Metadata

```
Course:   DS 5001
Module:   06 HW
Topic:    Similarity and Distance Measures
Author:   R.C. Alvarado
Date:     19 February (revised)
```

Instructions

In this week’s code exercise, you will compute and explore vector space distances between documents for a corpus of Jane Austen’s novels.

- Use the notebook from class as your guide, as well as any relevant previous notebooks.
- For source data, use the `LIB` and `CORPUS` tables you used last week for the Austem and Melville set. These are in the `/output` directory of the course repo.
- Note that you can use the functions you created last week to compute `TFIDF`; if you had problems with these, you may use functions in the homework key.
- Also, you will need to generate the `VOCAB` table from the Austen corpus; you can import your work from your last homework if you'd like.

To prepare to answer the questions below, complete the following tasks:

- Add a feature to the `LIB` table for the publication year of the book, using the data provided below.
  - Create a label for each book using a combination of the year and the book title.
  - *Scholarly side note*: This is the *publication* year in most cases. For works published posthumously, the year refers to when scholars think the work was actually completed. Note also, there is often a lag between date of completion and data of publication. We will not concern ourselves with these nuances here, but it is always helpful to understand how your data are actually produced.
- Bring into your notebook the functions you created previously to generate a `BOW` table and compute `TFIDF` values. Extend the `TFIDF` function so that it also returns the `DFIDF` value for each term in the `VOCAB`.
- Apply these functions to the corpus of Austen's works only, and using *chapters* as bags and `max` as the `TF` count method.
- Reduce the number of features in the returned `TFIDF` matrix to the **1000 most significant terms**, using `DFIDF` as your significance measure and only using terms whose maximum part-of-speech belongs to this set: `NN NNS VB VBD VBG VBN VBP VBZ JJ JJR JJS RB RBR RBS`. Note, these are all **open** categories, excluding proper nounns.

- "Collapse" `TFIDF` matrix that it contains mean `TFIDF` of each term by book. This will result in a matrix with book IDs as rows, and significant terms as columns.
- Use the reduced and collapsed `TFIDF` matrix to compute distance missures between all pairs of books, as we computed in Lab (using `pdist()`). See the table below for the measures to take.
  - As in the template, use the appropriate normed vector space for each metric.
  - You will need to create a table of book pairs (e.g. `PAIRS`).
  - You do *not* need to compute k-means clusters.
- Create hierarchical agglomerative cluster diagrams for the distance measures, using the appropriate linkage type for each distance measure. Again, see the table below for the appropriate linkage type.
  - Use the labels you created in the `LIB` in your dendograms to help interpret your results.

Once you have completed these tasks, answer the questions below.

Distance Measure and Linkage Method Combos

Distance Measure	Norming	Linkage
cityblock	None	weighted
cosine	None	ward
euclidean	L2	ward
jaccard	L0	weighted
jensenshannon	L1	weighted

Dates of Austen's Works

book_id	year	title
158	1815	Emma
946	1794	Lady Susan
1212	1790	Love And Freindship And Other Early Works
141	1814	Mansfield Park
121	1803	Northanger Abbey
105	1818	Persuasion
1342	1813	Pride and Prejudice
161	1811	Sense and Sensibility

Q1

What are the top 10 nouns by `DFIDF`, sorted in descending order? Include plural nouns, but don't include proper nouns.

Don't worry if your list does not include some terms that have the same weights as words in the list. Just take what Pandas gives you with `.head(10)` after sorting with ascending set to `False`.

Answer:

```
respect     174 NN  177.266344  0.008183
fortune     222 NN  177.261968  0.010642
marriage    246 NN  177.261968  0.013575
question    171 NN  177.258990  0.006774
ladies      240 NNS 177.258990  0.011303
behaviour   200 NN  177.240001  0.010849
farther     181 NN  177.240001  0.006979
advantage   166 NN  177.217644  0.007974
girl        254 NN  177.209470  0.012677
voice       228 NN  177.209470  0.009163
```

Q2

Grouping your `TFIDF` results by book, and taking the mean `TFIDF` of all terms per book, what is Austen's most "significant" book?

This value is computed from the `TFIDF` matrix your function returned.

Answer: Northanger Abbey

Q3

Using the dendograms you generated, which distance measure most clearly distinguishes Austen's two youthful works from her later works?

That is, which measure show the greatest separation between the first two work and the rest?

Note that the two youthful works were published before 1800.

Answer: **Jaccard weighted** shows the most distance between the early and the later works.

Q4

Do any of the distance measures produce dendrograms with works sorted in the exact order of their publication years?

Answer: **No**.

Q5

Some literary critics believe that Northanger Abbey is, among Austen's mature works, the one that most resembles her Juvenilia, i.e. her two works written as a young adult. Which distance measure dendrograms appear to corroborate this thesis? In other words, do any of them show that Northanger Abbey is closer to her juvenalia than the her other adult works?

Answer: The all show that the *Northanger Abbey* is closer to her juvenalia than her other adult works.

Code

Set Up

In [1]:
```
data_home = '../../repo/lessons/data'
data_prefix = 'austen-melville'
```

In [2]:
```
OHCO = ['book_id', 'chap_id', 'para_num', 'sent_num', 'token_num']
PARA = OHCO[:3]
CHAP = OHCO[:2]
BOOK = OHCO[:1]
```

In [3]:
```
import pandas as pd
import numpy as np
import re
from numpy.linalg import norm
from scipy.spatial.distance import pdist, squareform
```

Prepare the Data

Get `LIB`, `CORPUS`, and `VOCAB` for Jane Austen's works.

