M03 Homework

Michael Vaden, mtv2eva

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

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.

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

Get TOKENS and VOCAB

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

```
In [ ]: text_file = f"{data_home}/pg42324.txt"
```

```
In [ ]: OHCO = ['chap_num', 'para_num', 'sent_num', 'token_num']

        LINES = pd.DataFrame(open(text_file, 'r', encoding='utf-8-sig').readlines(), columns=['line_str'])
        LINES.index.name = 'line_num'
        LINES.line_str = LINES.line_str.str.replace(r'\n+', ' ', regex=True).str.strip()

        title = LINES.loc[0].line_str.replace('The Project Gutenberg EBook of ', '')
        print(title)
```

```
Frankenstein, by Mary W. Shelley
```

```
In [ ]: clip_pats = [
            r"\*\*\*\s*START OF (?:THE|THIS) PROJECT",
            r"\*\*\*\s*END OF (?:THE|THIS) PROJECT"
        ]

        pat_a = LINES.line_str.str.match(clip_pats[0])
        pat_b = LINES.line_str.str.match(clip_pats[1])

        line_a = LINES.loc[pat_a].index[0] + 1
        line_b = LINES.loc[pat_b].index[0] - 1
        print(line_a, line_b)

        LINES = LINES.loc[line_a : line_b]
```

```
19 7671
```

```
In [ ]: chap_pat = r'^(?:LETTER|CHAPTER)\b'

        chap_lines = LINES.line_str.str.match(chap_pat, case=True) # Returns a truth vector
```

```
In [ ]: LINES.loc[chap_lines]
```

Out[ ]:

	line_str
line_num	
343	LETTER I.
467	LETTER II.
594	LETTER III.
636	LETTER IV.
918	CHAPTER I.
1085	CHAPTER II.
1299	CHAPTER III.
1555	CHAPTER IV.
1789	CHAPTER V.
2028	CHAPTER VI.
2292	CHAPTER VII.
2655	CHAPTER VIII.
2958	CHAPTER IX.
3165	CHAPTER X.
3385	CHAPTER XI.
3651	CHAPTER XII.
3856	CHAPTER XIII.
4061	CHAPTER XIV.
4245	CHAPTER XV.
4552	CHAPTER XVI.
4861	CHAPTER XVII.
5046	CHAPTER XVIII.
5324	CHAPTER XIX.
5572	CHAPTER XX.
5905	CHAPTER XXI.
6274	CHAPTER XXII.
6615	CHAPTER XXIII.
6867	CHAPTER XXIV.

In [ ]:
```
LINES.loc[chap_lines, 'chap_num'] = [i+1 for i in range(LINES.loc[chap_lines].shape[0])]
```

In [ ]:
```
LINES.chap_num = LINES.chap_num.ffill()

LINES = LINES.dropna(subset=['chap_num']) # Remove everything before Chapter 1

LINES = LINES.loc[~chap_lines] # Remove chapter heading lines; their work is done
LINES.chap_num = LINES.chap_num.astype('int') # Convert chap_num from float to int
```

In [ ]:
```
CHAPS = LINES.groupby(OHCO[:1])\
    .line_str.apply(lambda x: '\n'.join(x))\
    .to_frame('chap_str')

CHAPS['chap_str'] = CHAPS.chap_str.str.strip()
```

In [ ]:
```
para_pat = r'\n\n+'
# CHAPS['chap_str'].str.split(para_pat, expand=True).head()
PARAS = CHAPS['chap_str'].str.split(para_pat, expand=True).stack()\
    .to_frame('para_str').sort_index()
PARAS.index.names = OHCO[:2]
```

In [ ]:
```
PARAS['para_str'] = PARAS['para_str'].str.replace(r'\n', ' ', regex=True)
PARAS['para_str'] = PARAS['para_str'].str.strip()
PARAS = PARAS[~PARAS['para_str'].str.match(r'^\s*$')]
```

In [ ]:
```
sent_pat = r'[.?!;:]+'
SENTS = PARAS['para_str'].str.split(sent_pat, expand=True).stack()\
    .to_frame('sent_str')
SENTS.index.names = OHCO[:3]
SENTS = SENTS[~SENTS['sent_str'].str.match(r'^\s*$')] # Remove empty paragraphs
SENTS.sent_str = SENTS.sent_str.str.strip() # CRUCIAL TO REMOVE BLANK TOKENS
```

In [ ]:
```
token_pat = r"[\s',-]+"
TOKENS = SENTS['sent_str'].str.split(token_pat, expand=True).stack()\
    .to_frame('token_str')
TOKENS.index.names = OHCO[:4]

TOKENS['term_str'] = TOKENS.token_str.replace(r'[\W_]+', '', regex=True).str.lower()
TOKENS
```

Out[ ]:

chap_num	para_num	sent_num	token_num	token_str	term_str
1	0	0	0	_To	to
			1	Mrs	mrs
		1	0	Saville	saville
			1	England	england
		2	0	_	
...	...	...	...	...	...
28	86	0	7	Frankenstein	frankenstein
			8	by	by
			9	Mary	mary
			10	W	w
		1	0	Shelley	shelley

