HW2

January 21, 2022

1 Homework 2: Markov Models of Natural Language

1.0.1 Name: [Jiaxin Luo]

1.0.2 Collaborators: [Your collaborators]

This homework focuses on topics related to string manipulation, dictionaries, and simulations.

I encourage collaborating with your peers, but the final text, code, and comments in this homework assignment should still be written by you.

Submission instructions: - Submit `HW2.py` on Gradescope under "HW2 - Autograder". Do **NOT** change the file name. The grade you see is the grade you get for the accuracy portion of your code (so no surprises). The style and readability of your code will be checked by the reader aka human grader. - Convert this notebook into a pdf file and submit it on GradeScope under "HW2 - PDF". Make sure your text outputs in the latter problems are visible.

1.1 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.

[1]:
```
# This cell imports your functions defined in HW2.py

from HW2 import count_characters, count_ngrams, markov_text
```

1.2 Data

Our training text for this exercise comes from Jane Austen's novel *Emma*, which Professor Chodrow 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.

[2]:
```
with open('emma-full.txt', 'r') as f:
    s = f.read()
```

2 Problem 1: Define `count_characters` in HW2.py

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. Your function should then return this dictionary.

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

2.0.1 Example usage:

```
count_characters("Torto ise!")
{'T': 1, 't' : 1, 'o' : 2, 'r' : 1, 'i' : 1, 's' : 1, 'e' : 1, ' ': 1, '!': 1}
```

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

[3]:
```
# test your count_characters
print(count_characters("Torto ise!"))
```

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

How many times does 't' appear in Emma? How about '!'?

How many different types of characters are in this dictionary?

[4]:
```
# write your answers here
ans = count_characters(s)
print(ans)
print (ans.get("t"))
print (ans.get("!"))
print (len(ans))
```

```
{'C': 647, 'H': 1740, 'A': 708, 'P': 257, 'T': 1132, 'E': 1501, 'R': 215, ' ':
155119, 'I': 4013, '\n': 4711, 'm': 17907, 'a': 53664, 'W': 1355, 'o': 52894,
'd': 28329, 'h': 40827, 'u': 20606, 's': 41556, 'e': 84502, ',': 12020, 'n':
46986, 'c': 14815, 'l': 27537, 'v': 7645, 'r': 40699, 'i': 42586, 'w': 14933,
't': 58067, 'f': 14598, 'b': 10532, 'p': 10284, 'y': 15268, 'g': 13525, 'x':
1346, ';': 2353, '-': 574, '.': 8881, 'S': 951, 'q': 895, '’': 1114, 'M': 2795,
'B': 598, 'j': 688, 'k': 4351, '–': 3102, ':': 174, '?': 621, '(': 107, ')':
107, 'L': 133, 'O': 304, 'N': 300, '"': 2099, '!': 1063, '”': 2090, 'J': 432,
'Y': 438, 'K': 412, 'D': 253, '‘': 112, 'z': 174, 'F': 541, 'G': 147, 'U': 39,
'V': 101, 'Q': 15, '8': 3, '2': 5, '3': 1, 'ï': 4, 'é': 5, 'X': 32, '4': 1, '&':
3, 'ê': 8, 'à': 4, '7': 1, '1': 2, '0': 8, '[': 1, ']': 1, '6': 1}
```

```
58067
1063
82
```

3 Problem 2: Define `count_ngrams` in HW2.py

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`. Your function should return the dictionary. You should be able to do this by making only a small modification to `count_characters`.

3.0.1 Example usage:

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

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

[5]:
```
# test your count_ngrams here
print(count_ngrams("tortoise", n = 2))
```

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

How many different types of 2-grams are in this dictionary?

[6]:
```
# write your answer here
print (len(count_ngrams(s)))
```

```
82
```

4 Problem 3: Define `markov_text` in HW2.py

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:

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

You have already done this in Exercise 2!

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

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

4.0.3 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.

4.1 What you should do

Write a function `markov_text` 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`.

4.2 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
```

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

```
Emma Woodhouse!"-Emma, as love,            Kitty, only this person no infering ever, while, 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
```

4.3 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.

[7]:
```
# test your markov_text here
print('-------- n = 2 ----------')
print(markov_text(s, n = 2, length = 200, seed = "Emma Woodhouse"))
print('-------- n = 4 ----------')
print(markov_text(s, n = 4, length = 200, seed = "Emma Woodhouse"))
print('-------- n = 10 ----------')
print(markov_text(s, n = 10, length = 200, seed = "Emma Woodhouse"))
```

```
-------- n = 2 ----------
Emma Woodhouse gir. Kniounter, way, "Mr. Woon's of to not got thady der the of
youcklicklibe thriecioncein, athed begaire."

"But lard sk offe hishat regairs's terfaimenry theaks lite excul, elf troposen
was acend
-------- n = 4 ----------
Emma Woodhouse, thing she could better-but that convening. Woodhouse the us you,
who having almost fetch had give aware to be in so go,' said he, "and don't
closed the charactions. I can your chief to hear to entio
-------- n = 10 ----------
Emma Woodhouse's most obliging as to leaving you, it is what we happily have
never betrayed into paying a visit, which was to conceal his real situations
increasing, not lessening, Mr. Woodhouse, if you remember it
```

5 Problem 4

Try out your function for varying values of `n`. 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.

(extra credit) What do you think could happen if you were to repeat this assignment but in unit of words and not in unit of characters? For example, 2-grams would indicate two words, and not two characters.

(extra credit) What heuristics would you consider adding to your model to improve its prediction performance?

6 write your observations and thoughts here

As n gets bigger, the text generated becomes smoother and more specific or says more like Emma. The reason why this occurs can be simply explained as that when we have more past data, we can get a more accurate prediction. As n gets bigger, the function needs more time to run since there are fewer matched grams, which takes a long time to detect.

If the code aims to generate the text based on the unit of words, the accuracy would be much higher I suppose. However, it might take a long time based on the result of bigger n from above.

A grammar model to detect the mistakes and record them to avoid such mistakes.

[10]:
```
# run your markov_text here
print('-------- n = 1 ----------')
print(markov_text(s, n = 2, length = 200, seed = "Emma Woodhouse"))
print('-------- n = 3 ----------')
print(markov_text(s, n = 4, length = 200, seed = "Emma Woodhouse"))
print('-------- n = 7 ----------')
print(markov_text(s, n = 10, length = 200, seed = "Emma Woodhouse"))
print('-------- n = 11 ----------')
print(markov_text(s, n = 2, length = 200, seed = "Emma Woodhouse"))
print('-------- n = 19 ----------')
print(markov_text(s, n = 4, length = 200, seed = "Emma Woodhouse"))
print('-------- n = 100 ----------')
print(markov_text(s, n = 10, length = 200, seed = "Emma Woodhouse"))
```

```
-------- n = 1 ----------
Emma Woodhousee not tabot bertaine, haugh ust fou th hould be igive carde une
ding had to athrom, ing of ant wask thatillannignion hen ons paying,

"I but for thiss-quit dis conly call as aged artion is bor mom. I
-------- n = 3 ----------
Emma Woodhouse, which me, that can such disdain. The would been ventionate
farther, I was nother to put as her."

"Wrong a stout bids that they warm for only what her is not know, beyond the
pretty answer to myself
-------- n = 7 ----------
Emma Woodhouse had been the shock of finding in Harriet could not get rid of
every thing-to do any thing else. He took some music from a child, as that of
all her friend related. Mrs. Dixon, I mean-I do not know wh
-------- n = 11 ----------
Emma Woodhouse, he she end a side re comery eng territhe ing obodhou athe note
threar froblity, ankink wounlivery of to card, wanselithopoords do parried but
bodiculto lat loorearm. Emma," suffers. Mr. Mr. Chur ve
-------- n = 19 ----------
Emma Woodhouse's odiously, the help; and regularly, she was in and triving
Highbury, and they were both is before family; but a points an afternal seem to
be so grief. I could not last sation the particulty. "We co
-------- n = 100 ----------
Emma Woodhouse?-what can it possible, the hour, before we condemn her taste was
not obliged Miss Bates may very like therefore, earnest. She talked of, and with
the fortitude of her little while ago, every letter h
```

7 (Extra credit) Problem 5

Try running your program with a different text!

You can - find any movie script from https://imsdb.com/ or a book by Shakespeare, Hemingway, Beowulf, O.Henry, A.A. Milne, etc from https://www.gutenberg.org/ - ctrl + a to select all text on the page - copy paste into a new .txt file. let's call it puppycat.txt. - put puppycat.txt in the same folder as emma-full.txt. - run the following code to read the file into variable `s`.

```
with open('puppycat.txt', 'r') as f:
    s = f.read()
```

- run `markov_text` on `s` with appropriate parameters.

Show your output here. Which parameters did you pick and why? Do you see any difference from when you ran the program with Emma? How so?

[11]:
```
# run your new
with open('puppycat.txt', 'r') as f:
    s = f.read()
```

[13]:
```
print('-------- n = 1 ----------')
print(markov_text(s, n = 2, length = 200, seed = "Joker"))
print('-------- n = 5 ----------')
print(markov_text(s, n = 2, length = 200, seed = "Joker"))
print('-------- n = 10 ----------')
print(markov_text(s, n = 2, length = 200, seed = "Joker"))
```

```
-------- n = 1 ----------
Jokereat. He off to kill.            Jok.
                  whistim.


         nowelen
                          Joks ing up gerkin is hiery seed
                                                Jok.

-------- n = 5 ----------
Joker mass ither, TV)
                         7.


   He's som, Sopeandes.

        JOKER
          Art, LOW CUTS NE Stre he mome. I nod smallown the do gund ush. TV)

-------- n = 10 ----------
Jokerying out oted but ore-yeappendings bad?
```

```
  ther


                                    (shad.

He over forks the som, "bre he's vice lin whooked to runds to tan
                OF RON ELED And cheary
```

I choose the script of Joker as the parameter. One of the big differences is the format, unlike Emma, the script of Joker displays separately, which causes the incoherence of generated text.

[ ]: