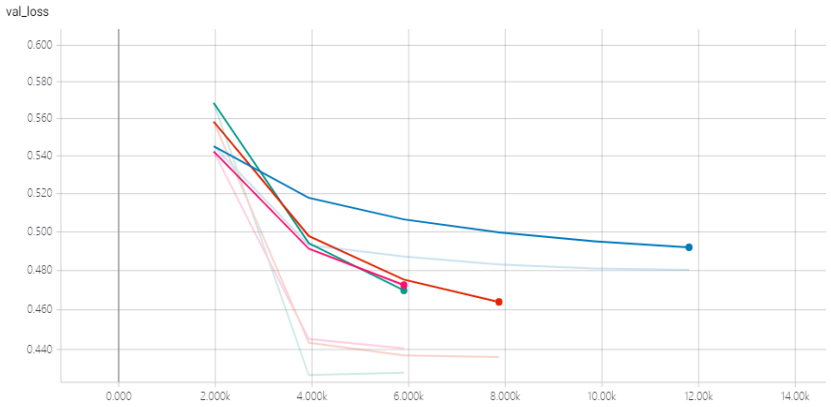

Figure 4.12: Valiation Loss

that for such loss function $\log N$ should be the upper bound of loss, where N is the number of output categories ($N = 2$ in our case). It means the model is not certain about which category should test inputs lie in. Loss of validation set and training set are close, which indicates that the model may be underfitted.

There should be three possible causes for the result.

First, the model is underfitted. The model structure is naive and relatively simple. However, even if more hidden layers are added (in places between output and RNN), the performance does not improve.

Second, the Word2Vec model is not accurate. Word2Vec generates models with good quality, when the input is properly preprocessed. But in our data, the e-mails are hard to preprocess. For example, there are numbers, symbols (serve as content delimiters, in common emails), manual confusion by adding spaces inside words, and lexical changes (partially reduced by lemmatization).

Third, improper model parameter selection because of hardware limitation. In real runs, the number of words cut from each e-mail body, and/or the length of word vector, must be limited. Otherwise, training requires huge amount of memory and an out of memory exception is easily thrown. When using Gensim's Word2Vec implementation, the number of words is set to 100 and vector length is set to 25. When using the Word2Vec implemented from scratch, the values are 18 and 150. For a general length e-mail with medium complexity, none of the settings seems to be able to capture the characteristics of e-mail texts well. This issue should be able to solve using better device, but unfortunately, we do not have one.

4.7. Java Application

Authors: Alex Anthis & Konstantinos Messanakis

4.7.1. 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.

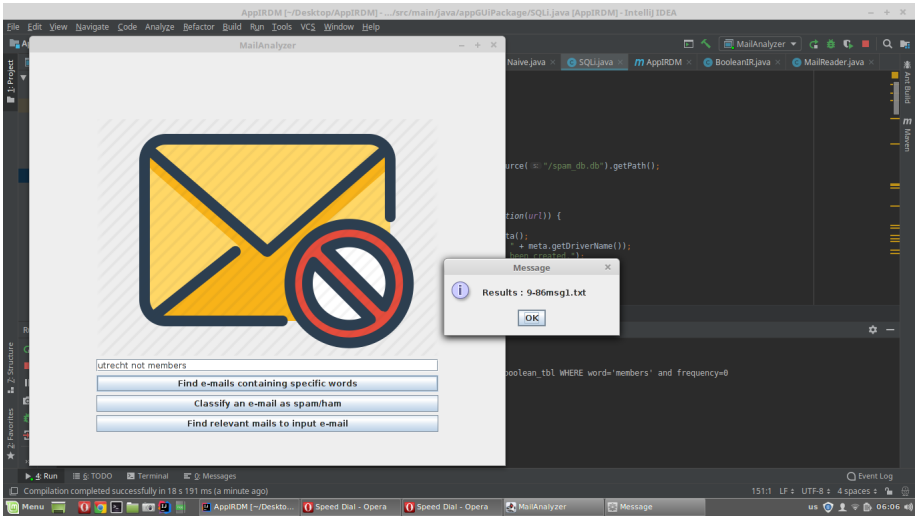

Figure 4.13: Querying Data from the Application

The above image presents the functionality of the Boolean IR component where a query is given as input to the database and an result is received. The query "utrecht not members" returns those emails who 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 2700 mails. For this specific queries are suggested to be executed in order to retrieve results. The functionality provided 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 served 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 presses the classification button the application will execute the machine learning process of training the 90% of the dataset, compute its accuracy on the rest 10% using it as test data and finally use the probabilities per term in order to estimate the class of the given email.

