Info

Name: Jacqui Unciano **Date:** 1 Feb. 2024 **Assignment:** Homework 3

Follow this pattern:

Create a new notebook for your work. Parse the Frankenstein text to generate TOKENS and VOCAB tables.

In [1]:

```
import pandas as pd
import numpy as np
import configparser

config = configparser.ConfigParser()
config.read('../../../env.ini')
data_home = config['DEFAULT']['data_home']
output_dir = config['DEFAULT']['output_dir']
```

In [2]:

```
text_file = f'{data_home}/gutenberg/pg42324.txt'
```

In [3]:

```
clip_pats = [
    r"\*\*\*\s*START OF (?:THE|THIS) PROJECT",
    r"\*\*\*\s*END OF (?:THE|THIS) PROJECT"
]
chap_pat = r"(^\s*(PREFACE\.)\s+)|(^\s*(CHAPTER|LETTER)\s+X{0,3}(IX|IV|V?I{0,3})\.*
```

In [4]:

```
import sys
local_lib = config['DEFAULT']['local_lib']
sys.path.append(local_lib)
from textimporter import TextImporter
```

In [5]:

```
my_text = TextImporter(src_file=text_file, ohco_pats=[('chap', chap_pat, 'm')], cli
my_text.import_source()
my_text.parse_tokens()
my_text.extract_vocab()
```

```
Importing  /Users/jacqu/OneDrive/Documents/MSDS-at-UVA-2023/DS5001/data/gutenberg/pg
42324.txt
Clipping text
Parsing OHCO level 0 chap_id by milestone (^\s*(PREFACE\.)\s+)|(^\s*(CHAPTER|LETTER)
\s+X{0,3}(IX|IV|V?I{0,3})\.*)
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',-]+
```

```
C:\Users/jacqu/OneDrive/Documents/MSDS-at-UVA-2023/DS5001/repo/lessons/lib\textimpor
ter.py:117: UserWarning: This pattern is interpreted as a regular expression, and ha
s match groups. To actually get the groups, use str.extract.
  div_lines = df[src_col].str.contains(div_pat, regex=True, case=False) # May want t
o parametize case
```

Out[5]:

```
<textimporter.TextImporter at 0x1ab5729aa10>
```

In [6]:
```
TOKEN = my_text.TOKENS
TOKEN.head()
```

Out[6]:

				token_str	term_str
chap_id	**para_num**	**sent_num**	**token_num**		
1	**0**	**0**	**0**	_To	to
			1	Mrs	mrs
		1	**1**	Saville	saville
			2	England	england
		2	**0**	_	

In [7]:
```
my_text.VOCAB.head()
```

Out[7]:

	n	n_chars	p	s	i	h
term_str						
the	4200	3	0.055424	18.042857	4.173356	0.231302
and	2976	3	0.039272	25.463710	4.670371	0.183413
i	2854	1	0.037662	26.552207	4.730760	0.178168
of	2650	2	0.034970	28.596226	4.837753	0.169175
to	2105	2	0.027778	36.000000	5.169925	0.143609

Create a list of sentences from the TOKENS table and a list of terms from the VOCAB table.

In [8]:
```
SENTS = my_text.gather_tokens(level=2)
sents_list = SENTS.sent_num_str.tolist()
sents_list[:5]
```

Out[8]:
```
['to mrs', 'saville england', '', 'st', 'petersburgh dec']
```

In [9]:
```
term_list = my_text.VOCAB.index.tolist()
term_list[:5]
```

Out[9]:
```
['the', 'and', 'i', 'of', 'to']
```

Generate ngram type tables and models, going up to the trigram level.

In [10]:
```
def get_ngrams(TOKEN, n=2, sent_key='sent_num'):

    OHCO = TOKEN.index.names
    grouper = list(OHCO)[:OHCO.index(sent_key)+1]

    PADDED = TOKEN.groupby(grouper)\
        .apply(lambda x: '<s> ' + ' '.join(x.term_str) + ' </s>')\
        .apply(lambda x: pd.Series(x.split()))\
        .stack().to_frame('term_str')
    PADDED.index.names = grouper + ['token_num']

    NGRAMS = PADDED.groupby(grouper)\
        .apply(lambda x: pd.concat([x.shift(0-i) for i in range(n)], axis=1)).reset
    NGRAMS.index = PADDED.index
    NGRAMS.columns = [f'w{j}' for j in range(n)]

    return NGRAMS
```

In [11]:

```
ngrams = 3
widx = [f"w{i}" for i in range(ngrams)]
```

In [12]:

```
def ngrams_to_models(ngrams):
    global widx
    n = len(ngrams.columns)
    model = [None for i in range(n)]
    for i in range(n):
        if i == 0:
            model[i] = ngrams.value_counts('w0').to_frame('n')
            model[i]['p'] = model[i].n / model[i].n.sum()
            model[i]['i'] = np.log2(1/model[i].p)
        else:
            model[i] = ngrams.value_counts(widx[:i+1]).to_frame('n')
            model[i]['cp'] = model[i].n / model[i-1].n
            model[i]['i'] = np.log2(1/model[i].cp)
    return model
```

In [13]:

```
NG3 = get_ngrams(TOKEN, n=3)
NG3.loc[(1,0,0)]
```

Out[13]:

	w0	w1	w2
token_num			
0	<s>	to	mrs
1	to	mrs	</s>
2	mrs	</s>	None
3	</s>	None	None

In [14]:

```
M3 = ngrams_to_models(NG3)
uni = M3[0].sort_values('n')
bi = M3[1].sort_values('n')
tri = M3[2].sort_values('n')
```

In [15]:

```
uni
```

Out[15]:

	n	p	i
w0			
irresolution	1	0.000012	16.387580
termination	1	0.000012	16.387580
brute	1	0.000012	16.387580
thonon	1	0.000012	16.387580
bruised	1	0.000012	16.387580
...	...	...	...
i	2854	0.033289	4.908810
and	2976	0.034712	4.848421
the	4200	0.048989	4.351406
</s>	5153	0.060105	4.056383
<s>	5153	0.060105	4.056383

6979 rows × 3 columns

In [16]:

```
bi
```

Out[16]:

		n	cp	i
w0	**w1**			
which	**shot**	1	0.001792	9.124121
hideous	**that**	1	0.090909	3.459432
	than	1	0.090909	3.459432
	narration	1	0.090909	3.459432
	monster	1	0.090909	3.459432
...	**...**	...	...	...
<s>	**the**	366	0.071027	3.815497
	and	418	0.081118	3.623838
	but	457	0.088686	3.495147
of	**the**	530	0.200000	2.321928
<s>	**i**	820	0.159131	2.651717

40841 rows × 3 columns

In [17]:

```
tri
```

Out[17]:

			n	cp	i
w0	**w1**	**w2**			
the	**most**	**learned**	1	0.017857	5.807355
a	**vast**	**and**	1	0.333333	1.584963
		portion	1	0.333333	1.584963
		sheet	1	0.333333	1.584963
	vehicle	**</s>**	1	1.000000	0.000000
...	**...**	**...**	...	...	...
<s>	**i**	**had**	50	0.060976	4.035624
	and	**i**	62	0.148325	2.753163
	i	**was**	63	0.076829	3.702200
	it	**was**	67	0.515385	0.956279
	but	**i**	101	0.221007	2.177839

64838 rows × 3 columns

Write the code to answer the following questions:

Question 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.

