Foundations of AI
Instructor: Rasika Bhalerao
Assignment 1
Due September 16

Your first homework assignment is to write a generative language model!

Learning goals:

- Review Python
    - Practice choosing appropriate data structures
    - Practice decomposing an algorithm down into smaller methods / refactoring
    - Practice debugging code
- Become familiar with vectors in Numpy
- Refresh knowledge of probability
- Think deeply about the implications of choosing training data for a language model

How the model will work:
We will implement an algorithm that takes a few (`n`) characters and predicts the character which is most likely to come next. For example, if `n` is 3, and a user types `"hap"`, then the character which is most likely to come next is `'p'` (because `"happ"` is a very common sequence in English, often a part of `"happy"` or `"happen"`.)

We can then repeat the process: after having generated that `'p'`, we have `"happ"`, so the 3 most recent letters are then `"app"`. So, the next character is likely `'y'` or possibly `'e'`. We can keep generating the next letter given the `n` most recent letters until a special `"<end-of-sequence>"` character is generated.

What to do:

1. Create a class called `CharNGramLanguageModel`. Give it a constructor which takes `n` as a parameter.
2. In order to generate text, we first need to learn the necessary probabilities from an existing training dataset.
    a. You can choose the dataset from which you learn the probabilities. (https://www.kaggle.com/datasets/mehaksingal/olivia-rodrigo-lyrics-datasetl is a dataset of the lyrics from Olivia Rodrigo's songs, if you're looking for quick inspiration.)
    b. For each combination of `n` letters, calculate the relative frequency of the following letter. For example, you might find that (when `n` is 2) `"pr"` is followed by `'a'` 20% of the time, `'e'` 15% of the time, `'i'` 15% of the time, `'o'` 20% of the time, `'u'` 15% of the time, and `'y'` 15% or the time.
    c. You will need to choose an appropriate data structure to store these percentages. Iterate through the dataset, a sliding window of `n` characters at a time, updating

counts. After counting everything in the dataset, convert the raw counts into percentages.

   d. Don't forget to add a special `"<end-of-sequence>"` character at the end of each sequence, and include it in the counting. Spaces and punctuation are also characters.
   e. Refactor the code into helper methods as needed. The calculations should be done when the constructor is called, but not all the code needs to be in the constructor.

3. Write a method called `generate_character(prompt)` which takes a string and generates the next character. It should take the last `n` characters of `prompt` and randomly choose the next character. When randomly choosing the next character, it should use the percentages learned in the previous step as weights in the random number generation.
   a. For example, if the last 4 characters were `"sika"` and we know that the next character is 40% likely to be `'.'` (period), 40% likely to be `'?'` (question mark), and 20% likely to be the special `"<end-of-sequence>"` character, then we should randomly generate a period (probability 0.4), a question mark (probability 0.4) or the `"<end-of-sequence>"` character (probability 0.2). The `random.choices()` function may be useful.
   b. You may use `None` to represent the `"<end-of-sequence>"` character, though it is not necessary.
   c. If it encounters a sequence of `n` characters which we did not have in our original training dataset, then it can generate the next character randomly without the weights.
      i. Or, as an alternative, have it choose the next character based only on the single previous character. This will require calculating the frequencies for all the individual characters while doing the training in the constructor.
4. Write a method called `generate(prompt)`. It should take a string prompt and repeatedly call `generate_character`, adding the generated character to the end of the prompt. It should keep going, adding letters to the prompt, until `generate_character` returns the `"<end-of-sequence>"` character. Then it should return the string that it built.
5. Test it out! In a `main` function, create an instance of `CharNGramLanguageModel`. Ask the user for a prompt (`input` function) and generate what comes next!

What to turn in:
Please submit these via Gradescope:

- Your Python file with `CharNGramLanguageModel` and the `main` function in it. (The `main` function should only run if this file is directly run, not if this file is imported.)
- The output of your program with a few different values of `n`. Try some small values (like 3 or 5) and some large values (like 20 or 50).
- A text or pdf file with your answers to these questions:

- Describe how the choice of training dataset affected your model. What happens if we train a model on one type of data (like music lyrics) and then ask it to work with a different type of data (such as medical reports)? Which datasets do popular language models use, and how might this affect their output? (1 paragraph)
- How long did this assignment take you? (1 sentence)
- Whom did you work with, and how? (1 sentence each)
  - Discussing the assignment with others is encouraged, as long as you don't share the code.
- Which resources did you use? (1 sentence each)
  - For each, please list the URL and a brief description of how it was useful.
- A few sentences about:
  - What was the most difficult part of the assignment?
  - What was the most rewarding part of the assignment?
  - What did you learn doing the assignment?
  - Constructive and actionable suggestions for improving assignments, office hours, and class time are always welcome.