HW2: Markov Models of Natural Language

Fill in your name and the names of any students who helped you below.

> I affirm that I personally wrote the text, code, and comments in this homework assignment.
>
> I received help from [other student] who gave me suggestions on [problem].

- Austin Wuthrich 4/14/22

Comments are required for problems 1,2, and 3. They are not required for problem 4

Language Models

Many of you may have encountered the output of machine learning models which, when "seeded" with a small amount of text, produce a larger corpus of text which is expected to be similar or relevant to the seed text. For example, there's been a lot of buzz about the new GPT-3 model, related to its carbon footprint, bigoted tendencies, and, yes, impressive (and often humorous) ability to replicate human-like text in response to prompts.

We are not going to program a complicated deep learning model, but we will construct a much simpler language model that performs a similar task. Using tools like iteration and dictionaries, we will create a family of **Markov language models** for generating text. For the purposes of this assignment, an n-th order Markov model is a function that constructs a string of text one letter at a time, using only knowledge of the most recent n letters. You can think of it as a writer with a "memory" of n letters.

Data

Our training text for this exercise comes from the first 10 chapters of Jane Austen's novel *Emma*, which I retrieved from the archives at Project Gutenberg. Intuitively, we are going to write a program that "writes like Jane Austen," albeit in a very limited sense.

In [1]:
```
s = "CHAPTER I Emma Woodhouse, handsome, clever, and rich, with a comfortable home and happy disposition, seemed to unite
```

Comments and Docstrings

Here's what we expect:

- **Comments**: Use comments liberally to explain the purpose of short snippets of code.
- **Docstrings**: Functions (and, later, classes) should always be accompanied by a *docstring*. Briefly, the docstring should provide enough information that a user could correctly use your function ***without seeing the code.*** In somewhat more detail, the docstring should include the following information:
  - One or more sentences describing the overall purpose of the function.
  - An explanation of each of the inputs, including what they mean, their required data types, and any additional assumptions made about them.
  - An explanation of the output.

In future homeworks, as well as on exams, we will be looking for clear and informative comments and docstrings.

Code Structure

In general, there are many good ways to solve a given problem. However, just getting the right result isn't enough to guarantee that your code is of high quality. Check the logic of your solutions to make sure that:

- You aren't making any unnecessary steps, like creating variables you don't use.
- You are effectively making use of the tools in the course, especially control flow.
- Your code is readable. Each line is short (under 80 characters), and doesn't have long tangles of functions or `()` parentheses.

Ok, let's go!

Exercise 1

Write a function called `count_characters()` that counts the number of times each character appears in a user-supplied string `s`. Your function should loop over each element of the string, and sequentually update a `dict` whose keys are characters and whose values are the number of occurrences seen so far. You may know of other ways to achieve the same result. However, you should use the loop approach, since this will generalize to the next exercise.

Note: while the construct `for letter in s:` will work for this exercise, it will not generalize to the next one. Use `for i in range(len(s)):` instead.

Example usage:

```
count_characters("tortoise")
{'t' : 2, 'o' : 2, 'r' : 1, 'i' : 1, 's' : 1, 'e' : 1}
```

Hint: Yes, you did a problem very similar to this one on HW1.

Your Solution

In [2]:

```
# write count_characters() here
def count_characters(s):
    """
    A function that inputs a string and counts the number of instances of each
        character in the string and places the information into a dictionary.
    Param s: string of any size with characters to be counted.
    Returns: dictionary with keys of each appearing character along with its
        corresponding integer values representing number of occurences in the string.
    """

    D = {}                      #Dictionary declared

    for i in range(len(s)):  #iterates through the length of the string
        key = s[i]           #key for dictionary is the string char at a particular index i
        if key in D.keys():  #checks if key already exists in the dictionary
            val = D[key]     #if so, pulls value for given key from dictionary
            val+=1           #incriments value for key
            D.update({key:val}) #updates dictionary with incrimentation
        else:                #occurs if key not already in dictionary
            val = 1          #sets occurance value to one
            D.update({key:val}) #updates dictionary with new key:value entry
    return D                 #return new dictionary

count_characters("tortoise") #Using example from above
```

Out[2]:

```
{'t': 2, 'o': 2, 'r': 1, 'i': 1, 's': 1, 'e': 1}
```

Exercise 2

An `n` -*gram* is a sequence of `n` letters. For example, `bol` and `old` are the two 3-grams that occur in the string `bold` .

Write a function called `count_ngrams()` that counts the number of times each `n` -gram occurs in a string, with `n` specified by the user and with default value `n = 1` . You should be able to do this by making only a small modification to `count_characters()` .

Example usage:

```
count_ngrams("tortoise", n = 2)
```

```
{'to': 2, 'or': 1, 'rt': 1, 'oi': 1, 'is': 1, 'se': 1} # output
```

Your Solution

In [3]:

```
# write count_ngrams() here
def count_ngrams(s,n=1):
    """
    Function that inputs a string and counts the number of instances
        of each character in the string and places that information into a dictionary.
    Param s: string of any size with characters to be counted.
    Returns: dictionary with keys that are each appearing character and corresponding
        integer values representing frequency of occurence.
    """

    D = {}                      #Dictionary declared

    for i in range(len(s)):    #iterates through the length of the string
        key = s[i:(i+n)]       #dictionary key is the string of n char from particular index i
        if (len(key) != n):    #ensures ngrams are desired size
            continue
        elif key in D.keys(): #checks if key already exists in the dictionary
            val = D[key]       #if so, pulls value for given key from dictionary
            val+=1             #incriments value for key
            D.update({key:val}) #updates dictionary with incrimentation
        else:                  #occurs if key not already in dictionary
            val = 1            #sets occurance value to one
            D.update({key:val}) #updates dictionary with new key:value entry
    return D                   #returns new ngram dictionary

count_ngrams("tortoise",2)    #demonstration with example given
```

Out[3]:

```
{'to': 2, 'or': 1, 'rt': 1, 'oi': 1, 'is': 1, 'se': 1}
```

Exercise 3

Now we are going to use our `n` -grams to generate some fake text according to a Markov model. Here's how the Markov model of order `n` works:

A. Compute (`n` +1)-gram occurrence frequencies

You have already done this in Exercise 2!

B. Pick a starting (`n` +1)-gram

The starting (`n` +1)-gram can be selected at random, or the user can specify it.

C. Generate Text

Now we generate text one character at a time. To do so:

1. Look at the most recent `n` characters in our generated text. Say that `n = 3` and the 3 most recent character are `the` .
2. We then look at our list of `n+1` -grams, and focus on grams whose first `n` characters match. Examples matching `the` include `them` , `the` , `thei` , and so on.
3. We pick a random one of these `n+1` -grams, weighted according to its number of occurrences.
4. The final character of this new `n+1` gram is our next letter.

For example, if there are 3 occurrences of `them` , 4 occurrences of `the` , and 1 occurrences of `thei` in the n-gram dictionary, then our next character is `m` with probabiliy 3/8, `[space]` with probability 1/2, and `i` with probability `1/8` .

Remember: the ***3rd***-order model requires you to compute ***4***-grams.

What you should do

Write a function that generates synthetic text according to an `n` -th order Markov model. It should have the following arguments:

- `s` , the input string of real text.

- `n` , the order of the model.
- `length` , the size of the text to generate. Use a default value of 100.
- `seed` , the initial string that gets the Markov model started. I used `"Emma Woodhouse"` (the full name of the protagonist of the novel) as my `seed` , but any subset of `s` of length `n+1` or larger will work.

Demonstrate the output of your function for a couple different choices of the order `n` .

Expected Output

Here are a few examples of the output of this function. Because of randomness, your results won't look exactly like this, but they should be qualitatively similar.

```
markov_text(s, n = 2, length = 200, seed = "Emma Woodhouse")
```

```
Emma Woodhouse ne goo thimser. John mile sawas amintrought will on I kink you kno but every sh inat he
fing as sat buty aft from the it. She cousency ined, yount; ate nambery quirld diall yethery, yould hat
earatte
```

```
markov_text(s, n = 4, length = 200, seed = "Emma Woodhouse")
```

```
Emma Woodhouse!”—Emma, as love,            Kitty, only this person no infering ever, while, and tried
very were no do be very friendly and into aid,    Man's me to loudness of Harriet's. Harriet belonger
opinion an
```

```
markov_text(s, n = 10, length = 200, seed = "Emma Woodhouse")
```

```
Emma Woodhouse's party could be acceptable to them, that if she ever were disposed to think of nothing
but good. It will be an excellent charade remains, fit for any acquainted with the child was given up to
them.
```

Notes and Hints

Hint: A good function for performing the random choice is the `choices()` function in the `random` module. You can use it like this:

```
import random
```

```
options = ["One", "Two", "Three"]
weights = [1, 2, 3] # "Two" is twice as likely as "One", "Three" three times as likely.

random.choices(options, weights)

    ['One'] # output
```

The first and second arguments must be lists of equal length. Note also that the return value is a list -- if you want the value *in* the list, you need to get it out via indexing.

Hint: The first thing your function should do is call `count_ngrams` above to generate the required dictionary. Then, handle the logic described above in the main loop.

In [88]:

```
# write markov_text() here

def markov_text(s, n, length = 100, seed = "Emma Woodhouse"):
    """
    Function that performs the markov model for text generation using ngrams on a provided string.
    Param s: the inputted string from which n-gram probabilities and text are generated.
    Param n: the desired order of the model (size of each n-gram unit) as an integer.
    Param length: index length of the desired generated string as an integer. Default length 100
    Param seed: string of at least length n+1 that the markov model builds off. Default is "Emma Woodhouse"
    Returns: markov string probabilistically built from inputs
    """

    import random                               #In order to use random.choices()

    probability_dict = count_ngrams(s,n+1) #Build dictionary of ngram probabilities from s

    prob_dict_keys=probability_dict.keys() #Declare + initialize list of keys from dictionary
    ngram_keys=[]               #Declare empty list for n+1 ngrams in s matching ngram of seed

    #each iteration appends 1 char to end of growing seed string. Grows seed by length
    for i in range(length):
        #initialize ngram as last n chars of the seed
        ngram=seed[-n:]

        #initalize list of n+1 ngrams that match ngram up to nth index
        ngram_keys=[i for i in prob_dict_keys if i[:n]==ngram]

        #initialize list of probabilities for each potential n+1 ngram
        weights = [probability_dict[i] for i in ngram_keys]
        #use choice function from random to select random n+1 ngram based on weights
        choice = random.choices(ngram_keys, weights)

        #random.choices() returns list, so convert list into string as ngram
        ngram = choice[0]

        #concatenate seed with last char of n+1 ngram growing the seed by 1 length
        seed = seed + ngram[-1]

    #return string built char by char from the seed and data provided of s
    return seed
```

In [121…

```
# try out your function for a few different values of n
markov_text(s,2,250,"Emma Woodhouse") #n=2
```

Out[121…

```
"Emma Woodhouses agre; atted youghtly doneste an Knink hill. Elto not eve shen withat his trying eve ma's quirs.“Esplest
is wand winfor whou, himproart, he covedint on; but wit youst as broathert beemorseld in a mand way,” cas as I the exclit
wort and siould has ma"
```

In [120…

```
# try another value of n!
markov_text(s,6,250,"Emma Woodhouse") #n=6
```

Out[120…

```
"Emma Woodhouse's performances and she think so totally feeling. She had fall behave? Who cried Emma; “to meet with the l
etter. If I were very like Mr. Martin, I supposes ended on; and niece. Her father, to find it all out. She knew Mr. Knigh
tley, “who had never be"
```

In [124…

```
# try third value!
markov_text(s,10,250,"Emma Woodhouse") #n=10
```

Out[124…

```
"Emma Woodhouse, I would marry to-morrow. It was most unlikely that he should think and say, 'Grandpapa, can you give me
such kind encourager, for his admiration made him think every thing else appear!—I feel now as if I could think of those
two lines are”——“The be"
```

Exercise 4

Using a `for` -loop, print the output of your function for `n` ranging from `1` to `10` .

Then, write down a few observations. How does the generated text depend on `n` ? How does the time required to generate the text depend on `n` ? Do your best to explain each observation.

```
In [129…
# code here
for n in range(1,11):
    print('Ngram size '+str(n)+': ',markov_text(s,n,200,"Emma Woodhouse"),"\n")
```

```
Ngram size 1:  Emma Woodhouse f athe wherlagis Hand Exclines m wersurt y k werseranoth ond te pig waneryomanert bes a wo
r. lissieaf, t, way higus, ossig hin Do auten We h mesin fore t.”“I hendoulate bo Somontll owirope ie s—het c

Ngram size 2:  Emma Woodhouse fand lon comily to pietake der, be: nown her of warrot for she congs man deas favict abide
ple the youghtly: ing, be mand was to whou antior pre provention,”—but to kney, as ack fachould might but la

Ngram size 3:  Emma Woodhouse married, “At them! butterday, with plestily with a did Mr. Knight. Mr. Mr. Were herself. Sh
e a dea of be such wanten held not know him. This, Harting! but he way surry, when to reat appy minutes!—the

Ngram size 4:  Emma Woodhousekeeper who had belong with there converself fortune, and obscurity, so much she; “I am were
could purpose, which she mouth—I had be. It waiting lady with imprudent indepent, Mr. Woodhouse, and I ever

Ngram size 5:  Emma Woodhouse. The poor Miss Woodhouse in all that all on horses and her been want match at therefore sho
uld have been going of the eager than that point. He was there was a strangest to Emma herself entirely; but

Ngram size 6:  Emma Woodhouse in himself, or frightful way!”“But what you,” said Emma herself for her opinion, to absent
friend the tyrannic influence; and believe I have seldom from pushing the very happy to command that there w

Ngram size 7:  Emma Woodhouse's kindness, felt his person, and then the more knowledge and pleading at last four years ol
d—how she might draw back again, seemed to understand the uncle not let us be too solemnly.“But they make ev

Ngram size 8:  Emma Woodhouse's family that could be no doubt he is determination range and work at Harriet, the morrow,
and acknowledge of the utmost advantage of a house of her own line of society of Mr. Elton only looked like

Ngram size 9:  Emma Woodhouse's help, to get a great deal in the smallest doubt that her father to prevent the young ladi
es that his manner, and the pleasure to herself, and tried very much amiss about her.”“I have no doubt of hi

Ngram size 10:  Emma Woodhouse gratefully observed, “It is very clever, and rich, with a complete education, to any conne
xions; at the same kind office done for him! I think very well chosen, and very mutually attached, and Emma d
```

Discussion Section:

Observations: As the ngram size specified in the function increases, the readability of generated text improves. The text becomes more like english in coherence, structure, syntax, and diction. The overall "errors" decrease on average from n=1 to n=10; however, even at n=10, the generated text could not pass as coherent writing, let alone by Jane Austen. Also of note, it took significantly more time to generate the text with larger ngrams than the text with smaller ngrams.

Discussion: The generated text varies with the size of n because the algorithm processes the data in n+1 ngram chunks. So, as the size of the n increases, the ngram segments the algorithm observes and picks from to append the next character to the seed are larger segments of coherent text. The algorithm is able to identify and select from more specific fragments of text, which yeild the next characters to append to the generated text that are on average more relevant (less out of the blue like a random "). The function probably takes longer to generate text as the ngram size increases because of the larger amounts of data required to process at one time. At n=1, the function deals essentially segments the string into and constructs dictionaries and lists by individual characters; whereas, for n=10, the function works with index length 10 strings. A one character string takes up much less memory than a longer string, and so it is most likely easier for the computer to work with small, unspecific ngrams instead of larger, less frequent and more unique ngrams.

```
In [ ]:
```