Lisa Yan and Jerry Cain
CS 109

Problem Set #4
October 16, 2020

Problem Set #4
Due: 1:00pm on Monday, October 26

With problems by Mehran Sahami and Chris Piech

- Submit on Gradescope by 1:00pm Pacific on Monday, October 26th, for a small, "on-time" bonus.
- All students have a pre-approved extension, or "grace period" that extends until Wednesday 1:00pm Pacific, when they can submit with no penalty. **The grace period expires on 1:00 Pacific on Wednesday, October 28th**, after which we cannot accept further late submissions.
- **Collaboration policy:** You are encouraged to discuss problem-solving strategies with each other as well as the course staff, but you must write up your own solutions and submit individual work. Please cite any collaboration at the top of your submission.
- **Tagging written problems**: When you submit your written PDF on Gradescope you must tag your PDF, meaning that you must assign pages of your PDF as answers to particular questions so that we can properly grade your submission. For problem sets, we are deducting **2 points** for any submissions that do not have all questions tagged.
- **For each problem, briefly explain/justify how you obtained your answer.** Brief explanations of your answer are necessary to get full credit for a problem even if you have the correct numerical answer. The explanations help us determine your understanding of the problem whether or not you got the correct answer. Moreover, in the event of an incorrect answer, we can still try to give you partial credit based on the explanation you provide. It is fine for your answers to include summations, products, factorials, exponentials, or combinations; you don't need to calculate those all out to get a single numeric answer.
- If you handwrite your solutions, you are responsible for making sure that you can produce **clearly legible** scans of them for submission. You may also use any word processing software you like for writing up your solutions. On the CS109 webpage we provide a template file and tutorial for the LaTeX system, if you'd like to use it.

Written Problems

Submit your solutions to these written problems as a single pdf file on Gradescope.

1. Say that of all the students who will attend Stanford, each will buy at most one laptop computer when they first arrive at school. 40% of students will purchase a PC, 30% will purchase a Mac, 10% will purchase a Linux machine and the remaining 20% will not buy any laptop at all. If 15 students are asked which, if any, laptop they purchased, what is the probability that exactly 6 students will have purchased a PC, 4 will have purchased a Mac, 2 will have purchased a Linux machine, and the remaining 3 students will have not purchased any laptop?

Continued on next page...

2. Users independently sign up for two online social networking sites, Lookbook and Quickgram. On average, 7.5 users sign up for Lookbook each minute, while on average 5.5 users sign up for Quickgram each minute. The number of users signing up for Lookbook and for Quickgram each minute are independent. A new user is defined as a new account, i.e., the same person signing up for both social networking sites will count as two new users.
   a. What is the probability that more than 10 new users will sign up for the Lookbook social networking site in the next minute?
   b. What is the probability that more than 13 new users will sign up for the Quickgram social networking site in the next 2 minutes?
   c. What is the probability that the company will get a combined total of more than 40 new users across both websites in the next 2 minutes?

3. You go on a camping trip with two friends who each have a mobile phone. Since you are out in the wilderness, mobile phone reception isn't very good. One friend's phone will independently drop calls with 20% probability. Your other friend's phone will independently drop calls with 30% probability. Say you need to make 6 phone calls, so you randomly choose one of the two phones and you will use that *same* phone to make all your calls (but you don't know which has a 20% versus 30% chance of dropping calls). Of the first 3 (out of 6) calls you make, one of them is dropped. What is the conditional expected number of dropped calls in the 6 total calls you make (conditioned on having already had one of the first three calls dropped)?

4. Our ability to fight contagious diseases depends on our ability to model them. One person is exposed to llama-flu. The method below models the number of individuals who will get infected.

```
from scipy import stats
"""
Return number of people infected by one individual.
"""
def num_infected():
  # most people are immune to llama flu.
  # stats.bernoulli(p).rvs() returns 1 w.p. p (0 otherwise)
  immune = stats.bernoulli(p = 0.99).rvs()
  if immune: return 0

  # people who are not immune spread the disease far by
  # making contact with k people (up to 100).
  spread = 0
  # returns random # of successes in n trials w.p. p of success
  k = stats.binom(n = 100, p = 0.25).rvs()
  for i in range(k):
    spread += num_infected()

  # total infections will include this individual
  return spread + 1
```

What is the expected return value of `numInfected()`?

5. Choose a number X at random from the set of numbers $\{1, 2, 3, 4, 5\}$. Now choose a number at random from the subset no larger than X, that is from $\{1, \ldots, X\}$. Let Y denote the second number chosen.
   a. Determine the joint probability mass function of X and Y.
   b. Determine the conditional mass function of X given $Y = i$. Do this for $i = 1, 2, 3, 4, 5$.
   c. Are X and Y independent? Justify your answer.

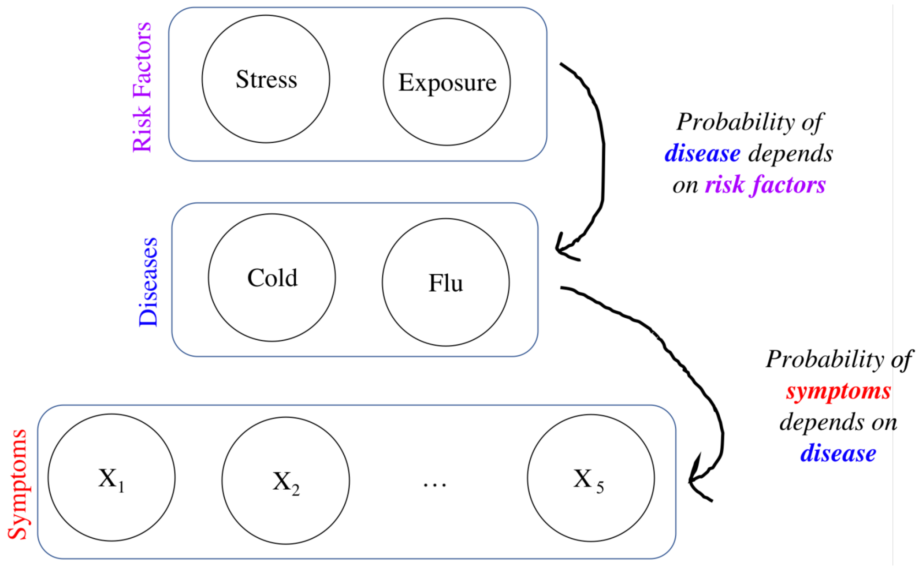
6. Let $X_1, X_2, \ldots$ be a series of independent random variables which all have the same mean μ and the same variance σ^2. Let $Y_n = X_n + X_{n+1}$. For $j = 0$, 1, and 2, determine $\text{Cov}(Y_n, Y_{n+j})$. Note that you may have different cases for your answer depending on the value of j.

Coding Problems

Download the starter code and implement the functions as described. You MUST use the starter code. Do not add import statements to the starter code. When you are ready to submit, upload only and `webmd.py` and `federalist.py` directly to Gradescope under "PSet4 - Coding" (not a zip file). Do not modify the file name or any function names. Do not submit any other files, such as `util.py` or `probabilities.py`; if you submit extra files, they'll simply be ignored during grading.

The Gradescope autograder will report your problem score; there are no hidden tests. You can submit an unlimited number of times. We will only use the score from your most recent submission. No partial credit will be given if you do not pass the autograder.

7. We are writing a WebMD program that is slightly larger than the one we worked through in class. In this program we predict whether a user has a flu ($F = 1$) or cold ($C = 1$) based on knowing any subset of 5 potential binary symptoms (e.g., headache, sniffles, fatigue, cough, fever) and a subset of binary risk factors (exposure, stress).


Risk Factors
Stress
Exposure
Probability of disease depends on risk factors
Diseases
Cold
Flu
Probability of symptoms depends on disease
Symptoms
X_1
X_2
…
X_5

The file `probabilities.py` implements the below functions. You **should not** modify these functions as we may use different probabilities when grading.

- The functions `probStress()` and `probExposure()` return the prior probabilities for stress and exposure, respectively.
- The functions `probCold(s, e)` and `probFlu(s, e)` return the probability that a patient has a cold or flu, given the state of the risk factors stress (`s`) and exposure (`e`).
- The function `probSymptom(i, f, c)` returns the probability that the `i`th symptom (X_i) takes on value 1, given the state of cold (`c`) and flu (`f`): $P(X_i = 1|F = f, C = c)$, for $i = 1, \dots, 5$.

We would like to write code that computes the probability $P(\text{Flu} = 1|\text{Exposure} = 1, X_2 = 1)$: the probability of flu *conditioned on observing* that the patient has had exposure to a sick friend and that they are experiencing Symptom 2 (sniffles):

Implement the function `inferProbFlu()` in `webmd.py` that computes the probability $P(\text{Flu} = 1|\text{Exposure} = 1, X_2 = 1)$ using **Rejection Sampling**. This function takes one parameter `ntrials`, which is the number of observations to generate:

```
def inferProbFlu(ntrials=1000000) # P(Flu = 1 | Exposure = 1 and X_2 = 1)
```

The Federalist Papers

The Federalist Papers is a body of essays written between 1787 and 1788 by Alexander Hamilton, James Madison, and John Jay under the collective pseudonym "Publius." Suppose that James Madison and Alexander Hamilton both claim to have written Federalist Paper No. 53 (we do not consider John Jay as a possible author in this problem). Over the next few problems, we will tackle the problem of deciding who wrote this paper probabilistically, given the contents of the document and historical evidence of the two authors' penmanship. It's a big task, but you all are ready :-)

Chapter 1: Mathematical Model

Before we get to the code, we need to first understand the model, which we discuss in Lecture Notes 11 as follows: Let there be n total words in the unknown document. There are m unique words in the document, where c_i is the number of times word i appears, and $\sum_{i=1}^{m} c_i = n$. If Hamilton wrote the document, then each of the n words has probability p_i of being unique word i, where $i = 1, \dots, m$. If Madison wrote the document, then each of the n words has probability q_i of being unique word i.

8. **[Written]** We set up the problem as comparing the probabilities that each author wrote the document: If the probability that Hamilton is the author (given the document's contents) is greater than the probability that Madison is the author (given the document's contents), then we predict that Hamilton is the author (Equation (1) in Lecture Notes 11, Section 3). Otherwise, we predict Madison.

   We make two assumptions: first, prior to knowing the document's contents, Hamilton and Madison are equally likely to be the author; second, given the author, the document's contents can be modeled using a Multinomial random variable, where the probability of producing word i is known (p_i for Hamilton, q_i for Madison). Use Bayes' Theorem to show that Equation (1) is equivalent to the below expression (Equation (2) in Lecture Notes 11):

$$\frac{\prod_{i=1}^{m} p_i^{c_i}}{\prod_{i=1}^{m} q_i^{c_i}} > 1$$

If the above inequality holds, then we predict Hamilton as the author. Otherwise, we predict Madison. (Lecture Notes 11: `http://web.stanford.edu/class/cs109/lectureNotes/LN11_joint_rvs.pdf`)

Chapter 2: Data processing

It's time to code! While the Gradescope autograder tests each function you write, we **urge** you to develop incrementally and **test locally**. We suggest that you verify that you pass all Gradescope tests on previous parts *before* moving to next parts, though it may help to read ahead.

To run and test your code locally, run `python3 federalist.py` (if OS X; if Windows, run `python federalist.py`) in your Python environment. This will run the `main` method, which runs each of the functions you are asked to write. We won't be grading anything in `main`, but the provided code shows how the functions might be used; you can modify `main` as you like.

9. We collect each author's word frequency data from existing documents, which involves recording word counts (which we've implemented for you) and computing word probabilities (your task).

   In `util.py`, the provided function `make_word_count_map` takes a *word list* (i.e., a Python list of strings) and returns a dictionary with key:value pairs (word: number of times the key appeared in the word list). Do not modify this function. For example:

```
>>> word_list = ['you', 'are', 'a', 'friend', 'and', 'a', 'role', 'model']
>>> word_counts = make_word_count_map(word_list))
>>> print(word_counts)
{'you': 1, 'are': 1, 'a': 2, 'friend': 1, 'and': 1, 'role': 1, 'model': 1}
```

   a. **[Coding]** In `federalist.py`, implement the function `make_word_prob_map`, which takes a word list and returns a dictionary with key:value pairs (word: probability that this word appears in the word list). Approximate the probability as the frequency of the word in the word list, i.e. the probability that the word is drawn if all words in the word list are equally likely. (Hint: It might be useful to use the provided `make_word_count_map` function.)

   b. **[Written]** The provided function `get_probability` in `util.py` takes two parameters—a map of word probabilities (i.e., the return value of `make_word_prob_map`) and a single word—and returns the probability of the word `word` appearing. Notice that it returns a small `EPSILON` value if the word does not exist in the `word_map` dictionary. Why is this `EPSILON` term necessary? Briefly explain in 1-3 sentences. (Do not modify this function; just read the implementation.)

Chapter 3: A Small Lesson in Underflow

A good practice in engineering is to first test our implementation on a small task before running it on the final task. The next part of this problem will show that while our prediction model is mathematically correct, it is numerically unstable and will not work on large documents.

We aren't using `unknown.txt` in this chapter—we're testing our model on a smaller file. All the information you need for this part is in the data table below, but if you're interested, the smaller file is also included in the starter code as `underflow.txt`.

10. **[Written]** Before writing code, you'll test the theory out by hand. Try to compute the ratio $(\prod_{i=1}^{m} p_i^{c_i})/(\prod_{i=1}^{m} q_i^{c_i})$ for a smaller document whose p_i, q_i and c_i values are provided in the table below (taken from `hamilton.txt`, `madison.txt`, and `underflow.txt`, respectively). You may assume that the below words are the only words that appear in this smaller document.

i	word	p_i	q_i	c_i
1	"the"	0.08	0.094	193
2	"of"	0.059	0.056	128
3	"to"	0.037	0.031	73
4	"and"	0.032	0.031	62

Feel free to use a calculator, Python, WolframAlpha, or whatever computing tool you like. What you will most likely encounter is an issue related to *arithmetic underflow*: The smallest representable number in Python is approximately 5×10^{-324}, beyond which Python rounds down to zero (Wikipedia link for those interested: `https://en.wikipedia.org/wiki/Double-precision_floating-point_format`)

Using the information above, briefly explain why arithmetic underflow will occur in this (very plausible) situation. We aren't grading code for this question, so please include your work (e.g., reasoning, numeric justification, or code screenshot) only on the PDF that you upload to Gradescope.

11. **[Written]** The ratio of probabilities we used above leads to arithmetic underflow. In practice, we can use the properties of logarithms to produce numerically stable results.

Use the properties of logarithms to show that the inequality in Problem 8 is equivalent to the below expression (Equation (3) on Lecture Notes 11). If the below inequality holds, then we predict Hamilton as the author (otherwise, we predict Madison):

$$\sum_{i=1}^{m} c_i \log p_i - \sum_{i=1}^{m} c_i \log q_i > 0$$

Chapter 4: The Final Stretch

It's time to bring all of our insights together to design a computationally tractable text analyzer!

12. **[Coding]** Let's use the log expression you derived in Problem 11 to determine the author of `unknown.txt`.

a. In `federalist.py`, implement `calculate_doc_log_prob`. This function takes two parameters:

   - a word probability map (i.e. the return value from `make_word_prob_map`) for a known author, either Madison or Hamilton, and
   - a map of word counts (i.e. the return value from `make_word_count_map`) for a mystery document whose author we're trying to predict.

   This function should return $\sum_{i=1}^{m} c_i \log f_i$, where f_i is the probability of word i, drawn from the first parameter to the function. This quantity is one of the terms in the expression you derived in Problem 11. (In practice, f_i will be either p_i or q_i, depending on the arguments that are passed to the function.)

   While you could theoretically use any log function, the Gradescope autograder expects that you will use the natural logarithm function in NumPy, which we've already imported for you (e.g., `log(10)` will return approximately `2.302`).

   If you're having trouble with this function, we recommend tracing out the logic on Problem 10's small example.

b. In `federalist.py`, implement `determine_author`, which takes four parameters and is called from the `main` function as below:

   - `word_list1`: a list of words that appear in `hamilton.txt`
   - `author1`: the string `"Alexander Hamilton"`
   - `word_list2`: a list of words that appear in `madison.txt`
   - `author2`: the string `"James Madison"`
   - `unknown_word_list`: a list of words that appear in `unknown.txt`

   This function should return your prediction of the author: either `author1`, i.e., `"Alexander Hamilton"`, or `author2`, i.e., `"James Madison"`. (Hint: you'll want to use both the provided functions in `util.py` and the functions you wrote in previous parts.)

You've just written a program that predicts the author of an unknown document. Give yourself a pat on the back! Exciting!