Import data from previous work

In [4]:
```
LIB_raw = pd.read_csv(f'{data_home}/output/{data_prefix}-LIB.csv').set_index('b
CORPUS_raw = pd.read_csv(f'{data_home}/output/{data_prefix}-CORPUS.csv').set_ir
```

Select Austen's works from `LIB`

In [5]:
```
LIB = LIB_raw[LIB_raw.author.str.contains("AUS")].copy().sort_index()
```

In [6]:
```
LIB
```

Out[6]:

book_id	source_file_path	author	title	chap_regex	book_len	n_chaps
105	../data/gutenberg/austen-melville-set/AUSTEN_J...	AUSTEN, JANE	PERSUASION	^Chapter\s+\d+$	83624	24
121	../data/gutenberg/austen-melville-set/AUSTEN_J...	AUSTEN, JANE	NORTHANGER ABBEY	^CHAPTER\s+\d+$	77601	31
141	../data/gutenberg/austen-melville-set/AUSTEN_J...	AUSTEN, JANE	MANSFIELD PARK	^CHAPTER\s+[IVXLCM]+$	160378	48
158	../data/gutenberg/austen-melville-set/AUSTEN_J...	AUSTEN, JANE	EMMA	^\s*CHAPTER\s+[IVXLCM]+\s*$	160926	55
161	../data/gutenberg/austen-melville-set/AUSTEN_J...	AUSTEN, JANE	SENSE AND SENSIBILITY	^CHAPTER\s+\d+$	119873	50
946	../data/gutenberg/austen-melville-set/AUSTEN_J...	AUSTEN, JANE	LADY SUSAN	^\s*[IVXLCM]+\s*$	23116	41
1212	../data/gutenberg/austen-melville-set/AUSTEN_J...	AUSTEN, JANE	LOVE AND FREINDSHIP SIC	^\s*LETTER .* to .*$	33265	24
1342	../data/gutenberg/austen-melville-set/AUSTEN_J...	AUSTEN, JANE	PRIDE AND PREJUDICE	^Chapter\s+\d+$	122126	61

Add publication dates to `LIB`

In [7]:
```
YYYY = """
book_id year    title
158     1815    Emma
946     1794    Lady Susan
1212    1790    Love And Freindship And Other Early Works
141     1814    Mansfield Park
121     1803    Northanger Abbey
105     1818    Persuasion
1342    1813    Pride and Prejudice
161     1811    Sense and Sensibility
""".split('\n')[1:-1]
```

In [8]:
```
YEARS = pd.DataFrame([line.split()[:2] for line in YYYY][1:], columns=['book_ic
YEARS.book_id = YEARS.book_id.astype('int')
YEARS = YEARS.set_index('book_id')
```

In [9]:
```
LIB['year'] = YEARS
```

In [10]:
```
LIB['label'] = LIB.apply(lambda x: f"{x.year}: {x.title}", 1)
```

In [11]:
```
LIB['label']
```

Out[11]:
```
book_id
105                    1818: PERSUASION
121              1803: NORTHANGER ABBEY
141                 1814: MANSFIELD PARK
158                          1815: EMMA
161         1811: SENSE AND SENSIBILITY
946                    1794: LADY SUSAN
1212      1790: LOVE AND FREINDSHIP SIC
1342           1813: PRIDE AND PREJUDICE
Name: label, dtype: object
```

Select Austen's works from `CORPUS`

In [12]:
```
CORPUS = CORPUS_raw.loc[LIB.index.values.tolist(), ['pos', 'token_str', 'term_s
```

In [13]:
```
CORPUS
```

Out[13]:

					pos	token_str	term_str
book_id	**chap_id**	**para_num**	**sent_num**	**token_num**			
105	**1**	**1**	**0**	**0**	NNP	Sir	sir
				1	NNP	Walter	walter
				2	NNP	Elliot,	elliot
				3	IN	of	of
				4	NNP	Kellynch	kellynch
...	**...**	**...**	**...**	**...**	...	...	...
1342	**61**	**18**	**0**	**8**	CC	and	and
				9	NNP	Prejudice,	prejudice
				10	IN	by	by
				11	NNP	Jane	jane
				12	NNP	Austen	austen

780873 rows × 3 columns

Generate Austen's `VOCAB`

In [14]:
```
VOCAB = CORPUS.term_str.value_counts().to_frame('n')
VOCAB.index.name = 'term_str'
VOCAB['max_pos'] = CORPUS.value_counts(['term_str','pos']).unstack().idxmax(1)
```

In [15]:
```
VOCAB
```

Out[15]:

term_str	n	max_pos
the	28274	DT
to	26029	TO
and	24060	CC
of	22927	IN
a	14301	DT
...	...	...
contagion	1	NN
purposed	1	VBN
stanwix	1	NNP
principled	1	VBN
pollution	1	NN

14745 rows × 2 columns

Vectorize the Data

Generate a `BOW` and computer `TFIDF` and derived quantities.

In [16]:
```
tf_method = 'max'
bag = CHAP
vocab_filter = 'dfidf'
n_terms = 1000
# pos_list = "CC CD DT EX FW IN MD PDT POS PRP PRP$ RP SYM TO UH WDT WP WP$ WRB
pos_list = "NN NNS VB VBD VBG VBN VBP VBZ JJ JJR JJS RB RBR RBS".split() # Oper
```

Define functions

Use the function you created previously.

In [17]:
```
def create_bow(CORPUS, bag, item_type='term_str'):
    BOW = CORPUS.groupby(bag+[item_type])[item_type].count().to_frame('n')
    return BOW
```

In [18]:
```
def get_tfidf(BOW, tf_method='max', df_method='standard', item_type='term_str')

    DTCM = BOW.n.unstack() # Create Doc-Term Count Matrix

    if tf_method == 'sum':
        TF = (DTCM.T / DTCM.T.sum()).T
    elif tf_method == 'max':
        TF = (DTCM.T / DTCM.T.max()).T
    elif tf_method == 'log':
        TF = (np.log2(DTCM.T + 1)).T
    elif tf_method == 'raw':
```

```
        TF = DTCM
    elif tf_method == 'bool':
        TF = DTCM.astype('bool').astype('int')
    else:
        raise ValueError(f"TF method {tf_method} not found.")

    DF = DTCM.count() # Assumes NULLs
    N_docs = len(DTCM)

    if df_method == 'standard':
        IDF = np.log2(N_docs/DF) # This what the students were asked to use
    elif df_method == 'textbook':
        IDF = np.log2(N_docs/(DF + 1))
    elif df_method == 'sklearn':
        IDF = np.log2(N_docs/DF) + 1
    elif df_method == 'sklearn_smooth':
        IDF = np.log2((N_docs + 1)/(DF + 1)) + 1
    else:
        raise ValueError(f"DF method {df_method} not found.")

    TFIDF = TF * IDF

    DFIDF = DF * IDF

    TFIDF = TFIDF.fillna(0)

    return TFIDF, DFIDF
```

Get `BOW` by chapter with `max`

In [19]:
```
bag
```

Out[19]:
```
['book_id', 'chap_id']
```

In [20]:
```
BOW = create_bow(CORPUS, bag)
```

In [21]:
```
tf_method
```

Out[21]:
```
'max'
```

In [22]:
```
TFIDF, DFIDF = get_tfidf(BOW, tf_method)
```

In [23]:
```
TFIDF[VOCAB.sort_values('n', ascending=False).head(200).sample(10).index].sampl
```

Out[23]:

	term_str	much	soon	every	love	said	elinor	till	
book_id	chap_id								
121	16	0.007653	0.004519	0.006778	0.008244	0.029756	0.000000	0.013246	0.0
161	8	0.006353	0.011254	0.011254	0.041066	0.038427	0.316839	0.000000	0.0
1342	18	0.007867	0.005575	0.004646	0.000000	0.028551	0.000000	0.010894	0.0
105	7	0.005612	0.015622	0.004261	0.015546	0.020782	0.000000	0.012489	0.0
946	24	0.005856	0.003458	0.010374	0.006309	0.025300	0.000000	0.000000	0.0
141	5	0.011338	0.006025	0.010042	0.021985	0.020572	0.000000	0.011774	0.0
1342	40	0.017268	0.015295	0.015295	0.013952	0.027976	0.000000	0.011208	0.0
	41	0.007401	0.008740	0.006555	0.007972	0.028775	0.000000	0.006404	0.0
141	27	0.008342	0.004030	0.002687	0.009804	0.021624	0.000000	0.003938	0.0
105	8	0.005863	0.004451	0.002968	0.005414	0.015199	0.000000	0.004349	0.0

Reduce `VOCAB` to `n` most significant terms

In [24]:
```
# DFIDF.sort_values(ascending=False)
```

In [25]:
```
VOCAB['dfidf'] = DFIDF
```

In [26]:
```
VOCAB['mean_tfidf'] = TFIDF.mean()
```

In [27]:
```
n_terms
```

Out[27]:
```
1000
```

In [28]:
```
vocab_filter
```

Out[28]:
```
'dfidf'
```

In [29]:
```
VIDX = VOCAB.loc[VOCAB.max_pos.isin(pos_list)]\
    .sort_values(vocab_filter, ascending=False)\
    .head(n_terms).index
```

Reduce `TFIDF` feature space

Collapse `TFIDF` by mean bag

In [30]:
```
M = TFIDF[VIDX].fillna(0).groupby('book_id').mean() # MUST FILLNA
```

In [31]:
```
M
```

Out[31]:

term_str	greatest	stay	respect	thinking	forward	fortune	assure	marriage
book_id								
105	0.003233	0.010584	0.006140	0.011340	0.009262	0.013589	0.010205	0.011468
121	0.007326	0.008810	0.004111	0.008241	0.006748	0.010054	0.008954	0.005660
141	0.010356	0.009412	0.009049	0.010821	0.009500	0.006946	0.004308	0.007412
158	0.007157	0.011392	0.011402	0.014596	0.008882	0.010334	0.014604	0.009475
161	0.009488	0.008128	0.006497	0.005347	0.006474	0.012934	0.009847	0.013288
946	0.015570	0.005638	0.010156	0.001255	0.010188	0.005375	0.005513	0.021950
1212	0.010051	0.000884	0.003126	0.003517	0.001501	0.011411	0.011138	0.016403
1342	0.005553	0.012313	0.009518	0.007312	0.006757	0.014328	0.013715	0.020465

8 rows × 1000 columns

In [32]:
```
# M2 = TFIDF.fillna(0).groupby('book_id').mean()[VIDX] # MUST FILLNA
```

In [33]:
```
# M2
```

Normalize `TFIDF` for distance measuring

In [34]:
```
L0 = M.astype('bool').astype('int') # Binary (Pseudo L)
L1 = M.apply(lambda x: x / x.sum(), 1) # Manhattan (Probabilistic)
L2 = M.apply(lambda x: x / norm(x), 1) # Euclidean
```

Generate doc pairs

In [35]:
```
PAIRS = M.T.corr().stack().to_frame('correl')
PAIRS.index.names = ['doc_a','doc_b']
PAIRS = PAIRS.query("doc_a > doc_b") # Remove identities and reverse duplicates
```

In [36]:
```
general_method = 'weighted' # single, complete, average, weighted
euclidean_method = 'ward' # ward, centroid, median
combos  = [
    (L2, 'euclidean', 'euclidean', euclidean_method),
    (M,  'cosine', 'cosine', euclidean_method),
    (M,  'cityblock', 'cityblock', general_method),
    (L0, 'jaccard', 'jaccard', general_method),
    (L1, 'jensenshannon', 'js', general_method),
]
```

In [37]:
```
for X, metric, label, _ in combos:
    PAIRS[label] = pdist(X, metric)
```

In [38]:
```
PAIRS.style.background_gradient('GnBu', high=.5)
```

Out[38]:

doc_a	doc_b	correl	euclidean	cosine	cityblock	jaccard	js
121	105	0.208149	0.582742	0.169794	3.548524	0.009009	0.231993
141	105	0.320335	0.500855	0.125428	2.949468	0.006000	0.192324
	121	0.239239	0.519355	0.134865	3.230207	0.010000	0.199676
158	105	0.299327	0.578299	0.167215	3.701946	0.006000	0.227238
	121	0.175196	0.843346	0.355616	5.352625	0.126000	0.369170
	141	0.261761	0.852840	0.363668	5.466981	0.108216	0.377576
161	105	0.130100	0.558419	0.155916	3.608176	0.007000	0.218743
	121	0.215281	0.560630	0.157153	3.263632	0.007000	0.215699
	141	0.047429	0.593963	0.176396	3.653082	0.011000	0.227628
	158	0.133219	0.579140	0.167701	3.535098	0.007000	0.217987
946	105	-0.058299	0.818947	0.335337	5.171677	0.125125	0.360517
	121	0.061797	0.856388	0.366700	5.390646	0.109218	0.374985
	141	0.077569	0.557344	0.155316	3.514433	0.008000	0.215576
	158	0.042805	0.521184	0.135816	2.910368	0.004004	0.184805
	161	0.094979	0.591501	0.174937	3.571928	0.002000	0.218929
1212	105	0.056996	0.786469	0.309267	4.919311	0.122000	0.344624
	121	0.089020	0.861725	0.371285	5.458439	0.108000	0.375517
	141	0.013091	0.512459	0.131307	3.294148	0.003000	0.196656
	158	0.053664	0.576470	0.166159	3.761047	0.006000	0.220952
	161	0.221230	0.806448	0.325179	5.361057	0.126000	0.347914
	946	0.147323	0.853668	0.364375	5.829864	0.108216	0.375103
1342	105	0.168656	0.574369	0.164950	3.607296	0.007000	0.217466
	121	0.258590	0.786825	0.309547	5.139736	0.122000	0.346270
	141	0.266045	0.788674	0.311003	5.308148	0.108000	0.351501
	158	0.117957	0.516489	0.133381	3.201975	0.003000	0.195426
	161	0.285853	0.917422	0.420832	5.569192	0.189990	0.421410
	946	0.162648	0.757459	0.286872	4.998955	0.119238	0.334455
	1212	0.188111	0.799262	0.319410	5.397732	0.109000	0.352887

In [39]:

```
PAIRS.corr().style.background_gradient(cmap='GnBu', high=.5)
```

Out[39]:

	correl	euclidean	cosine	cityblock	jaccard	js
correl	1.000000	0.093538	0.100085	0.105621	0.147002	0.126429
euclidean	0.093538	1.000000	0.998540	0.985599	0.962054	0.996756
cosine	0.100085	0.998540	1.000000	0.980221	0.963509	0.996019
cityblock	0.105621	0.985599	0.980221	1.000000	0.943970	0.985831
jaccard	0.147002	0.962054	0.963509	0.943970	1.000000	0.973634
js	0.126429	0.996756	0.996019	0.985831	0.973634	1.000000

Visualize

Define function

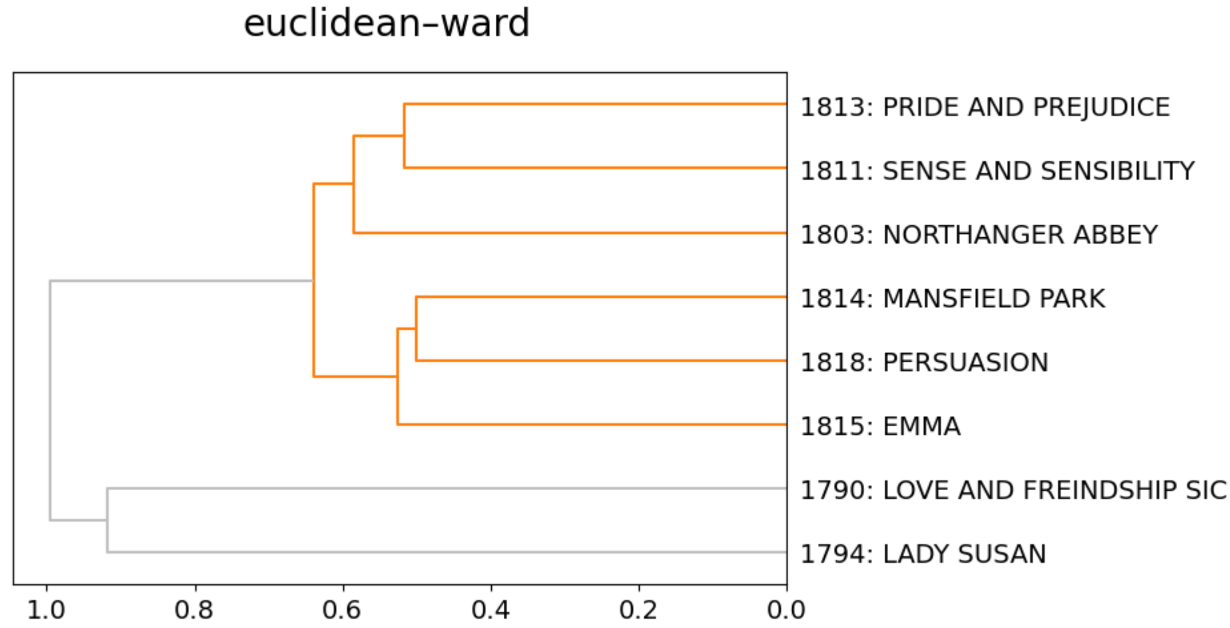
In [40]:
```
import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt
```

In [41]:
```
def draw_hca(sims, linkage_method='complete', figsize=(7.5, 5)):
    global LIB

    tree = sch.linkage(sims, method=linkage_method)
    color_thresh = pd.DataFrame(tree)[2].mean()

    labels = LIB.label.values

    plt.figure()
    fig, axes = plt.subplots(figsize=figsize)
    dendrogram = sch.dendrogram(tree,
                                labels=labels,
                                orientation="left",
                                count_sort=True,
                                distance_sort=True,
                                above_threshold_color='.75',
                                color_threshold=color_thresh,
                               )
    plt.tick_params(axis='both', which='major', labelsize=14)
    fig.suptitle(f"{label}-{linkage_method}", fontsize=20)

#     return fig
```

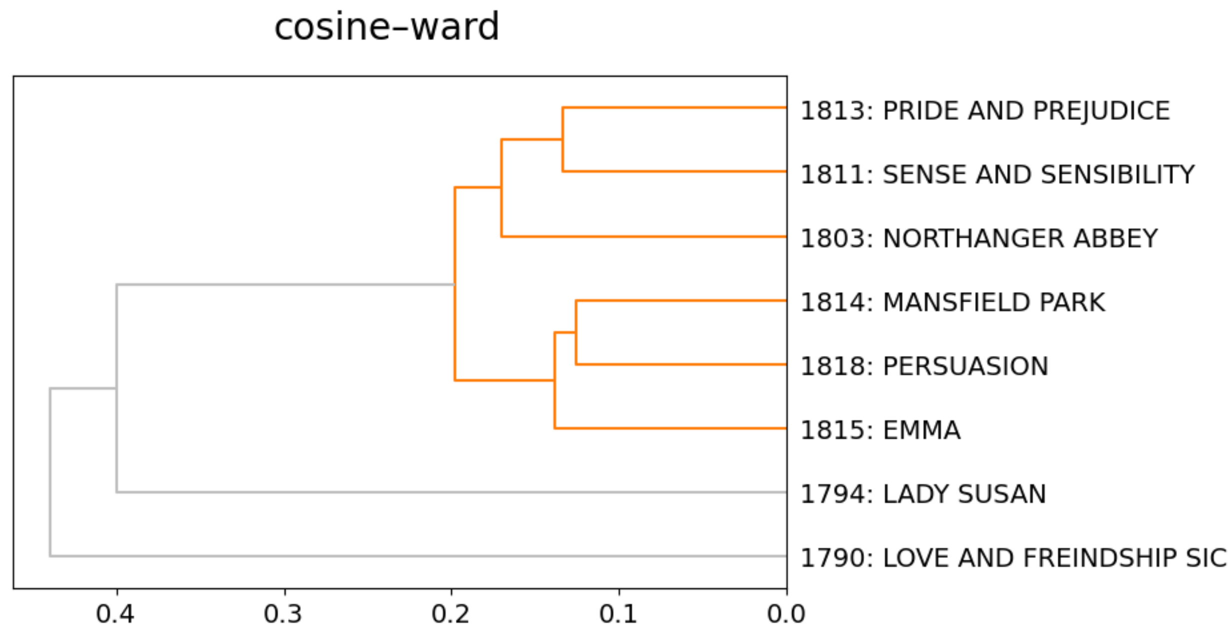
Generate for each combo

In [42]:
```
for X, metric, label, linkage in combos:
    draw_hca(PAIRS[label], linkage_method=linkage)
```

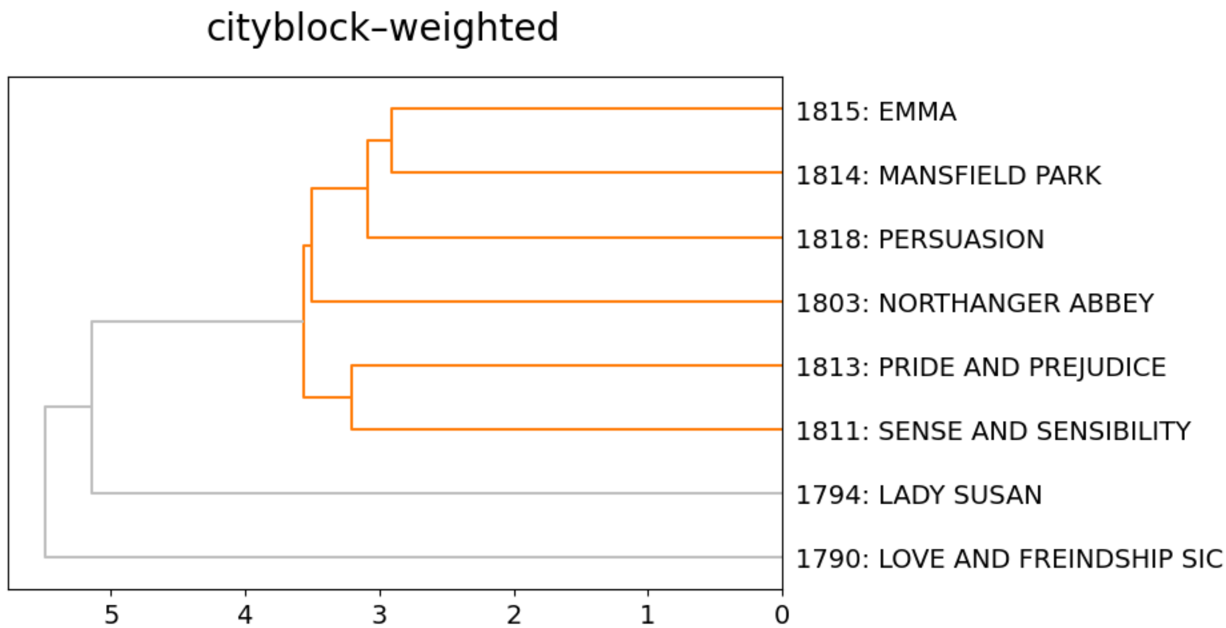
```
<Figure size 640x480 with 0 Axes>
```



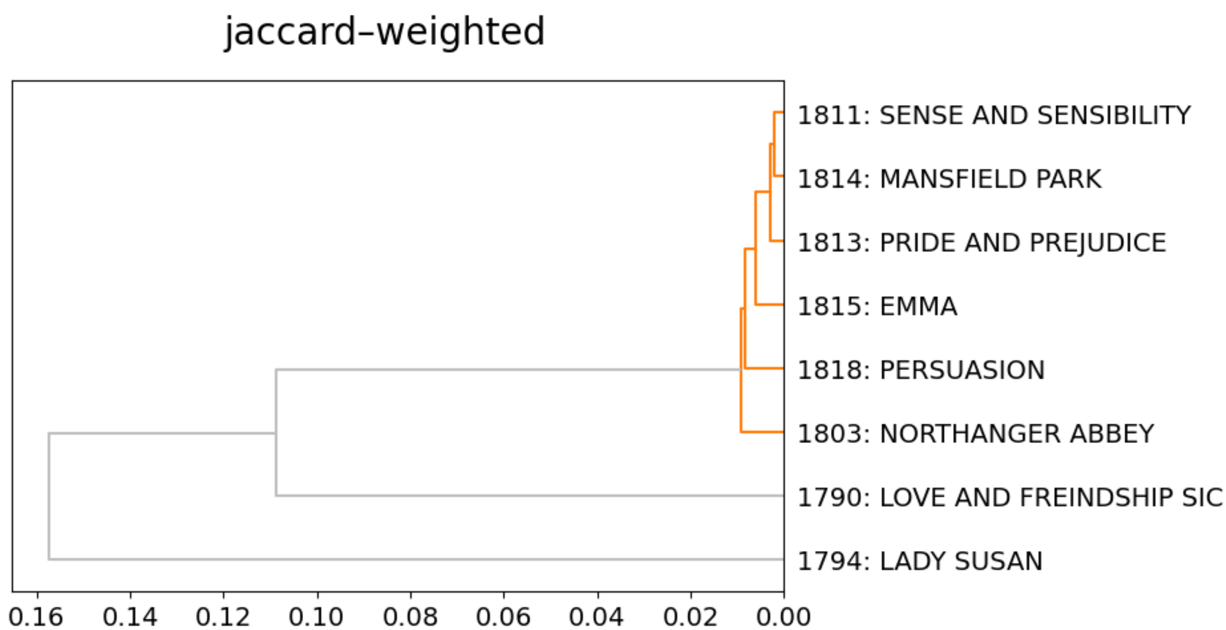

```
<Figure size 640x480 with 0 Axes>
```



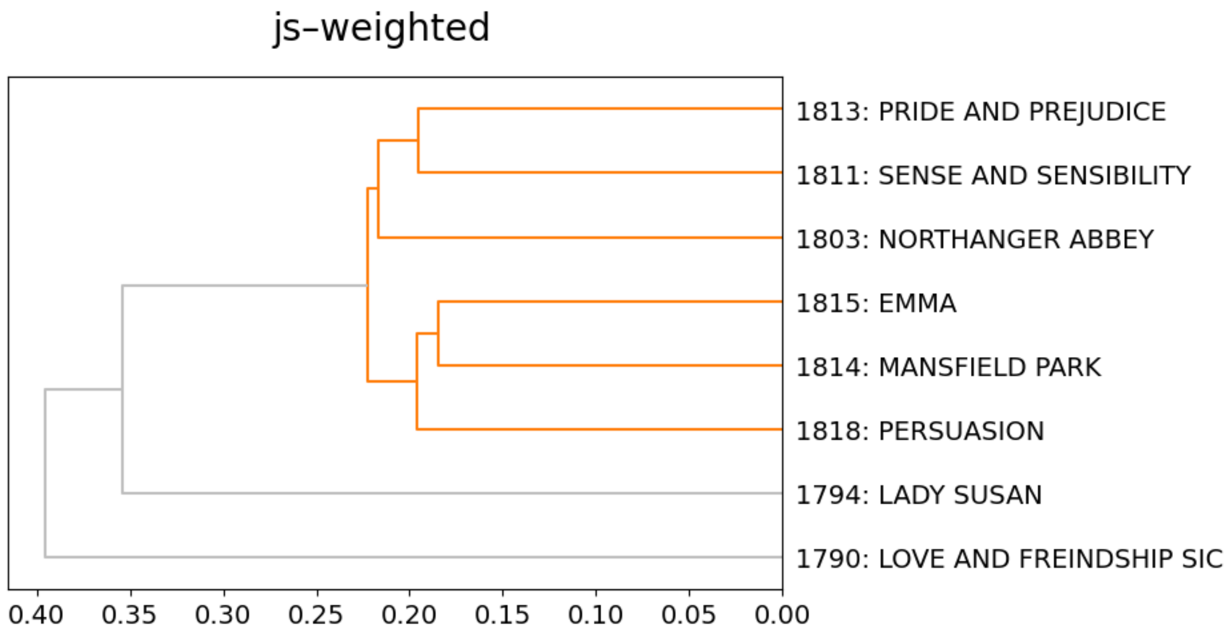

```
<Figure size 640x480 with 0 Axes>
```



<Figure size 640x480 with 0 Axes>



<Figure size 640x480 with 0 Axes>



Save

In [43]:
```
BOW_REDUCED = M.stack().to_frame('tfidf_mean')
BOW_REDUCED['binary']   = L0.stack()
BOW_REDUCED['tfidf_l1'] = L1.stack()
BOW_REDUCED['tfidf_l2'] = L2.stack()
BOW_REDUCED = BOW_REDUCED.sort_index()
```

In [44]:
```
BOW_REDUCED
```

Out[44]:

book_id	term_str	tfidf_mean	binary	tfidf_l1	tfidf_l2
105	able	0.007063	1	0.000977	0.027764
	absence	0.004201	1	0.000581	0.016514
	absolutely	0.003214	1	0.000445	0.012635
	accept	0.004572	1	0.000633	0.017974
	accepted	0.002638	1	0.000365	0.010370
...	...	...	...	...	...
1342	yes	0.013016	1	0.001619	0.046311
	yesterday	0.005150	1	0.000641	0.018323
	younger	0.017307	1	0.002153	0.061578
	yours	0.009408	1	0.001170	0.033472
	youth	0.004538	1	0.000564	0.016144

8000 rows × 4 columns

Answers

Q1

Top 10 Nouns using DFIDF?

In [45]:
```
VOCAB['dfidf'] = DFIDF
```

In [46]:
```
VOCAB.sort_values('dfidf', ascending=False).query('max_pos == "NN" or max_pos =
```

Out[46]:

term_str	n	max_pos	dfidf	mean_tfidf
respect	174	NN	177.266344	0.008183
fortune	222	NN	177.261968	0.010642
marriage	246	NN	177.261968	0.013575
question	171	NN	177.258990	0.006774
ladies	240	NNS	177.258990	0.011303
behaviour	200	NN	177.240001	0.010849
farther	181	NN	177.240001	0.006979
advantage	166	NN	177.217644	0.007974
girl	254	NN	177.209470	0.012677
voice	228	NN	177.209470	0.009163

Q2

Most significant book?

NOTE: The anwser to this question depends on two factors:

- Whether the `TFIDF` table has nulls or not. It should have nulls replaced by 0s, using `fillna(0)` but early I had told the students to keep nulls in the table in order to more easily compute `DF` from the `TFIDF` table.
- Whether they use the full or the reduced `TFIDF` table. The intent of the question was to use the full table, but I can see that this is not completely clear.

We will accept all combinations.

Collapse TFIDF by book

In [47]:
```
LIB['mean_tfidf'] = TFIDF.stack().groupby('book_id').mean()
```

In [48]:
```
LIB.loc[LIB.mean_tfidf.idxmax()].title
```

Out[48]:
```
'NORTHANGER ABBEY'
```

In [49]:
```
TFIDF2 = TFIDF.fillna(0)
```

In [50]:
```
LIB['mean_tfidf2'] = TFIDF2.stack().groupby('book_id').mean()
```

In [51]:
```
LIB.loc[LIB.mean_tfidf2.idxmax()].title
```

Out[51]:
```
'NORTHANGER ABBEY'
```

Class

In [52]:
```
class TfidfVectorizer():

    item_type:str = 'term_str'
    tf_method:str = 'max'
    df_method:str = 'standard'
    V:pd.DataFrame = None


    def __init__(self, CORPUS:pd.DataFrame, VOCAB:pd.DataFrame):
        self.CORPUS = CORPUS
        self.VOCAB = VOCAB
        self.OHCO = list(CORPUS.index.names)

    def create_bow(self, ohco_level):
        self.bag = self.OHCO[:ohco_level]
        self.BOW = self.CORPUS.groupby(self.bag+[self.item_type])\
            [self.item_type].count().to_frame('n')

    def get_tfidf(self):

        DTCM = self.BOW.n.unstack() # Create Doc-Term Count Matrix w/NULLs
        self.V = pd.DataFrame(index=DTCM.columns)

        if 'max_pos' in VOCAB:
            self.V['max_pos'] = self.VOCAB.max_pos

        if self.tf_method == 'sum':
            TF = (DTCM.T / DTCM.T.sum()).T
        elif self.tf_method == 'max':
            TF = (DTCM.T / DTCM.T.max()).T
        elif self.tf_method == 'log':
            TF = (np.log2(1 + DTCM.T)).T
        elif self.tf_method == 'raw':
            TF = DTCM
        elif self.tf_method == 'bool':
            TF = DTCM.astype('bool').astype('int')
        else:
            raise ValueError(f"TF method {tf_method} not found.")

        DF = DTCM.count()
        N_docs = len(DTCM)
```

```
        if self.df_method == 'standard':
            IDF = np.log2(N_docs/DF) # This what the students were asked to use
        elif self.df_method == 'textbook':
            IDF = np.log2(N_docs/(DF + 1))
        elif self.df_method == 'sklearn':
            IDF = np.log2(N_docs/DF) + 1
        elif self.df_method == 'sklearn_smooth':
            IDF = np.log2((N_docs + 1)/(DF + 1)) + 1
        else:
            raise ValueError(f"DF method {df_method} not found.")

        TFIDF = TF * IDF

        self.BOW['tfidf'] = TFIDF.stack()
        self.BOW['tf'] = TF.stack()
        self.V['df'] = DF
        self.V['idf'] = IDF
        self.N_docs = N_docs

    def get_dfidf(self):
        self.V['dfidf'] = self.V.df * self.V.idf

    def get_mean_tfidf_for_VOCAB(self):
        self.V['mean_tfidf'] = self.BOW.groupby('term_str').tfidf.mean()
```

In [53]:

```
# tv = TfidfVectorizer(CORPUS, VOCAB)
# tv.create_bow(2)
# tv.get_tfidf()
# tv.get_dfidf()
```

Experiment

Effect of filling nulls at various points

Means vary becuase N varies in the denominator.

In [54]:

```
M0 = TFIDF.groupby('book_id').mean()            # FILLNA not done
M1 = TFIDF.fillna(0).groupby('book_id').mean() # FILLNA before grouping -- CORR
M2 = TFIDF.groupby('book_id').mean().fillna(0) # FILLNA after grouping (same as
```

In [55]:

```
M0
```

Out[55]:

term_str	0	1	10	10000	10th	11th	12	12th	
book_id									
105	0.000000	0.004962	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
121	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
141	0.000000	0.000000	0.003480	0.000000	0.000000	0.000000	0.000000	0.000000	0
158	0.000000	0.000000	0.000000	0.004234	0.000000	0.000000	0.000000	0.000000	0
161	0.000000	0.001522	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
946	0.000000	0.000000	0.004482	0.000000	0.000000	0.000000	0.000000	0.000000	0
1212	0.001968	0.000000	0.000000	0.000000	0.009915	0.000984	0.006986	0.000984	0
1342	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0

8 rows × 14745 columns

In [56]:
```
M1
```

Out[56]:

term_str	0	1	10	10000	10th	11th	12	12th	
book_id									
105	0.000000	0.004962	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
121	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
141	0.000000	0.000000	0.003480	0.000000	0.000000	0.000000	0.000000	0.000000	0
158	0.000000	0.000000	0.000000	0.004234	0.000000	0.000000	0.000000	0.000000	0
161	0.000000	0.001522	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
946	0.000000	0.000000	0.004482	0.000000	0.000000	0.000000	0.000000	0.000000	0
1212	0.001968	0.000000	0.000000	0.000000	0.009915	0.000984	0.006986	0.000984	0
1342	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0

8 rows × 14745 columns

In [57]:
```
M2
```

Out[57]:

term_str	0	1	10	10000	10th	11th	12	12th	
book_id									
105	0.000000	0.004962	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
121	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
141	0.000000	0.000000	0.003480	0.000000	0.000000	0.000000	0.000000	0.000000	0
158	0.000000	0.000000	0.000000	0.004234	0.000000	0.000000	0.000000	0.000000	0
161	0.000000	0.001522	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
946	0.000000	0.000000	0.004482	0.000000	0.000000	0.000000	0.000000	0.000000	0
1212	0.001968	0.000000	0.000000	0.000000	0.009915	0.000984	0.006986	0.000984	0
1342	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0

8 rows × 14745 columns

In [ ]: