Final Project: Spam Filter

Mattalika Intarahom

March 9, 2023

1 Implementation

In this section, I will explains in detail how the spam filter and the extensions are implemented.

1.1 `corpus.py`

`corpus.py` mainly deal with preprocessing text and preparing training data. It contains the following functions:

1.1.1 `tokenize(text, remove_punctuation, remove_stopwords, check_nonwords)`

In addition to the original function which accept only one parameter, `text`, I extend this function by adding three additional parameters, `remove_punctuation`, `remove_stopwords` and `check_nonwords`.

When `remove_punctuation=True`, all punctuation in a plain email will be removed using regular expression which will replace any punctuation with an empty string. After removing punctuation, the plain email will be tokenized using `word_tokenize` from NLTK. Similarly, when `remove_stopwords=True`, any token that exists in a stopwords corpus will be removed from the tokenized text.

When `check_nonwords=True`, each token will be checked if it is a non-word. In order to do so, tokens have to be classified into their parts of speech and labelled accordingly. Using `pos_tagger`, I yield tuples which consist of a word and its POS. Tuples are then fed to `WordNetLemmatizer`, which returns a lemma form for each input word. For example, it will return `'read'` for a tuple `('reading', 'VBG')`. The reason why words have to be grouped into one lemma form is because dictionaries do not cover all the variations of a word.

Lastly, a lemmatized word is check if it exists in a dictionary. This is done by using three dictionaries, `wordnet`, `words`, and `names` from NLTK corpus, with multiple hand-coded rules. If the word starts with alphabet and does not exist in dictionaries, I further check: (1) if the word exists in `names` dictionary, it is mapped to a token `"__NAME__"`, (2) else it is mapped to `"__NON_WORD__"`. (3) If the word starts with a number, it is mapped to `"__NUM__"`, (4) else the word is mapped to `"__UNK__"`. Additionally, (5) If an email does not contain any content at all, it is mapped to `"__NO_CONTENT__"`.

I decide not to use a word correction before looking up words in dictionaies, therefore misspelled words will also be mapped to `"__NON_WORD__"`.

In the final version of a program, `remove_punctuation` and `remove_stopwords` are set to `True`, while `check_nonwords` is set to `False` as a default value. This is because even though the initial purpose of `check_nonwords` is to group intentional spelling variants in spam emails to `"__NON_WORD__"` token, this function takes extremely long to run.

Overall, these three boolean parameters allow me to compare the performance of a spam filter in different settings (i.e., with or without stopwords removal) and later optimize which of three features I want to use in the final version without directly modifying the code.

1.1.2 `read_file(path)`

This function is designed to keep only the body of an email by removing a subject line and a block of forwarded email header. The former is done by deleting the first line of an email if the line starts with a word, `subject`. The latter is done by detecting a line starts with `'- - - -'` which indicates the start of a header block, then iterates through each line in a block and continue to delete the line until the first line of the forwarded email content is found. For example, texts in the red box below will be removed from the email.

```
Subject: leadership development pilot
sally :
what timing , ask and you shall receive . as per our discussion , listed below
is an update on the leadership pilot . your vendor selection team will
receive an update and even more information later in the week .
on the lunch & learn for energy operations , the audience and focus will be
your group . we are ready to start up when appropriate .
thank you for your time today . please call me if you have any questions at
x 33597 .
- - - - - - - - - - - - - - - - - - - - - - - - forwarded by julie armstrong / corp / enron on 01 / 17 / 2000
06 : 44 pm - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
from : susan runkel @ ect 01 / 17 / 2000 03 : 22 pm
to : cindy skinner / hou / ect @ ect , brad mcsherry / hou / ect @ ect , norma
villarreal / hou / ect @ ect , kimberly rizzi / hou / ect @ ect , fran l mayes / hou / ect @ ect ,
gary buck / hou / ect @ ect , robert jones / corp / enron @ enron , sheila
walton / hou / ect @ ect , philip conn / corp / enron @ enron , mary overgaard / pdx / ect @ ect ,
kim melodick / hou / ect @ ect , valeria a hope / hou / ect @ ect
cc : david oxley / hou / ect @ ect , susan carrera / hou / ect @ ect , jane
allen / hou / ect @ ect , christine shenkman / enron _ development @ enron _ development ,
kathryn mclean / hou / ect @ ect , gracie s presas / hou / ect @ ect , janice
riedel / hou / ect @ ect , julie armstrong / corp / enron @ enron
subject : leadership development pilot
good news regarding the ena leadership curriculum ! through the help of a
vendor selection team from eops , we ' ve chosen southwest performance group and
wilson learning products as one of our primary vendors for the leadership
curriculum and programs . we are ready to conduct a pilot on february 8 - 10 of
six modules . the purpose of the pilot is to evaluate for fine - tuning the
wilson learning materials and facilitators and to present just a portion of
the leadership curriculum .
```

1.1.3 `get_paths(path)`

The purpose of this function is to retrieve a file name of all files in a directory `path`, then return a list of file names `file_names`.

1.1.4 `read_dataset(path)`

This function retrieves files in the given folder using `get_paths()`, then preprocesess, tokenizes, and prepares tokenized data into a proper format ready to be trained.

1.2 `nb.py`

`nb.py` contains a class `SpamFilter()` which mainly deal with tokenized texts and calculations. It contains the following functions:

1.2.1 `__init__(self)`

A class constructor, `__init__(self)` takes zero parameter.

1.2.2 `train(self, emails)`

This function counts the frequencies of how many emails does a word occur in spam and ham emails, then store them to dictionaries `spam_freq` and `ham_freq`, with a word as a key and a frequency as a value. The keys of these two dictionaries are merged to create a list of `vocab` with no duplicate. Two numpy lists, `p_word_spam` and `p_word_ham` of the same length with `vocab` are initialized using `numpy.zeros()`, which will later be used to store the probability of each word in `vocab` occurring in spam and ham email respectively.

1.2.3 `classify(self, email)`

This functions utilizes `vocab`, `p_word_spam` and `p_word_ham` from `train()` to calculate a spam score. Additionally, to avoid multiplying by zero and floating point underflow, I add `+1` to every probability before multiplying them together. If the model encounter unknown words in a testing data, it will skip that word. Finally, threshold is set to 0.49, any email with equal to or higher than this threshold will be classified as `spam`. A result including a spam score and a predicted label are returned.

1.3 main.py

`main.py` is mainly dealing with user interactions. I implement a console-based user interface which allows users to navigate through all the menus by typing a command, for example enter [T] for training (For more details please check a documentation section). This file contains the following functions:

1.3.1 write_result(result_lines)

This function first checks if `.\result` folder exists in the current directory, if not it will create a new result folder. Result line is then written on a text file which is later saved in `.\result`.

1.3.2 do_train(spam_path, ham_path)

This function preprocesses the training data before passing them to `train()`. After the training is done, a trained `SpamFilter()` object will be saved as `training_spam.pickle` file for later use, thus users do not have to train the spam filter every time they use the program.

1.3.3 classify_single(filepath)

This function accepts a file path to a single email. It will proceed to `calssify()` only if a pretrained `training_spam.pickle` exists. Result consisting a file name, a spam score, and a predicted label is directly printed on the console.

1.3.4 classify_batch(folderpath)

This function accepts a file path to a folder containing emails. Similair to `classify_single()`, it will proceed to `calssify()` only if a pretrained `training_spam.pickle` exists. Finally, `result_lines` containing a file name retrieved from `get_paths()`, a spam score, and a predicted label for each email is passed to `write_result()` in order to export results into one single file.

For all three functions, exception handling is applied to catch an error such as `FileNotFoundError` that could frequently occur when users pass an inappropriate path, others errors are handled using `except Exception as e`.

1.4 eval.py

Additionally, I include the extension `eval.py`, containing two functions that allow me to further evaluate the performance of my spam filter.

1.4.1 read_results(filepath, gold_label)

This function takes a result file, reads through lines and maps a gold label and a predicted label as a tuple `(gold_label, predicted_label)`.

1.4.2 get_eval(result_set))

This function accept tuples of `(gold_label, predicted_label)` and calculate precision, recall and F1 score as follows:

$$Precision = \frac{TruePositive}{TruePositve + FalsePositive} \tag{1}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{2}$$

$$F1 = 2 * (\frac{1}{Precision}) + (\frac{1}{Recall}) \tag{3}$$

2 Results and Evaluations

In this section, I will discuss about the result of my experiment with different spam score threshold values with the setting `remove_punctuation=True, remove_stopwords=True, check_nonwords=False`. Results for each threshold are then collected and calculated using precision, recall and F1 score.

Threshold	Precision	Recall	F1
0.45	0.51	0.97	0.66
0.475	0.53	0.91	0.67
0.48	0.56	0.89	0.64
0.485	0.58	0.86	0.69
0.49	0.63	0.80	0.71
0.495	0.71	0.63	0.67
0.50	0.95	0.26	0.40
0.51	1.00	0.0007	0.0013

Table 1: Precision, Recall and F1 over variations of threshold value.

