Programming for Computational Linguistics 2022/2023

Spam Filter: Full Specification

In this document, we will fully enumerate the required components of your naive bayes spam filter. You will be graded on how well your program meets this specification. In order to receive full credit, you must implement this specification exactly, with all files, classes, functions, and methods named and implemented as described.

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. `nb.py` will implement the core logic of the spam filter, and will be responsible for predicting tokens and generating text. `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 module.

1 `corpus.py`

This module should implement the following functions:

1.1 `tokenize(text)`

This function should accept as input a string `text`, and should return a list of tokens, with each token consisting of a string. The exact implementation is up to you — you can do this as simply or as complicatedly as you like. As long as you generate a list of tokens in some manner, you will receive full credit for this section. However, cleverer tokenization techniques may yield subjectively better final text.

1.2 `read_file(path)`

This function should accept as input a string `path`, open the file at the specified path, and return the file contents as a list of tokens. Each file starts with one line containing the subject of the email, starting with `"Subject: "` and an arbitrary amount of lines containing the body of the email. For our purposes, you can discard the subject of the email and return the body as a list of tokens.

1.3 `read_dataset(path)`

This function should accept as input a string `path` which points to a dataset. The dataset that we provide contains two directories, `ham` and `spam`. Your func-

tion should assume that these directories exist in the provided `path`, discover all email files contained in them and read their contents using `read_file`. Every email should be combined into a tuple with its respective label (`"ham"` or `"spam"`) to form a training instance, e.g.

```
    (["buy", "cheap", "medications", "online"], "spam")
```

The function should return a list of training instances

2 nb.py

This module should contain a class `SpamFilter`. This class will deal only with token sequences as provided by `corpus.py`, not directly with untokenized text. That class should implement the following methods:

2.1 `__init__(self)`

The constructor should create a new, untrained `SpamFilter`, and initialize whatever is needed. You may provide additional parameters if you think they might be useful to you, but they must have default values — it should be possible to instantiate a `SpamFilter` by calling `SpamFilter()` without any additional arguments.

2.2 `train(self, emails)`

This method should train your spam filter, such that it learns the statistics of `emails`. This method alone will not return anything, but after this method is called, the method `classify` should work properly, according to the statistics of `emails`. `emails` is a list training instances as returned by `read_dataset`, where each training instance contains a list of tokens of an email and a label (`"spam"` or `"ham"`)

2.3 `classify(self, email)`

This method should classify `email` into spam or ham. `email` is a list of tokens representing the contents of an email. The return value of this function should be a tuple consisting of both the spam score as a floating point number and the spam classification result for the given email as a string (`"spam"` or `"ham"`).

3 main.py

This module 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 user instructions. This program should allow the user to:

- Train a spam filter from a dataset of emails
- Classify an email into spam or ham

- Batch classify an entire dataset of emails

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).