75941 rows × 2 columns

In [ ]:

```
TOKENS['term_str'] = TOKENS.token_str.replace(r'[\W_]+', '', regex=True).str.lower()
VOCAB = TOKENS.term_str.value_counts().to_frame('n').reset_index().rename(columns={'index':'term_str'})
VOCAB.index.name = 'term_id'
VOCAB
```

Out[ ]:

term_id	term_str	n
0	the	4200
1	and	2976
2	i	2854
3	of	2650
4	to	2105
...	...	...
6973	indecent	1
6974	pretended	1
6975	warmly	1
6976	hesitate	1
6977	shelley	1

6978 rows × 2 columns

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

In [ ]:

```
def token_to_padded(token, grouper=['sent_num'], term_str='term_str'):
    ohco = token.index.names # We preserve these since they get lost in the shuffle
    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 = ohco
    return padded
```

In [ ]:

```
PADDED = token_to_padded(TOKENS, grouper=OHCO[:3], term_str='term_str')
```

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

In [ ]:

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

In [ ]:

```
def padded_to_ngrams(padded, grouper=['sent_num'], n=2):

    ohco = padded.index.names
    ngrams = padded.groupby(grouper)\
        .apply(lambda x: pd.concat([x.shift(0-i) for i in range(n)], axis=1))\
        .reset_index(drop=True)
    ngrams.index = padded.index
    ngrams.columns = widx

    # ngrams = pd.concat([padded.shift(0-i) for i in range(n)], axis=1)
    # ngrams.index.name = 'ngram_num'
    # ngrams.columns = widx
    # ngrams = ngrams.fillna('<EOF>')

    return ngrams
```

In [ ]:

```
ngrams = 1
widx = [f"w{i}" for i in range(ngrams)]

NGRAMS1 = padded_to_ngrams(PADDED, OHCO[:3], ngrams)
```

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

        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)
                model[i] = model[i].sort_index()
            return model
        M = ngrams_to_models(NGRAMS)
```

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

        NGRAMS3 = padded_to_ngrams(PADDED, OHCO[:2], ngrams)

        M3 = ngrams_to_models(NGRAMS3)
```

```
In [ ]: ngrams = 1
        widx = [f"w{i}" for i in range(ngrams)]

        NGRAMS1 = padded_to_ngrams(PADDED, OHCO[:3], ngrams)

        M1 = ngrams_to_models(NGRAMS1)
```

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.

```
In [ ]: stop_words = ['a', 'an', 'the', 'this', 'that', '<s>']

        NGRAMS.query("w1 == 'monster' & w0 not in @stop_words")
```

Out[ ]:

chap_num	para_num	sent_num		w0	w1
9	3	17	25	miserable	monster
14	8	0	1	abhorred	monster
19	25	4	23	detestable	monster
20	28	0	1	hideous	monster
28	4	9	5	hellish	monster
	17	6	2	gigantic	monster

```
In [ ]: print(list(NGRAMS.query("w1 == 'monster' & w0 not in @stop_words")['w0']))

['miserable', 'abhorred', 'detestable', 'hideous', 'hellish', 'gigantic']
```

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 [ ]: TEST_SENTS = pd.DataFrame({'sent_str': ['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.']})

        TEST_SENTS.index.name = 'sent_num'
        TEST_SENTS
```

Out[ ]:

sent_num	sent_str
0	The monster is on the ice.
1	Flowers are happy things.
2	I have never seen the aurora borealis.
3	He never knew the love of a family.

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

#TEST_TOKENS.head()
```

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

In [ ]:
```
TEST_PADDED = token_to_padded(TEST_TOKENS)
```

In [ ]:
```
TEST_NGRAMS = padded_to_ngrams(TEST_PADDED, 'sent_num', ngrams)
```

In [ ]:
```
TEST_NGRAMS = TEST_NGRAMS.reset_index().rename({'level_1':'token_num'}, axis=1).groupby(['sent_num', 'token_num']).sum()
```

In [ ]:
```
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(M[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)

R = test_model(M, TEST_NGRAMS, TEST_SENTS)
```

In [ ]:
```
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 [ ]:
```
PP = compute_perplexity(R, TEST_SENTS)
PP
```

Out[ ]:

	sent_str	pp0	pp1
sent_num			
0	The monster is on the ice.	116.056265	80.733835
1	Flowers are happy things.	586.369721	534.302604
2	I have never seen the aurora borealis.	340.789117	138.907338
3	He never knew the love of a family.	170.793655	137.060591

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.

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

In [ ]:
```
matrix_df = M[1].unstack()

matrix_df.loc[['he', 'she']].loc[:, (['n', 'i'], ['said', 'heard'])]
```

Out[ ]:

	n		i	
w1	**said**	**heard**	**said**	**heard**
w0				
he	21.0	5.0	4.857981	6.928370
she	3.0	3.0	6.409391	6.409391

4.Generate 20 sentences using the generate_text() function. Display the results.

In [ ]:
```
def generate_text(M, n=250):

    if len(M) < 3:
        raise ValueError("Must have trigram model generated.")

    # Start list of words
    first_word = M[1].loc['<s>'].sample(weights='cp').index[0]

    words = ['<s>', first_word]
```

```python
    for i in range(n):

        bg = tuple(words[-2:])

        # Try trigram model
        try:
            next_word = M[2].loc[bg].sample(weights='cp').index[0]

        # If not found in model, back off ...
        except KeyError as e1:
            try:
                # Get the last word in the bigram
                ug = bg[1]
                next_word = M[1].loc[ug].sample(weights='cp').index[0]

            except KeyError as e2:
                next_word = M[0].sample(weights='p').index[0]

        words.append(next_word)


    text = ' '.join(words[2:])
    print('\n\n'.join([str(i+1) + ' ' + line.replace('<s>','')\
        .strip().upper() for i, line in enumerate(text.split('</s>'))]))
```

In [ ]:
```python
generate_text(M3, n = 270)
```

```
1 THE KNOWLEDGE WHICH I ALONE POSSESSED WAS THE PERIOD FIXED FOR THE ENJOYMENT OF PLEASURE

2

3 YOU ARE WELL ACQUAINTED WITH HIM AT PRESENT EXISTING IN THE SUCCESS OF MY FATHER IS IN DEATH

4

5 THE SOFT AIR JUST RUFFLED THE WATER

6 I SAW MY FRIENDS MY WIFE AND MY HEART WHICH WAS TO DISCOVER ANY CLUE BY WHICH I REGARD MYSELF

7 I BELIEVED IN HER GUILT

8 THIS CHILD WAS THIN AND VERY HAPPY ONLY A FEW DAYS AT LAUSANNE IN THIS JOURNEY HAD BEEN THE CAUSE

9 AT LENGTH ARRIVED

10 I TROD HEAVEN IN MY OWN MIND BEGAN TO PLAY AND TO BECOME MY FELLOW CREATURES THEN COULD I DO MY DUTY

11 SHE SOMETIMES BEGGED JUSTINE TO FORGIVE HER UNKINDNESS BUT MUCH OFTENER ACCUSED HER OF HAVING CAUSED THE BEST MEANS OF MATERIALLY ASS
ISTING THE PROGRESS OF YOUR MIND TO AN EXPRESSION OF WILDNESS AND EVEN MADNESS

12 BUT WHEN DANIEL NUGENT WAS CALLED SISTER OR AGATHA

13 THE PATRIARCHAL LIVES OF ALL EXCELLENCE AND ENDEAVOURED TO WELCOME ME

14 BESIDES SOME MONTHS IN PRISON

15 EVERY WHERE I AM LOST IN CONJECTURE AS TO CREATURES OF AN ENGLISH PHILOSOPHER THE KNOWLEDGE OF LANGUAGE

16 A FEW MISERABLE COWS AND OATMEAL FOR ITS HOSPITALITY

17 MELANCHOLY FOLLOWED BUT BY DEGREES ONE HERB FROM ANOTHER

18

19 THE WOUNDED DEER DRAGGING ITS FAINTING LIMBS TO SOME ONE A FRIEND OF MY LABOURS IN SOME DEGREE ALARMED ME

20 I RETIRED TO A HUMAN CREATURE
```

5.Compute the redundancy 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...

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

N is computed as the number of all possible combinations for each ngram. So, for the bigram model N is the number of unigrams (i.e. the vocabulary size plus the sentence boundary signs) squared, and for the trigram model the value is cubed, i.e. N = len(M[0].index)**{i+1}

In [ ]:
```python
M3[1]['p'] = M3[1].n /  M3[1].n.sum()
M3[2]['p'] = M3[2].n /  M3[2].n.sum()

M3[0]['h'] = M3[0]['p'] * M3[0]['i']
M3[1]['h'] = M3[1]['p'] * M3[1]['i']
M3[2]['h'] = M3[2]['p'] * M3[2]['i']
```

In [ ]:
```python
H1 = M3[0]['h'].sum()
H2 = M3[1]['h'].sum()
H3 = M3[2]['h'].sum()
```

In [ ]:
```python
n_terms1 = M3[0]['n'].count()
n_terms2 = M3[1]['n'].count()
n_terms3 = M3[2]['n'].count()
```

In [ ]:
```python
Hmax1 = np.log2(n_terms1)
Hmax2 = np.log2(n_terms2)
Hmax3 = np.log2(n_terms3)
```

```
R1 = 1 - (H1/Hmax1)
R2 = 1 - (H2/Hmax2)
R3 = 1 - (H3/Hmax3)
```

In [ ]:
```
print(f"Unigram Redundancy: {R1}")
print(f"Bigram Redundancy: {R2}")
print(f"Trigram Redundancy: {R3}")
```

```
Unigram Redundancy: 0.30885786469975895
Bigram Redundancy: 0.6843823899586926
Trigram Redundancy: 0.8838055825847118
```