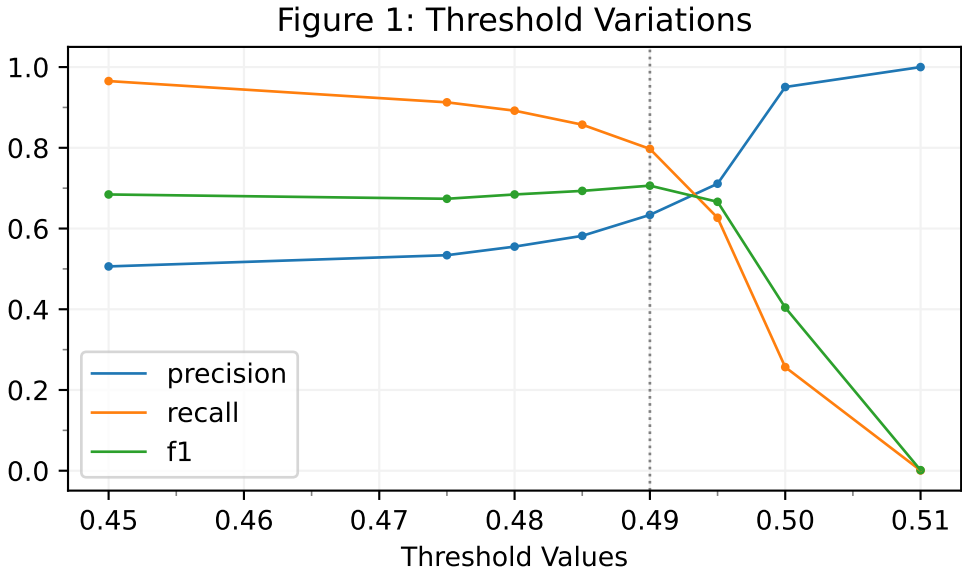

Figure 1 illustrate the change in precision, recall, and F1 score across different threshold values ranging from 0.45 to 0.51. The vertical axis represents scores, while the horizontal axis represents threshold values. Overall, there is a sudden drop of recall and F1 score where threshold is 0.495, while precision increases significantly. This indicates that the higher the threshold, the more precise the model can predict spam. However, there is a trade-off between accuracy and robustness which leads to a drop of recall and F1, suggesting that even though a model is very precise, it can retrieve very few spam emails.

In the final version of my program, I decide to use a threshold value of 0.49 since it maximizes the F1 score, which is a balanced measurement between precision and recall.

Lastly, in order to improve the performance of a model, I suggest that in addition to a document frequency (how many documents does a word appear), we also have to take a term frequency (how many times does a word appear in a class) and the length of a document into account. For example, if we set `check_nonwords=True`, the token `"__NON_WORD__"` which groups intentional spelling variants together will appear in spam email more often than in ham emails. However, the provided equations do not take word frequencies into account, therefore the model would not benefit much from `check_nonwords=True` especially when this function might take hours to run.

3 Documentation

Users can navigate through menu by typing the following commands:

- Enter [T] to train a spam filter
- Enter [F] to filter email(s)

- Enter [H] for help
- Enter [E] for exit

3.1 Enter [T]

Enter [T] will allow users to train a spam filter using their training data. In order to train, users will have to provide (1) a path to spam folder and (2) a path to ham folder. Once the training is complete, users can utilized this spam filter to classify emails without having to train the model again every time they start the application. However, it is possible to train the model again if users want to train with different training data.

```
>> Please enter your command: t
>> Enter a path to your [spam] training data. (Example: data/train/spam/): data/train/spam/
>> Enter a path to your [ham] training data. (Example: data/train/ham/): data/train/ham/
Start preprocessing training data. . .
100%|██████████| 4500/4500 [00:04<00:00, 1004.11it/s]
100%|██████████| 4500/4500 [00:04<00:00, 1027.59it/s]

Start training Spam Filter. . .
100%|██████████| 9000/9000 [00:00<00:00, 32491.09it/s]
Training completed.
You can now use your spam filter.
>> Please enter your command:
```

3.2 Enter [F]

Enter [E] will allow users to filter email(s) as a single email (enter [s]) or as a batch (enter [B]).

For a single email, users will have to provide a path to a file. A result, consists of a filename, a spam score, and a predicted label will be printed on the console.

```
>> Please enter your command: f
   >> Enter [S] to filter a single email, [B] to filter as a batch: s
   >> Enter a path to your email file. (Example: data/train/ham/email.txt): data/train/ham/0001.2000-01-17.beck.ham.txt
Result : data/train/ham/0001.2000-01-17.beck.ham.txt 0.4591 ham
```

For a batch, users will have to provide a path to a folder. Results, consists of a filename, a spam score, and a predicted label will be exported as a single file.

```
>> Please enter your command: f
   >> Enter [S] to filter a single email, [B] to filter as a batch: b
    >> Enter a path to your folder (Example: data/test/ham/): data/test/ham/

Start classifying. . .
100%|██████████| 1500/1500 [00:17<00:00, 84.73it/s]
Result file 2023-03-09_18-49-46_results.txt is successfully exported
to                                                                \Project\main\result
```

3.3 Enter [H]

Enter [H] will print instructions regarding how to train and filter email(s).

3.4 Enter [E]

Enter [E] will directly close the application.