Programming for Computational Linguistics 2022/2023

Final Project — Naive Bayes Spam Filter

Due 9.03.2023

1 Introduction

Your task in this project will be to **design, implement, and document a naive bayes spam filter in python**. This spam filter will be able to learn to classify spam emails from provided training data, and can then be used to classify new emails. The further sections will describe the theory in more detail, specify the obligatory functionality of your spam filter and give an overview of how the project will be graded.

2 Theory

In this section, we will define the theoretical background of a naive bayes spam filter. This will be an abstract description, and will not necessarily correspond directly to your implementation.

2.1 Spam

Spam or junk email refers to unsolicited email messages that are used for advertising or malicious purposes such as phishing and delivering malware. Spam email is typically sent out in bulk by automated systems and is estimated to make up around half of all emails delivered.

2.2 Spam Filter

A spam filter is an automated system that classifies emails into spam or not spam (sometimes also referred to as "ham").

It typically does so by predicting a spam score which is in turn used to make a prediction into "spam" and "ham" if the score exceeds a certain threshold.

One such system is called naive bayes spam filter which leverages bayes theorem to estimate the probability that a certain word appears in a spam email.

2.3 Bayes' Theorem

Bayes' Theorem models the probability of an event given evidence of the occurence of another event.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1}$$

2.4 Naive Bayes Spam Filter

Our spam filter uses bayes' theorem to classify emails into "spam" and "ham" based on the words of the emails. It leverages bayes' theorem to estimate the probability that an email is spam given knowledge that a word w_i appears in it.

$$P(\text{“spam”}|w_i) = \frac{P(w_i|\text{“spam”})\,P(\text{“spam”})}{P(w_i)} \tag{2}$$

In order to estimate the probability $P(\text{“spam”}|w_i)$, we need to estimate the following three terms:

$P(w_i\|\text{“spam”})$	*the probability that w_i appears in a spam email*
$P(\text{“spam”})$	*the probability that a given email is spam*
$P(w_i)$	*the probability that w_i appears in an email*

We could estimate the *a-priori* probability $P(\text{“spam”})$ as the overall probability that an email is spam, though in practice we can get a more reliable classifier by setting it to $P(\text{“spam”}) = P(\text{“ham”}) = 0.5$ and using a balanced training dataset that contains equal amounts of spam and ham email.

Additionally, rewriting $P(w_i)$ as $P(w_i|\text{“spam”})\,P(\text{“spam”}) + P(w_i|\text{“ham”})\,P(\text{“ham”})$ allows us to simplify above equation to:

$$P(\text{“spam”}|w_i) = \frac{P(w_i|\text{“spam”})}{P(w_i|\text{“spam”}) + P(w_i|\text{“ham”})} \tag{3}$$

In this formulation we only need to estimate the two terms, $P(w_i|\text{“spam”})$ and $P(w_i|\text{“ham”})$, which we can do using our training dataset. The probability $P(w_i|\text{“spam”})$ can be obtained by counting the number of spam emails containing w_i divided by the total number of emails containing w_i

$$P(w_i|\text{“spam”}) \sim \frac{|\{\text{email}|w_i \in \text{email} \wedge \text{email} \in \text{“}spam''\}|}{|\{\text{email}|w_i \in \text{email}\}|} \tag{4}$$

$P(w_i|\text{“ham”})$ can be estimated accordingly.

Putting it all together, we can get the overall probability that a given email is spam using the following formula:

$$p = \frac{p_1 p_2 \dots p_n}{p_1 p_2 \dots p_n + (1-p_1)(1-p_2)\dots(1-p_n)} \tag{5}$$

where p_i is the probability $P(\text{“spam”}|w_i)$.

This probability is also called spam score and we classify an email as spam if the spam score exceeds a certain threshold.

3 Specification and Baseline Functionality

In this section, we will enumerate the required components of your naive bayes spam filter. You will be graded on how well your program meets this specification. In the following days and weeks, we will provide more detailed recommendations for how you might want to implement this specification.

3.1 Specification

Your program will be split into three separate modules: `corpus.py`, `nb.py`, and `main.py` (although you may implement additional modules to use if you would like).

