University of Applied Sciences and Arts
Northwestern Switzerland

Development and Evaluation of a Java-based Spam Detector

Authors

Jibin Mathew Peechatt

Mike Rey

Sascha Vetsch

Instructor

Prof. Dr. Bradley Richards

Course

Programming 2-BIT FT 2023

Date of submission

07.05.2023

Table of Content

1. Introduction 3
2. Theoretical Foundation 3
3. Practical Implementation 4
4. Results 5
5. Summary 6
6. References 7
7. Table of Figures 8

1. Introduction

In today's information age, the exposure and declaration of spam emails play a significant role in people's lives. Not only because of the useless information spam contains, but also because the recipient may open a malware-containing link or mistakenly enter sensitive information. To respond to this issue, a spam detector is the most obvious solution. And that is exactly what this Java-based application aims to do.

The Java application was developed using the Model-View-Controller (MVC) architecture, which allows efficient separation of concerns and makes the code more maintainable and understandable. The project consists of four primary classes, “spamMain”, “spamModel”, “spamView”, and “spamController”, that all work together to detect spam emails.

The “spamModel” class represents the core data of the application and implements the spam detection logic. It uses methods to scan the email and updates the statistics of the program, such as whether the emails are classified as spam or not. In our case, the “spamModel” has the checker for the domain of the email.

The “SpamView” class enables the user to interact with the program by establishing JavaFX Elements such as Button controls. The spamView class shows us what the application would look like. It would display the parsed email. And the “SpamController” class is the intermediary part between the Model and View class. The “spamController” has a checker as well for the content in the mail, which is not the best practice in MVC, as logic is usually written in the model class. It manages the user input and interacts with the Model and View class in the spam detection process.

2. Theoretical Foundation

The idea is that certain words and phrases are commonly found in spam emails which can be used to identify a specific pattern of spam. So, we used a rule-based approach to identify spam by looking for keywords that are stored in an Array List as String characters and which the program uses to determine whether it is spam or not. The Application uses an MVC architecture as a design pattern to separate the logic of the program into manageable, interconnected components. To detect spam in an email effectively the program must separate the sender's domain and content when it analyzes the target .EML file. An EML is the file format of an email message.

As we had to parse our receiving emails into different parts like to, from, subject, and content, we decided to use the library JavaMail API. It provides a cool interface for sending and receiving emails. With the usage of this API, email parsing has been quite easy.

For larger datasets with more complex detection scenarios, Hash tables and Trees would have been more appropriate and efficient in lookups and insertions. However, as this is a small-scale application where the main manipulation mostly would include adding new keywords at the end of the list, a linear data structure, an Array List, would be the most suitable. They are easy to understand and implement but offer easy access and manipulation.

3. Practical Implementation

Firstly, We decided to implement the Filechooser as we needed to select email files from our system. This was a requirement for the project as well. It was implemented through the javafx.stage.FileChooser. In our program, we have set up a button which on set action, would go to the file chooser. We had to learn how to set the file name filter. This had to be done as we only wanted the EML Format, which is the standard format for an email (*JavaFX FileChooser*, n.d.).

Our spam project returns an output called “Spam Score” of zero or one. When the spam score is at one, the probability of spam is high. Zero, on the other hand, indicates a low spam score. To retrieve and display the mail in the view of the program, Java Mail API is used. As discussed already, this library provides a class that models a mail system. To successfully use the application, Java Mail API and Java Activation Framework (JAF) need to be installed on the user's system. The activation file, from JAF, is essential and must not be forgotten (JavaMail API - Environment Setup, n.d.).

Two labels are set up to inform the user about the spam score. The first label, called “spamScoreLabel” reviews the sender's domain. While the second label called “spamScoreofContent” scans through the content of the email. Therefore, there are two kinds of detection procedures in the program to decide whether the message is spam or not.

In the output of the "spamScoreLabel", the system examines the sender's address listed in the "From:" field and determines whether it is spam or not based on predefined keywords. For this purpose, two Array lists called "goodMails" and "spamdomains" have been created. In the list of "goodMails" common domains which are considered legitimate are contained. In contrast, "spamdomains" contains known domains associated with spam or other malicious activities.

The sender's domain is separated using a Regular Expression and compared against two separate array lists. If the domain is found in the "goodMails" array list, the spam score is set to zero, indicating a low probability that the email is spam. Conversely, if the domain is found in the "spamdomains" list, the spam score is set to one, indicating a high probability that the email is spam. We used Stack Overflow to find the regex which will help us find the domain name from the email (*Regex Get Domain Name from Email - Stack Overflow*, n.d.)**.** We have tested this regex online to make sure it has been working as well (*Regex101: Build, Test, and Debug Regex*, n.d.).

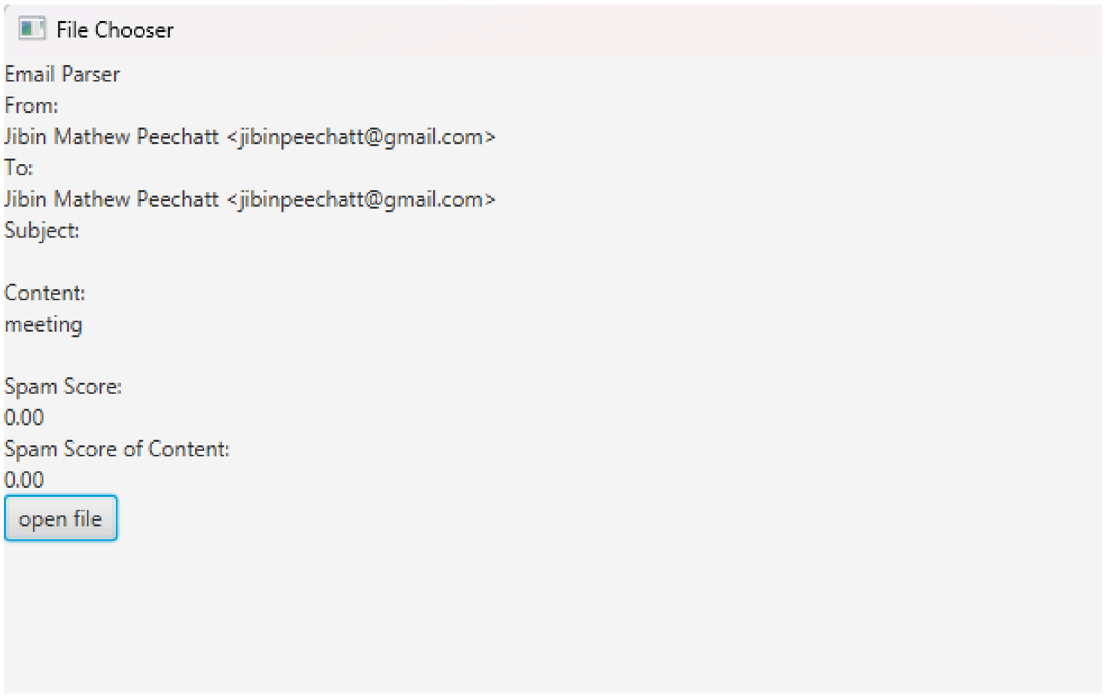
If it is neither of these situations, that is it is not in the goodMails or spamdomains, then the spam score is now set to one and the domain is now written to a text file and added to the “spamdomains” list. The user can ascertain the domain by opening the text file and, if desired, it can be corrected manually. The file’s path would be displayed in the console during such a situation.

For the "spamScoreofContent" output, the email content is analyzed by comparing it against two predefined lists of keywords in the "SpamController" class to determine its spam probability. The first list, "spamKeywords", contains common spam phrases and content, while the second list, "hamKeywords", contains typical keywords found in important or legitimate emails.

The content of the mail received is brought out through the JavaMail API. Now the application checks both lists and displays the respective score. The default spam score is set as 1.0 and can be changed at any time.

4. Results

The main goal of the project was to develop a Java-based application for spam detection in emails by analyzing the sender's content and domain. To achieve the project's goal, the developers decided to use an MVC architecture for the program. This makes the whole program easier to manage and understand, which is an important requirement for a team with beginner-level skills.

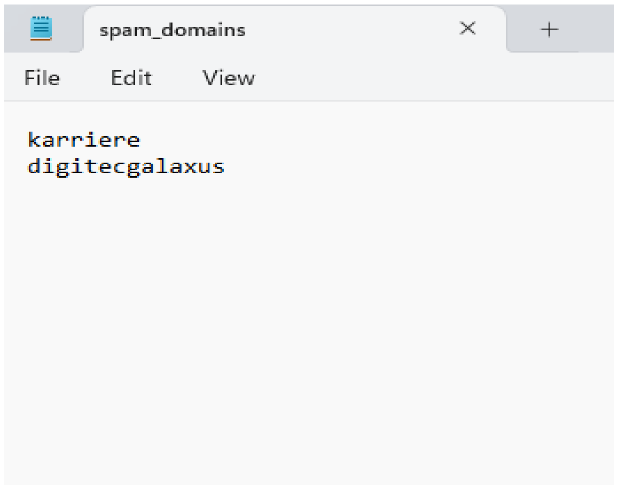


```
File Chooser
Email Parser
From:
Jibin Mathew Peechatt <jibinpeechatt@gmail.com>
To:
Jibin Mathew Peechatt <jibinpeechatt@gmail.com>
Subject:

Content:
meeting

Spam Score:
0.00
Spam Score of Content:
0.00
```

Figure 1: Result of a testing

In Figure 1, we can see that the spam score is 0.0, as the domain, in this case, is Gmail and it is written in our goodMails list. The spam Score of content is 0.0, as “meeting” is defined as a good keyword by our hamKeywords list for the content of a mail. Hence the final result of this mail is that it has a very low probability of being spam.

The written program has proven to be highly accurate and precise in identifying spam in a sender's domain and the content of an email, based on predefined keywords. It is easy to use and runs on every machine, which makes it a convenient tool for spam detection. We have designed the application to allow easy addition and deletion of spam keywords, both for domain and content. If the spam detector detects the domain is neither in goodMails nor spamdomains, the software will automatically generate a text file that will allow the user to see the domain name of the email and also has the option to adjust the keyword list manually in such a case. This allows the program to become smarter with each use and improve its detection capability with the help of the user.



```
karriere
digitecgalaxus
```

Figure 2: The text file that would be generated for domain classification.

However, it is important to note that this is a very small-scale application that is only remotely like a professional spam detector. For this reason, we decided to use a linear data structure, which is easy to handle, but not suitable for larger datasets due to its fixed size and its fundamental properties.

In Figure 2, we can notice the restrictions of the program here. We all know that digitecgalaxus is not a spam domain after it has been written to the file, but it has been set to a spam score of 1.0. This kind of restriction happens in our program, as the checking is based on an array list. We cannot possibly write down the domains of all companies existing in Switzerland. But these can be manually corrected in our program easily. As we know digitechgalaxus is a good domain, we have to manually add it to the array list of goodMails. So with time, our program gets more and more intelligent.

Overall, the spam detection program has demonstrated the effectiveness of pattern recognition based on a set of rules. And we also gained useful insides into the consideration and limitations when developing such tools.

5. Summary

Although our spam detector program has shown promising results, it is not flawless. To achieve a higher level of accuracy, we need to implement advanced coding techniques, or machine learning models, or explore different web packages. However, the file-writing feature of our program enables it to learn and become more intelligent over time, which should be considered an important feature of the software. Also, what we found interesting was Java Mail API, as it was really useful in the whole development of the system.

In addition, we plan to expand our spam detection capabilities to include checking the email header/subject by incorporating two additional Array Lists and two update methods in the spam controller. Unfortunately, due to time constraints, we were unable to complete this task. But we plan to work on this in the future to improve the overall effectiveness of our spam detector.

6. References

Java Archive Downloads - Java Platform Technologies. (n.d.). Retrieved May 4, 2023, from https://www.oracle.com/java/technologies/java-archive-downloads-java-plat-downloads.html#jaf-1.1.1-fcs-oth-JPR

JavaFX FileChooser. (n.d.). Retrieved May 4, 2023, from https://jenkov.com/tutorials/javafx/filechooser.html

JavaMail API - Environment Setup. (n.d.). Retrieved May 3, 2023, from https://www.tutorialspoint.com/javamail_api/javamail_api_environment_setup.htm

javax.mail (Java(TM) EE 7 Specification APIs). (n.d.). Retrieved May 4, 2023, from https://docs.oracle.com/javaee/7/api/javax/mail/package-summary.html

Regex get domain name from email - Stack Overflow. (n.d.). Retrieved May 4, 2023, from https://stackoverflow.com/questions/39027204/regex-get-domain-name-from-email

regex101: build, test, and debug regex. (n.d.). Retrieved May 4, 2023, from https://regex101.com/

7. Table of Figures

Figure 1: Result of a testing 5
Figure 2: The text file that would be generated for domain classification. 6