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

Below the functionality of the clustering's component functionality is presented. The absolute path of the email is served as input in the application and a message is handed

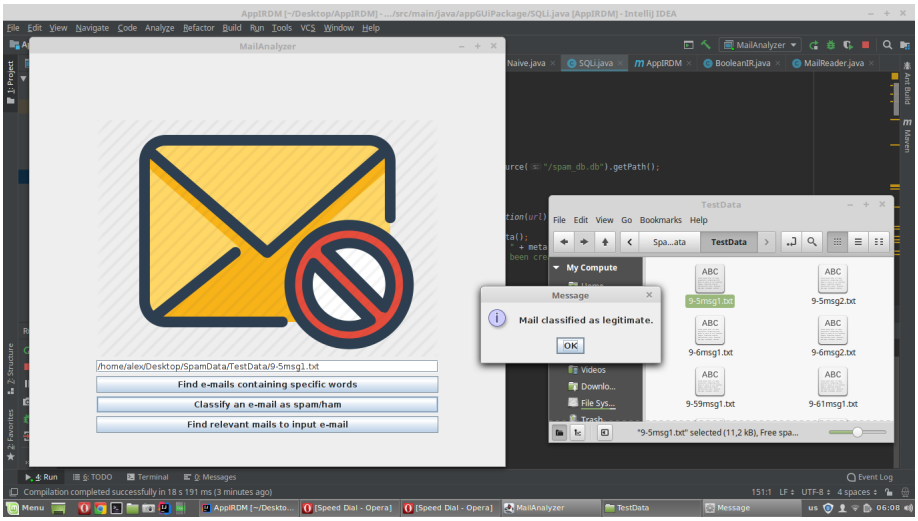

Figure 4.14: Spam Mail Detection by the Application

as an output showing a list of emails similar to the input e-mail (most similar mails in the same cluster).

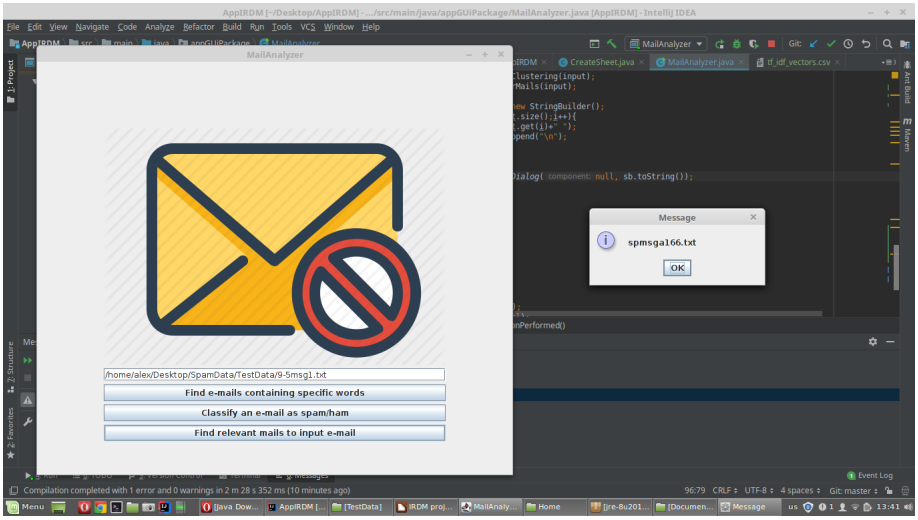

Figure 4.15: Retrieving Similar Mail by the Application

5. System Summary

The main result is that every component designed is able to work, but at different levels. Boolean IR and the Naive Bayes filter are fully functional and work as expected. These functions are also presented by a fine GUI application. Word2Vec is implemented but

the result cannot be easily verified. However compared to a "standard" implementation in Gensim it should be working correctly. The RNN model can be treated as a classifier but its accuracy is lower than the Naive Bayes filter, due to some possible factors.

For the whole project, one apparent limitation is there are two main languages (Java and Python) in used in the project, which are hard to interop. This limits the reuse of components - an language adapter was designed and implemented at first, but not used later in the project. And that leads to complete different usage of frameworks and libraries, and the only feature that can be related is the spam filter accuracy.

Another limitation is the implementations may not be well optimized. For a vocabulary of 48000 words and 2700 documents, searching the whole database takes unacceptable amount of time. However, as the method itself has been proved correct, it can be improved by using more optimized algorithms, or using devices with higher specs.

6. Conclusion

In this project, we try to use web information retrieval and data mining knowledge built one application to do email filtering and querying. The project used three models Naïve Bayes, Clustering, and RNN. These three models fed by different information retrieval methods. Naïve Bayes gets the data by Boolean IR; clustering gets the data by TF-IDF vectorization; and RNN gets the data by word2vec. Based on final results, Naïve Bayes is the optimize model we would like to suggest.

During the project, we met some challenges. One of them is the big size of the data set, it requires long time period to train the models, feeding all the data into database and perform the TF-IDF vectorization. Also, the project was implemented using two languages, python and java. In the end,because of time limitation, we were not able to integrate them together, so in the application demo we only show the app built in java.

For future research, the project can focus on training the models based on a bigger data set, as well as integrating the two languages components into one.

Appendices

A. Java Application Installation Guide

The project can be loaded as an IntelliJ project on Linux environment with Java 1.8 installed. Once compilation build is completed the application can be launched by executing the `MailAnalyzer.main()` method. Doing so must display the main window of the application. From here you can enter the desired queries or file paths in the search bar and explore the different functionalities our application has to offer.