Spam Filter

January 5, 2025

Šimon Brandner
brandsi1@cvut.cz
Faculty of Electrical Engineering,
Czech Technical University

Jiří Král
kralji17@cvut.cz
Faculty of Electrical Engineering,
Czech Technical University

Abstract

We were tasked with implementing a spam filter in Python in whatever way. We chose the Beautiful Soup library for parsing individual e-mails and Naive Bayes filtering for classification. We achieved fast and accurate results (75% - 95% depending on the dataset). In this report, we also mention other tools used during development as well as the organization of teamwork.

1 Introduction

This semestral work aimed to implement a spam filter in Python – a binary classifier that categorizes emails as either *spam* or *ham* (not spam). Individual emails are represented as text files formatted according to RFC 5322. Given a *dataset* (a directory with e-mails), the spam filter creates a special file `!prediction.txt` (according to the specification) which contains the classification of each e-mail in the form of key-value pairs in the format `<email-file-name> <OK | SPAM>` separated by newlines.

However, before the spam filter can classify e-mails, it must first be trained on a dataset where the correct classification of all e-mails is known beforehand and written in file `!truth.txt` (which has the same format as `!prediction.txt`).

The spam filter adheres to the specification which can be found on the course's webpage and its source code has been made public on GitHub.

2 Implementation

We focus on parsing and filtering in this section as that is the core of the spam filter and the other work that was done is not particularly interesting.

2.1 Parsing

Given an email formatted according to the RFC 5322 standard, we split it into a head section containing all the metadata and a body section containing the content of the email itself.

From the head, we extract the `From` entry, which contains the address of the sender, and the `Subject` entry of the email containing the subject. The `From` entry is mandatory and should be present in every email formatted according to the standard. The `Subject` entry is not mandatory but is present in almost every email. If the `Subject` entry is not present, it is treated as if it was an empty string.

If HTML markup is present in the body of the email, it is removed using Python's Beautiful Soup (1) library like so:

```
from bs4 import BeautifulSoup
# ...
soup = BeautifulSoup(text_with_markup,
'html.parser')
text_without_markup = soup.get_text()
```

`From`, `Subject` and the body of the e-mail is then split into words at whitespace characters with each word being converted to lowercase. The set of all these words is called a *bag of words* (see Section 2.2) used as input for the Naive Bayes filter described in the next section.

2.2 Naive Bayes

We decided to opt for Naive Bayes (2) filtering as it seemed relatively straightforward and quick to implement, so it would offer a quick win if it worked. Another benefit of this approach is its speed. As the name implies, this approach is considered naive since it uses what we call a *bag of words* – it does not consider the word order at all (2). That said, it still produces relatively accurate results (see Section 4).

2.2.1 Notation

As e-mails often include the same word multiple times, we will use the concept of a *multiset* which, unlike a regular *set*, allows for multiple instances of

the same element. We define $|A|$ as the size of the multiset A, $A \uplus B$ as the union of multisets A and B and $\text{freq}(m, M)$ as the number of occurrences of element m in the multiset M.

Let E be the multiset of all e-mails, $S \subseteq E$ be the multiset of all spams and $H \subseteq E$ be the multiset of all hams. We view any e-mail $e \in E$ as a multiset of words $w \in e$. Let $W_{\text{spam}} = \biguplus_{s \in S} s$ be the multiset of all words in spams and $W_{\text{ham}} \in \biguplus_{h \in H} h$ be the multiset of all words in hams. Let $W = W_{\text{spam}} \uplus W_{\text{ham}}$ be the multiset of all words.

2.2.2 Training

During training, we compute the likelihood that any e-mail is spam

$$\tilde{P}_{\text{spam}} = \frac{|S|}{|E|}, \tag{1}$$

and the likelihood that any e-mail is ham

$$\tilde{P}_{\text{ham}} = \frac{|H|}{|E|}. \tag{2}$$

Later during training, for each word $w \in W$ we compute the likelihood that it appears in spam

$$\hat{P}_{\text{spam}}(w) = \frac{\text{freq}(w, W_{\text{spam}}) + 1}{|W_{\text{spam}}|}, \tag{3}$$

and the likelihood that the word $w \in W$ appears in ham

$$\hat{P}_{\text{ham}}(w) = \frac{\text{freq}(w, W_{\text{ham}}) + 1}{|W_{\text{ham}}|}. \tag{4}$$

2.2.3 Filtering

During filtering, for any given e-mail $e \in E$, we compute the likelihood that it is spam

$$P_{\text{spam}}(e) = \tilde{P}_{\text{spam}} \prod_{w \in e} \hat{P}_{\text{spam}}(w), \tag{5}$$

and the likelihood that the e-mail is ham

$$P_{\text{ham}}(e) = \tilde{P}_{\text{ham}} \prod_{w \in e} \hat{P}_{\text{ham}}(w). \tag{6}$$

If $P_{\text{spam}} < P_{\text{ham}}$, we say a given e-mail is ham, otherwise, it is spam.

We can now also see the $+1$ in Equation 3 and Equation 4 is very important. Without it, if a $w \notin W_{\text{spam}}$ or $w \notin W_{\text{ham}}$ but $w \in e$, $P_{\text{spam}}(e)$ or $P_{\text{ham}}(e)$ would equal 0 respectively (2).

3 Methodology

We were provided with two data sets, in this section we describe our methodology for testing the spam filter on these two datasets.

3.1 Quality of a spam filter

We base our measurement of the quality of the filter on the *binary confusion matrix* (see Table 1) which is the result of comparing the predicted classification (the one generated by the classifier) with the actual classification. We consider the case where e-mail is spam to be positive.

		Predicted class	
		Positive	**Negative**
Actual class	**Positive**	True Positive (**TP**)	False Negative (**FN**)
	Negative	False Positive (**PN**)	True negative (**TN**)

Table 1: Binary confusion matrix

After the spam filter classifies e-mails in a given dataset, we compute how many e-mails are in each category (TP, TN, FP, FN). Based on this we compute the quality

$$q = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + 10\ \text{FP} + \text{FN}} \tag{7}$$

of the filter for the dataset. In Equation 7 we multiply FP 10 times since we consider the case where a user does not receive a ham to be worse than if they received spam.

3.2 Data splitting

A testing script (located in `src/test.py`) was created for running the spam filter on the provided datasets. First, it randomly splits the dataset into two parts — the *training set* and the *testing set* (with a ratio of 4:1). Later, it runs the `BaseFilter.train()` method on the training set and `BaseFilter.test()` on the testing set. This process can be run multiple times for each dataset to make sure the randomness did not introduce any bias.

4 Results

Our spam filter implementation gives quite good results and achieves high-quality q. Specifically, it gets $q_1 = 93\%$ for the first provided dataset and $q_2 = 89\%$ for the second provided dataset with 250 runs of the testing code (as described in Section 3.2).

We were surprised by how well Naive Bayes handles non-words in email and especially how well it performs even when it treats words that have punctuation before or after them as separate words. By non-words in this case, we mean any content other than normal words and HTML markup (which we have stripped out) that we can find in an email, such as URLs, random data etc.

Removing all punctuation from the text of an email, and limiting the length of what we still consider a word to 25 characters, made the dictionary of words captured by the algorithm smaller and cleaner. However, it did not improve the results noticeably.

While punctuation characters such as `!` or `?` play an important role in the overall tone and meaning of an email, other characters such as `(` and `,` may play a much smaller role and probably don't hold any information that would give a word much of an extra meaningful value for it to make sense to be treated differently.

We were also surprised by how effective the bag of words approach can be even though it glosses over a lot of nuance.

5 Teamwork

This semestral work was a team project, in this section, we detail how the work was organized and performed, what tools were used for communication and collaboration etc.

5.1 Division of labour

In table Table 2 we show what tasks were done by each team member.

Task	Author
Email parsing	Král
Text normalization	Král
Naive Bayes filtering	Brandner
Report	Both

Table 2: Division of labour

5.2 Tools

We opted for Git as our version control software since it is the de facto standard. We used GitHub as the cloud service on which to host our Git repository. GitHub was also a great choice given its code collaboration features such as pull requests which enabled us to effectively communicate our ideas about code.

For real-time communication, we tried several services (e.g. Matrix, WhatsApp) but in the end stuck to WhatsApp as it turned out to be the most stable. However, we were not happy with the features WhatsApp offered. While its text formatting features have improved in recent years, they are nowhere close to what some other services offer (code blocks etc.). The closed-source nature of WhatsApp is also not entirely to our liking, so we'd certainly like to give protocols such as Matrix a second chance in the future.

For writing this report, we decided to try Typst, a modern alternative to LaTeX written in Rust, which offers simpler syntax, faster compile times and builtin online editor. While it does not offer complete feature parity with LaTeX, we have enjoyed working with it, as its simpler syntax makes writing text faster and it does not produce hard-to-parse error messages. We would say it was more than adequate for our needs.

6 Conclusion

Our task was to implement a spam filter in Python using whatever algorithm. We opted for parsing the emails' content and Naive Bayes filtering. Using these methods we were able to implement a filter with high accuracy. The project allowed us to learn new things and gain experience in the fields of statistics, analysis, Python, code collaboration, versioning, testing, and teamwork management.

7 Bibliography

1. LEONARD RICHARDSON. Beautiful Soup 4.12.0 documentation. Online. Available from: https://www.crummy.com/software/BeautifulSoup/bs4/doc/#quick-start
2. *Naive Bayes, Clearly Explained!!!.* Online. Available from: https://www.youtube.com/watch?v=O2L2Uv9pdDA