In [18]:

```
bi.query('w1 == "monster"').sort_values('n', ascending=False)
```

Out[18]:

		n	cp	i
w0	**w1**			
the	**monster**	20	0.004762	7.714246
a	**monster**	3	0.002160	8.854868
hideous	**monster**	1	0.090909	3.459432
hellish	**monster**	1	0.142857	2.807355
detestable	**monster**	1	0.500000	1.000000
gigantic	**monster**	1	0.166667	2.584963
this	**monster**	1	0.002488	8.651052
miserable	**monster**	1	0.015385	6.022368
abhorred	**monster**	1	0.083333	3.584963
<s>	**monster**	1	0.000194	12.331197

Six words that precede the word, 'monster', are:

1. miserable
2. abhorred
3. gigantic
4. hideous
5. hellish
6. detestable

Question 2

List the following sentences in ascending order of bigram perplexity 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 [19]:

```
ngrams = 2
widx = [f"w{i}" for i in range(ngrams)]
```

In [20]:

```
def sentence_to_token(sent_list, file=True):

    # Convert list of sentences to dataframe
    if file:
        S = pd.read_csv("test_sentences.txt", header=None, names=['sent_str'])
    else:
        S = pd.DataFrame(sent_list, columns=['sent_str'])
    S.index.name = 'sent_num'

    # Convert dataframe of sentences to TOKEN with normalized terms
    K = S.sent_str.apply(lambda x: pd.Series(x.split())).stack().to_frame('token_st
    K['term_str'] = K.token_str.str.replace(r"[\W_]+", "", regex=True).str.lower()
    K.index.names = ['sent_num', 'token_num']

    return S, K
```

In [21]:

```
test_sentences = """
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]
test_sentences
```

Out[21]:

```
['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 [22]:

```
TEST_SENTS, TEST_TOKENS = sentence_to_token(test_sentences, file=False)
```

In [23]:

```
TEST_NGRAMS = get_ngrams(TEST_TOKENS)
TEST_NGRAMS.loc[0]
```

Out[23]:

	w0	w1
token_num		
0	<s>	the
1	the	monster
2	monster	is
3	is	on
4	on	the
5	the	ice
6	ice	</s>
7	</s>	None

In [24]:

```
def test_model(model, ngrams, sents):

    global widx

    assert len(model) == len(ngrams.columns)
    n = len(model)
    ohco = ngrams.index.names

    R = []
    for i in range(n):
        T = ngrams.merge(model[i], on=widx[:i+1], how='left')
        T.index = ngrams.index
        T = T.reset_index().set_index(ohco + widx).i #.to_frame(f"i{i}")

        # This how we handle unseen combos
        T[T.isna()] = T.max()
        R.append(T.to_frame(f"i{i}"))

    return pd.concat(R, axis=1)
```

In [25]:
```
NG2 = get_ngrams(TOKEN)
M = ngrams_to_models(NG2)
```

In [26]:
```
R = test_model(M, TEST_NGRAMS, TEST_SENTS)
```

In [27]:
```
def compute_perplexity(results, test_sents, n=2):
    for i in range(n):
        test_sents[f"pp{i}"] = np.exp2(results.groupby('sent_num')[f"i{i}"].mean())
    return test_sents
```

In [28]:
```
PP = compute_perplexity(R, TEST_SENTS)
PP.sort_values("pp1", ascending=True)
```

Out[28]:

	sent_str	pp0	pp1
sent_num			
0	The monster is on the ice	116.172431	80.655838
3	He never knew the love of a family	170.898523	136.954650
2	I have never seen the aurora borealis	341.056962	138.788045
1	Flowers are happy things	587.334984	534.302604

Question 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.

1. ['he','she'] to select the indices.
2. ['said','heard'] to select the columns.

In [46]:
```
q3 = M[1].sort_values('n')
q3 = q3.cp.unstack(fill_value=0)
```

In [47]:
```
q3.loc[(['he', 'she'], ['said', 'heard'])]
```

Out[47]:

w1	said	heard
w0		
he	0.034483	0.008210
she	0.011765	0.011765

Question 4

Generate 20 sentences using the .generate_text() method from the langmod.NgramLanguageModel class.

In [72]:
```
local_lib = config['DEFAULT']['local_lib']
sys.path.append(local_lib)
from langmod_class import NgramCounter
from langmod_class import NgramLanguageModel
```

In [73]:
```
bigram = NgramCounter(sents=sents_list, vocab=term_list, n=2)
bigram.generate()
```

In [80]:
```
bigram.S
```

Out[80]:

	sent_str	len
0	to mrs	4
1	saville england	4
2		2
3	st	3
4	petersburgh dec	4
...	...	...
5148	line 2863	4
5149	i do no not fear to die to i do now not fear t...	19
5150	fulfil the wishes of you parents to your parents	11
5151	end of the project gutenberg ebook of frankens...	13
5152	shelley	3

5153 rows × 2 columns

In [83]:
```
bimod = NgramLanguageModel(bigram)
```

In [84]:

```
bimod.apply_smoothing()
```

In [86]:

```
bimod.generate_text()
```

```
01. AS IF HER VOICE WOULD FOR WHEN I FEEL YOUR EYES AND HE SPOKE AND TRIUMPH AND SOM
ETIMES I HAD NO TRACE ITS CONCLUSION BY THE SUMMIT OF MY FAMILY IN STONE THAT OF MY
WORK KNELT AT LENGTH HE THEN THOUGHT IT NECESSARY KNOWLEDGE OF MY POWER SHE COULD DR
INK IN HORROR OF WHICH WAS MELANCHOLY SUBJECTS MY YOUTH BUT SAID THAT DAY MY DUTIES
ON EXAMINING AND EXCITED BY EVERY THING WAS DECEIVED ALAS BUT I HAVE BEEN PENSIVE I
AM CONTENT IF IMPATIENT TO UNDERSTAND THEIR LABOURS AND DELIGHT FRANKENSTEIN HAD CAS
T MY FRIENDS WITH ITS INTELLECTUAL EYE OF SWITZERLAND APPEARED IN BONDS OF RETURNING
ALAS I HAD FIRST LITTLE FOOD AND TALKS AS IF I SHOULD BE THE ANCIENT STUDIES YOU WEL
L IF THEREFORE IN A RESISTLESS AND ALLOWED TO IMITATE HER PROMISED THAT THAT POSSESS
ED BY THE DREADFUL MEANS GAINED ADDITIONAL LOVE WILLIAM SAYING THIS SHORT AND IMMACU
LATE BEINGS WHO COMMITTED AT THAT WHAT I HOPED TO PROGNOSTICATE PEACE I WAS TO BESTO
W ANIMATION WHEN I HAVE A LEAGUE IN LISTENING TO CLAIM THE BEAUTY THE HOVEL TO ME TH
EREFORE TO RECORD HAVE UNKNOWN HOW SHE DID NOT ESCAPE YET I CONFESS TO SECURE HIM AN
D PERISH IN PEACE.
```

Question 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 - (H/H-max), where H is the actual entropy of the model and H-max is its maximum entropy.

In [119…

```
NG3 = get_ngrams(TOKEN, n=3)
M3 = ngrams_to_models(NG3)
```

In [123…

```
p = M3[0].p
Hmax = np.log2(len(M[0].index))
uniH = sum(p*np.log2(1/p))
1-(uniH/Hmax)
```

Out[123…

```
0.3085090204282228
```

In [124…

```
p = M3[1].n /  M3[1].n.sum()
Hmax = np.log2(len(M[0].index)**2)
uniH = sum(p*np.log2(1/p))
1-(uniH/Hmax)
```

Out[124…

```
0.4466133279213331
```

In [125…

```
p = M3[2].n /  M3[2].n.sum()
Hmax = np.log2(len(M[0].index)**3)
uniH = sum(p*np.log2(1/p))
1-(uniH/Hmax)
```

Out[125…

```
0.6310755519475555
```

The redundancy increases as the choice of n-gram increases in length.