CIS 521: Homework 5 [100 points]

Release Date	Thursday, October 27, 2016
Due Date	11:59 pm on Thursday, November 3, 2016

Instructions

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

A skeleton file `homework5.py` containing empty definitions for each question has been provided. Since portions of this assignment will be graded automatically, none of the names or function signatures in this file should be modified. However, you are free to introduce additional variables or functions if needed.

You may import definitions from any standard Python library, and are encouraged to do so in case you find yourself reinventing the wheel. If you are unsure where to start, consider taking a look at the data structures and functions defined in the `collections`, `email`, `math`, and `os` modules.

You will find that in addition to a problem specification, most programming questions also include one or two examples from the Python interpreter. In addition to performing your own testing, you are strongly encouraged to verify that your code gives the expected output for these examples before submitting.

It is highly recommended that you follow the Python style guidelines set forth in PEP 8, which was written in part by the creator of Python. However, your code will not be graded for style.

Once you have completed the assignment, you should submit your file on Eniac using the following `turnin` command, where the flags `-c` and `-p` stand for "course" and "project", respectively.

```
turnin -c cis521 -p hw5 homework5.py
```

You may submit as many times as you would like before the deadline, but only the last submission will be saved. To view a detailed listing of the contents of your most recent submission, you can use the following command, where the flag `-v` stands for "verbose".

```
turnin -c cis521 -p hw5 -v
```

1. Spam Filter [95 points]

In this section, you will implement a minimal system for spam filtering. You will begin by processing the raw training data, which is assumed in the examples below to lie in the directory `data/train` relative to the skeleton file. 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 publicly available test set, located in `data/dev`.

1. **[5 points]** 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>']
```

2. **[30 points]** 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. Specifically, if the set V denotes the vocabulary of words in the emails, then the probabilities should be computed by taking the logarithms of

$$P(w) = \frac{\text{count}(w) + \alpha}{\left(\sum_{w' \in V} \text{count}(w')\right) + \alpha(|V| + 1)}, \qquad P(\langle \text{UNK} \rangle) = \frac{\alpha}{\left(\sum_{w' \in V} \text{count}(w')\right) + \alpha(|V| + 1)}$$

where w is a word in the vocabulary V, α is the smoothing constant (typically in the range $0 < \alpha \leq 1$), and $\langle \text{UNK} \rangle$ denotes a special word that will be substituted for unknown tokens at test time.

```
>>> 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
```

3. **[10 points]** Write an initialization method `__init__(self, spam_dir, ham_dir, smoothing)` in the `SpamFilter` class that creates two log-probability dictionaries corresponding to the emails in the provided spam and ham directories, then stores them internally for future use. Also compute the class probabilities $P(\text{spam})$ and $P(\neg\text{spam})$ based on the number of files in the input directories.

4. **[25 points]** 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 the lecture slides that for a given class $c \in \{\text{spam}, \neg\text{spam}\}$,

$$P(c \mid \text{document}) \sim P(c) \prod_{w \in V} P(w \mid c)^{\text{count}(w)},$$

where the normalization constant $1/P(\text{document})$ is the same for both classes and can therefore be ignored. Here, the count of a word is computed over the input document to be classified.

Remember that these computations should be computed 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
```

5. **[25 points]** Suppose we define the spam indication value of a word w to be the quantity

$$\log\left(\frac{P(w \mid \text{spam})}{P(w)}\right).$$

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

$$\log\left(\frac{P(w \mid \neg\text{spam})}{P(w)}\right).$$

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 at least one spam email and 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:']
```

2. Feedback [5 points]

1. **[1 point]** Approximately how long did you spend on this assignment?
2. **[2 points]** Which aspects of this assignment did you find most challenging? Were there any significant stumbling blocks?
3. **[2 points]** Which aspects of this assignment did you like? Is there anything you would have changed?