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].

- David Nguyen

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 (https://en.wikipedia.org/wiki/GPT-3), related to its carbon footprint (https://www.forbes.com/sites/robtoews/2020/06/17/deep-learnings-climate-change-problem/#2781c1b16b43), bigoted tendencies (https://medium.com/fair-bytes/how-biased-is-gpt-3-5b2b91f1177), and, yes, impressive (and often humorous (https://aiweirdness.com/)) ability to replicate human-like text in response to prompts. (https://www.technologyreview.com/2020/07/20/1005454/openai-machine-learning-language-generator-gpt-3-nlp/)

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 (https://www.gutenberg.org/files/158/158-h/158-h.htm#link2H_4_0001). Intuitively, we are going to write a program that "writes like Jane Austen," albeit in a very limited sense.

In [69]:
```
s = "CHAPTER I Emma Woodhouse, handsome, clever, and rich, with a comfortable home and happy disposition, seemed
```

Comments and Docstrings

This is the first homework assignment in which you will be graded in part on the quality of your documentation and explanation of your code. 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 [10]:
```
# write count_characters() here
''' take in a string and then count the number of times each character appears '''
def count_characters(s):
    tempDict = {} #initialize empty dictionary

    #loop to go through all elements in the sentence (s)
    for i in range(len(s)):
        if s[i] not in tempDict: #for adding elements not in dict already
            tempDict[s[i]] = 1
        elif s[i] in tempDict:   #increment elements already in dict
            tempDict[s[i]] += 1
    return tempDict #return dict with key and value pairs

myString = input('Please input a string: ') #prompt input
count_characters(myString) #call function
```

```
Please input a string: tortoise
```

Out[10]:
```
{'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 [111]:
```
# write count_ngrams() here
''' counts the number of times each n-gram occurs in a string,
    with n specified by the user and with default value n = 1 '''
def count_ngrams(s, n = 1): #takes in string s and n with n = 1 as default
    tempDict = {} #initialize empty dictionary

    #loop to go through all values of n grams
    for i in range(len(s) - (n-1)):
        tempString = '' #initialize placeholder string

        #making the n grams string
        for a in range(n):
            tempString += s[i+a]

        #adding to dict or incrementing it
        if tempString not in tempDict:
            tempDict[tempString] = 1
        elif tempString in tempDict:
            tempDict[tempString] += 1


    return tempDict #return the dict

userString = input('Input a string: ')
userNum = int(input('Input n: '))

count_ngrams(userString, userNum)
```

```
Input a string: tortoises
Input n: 2
```

Out[111]:
```
{'to': 2, 'or': 1, 'rt': 1, 'oi': 1, 'is': 1, 'se': 1, 'es': 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 fi
ng as sat buty aft from the it. She cousency ined, yount; ate nambery quirld diall yethery, yould hat ear
atte
```

```
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 ver
y were no do be very friendly and into aid,    Man's me to loudness of Harriet's. Harriet belonger opinio
n 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 bu
t good. It will be an excellent charade remains, fit for any acquainted with the child was given up to th
em.
```

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 [5]:
```
s = "CHAPTER I Emma Woodhouse, handsome, clever, and rich, with a comfortable home and happy disposition, seemed
```

In [96]:
```
# write markov_text() here

import random

''' function to generate some fake text according to a Markov model '''
def markov_text(s, n, length = 100, seed = 'Emma Woodhouse'):
    gram = count_ngrams(s, n+1) #computing n+1 gram in a dict

    myString = seed #initialize a string to add to letters to

    #while to generate new string until string reaches desired length
    while(len(myString) < length):
        tempString = myString[-n:] # string to check against
        newDict = dict() #initialize a new dict

        #filtering down the dictionary to match last n letters of each key
        for (key, value) in gram.items():
            if tempString in key[:n]:
                newDict[key] = value

        #getting random gram
        newLetter = random.choices( list(newDict.keys()),list(newDict.values()) )

        #adding last char of random gram to new string
        myString += newLetter[0][-1]

    return myString
```

In [97]:
```
# try out your function for a few different values of n
markov_text(s, n = 2, length = 200, seed = "Emma Woodhouse")
```

Out[97]:
```
"Emma Woodhoused.""Dide the nothe deather not whaps, se ing wedly was and sperying, to he prompt; end st thounc
ou spen of ther yed, frantleforembe must way;—any light ver canded. Woody's be hou th Emme"
```

In [98]:
```
# try another value of n!
markov_text(s, n = 4, length = 200, seed = "Emma Woodhouse")
```

Out[98]:
```
"Emma Woodhouse. The by speak. But shall certainly often.""To be decidedly—remark with which woman, as a case,
and her own have recollections. She had neight youth who have forwards to Mrs. Weston's fo"
```

In [110]:
```
# try third value!
markov_text(s, n = 10, length = 200, seed = "Emma Woodhouse")
```

Out[110]:
```
'Emma Woodhouse, do advise me.""Oh no, no! the letter was from him, from Mr. Elton speak with great delight. It
opens his designs to his family, and Miss Bates, or when Mrs. Perry drank tea with Mrs. a'
```

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 [108]:
```
# code here
#for loop starting at n = 1 and ends at n = 10
for n in range(1,11):
    print('n = ' + str(n))
    print(markov_text(s, n, length = 200, seed = "Emma Woodhouse"))
```

```
n = 1
Emma Woodhouse isord asheyolint t c, atotlout, be inerayo ve areshererershithis ma ms ininn I e a le ouiprkyo a,
f abely, sthothid ave t omint h Wer hett n cors teredouce iesathay heyort. t pe iere mmea
n = 2
Emma Woodhousend and ind's se ing se wher it of an emplare as atimen whichand andly briew hatell begraden a mag
irs. I poven evertur hationes. Yough muche mak ot topits Thiculd alway led bou handeabley
n = 3
Emma Woodhouse"—and posed thation thing ally slow provery breline of ever it vised was perfectly kind.Miss Smit
h you, you me. Her visition you williarly sting you, the keep comparefor Miss Taylor acqu
n = 4
Emma Woodhouse him friend.Miss Nash we are. I now quicksight!—she only was in read most submit away better belo
nger which you in views, and her by thoughter as gone to a man, recommiss Taylor things i
n = 5
Emma Woodhouse's giggle; she same glad I do not pleasure, as he reason sober and scalloped him fifty time to sa
y, but would be little who all that I had been the myself; and one rose taste of the thin
n = 6
Emma Woodhouse's giving him exceedingly incline her school, Harriet's habits of her, involved in a good share a
s longer—but there was enough to containing, before. The rest of her own danger of a most
n = 7
Emma Woodhouse has given you up.""Dear me, how clever!—Could it really almost always right in any man's coming
to herself as for a wife, who wanted—exactly so,' as he does seem to be sitting began; an
n = 8
Emma Woodhouse. "So prettily done! Just as your doing. You persuading him to marry her.""He is very lucky, for
I would not be overpower me with it—but it seems a case of necessity. Mr. John Knightley,
n = 9
Emma Woodhouse hoped very soon.""My dear Harriet, to do one good. How trifling they make every thing but good.
It will be accomplish, and study for compliment, so I will tell you, thank you, my dear p
n = 10
Emma Woodhouse could not tell. She was obliged to you for it. I almost long to attempt her likeness almost befo
re he does. Your views for Harriet. "Poor creatures! one can think of him? Do you think I
```

[Discuss here]

As n increases, the generated text depends more heavily the text that is currently present. This means when n is smaller there are much more instances that may be used to get the next letter in the generated text. This leads to the generated text being somewhat gibberish. The coherency of the text increases as n increases. However it is also important to note that the generate text takes longer as n increases as well. This is likely due when we filter, we need to check more characters.

In [ ]: