CS540 Spring 2024 Homework 2

Due: 8 February 2024, 11:00am

1 Probabilistic Language Identification

Your detectives arrived at company X only to find an important piece of evidence, a business letter, has been shredded. After painstakingly recovering the pieces, the detectives reported to you that all they could do was to count how many times each alphabet character appeared in the letter. Your mission, should you choose to accept it, is to estimate whether the letter was written in English or Spanish.

1.1 Build your own digital shredder

We gave you a python program skeleton **hw2.py**. You should complete the program. The first thing it should do is to read in a plain text letter and digitally shred it: print out the character counts.

- Your program should read from the input file named "letter.txt" in the same directory as hw2.py. We will supply this text file when we grade your program. For developing your code, copy any one of the *samples/letter*[0−4].*txt* files as letter.txt.
- The input file may contain any printable ASCII characters and can be short (e.g. a single word) or long (e.g. an article). In particular, there can be a mix of upper and lower case characters; numbers and other printable ASCII characters are possible too.
- The letter is known to be entirely written in only one language
- Your program should ignore case, i.e. merge "A" and "a" counts together, and so on. This is known as case-folding.
- **Your program should only count characters A to Z (after case-folding), and ignore all other characters such as space, punctuations, etc.**
- For this part, your program should output the string **Q1** on its own line, followed by exactly 26 lines. Each line contains the upper case letter, a space, the count of that letter (after case-folding). You can see the sample output in *samples/letter*$[0-4]$*_out.txt*.

For example, assume letter.txt is the following:

```
Hi! I'll go :-)
```

This part of your program should output:

```
Q1
A 0
B 0
C 0
D 0
E 0
F 0
G 1
H 1
I 2
J 0
K 0
L 2
M 0
N 0
O 1
P 0
Q 0
R 0
S 0
T 0
U 0
V 0
W 0
X 0
Y 0
Z 0
```

1.2 Language identification via Bayes rule

We arrange the 26 counts into a 26-dimensional count vector

$$X = (X_1, \ldots, X_{26}),$$

where X_1 is the count of A, X_{26} is the count of Z, and so on. X is the observed evidence. We are interested in the source language $y \in \{English, Spanish\}$. Our goal is to compute the conditional probability

$$P(Y = y \mid X).$$

For this, we will use the Bayes rule:

$$P(Y = y \mid X) = \frac{P(X \mid Y = y)P(Y = y)}{P(X)} = \frac{P(X \mid Y = y)P(Y = y)}{\sum_{y' \in \{English, Spanish\}} P(X \mid Y = y')P(Y = y')}.$$

We need to know the following:

- $P(Y = y)$ the prior probability that the letter is of language y, without seeing any shredded evidence. For this homework, assume

$$P(Y = English) = 0.6, P(Y = Spanish) = 1 - P(Y = English) = 0.4.$$

- $P(X \mid Y = y)$ the conditional probability of observing evidence X given language y. We will use the multinomial probability. Under this model, for $y = English$ let

$$e = (e_1, \ldots, e_{26})$$

be a 26-dimensional multinomial parameter vector of English where $e_i \geq 0$ for $i = 1 \ldots 26$, and $\sum_{i=1}^{26} e_i = 1$. e_i is the probability of the ith character in English, i.e. e_1 is for A, e_{26} is for Z. Similarly, let

$$s = (s_1, \ldots, s_{26})$$

be the parameter vector of Spanish. e and s are given to you as two plain text files **e.txt, s.txt** with self-explanatory format.

Then, the multinomial probability is defined as

$$P(X \mid Y = y) = C(X) \prod_{i=1}^{26} p_i^{X_i}$$

where p is e if $y = English$ or s if $y = Spanish$, and $C(X)$ is the multinomial coefficient:

$$C(X) = \frac{\left(\sum_{i=1}^{26} X_i\right)!}{X_1! \cdot \ldots \cdot X_{26}!}.$$

You now have everything needed to mathematically compute $P(Y = y \mid X)$ for $y \in \{English, Spanish\}$.

1.3 Computational considerations

However, as a computer scientist we want to avoid unnecessary computation and to avoid underflow. By inspecting $P(Y = y \mid X)$ and remembering the normalization $P(Y = English \mid X) + P(Y = Spanish \mid X) = 1$, we see that any terms that does not depend on y can be ignored, as long as we normalize in the end. Concretely, we say

$$P(Y = y \mid X) \propto P(Y = y) \prod_{i=1}^{26} p_i^{X_i}$$

where we removed $C(X)$ and $P(X)$. Equivalently, we can compute (recall p depends on y)

$$f(y) = P(Y = y) \prod_{i=1}^{26} p_i^{X_i}$$

and then normalize:

$$P(Y = y \mid X) = \frac{f(y)}{f(English) + f(Spanish)}.$$

We also do not like multiplying many small numbers for fear of underflow, so we want to compute things in the log domain:

$$F(y) = \log f(y) = \log P(Y = y) + \sum_{i=1}^{26} X_i \log p_i.$$

We will use the default base in python which is natural log. From $F(y)$ we can compute the desired conditional probability by

$$P(Y = y \mid X) = \frac{e^{F(y)}}{e^{F(English)} + e^{F(Spanish)}}.$$

However, $F()$ can be very negative if the letter is long. This leads to $e^{F()}$ underflow, and we may have a divide-by-zero error. To make the computation more robust, let's fix $y = English$ and note that

$$P(Y = English \mid X) = \frac{e^{F(English)}}{e^{F(English)} + e^{F(Spanish)}} = \frac{1}{1 + e^{F(Spanish)-F(English)}}.$$

Now, $F(Spanish)-F(English)$ can be very positively large and $e^{F(Spanish)-F(English)}$ overflows toward infinity. But we know the ratio should approximate 0 in that case. Conversely, $F(Spanish) - F(English)$ can be very negative and $e^{F(Spanish)-F(English)}$ underflows toward zero. But we know the ratio should approximate 1 in that case. Thus in your python program **you must do an if-else check as follows**:

- If $F(Spanish) - F(English) \geq 100$ then simply set $P(Y = English \mid X) = 0$;
- If $F(Spanish) - F(English) \leq -100$ then simply set $P(Y = English \mid X) = 1$;
- Otherwise compute

$$P(Y = English \mid X) = \frac{1}{1 + e^{F(Spanish)-F(English)}}.$$

1.4 Program output specification

We should be able to run your program with the following command:

```
python3 hw2.py
```

Your program should produce output that follow the specification in the following 4 questions.

Q1 ([25] points): Print "Q1" followed by the 26 character counts for letter.txt. Please see sample output files for the exact format. We will test it on different letter.txt to check your count output.

Q2 ([25] points): Compute $X_1 \log e_1$ and $X_1 \log s_1$. Print "Q2" then these values up to **4 decimal places** on two separate lines. Sample output for **letter.txt** is as follows:

```
-0.0000
-0.0000
```

Output format explanation: The first line is the value of $X_1 \log e_1$ and the second line is the value of $X_1 \log s_1$. There is no empty line in between or additional blank spaces. Notice that the values are 0 since 'A' (or 'a') does not appear in the text! The negative sign is due to python implementation, we also accept 0.0000 as an answer

Q3 ([25] points): Compute $F(English)$ and $F(Spanish)$. Similarly, print "Q3" followed by their values up to **4 decimal places** on two separate lines.

Q4 ([25] points): Compute $P(Y = English \mid X)$. Print "Q4" then this value up to **4 decimal places**.

You are expected to print out the answers to the above questions to stdout. In other words, your program should print the values to your screen. You can use the python **print()** function. Note that the autograder behaves strangely if passing multiple, comma-seperated arguments to print(). Instead, cast all your arguments to strings and join them with "+" before passing your answers to print(). Do not add additional blank lines between your outputs. Carefully follow the format described above as your submission will be automatically graded using an autograder. We provided four sample input and output files for your reference.

1.5 Files in the assignment

The homework is released as a single **hw2.zip** which contains the following files and directories:

- English character probability file **e.txt**
- Spanish character probability file **s.txt**
- directory **samples** contains sample letters and the desired output. Your outputs should be identical to the ones provided. HINT: One way to check whether your output matches the expected output is to use **vimdiff** on Linux Terminals (use CSL Machines) for a line by line comparison between two files. Usage:

```
>> vimdiff file1 file2
```

You can save your output to a file and then compare using vimdiff as follows

```
>> python3 hw2.py | tee your_output
>> vimdiff your_output letter_out.txt
```

- Skeleton code **hw2.py** to help you get started. It contains the following two functions:
  - **shred()**: Implements digital shredder functionality described in section 1.1. You are provided the definition and return value. You will need to complete the function to answer Q1.
  - **get_parameter_vectors()**: Reads "e.txt" and "s.txt" and returns the e and s vectors described in section 1.2. We have already implemented this function for you. You can directly use the returned values in your code.
- Take special note of the contents of letter4. This is a letter written in German. Our model only knows about English and Spanish. It is useful to think about how the model will respond to languages it doesn't know.

You are free to implement the rest of the assignment as you wish using your python coding skills.

1.6 Deliverables

Please submit your file named hw2.py to gradescope. Do not submit a Jupyter notebook .ipynb file. You can only use functions supported by python's standard library[1]. Do not import any packages that necessitate additional installation, including numpy. It will not be supported by Gradescope.

ALL THE BEST!

[1] https://docs.python.org/3/library/index.html