Mini-project: Email Spam Detection

Mini-project: Email Spam Detection

90/90 points (graded)

We are going to build a Naive Bayes classifier to detect spam. Download `naivebayes.zip`. After unzipping, you should see 2 python files and 3 subfolders.

You'll be editing:
`naivebayes.py` — your code goes here

Please don't modify:
`util.py` — python functions/classes that will be useful as you write your classifier

Data:
`data/spam/` — Spam emails
`data/ham/` — Non-spam (commonly called ham) emails
`data/testing/` — Test data for you to classify

(The data used in this assignment comes from a preprocessed version of the Enron email database. See V. Metsis, I. Androutsopoulos and G. Paliouras, *Spam Filtering with Naive Bayes – Which Naive Bayes?" Proceedings of the 3rd Conference on Email and Anti-Spam (CEAS 2006), Mountain View, CA, USA, 2006.*)

For this problem, you should assume that each email i has an unobserved class variable $C^{(i)}$ which takes on either the value “spam" or the value “ham". Recall the notion of *features*: in our case, these features will be binary word occurrences. Let $\{w_1, \ldots, w_J\}$ be the set of all words used in all emails. Then, let $Y_j^{(i)}$ be a binary random variable that takes on value 1 if w_j appears in email i, and 0 if it does not. You can imagine that words like “viagra" would be very likely to occur in spam, but not as likely in ham.

The joint probability of class labels and features is given by

$$p_{C^{(1)},Y_1^{(1)},\ldots,Y_J^{(1)},\ldots,C^{(n)},Y_1^{(n)},\ldots,Y_J^{(n)}}\left(c^{(1)}, y_1^{(1)}, \ldots, y_J^{(1)}, \ldots, c^{(n)}, y_1^{(n)}, \ldots, y_J^{(n)}; \theta\right) = \prod_{i=1}^{N}\left[p_C(c^{(i)};\theta)\prod_{j=1}^{J} p_{Y_j|C}(y_j^{(i)}|c^{(i)};\theta)\right],$$

where θ consists of all the parameters $s, p_1, \ldots, p_J, q_1, \ldots, q_J$. We will assume that:

$$p_{C^{(i)}}(c^{(i)};\theta) = \begin{cases} s & \text{if } c^{(i)} = \text{spam} \\ 1-s & \text{if } c^{(i)} = \text{ham} \end{cases}$$

$$p_{Y_j|C}(y_j^{(i)}|\text{ham};\theta) = \begin{cases} p_j & \text{if } y_j^{(i)} = 1 \\ 1-p_j & \text{if } y_j^{(i)} = 0 \end{cases}$$

$$p_{Y_j|C}(y_j^{(i)}|\text{spam};\theta) = \begin{cases} q_j & \text{if } y_j^{(i)} = 1 \\ 1-q_j & \text{if } y_j^{(i)} = 0 \end{cases}$$

The maximum likelihood estimate for the parameter p_j is just the fraction of ham emails that contain the word w_j, and similarly, the maximum likelihood estimate for q_j is the fraction of spam emails that contain the word w_j. You should also be able to see that the maximum likelihood estimate for s is the

fraction of training documents that are spam. In order to avoid issues with unseen words, you should use Laplace smoothing (for autograder purposes, please use Laplace smoothing where a pseudocount of 1 is added per possible outcome; **don't do Laplace smoothing for the parameter s**).

- **(a)***Training*. First, we'll estimate the parameters.
  - Implement the function `get_counts`, which counts the number of files each word occurs in.
  - Implement the function `get_log_probabilities`, which, for each word, computes the log of a smoothed frequency for that word.
  - Implement the function `learn_distributions`. This function takes two lists of filenames, one for spam and one for ham, and learns the parameters for each word for each class.

  You may find the class `Counter` and `defaultdict` provided in the built-in Python `collections` module helpful. For example, a dictionary where missing keys are given a default value of 0 could be implemented as either of the lines below:

```
my_counter = Counter()
my_counter = defaultdict(lambda : 0)
```

- **(b)***Testing.* Implement the function `classify_email`, which classifies a new email using the MAP rule and Equation eq:naive-bayes-sum. You can then try it out using

  ```
  python naivebayes.py data/testing/ data/spam/ data/ham/
  ```

  How well does your classifier perform?

- **(c)** Estimating s from data is heavily reliant on the sizes of the data. In the real world, it's often difficult to find good training examples for ham, since nobody wants to give out their private email for the world to read. As a result, spam datasets often have many more spam examples than ham

examples.

By setting s manually, we can adjust how much the algorithm favors catching spam at the expense of falsely flagging a ham email. Try setting s to a few values manually, and briefly explain what happens to your performance as s increases and decreases. Specifically look at what happens to the number of false positives (ham emails incorrectly classified as spam) and the number of true positives (spam emails correctly classified as spam). **This part is multiple-choice. Scroll all the way down to the bottom of this page to provide your answer. For just this part, the number of attempts is *1*.**

THIS IS WHERE YOU PASTE YOUR CODE

```
 9 def get_counts(file_list):
10     """
11     Computes counts for each word that occurs in the files in file_list.
12 
13     Inputs
14     ------
15     file_list : a list of filenames, suitable for use with open() or
16                 util.get_words_in_file()
17 
18     Output
19     ------
20     A dict whose keys are words, and whose values are the number of files the
21     key occurred in.
22     """
23     ### TODO: Comment out the following line and write your code here
24     from collections import Counter
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Test results

Hide output

CORRECT

Test: get_counts(['autograder_ham1', 'autograder_ham2', 'autograder_spam1', 'autograder_spam2', 'autograder_spam3', 'autograder_test1', 'autograder_test2', 'autograder_test3', 'autograder_test4', 'autograder_test5'])

Output:

```
{"jackrabbit": 1, "from": 1, "nine": 1, "space": 1, "snakes": 1, "plan": 2, "plane": 1, "cat": 5, "on": 1, "shark": 5, "interrobang": 4, "with": 1, "outer": 1, "a": 1, "cats": 1, "dog": 2}
```

Test: get_log_probabilities(['autograder_ham1', 'autograder_ham2', 'autograder_spam1', 'autograder_spam2', 'autograder_spam3', 'autograder_test1', 'autograder_test2', 'autograder_test3', 'autograder_test4', 'autograder_test5'])

Output:

```
{"jackrabbit": -1.791759469228055, "from": -1.791759469228055, "nine": -1.791759469228055, "space": -1.791759469228055, "snakes": -1.791759469228055, "plan": -1.3862943611198906, "plane": -1.791759469228055, "cat": -0.6931471805599453, "dog": -1.3862943611198906, "shark": -0.6931471805599453, "on": -1.791759469228055, "interrobang": -0.8754687373538999, "with": -1.791759469228055, "outer": -1.791759469228055, "cats": -1.791759469228055, "a": -1.791759469228055}
```

Test: learn_distributions([['autograder_spam1', 'autograder_spam2', 'autograder_spam3'], ['autograder_ham1', 'autograder_ham2']])

Output:

```
[[["cat", -0.2231435513142097], ["dog", -0.916290731874155], ["shark", -0.5108256237659907]], [["cat", -0.6931471805599453], ["dog", -0.6931471805599453], ["interrobang", -0.2876820724517809], ["plan", -0.6931471805599453], ["shark", -0.2876820724517809]], [-0.5108256237659907, -0.916290731874155]]
```

Test: classify_emails(['autograder_spam1', 'autograder_spam2', 'autograder_spam3'], ['autograder_ham1', 'autograder_ham2'], ['autograder_test1', 'autograder_test2', 'autograder_test3', 'autograder_test4', 'autograder_test5'])

Output:

```
spam    spam    ham     spam    ham
```

Hide output

Submit You have used 4 of 100 attempts

✔ Correct (90/90 points)

Mini-project: Email Spam Detection Part (c)

10/10 points (graded)

THIS IS WHERE YOU ANSWER PART (C)

As s increases: the number of false positives

- (•) tends to increase ✔
- ( ) tends to decrease
- ( ) stays roughly the same

and the number of true positives

- (•) tends to increase ✔
- ( ) tends to decrease
- ( ) stays roughly the same

Submit You have used 1 of 1 attempt

✔ Correct (10/10 points)

Discussion

Topic: Mini-project 3 / Mini-project: Email Spam Detection

Show Discussion