Metadata

```
Course: DS 5001
Module: 03: Homework KEY
Topics: Inferring and Interpreting Language Models
Author: R.C. Alvarado
```

Instructions

Use the the following libraries and source text to answer the questions in this assessment.

- `pg42324.txt`
- `textimporter.py`
- `langmod.py`

Follow this pattern:

- Create a new notebook for your work.
- Parse the *Frankenstein* text to generate TOKENS and VOCAB tables.
- Create a list of sentences from the TOKENS table and a list of terms from the VOCAB table.
- Pass the two lists to an `langmod.NgramCounter` object to generate ngram type tables and models, going up to the trigram level.
- Write the code to answer the following questions:
  1. List six words that precede the word "monster," excluding stop words (and sentence boundary markers). Stop words include 'a', 'an', 'the', 'this', 'that', etc. Hint: use the `df.query()` method.
  2. List the following sentences in ascending order of bigram perpexity according to the language model generated from the text:
     `The monster is on the ice.`
     `Flowers are happy things.`
     `I have never seen the aurora borealis.`
     `He never knew the love of a family.`
  3. Using the bigram model represented as a matrix, explore the relationship between bigram pairs using the following lists. Hint: use the `.unstack()` method on the feature `n` and then use `.loc[]` to select the first list from the index, and the second list from the columns.
     A. `['he','she']` to select the indices.
     B. `['said','heard']` to select the columns.
  4. Generate 20 sentences using the `.generate_text()` method from the `langmod.NgramLanguageModel` class.
  5. Compute the redundancy R for each of the n-gram models using the MLE of the joint probability of each ngram type. In other words, for each model, just use the

`.mle` feature as p in computing $H = \sum p(ng) \log_2(1/p(ng))$. Does R increase, decrease, or remain the same as the choice of n-gram increases in length? Hint: Remember that $R = 1 - \frac{H}{H_{max}}$, where H is the actual entropy of the model and H_{max} is its maximum entropy.

Hints:

- You may use the libraries or cut-and-paste code from the relevant notebooks.
- Use the `M03_LanguageModels.ipynb` to see how the objects from the libraries are used.
- The story begins with the Preface.
- Even though they are not called "chapters," treat the Preface and Letters as chapters.
- Don't worry about OOV words or creating and `<UNK>` term in your vocabulary.
- You don't have to use the "START OF PROJECT GUTENBERG ...", etc., to clip the text. Find the lines where you think the text actually begins and ends.

Solution

Config

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
data_home = "./"
local_lib = "./"
src_file_path = f'{data_home}/pg42324.txt'
```

In [3]:
```
import sys
sys.path.append(local_lib)
```

In [4]:
```
from textimporter import TextImporter
from langmod import NgramCounter, NgramLanguageModel
```

Import Data

In [5]:
```
ohco_pats = [
    ('chap', r"^(?:PREFACE|CHAPTER|LETTER)\s", 'm')
]
clip_pats = [
    r"^M\. W\. S\.\s*$",
    r"^THE END\.\s*$"
]
```

In [6]:
```
franky = TextImporter(src_file_path, ohco_pats=ohco_pats, clip_pats=clip_pats)
```

In [7]:
```
franky.import_source().parse_tokens().extract_vocab();
```

```
Importing  .//pg42324.txt
Clipping text
Parsing OHCO level 0 chap_id by milestone ^(?:PREFACE|CHAPTER|LETTER)\s
Parsing OHCO level 1 para_num by delimitter \n\n
Parsing OHCO level 2 sent_num by delimitter [.?!;:]+
Parsing OHCO level 3 token_num by delimitter [\s',-]+
```

In [8]:

```
franky.TOKENS
```

Out[8]:

chap_id	para_num	sent_num	token_num	token_str	term_str
1	**0**	**0**	**0**	_To	to
			1	Mrs	mrs
		1	**1**	Saville	saville
			2	England	england
		2	**0**	_	
...	**...**	**...**	**...**	...	...
28	**82**	**1**	**10**	lost	lost
			11	in	in
			12	darkness	darkness
			13	and	and
			14	distance	distance

75721 rows × 2 columns

In [9]:

```
franky.VOCAB
```

Out[9]:

term_str	n	n_chars	p	s	i	h
the	4197	3	0.055427	18.041696	4.173263	0.231312
and	2976	3	0.039302	25.443884	4.669247	0.183512
i	2852	1	0.037665	26.550140	4.730648	0.178178
of	2647	2	0.034957	28.606347	4.838263	0.169133
to	2101	2	0.027747	36.040457	5.171545	0.143493
...	...	...	...	...	...	...
overweigh	1	9	0.000013	75721.000000	16.208406	0.000214
pledge	1	6	0.000013	75721.000000	16.208406	0.000214
salvation	1	9	0.000013	75721.000000	16.208406	0.000214
timorous	1	8	0.000013	75721.000000	16.208406	0.000214
thinks	1	6	0.000013	75721.000000	16.208406	0.000214

6965 rows × 6 columns

In [10]:
```
franky.OHCO
```
Out[10]:
```
['chap_id', 'para_num', 'sent_num', 'token_num']
```

In [11]:
```
sents = franky.gather_tokens(2).sent_str.to_list()
```

In [12]:
```
sents[:10]
```
Out[12]:
```
['to mrs',
 'saville england',
 '',
 'st',
 'petersburgh dec',
 '11th 17',
 'you will rejoice to hear that no disaster has accompanied the commencement o
f an enterprise which you have regarded with such evil forebodings',
 'i arrived here yesterday',
 'and my first task is to assure my dear sister of my welfare and increasing c
onfidence in the success of my undertaking',
 'i am already far north of london']
```

In [13]:
```
vocab = franky.VOCAB.index.to_list()
```

In [14]:
```
vocab[:10]
```
Out[14]:
```
['the', 'and', 'i', 'of', 'to', 'my', 'a', 'in', 'was', 'that']
```

In [15]:
```
train = NgramCounter(sents, vocab)
```

In [16]:
```
train.generate()
```

In [17]:
```
# train.LM[2].n.unstack(fill_value=0)
```

Q1

List six words that precede the word "monster," excluding stop words (and sentence boundary markers). Stop words include 'a', 'an', 'the', 'this', 'that', etc.

Hint, use the `df.query()` method.

ISSUE: If you use `text_importer.py` you get a set of 6, if you parse it yourself you get 5 of the same but a different 6.

In [18]:
```
train.LM[1].query("w1 == 'monster'")
```

Out[18]:

w0	w1	n	mle
<s>	monster	1	0.000011
a	monster	3	0.000033
abhorred	monster	1	0.000011
detestable	monster	1	0.000011
gigantic	monster	1	0.000011
hellish	monster	1	0.000011
hideous	monster	1	0.000011
miserable	monster	1	0.000011
the	monster	20	0.000220
this	monster	1	0.000011

```
abhorred
detestable
gigantic
hellish
hideous
miserable
```

Trying it by hand ...

In [19]:
```
import re
```

In [20]:
```
big_line = open(src_file_path, 'r').read()
big_line = big_line.lower().replace("\n", ' ')
big_line = re.sub(r"[\W_]+", " ", big_line)
big_line = re.sub(r"\s+", " ", big_line)
tokens = big_line.split()
```

In [21]:
```
big_line[:500]
```

Out[21]:
```
' the project gutenberg ebook of frankenstein by mary w shelley this ebook is for the use of anyone anywhere at no cost and with almost no restrictions what soever you may copy it give it away or re use it under the terms of the projec t gutenberg license included with this ebook or online at www gutenberg org ti tle frankenstein or the modern prometheus author mary w shelley release date m arch 13 2013 ebook 42324 language english start of this project gutenberg eboo k frankenstein produced by greg w'
```

In [22]:
```
bg_data = []
for i in range(len(tokens)):
    bg_data.append(tokens[i:i+2])
BG = pd.DataFrame(bg_data, columns=['w0','w1']).drop_duplicates()
```

In [23]:
```
BG.query("w1 == 'monster'").sort_values('w0')
```

Out[23]:

	w0	w1
40878	a	monster
33259	abhorred	monster
48760	cried	monster
45661	detestable	monster
72064	gigantic	monster
70652	hellish	monster
48800	hideous	monster
18370	miserable	monster
19663	the	monster
19350	this	monster

Q2

List the following sentences in ascending order of bigram perpexity according to the language model generated from the text.

```
The monster is on the ice.
Flowers are happy things.
I have never seen the aurora borealis.
He never knew the love of a family.
```

In [24]:
```
model = NgramLanguageModel(train)
model.apply_smoothing()
```

In [25]:
```
test_sents = """
The monster is on the ice.
Flowers are happy things.
I have never seen the aurora borealis.
He never knew the love of a family.
""".split('\n')[1:-1]
```

In [26]:
```
test_sents = [s.lower() for s in test_sents]
```

In [27]:
```
test = NgramCounter(test_sents, vocab)
test.generate()
```

In [28]:
```
model.predict(test)
```

In [29]:
```
model.T.S
```

Out[29]:

	sent_str	len	ng_1_ll	pp1	ng_2_ll	pp2	ng_3_ll	pp3
0	the monster is on the ice.	9	-46.649460	36.334631	-74.688657	314.897754	-213.042107	1.335934e+07
1	flowers are happy things.	7	-44.532783	82.243297	-75.997581	1854.477868	-177.397939	4.254725e+07
2	i have never seen the aurora borealis.	10	-50.323281	32.725155	-87.041808	417.080128	-230.966554	8.969869e+06
3	he never knew the love of a family.	11	-65.633527	62.538999	-115.580343	1455.504786	-232.560952	2.313915e+06

In [30]:
```
model.T.S.sort_values('pp2').sent_str
```

Out[30]:
```
0                  the monster is on the ice.
2     i have never seen the aurora borealis.
3        he never knew the love of a family.
1                   flowers are happy things.
Name: sent_str, dtype: object
```

Q3

Using the bigram model represented as a matrix, explore the relationship between bigram pairs as done in the "Explore" section of the template notebook, but use the following lists. **What might you speculate about gender and communication given the results you see?**

- `['he','she']` to select the indices.
- `['said','heard']` to select the columns.

Hint: use `.unstack()` method on the feature `n` and then use `.loc[]` to select the first list from the index, and the second list from the columns.

In [31]:
```
BGX = model.LM[1].n.unstack()
```

In [32]:
```
print(BGX.loc[['he','she'],['said','heard']])
```

```
w1   said  heard
w0
he   21.0    5.0
she   3.0    3.0
```

Speculation: Men talk more than women.

Q4

Generate a text using the `generate_text` function.

In [33]:
```
model.generate_text()
```

```
01. NOT THAT BE AN IMPEDIMENT AND TRULY I REJOICED THAT THUS I RETURNED TO OUR
UNION WE SHOULD PROCEED TO VILLA LAVENZA AND SPEND OUR FIRST DAYS.

02. THIS WAS A PORTRAIT OF A PECULIAR INTEREST TO EVERY GLOOMY IDEA THAT AROS
E.

03. THE OLD MAN.

04. I COMMIT MY THOUGHTS UNCHECKED BY REASON TO RAMBLE IN THE HEATHS OF ENGLAN
D AND AMONG THESE MOUNTAINS I SHOULD SOON SEE GENEVA.

05. AND I FEEL I SWEAR.

06. I INDEED PERCEPTIBLY GAINED ON IT I WILL SOON EXPLAIN TO ME LIKE THE TORTU
RE.

07. AND ALTHOUGH THEY WERE MY BRETHREN MY FELLOW CREATURES.

08. ONE INSCRIPTION THAT HE WAS NOT COLD.

09. CONTINUING THUS I LOVED REMAINED BEHIND.

10. I HESITATED BEFORE I COULD DISTINGUISH WAS THE JUST TRIBUTE SHE SHOULD SUF
FER AS GUILTY.

11. AND NOW AS I GAZED WITH DELIGHT.

12. I FELT THE TORMENT OF A SNAKE THAT I SURVIVE WHAT I HAVE FAILED.

13. BUT YOU WILL.

14. WHAT DID THEIR TEARS IMPLY.

15. MISERY HAD HER DWELLING IN MY OWN.

16. NAY THEN I WAS BENEVOLENT.

17. AND AS I PROCEEDED I WEIGHED THE VARIOUS IMPROVEMENTS MADE BY A SOLEMN VOW
IN MY CONVALESCENCE HAD COMMENCED AND PROCEEDED TO EXECUTE THIS DEAR REVENGE W
ILL I GIVE UP MY SEARCH THAN TO HIM AN APPETITE.

18. AS NIGHT OBSCURED THE SHAPES OF OBJECTS A THOUSAND PLANS BY WHICH HE APPEA
RED TO CONSIDER IT AS AN EASIER TASK TO PERFORM THIS SACRIFICE.

19. IN THIS LAND OF PEACE AND SOLITUDE FOR A FEW MINUTES AFTER I ENTERED IT BY
INNUMERABLE CHINKS I FOUND THAT HE TRIED TO AWAKEN IN ME.

20. A MUMMY AGAIN ENDUED WITH ANIMATION COULD NOT CONSENT TO GO INSTANTLY TO G
ENEVA WITH ALL THE EVIDENCE OF FACTS A WEIGHT OF ANGUISH THAT I HOPED THAT CHA
NGE OF FEELING.
```

Q5

Compute the redundancy R for each of the n-gram models using the MLE of the joint probability of each ngram type. In other words, for each model, just use the `.mle` feature as p in computing $H = \sum p(ng) \log_2(1/p(ng))$

Remember that $R = 1 - \frac{H}{H_{max}}$, where H is the actual entropy of the model and H_{max} is its maximum entropy.

Does R increase, decrease, or remain the same as the choice of n-gram increases in length?

In [34]:
```
V = len(vocab)
```

In [35]:
```
R = []
for i in range(3):
    N = V**(i+1)
    H = (train.LM[i]['mle'] * np.log2(1/train.LM[i]['mle'])).sum()
    Hmax = np.log2(N)
    R.append(int(round(1 - H/Hmax, 2) * 100))
```

In [36]:
```
R
```

Out[36]:
```
[33, 48, 61]
```

ANSWER: Redundancy increases.

ISSUE: If you use the just the vector length of seen values, the redundancy will decrease. We accept both answers since some students were told to use only the seen values for the length.

Notes

Q2

```
self.T.S[f'ng_{ng}_ll'] = self.T.NG[i]\
    .join(self.LM[i].log_p, on=self.widx[:ng])\
    .fillna(self.Z1[i]).fillna(self.Z2[i])\
    .groupby('sent_num').log_p.sum()

self.T.S[f'pp{ng}'] = 2**( -self.T.S[f'ng_{ng}_ll'] /
self.T.S['len'])
```

In [37]:
```
# Bigram Prediction
X = test.NG[1]\
    .join(train.LM[1].log_p, on=['w0','w1'])\
    .fillna(model.Z1[1])\
    .fillna(model.Z2[1])
```

In [38]:
```
X.groupby('sent_num').log_p.sum() #.sort_values(ascending=False).to_frame('log_
```

Out[38]:
```
sent_num
0    -74.688657
1    -75.997581
2    -87.041808
3   -115.580343
Name: log_p, dtype: float64
```

In [39]:
```
test.S.ng_2_ll
```

Out[39]:
```
0    -74.688657
1    -75.997581
2    -87.041808
3   -115.580343
Name: ng_2_ll, dtype: float64
```

In [ ]: