Spam Filter From Scratch

1 Naive Bayes

In this assignment, you implement a basic spam filter using naive Bayes classification.

In this lab, you will implement a minimal system for spam filtering. You will begin by processing the raw training data (in `/data/train-ham/`[1] and `/data/train-spam/`). Next, you will proceed by estimating the conditional probability distributions of the words in the vocabulary determined by each document class. Lastly, you will use a Naive Bayes model to make predictions on the test set located in `/data/dev-ham/` and `/data/dev-spam/`.

1.1 load_tokens()

Questions

1. Making use of the `email` module, write a function `load_tokens(email_path)` that reads the email at the specified path, extracts the tokens from its message, and returns them as a list.

Specifically, you should use the `email.message_from_file(file_obj)` function to create a `message` object from the contents of the file, and the `email.iterators.body_line_iterator(message)` function to iterate over the lines in the message. Here, tokens are considered to be contiguous substrings of non-whitespace characters.

```
>>> ham_dir = "/data/train-ham/"
>>> load_tokens(ham_dir+"ham1")[200:204]
['of', 'my', 'outstanding', 'mail']
>>> load_tokens(ham_dir+"ham2")[110:114]
['for', 'Preferences', '-', "didn't"]
```

```
>>> spam_dir = "/data/train-spam/"
>>> load_tokens(spam_dir+"spam1")[1:5]
['You', 'are', 'receiving', 'this']
>>> load_tokens(spam_dir+"spam2")[:4]
['<html>', '<body>', '<center>', '<h3>']
```

1.2 log_probs()

Questions

2. Write a function `log_probs(email_paths, smoothing)` that returns a dictionary from the words contained in the given emails to their Laplace-smoothed log-probabilities.

[1] Yes, it's actually called **ham**!

Specifically, if the set V denote the vocabulary of words in the emails, then the probabilities should be computed by taking the logarithms of:

$$P(w) = \frac{count(w) + \alpha}{\sum_{w' \in V} count(w') + \alpha|V|} \tag{1}$$

where w is a word in the vocabulary V and α is the smoothing constant. Be sure to add an <*UNK*> token (whose count in the training data is 0) to the vocabulary to handle novel items in the development and test data.

```
>>> paths = ["/data/train-ham/ham%d" % i
...           for i in range(1, 11)]
>>> p = log_probs(paths, 1e-5)
>>> p["the"]
-3.6080194731874062
>>> p["line"]
-4.272995709320345
```

```
>>> paths = ["/data/train-spam/spam%d" % i
...           for i in range(1, 11)]
>>> p = log_probs(paths, 1e-5)
>>> p["Credit"]
-5.837004641921745
>>> p["<UNK>"]
-20.34566288044584
```

1.3 __init__

Questions

3. Write an initialization method `__init__(self, spam_dir, ham_dir, smoothing)` in the `SpamFilter` class that:
   (a) creates two log-probability dictionaries corresponding to the emails in the provided spam and ham directories, then stores them internally for future use.
   (b) computes the class probabilities *P(spam)* and *P(¬spam)* based on the number of files in the input directories

1.4 is_spam()

Questions

4. Write a method `is_spam(self, email_path)` in the `SpamFilter` class that returns a Boolean value indicating whether the email at the given file path is predicted to be spam.

Tokens which were not encountered during the training process should be converted into the special word <*UNK*> in order to avoid zero probabilities.

Recall from class that for a given class $c \in \{spam, \neg spam\}$:

$$P(c|document) \approx P(c) \prod_{w \in V} P(w|c)^{count(w)} \tag{2}$$

(In principle we should be dividing by the normalization constant $P(document)$ but since it's the same for both classes it can safely be ignored). Here the count of a word is computed over the input document to be classified (not the count in the training data).

Remember: these computations should be done in log-space to avoid underflow.

```
>>> sf = SpamFilter("/data/train-spam",
...      "/data/train/ham", 1e-5)
>>> sf.is_spam("/data/train-spam/spam1")
True
>>> sf.is_spam("/data/train-spam/spam2")
True
```

```
>>> sf = SpamFilter("/data/train-spam",
...      "/data/train/ham", 1e-5)
>>> sf.is_spam("/data/train-ham/ham1")
False
>>> sf.is_spam("/data/train-ham/ham2")
False
```

1.5 most_indicative{spam,ham}()

Suppose we define the spam indication value of a word w to be the quantity:

$$log(\frac{P(w|spam)}{P(w)}) \tag{3}$$

Similarly, we define the ham indication value of a word w to be:

$$log(\frac{P(w|\neg spam)}{P(w)}) \tag{4}$$

Questions

5. Write a pair of methods `most_indicative_spam(self, n)` and `most_indicative_ham(self, n)` in the `SpamFilter` class which return the n most indicative words for each category, sorted in descending order based on their indication values. You should restrict the set of words considered for each method to those which appear in both at least one spam email and at least one ham email.

Hint: The probabilities computed within the `__init__(self, spam_dir, ham_dir, smoothing)` method are sufficient to calculate these quantities.

```
>>> sf = SpamFilter("/data/train-spam",
...      "/data/train-ham", 1e-5)
>>> sf.most_indicative_spam(5)
['<a', '<input', '<html>', '<meta',
 '</head>']
```

```
>>> sf = SpamFilter("/data/train-spam",
...      "/data/train-ham", 1e-5)
>>> sf.most_indicative_ham(5)
['Aug', 'ilug@linux.ie', 'install',
 'spam.', 'Group:']
```