- `corpus.py` will be responsible for reading text files, and processing and tokenizing text. It should implement a function `tokenize` which accepts text as input and returns a list of tokens. The exact implementation is up to you — for example, simple tokenization can take only white space and punctuation into account, cleverer tokenization techniques however may yield better results. You can implement this function on your own (think about special cases, capitalization, etc.) or use some external library (for example NLTK). It should also implement a function `read_file` which accepts as input the path to a text file representing an email, reads the file and returns the contained text. Lastly it should implement a function `read_dataset`, which should accept as input a path to a dataset. The dataset that we provide contains two directories, `ham` and `spam`, each containing multiple files representing ham and spam emails respectively. You function should discover those files, read them using `read_file` and return a dataset of training instances consisting of emails with their respective label (ham or spam) However, cleverer tokenization techniques may yield subjectively better final text, which will factor into the subjective output evaluation.

- `nb.py` will implement the core logic of the naive bayes model, and will be responsible for classifying text. It should contain a definition of a class `SpamFilter` with methods such as `train` (trains your spam filter by estimating parameters from training data) and `classify` (calculates the spam score of a given email and classifies it into spam or ham if the spam score exceeds a threshold).

- `main.py` will be responsible for user interaction — users of the spam filter will invoke `main.py`, and all functionality will be exposed through this file. When run, it should provide the user with instructions, and ask what they would like to do. The exact interface is up to you, but your program should print enough instructions to make it self explanatory. We should be able to run this program, without reading your report, and figure out how to do what we want (However, you should still fully document the user interface in your report).

3.2 Baseline Functionality

The baseline functionality of your spam filter should allow the user to follow the steps below:

1. Train a spam filter on the training dataset provided on Ilias
2. Batch classify the test dataset and output the filename, the predicted spam score and classification result to a file

Run the steps above to classify the emails from the test dataset and submit the file with your report. The lines of the file should be formatted like in the following example:

```
1734.2003-02-14.XY.spam.txt 0.732 spam
1615.2002-07-03.XY.ham.txt 0.1042 ham
```

...

4 Report

Write a 2-page report (font size 10pt) describing what you did. Make sure that your report is **easy to read**. Remember about section titles, indentation, margins, etc.

Your report should describe and justify any specific design decisions you took with your spam filter. For example, how do you store the estimated parameters? How do you represent training instances? How do you perform tokenization? etc. You do not have to explain what is already described in the specification and you should definitely not copy anything from the specification (or any other provided file) into your report.

One of the most important parts of your report should be a section "Documentation" describing how users should run and use your program. This section is not code documentation so don't put your code with comments here. Instead, you should describe how to use `main.py`: what options does it have, what do they do, etc.

You should describe each extension that you implemented (see Section 5) in its own section of your report. Explain what the extension is, how it works, and why you chose to implement it (you can exceed the 2-page limit to describe your extensions).

5 Possible Extensions and Analysis

After you implemented the baseline functionality, we encourage you to extend it. Below are some ideas for possible extensions; if you have another idea for an extension you would like to try, ask us if it would be appropriate first. Every extension you add should be well documented in your report: What is the extension? What is its goal? How does it work? For example, if you decide to implement smoothing, describe what the goal of the method is, can you find an example where it improves over a model without smoothing and explain why.

1. Experiment with different spam score threshold values

2. Improve the feature representation

   - Filter stop words
   - Represent email with bigrams, trigrams, ..., n-grams instead of unigrams to model the words in their context (but be aware of sparsity)
   - Handle spelling mistakes or intentional spelling variants that spammers use to escape detection

3. Qualitative/quantitative evaluation of results

   - Identify the top-n words that maximize $P(\text{“spam”}|w_i)$. What are the probabilities associated with the words? Are there any unexpected entries in the list?
   - Find examples of misclassified emails and try to explain why the classification fails (insufficient training data? phenomena not captured by feature representation? naive bayes modeling approach inadequate?)
   - Change the size of the training data and analyze the influence on the classification performance on the test data.