INFO

Name: Jacqui Unciano **Assignment:** HW08 **Date:** 03/20/2024

Directions

In this week's homework, you will create topic models using Scikit Learn's LatentDirichletAllocation class. Using the notebooks from the previous week (Module 08), and the CORPUS and LIB tables for the novels collection (found in the novels subdirectory of the shared Dropbox folder), do the following:

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation as LDA
import plotly_express as px
```

In [2]:
```
import configparser
config = configparser.ConfigParser()
config.read('../../../env.ini')
data_home = config['DEFAULT']['data_home']
output_dir = config['DEFAULT']['output_dir']
```

In [3]:
```
data_prefix = 'novels'
colors = "YlGnBu"
```

In [4]:
```
CORPUS = pd.read_csv(f'{data_home}/{data_prefix}-CORPUS.csv')
LIB = pd.read_csv(f'{data_home}/{data_prefix}-LIB.csv')
```

In [5]:
```
CORPUS.head()
```

Out[5]:

	book_id	chap_id	para_num	sent_num	token_num	pos	term_str
0	secretadversary	1	0	1	0	DT	the
1	secretadversary	1	0	1	1	NNP	young
2	secretadversary	1	0	1	2	NNP	adventurers
3	secretadversary	1	0	1	3	NNP	ltd
4	secretadversary	1	1	0	0	JJ	tommy

In [6]:
```
LIB.head()
```

Out[6]:

	book_id	genre_id	author_id
0	secretadversary	d	christie
1	styles	d	christie
2	moonstone	d	collins
3	adventures	d	doyle
4	baskervilles	d	doyle

Generate two topic models, one for paragraphs as documents (i.e. bags), the other with chapters as documents. A topic model in this context comprises three tables: THETA, PHI, and TOPICS. For each model, use the following parameters:

CountVectorizer

1. max_features = 4000
2. stop_words = 'english'

LatentDirichletAllocation

1. n_components = 20
2. max_iter = 5
3. learning_offset = 50
4. random_state = 0

Hyperparameters

1. Use only nouns (NN and NNS)
2. Number of words used to characterize a topic: 7

Note: You may want to generalize the notebook code use to generate topic models by creating a class, or at least a library of functions, to perform various tasks. This will allow to quickly run topic models with bag as a parameter. An added benefit of creating a class for exploring topic models is that you can use it in your final projects.

self.n_topics = n_topics self.max_iter = max_iter self.learning_offset = learning_offset self.random_state = random_state self.colors = colors self.n_terms = n_terms self.n_gram_range = n_gram_range self.stop_words = stop_words , n_topics=20, max_iter=5, learning_offset=50., random_state=0, colors = "YlGnBu", n_terms=4000, n_gram_range=(1,2), stop_words='english'

In [7]:
```
class Topic:
    DOCS = pd.DataFrame()
    CORPUS = pd.DataFrame()
    THETA = pd.DataFrame()
    PHI = pd.DataFrame()
    TOPICS = pd.DataFrame()
    TERMS = []
    DTM = pd.DataFrame()
    TNAMES = []
    n_topics=20
```

```
    max_iter=5
    learning_offset=50
    random_state=0
    n_terms=4000
    ngram_range=(1,2)
    stop_words='english'


    def __init__(self, CORPUS):
        self.CORPUS = CORPUS

    def get_doc(self, bag=['book_id', 'chap_num', 'para_num'], filter_on=True, filt
        if filter_on==True:
            self.DOCS = self.CORPUS[self.CORPUS.pos.str.match(filter)]\
            .groupby(bag).term_str\
            .apply(lambda x: ' '.join(x))\
            .to_frame()\
            .rename(columns={'term_str':'doc_str'})
        elif filter_on==False:
            self.DOCS = self.CORPUS.groupby(bag).term_str\
            .apply(lambda x:' '.join(x)).to_frame('doc_str')
        return self.DOCS

    def vec_engine(self):
        count_engine = CountVectorizer(max_features=self.n_terms,
                                       ngram_range=self.ngram_range,
                                       stop_words=self.stop_words)
        count_model = count_engine.fit_transform(self.DOCS.doc_str)
        self.TERMS = count_engine.get_feature_names_out()
        return count_model

    def lda_engine(self):
        lda_engine = LDA(n_components=self.n_topics,
                         max_iter=self.max_iter,
                         learning_offset=self.learning_offset,
                         random_state=self.random_state)
        count_model = self.vec_engine()
        self.TNAMES = [f"T{str(x).zfill(len(str(self.n_topics)))}" for x in range(s
        self.DTM = pd.DataFrame(count_model.toarray(), index=self.DOCS.index, colum
        lda = lda_engine.fit_transform(count_model)
        self.lda_components = lda_engine.components_
        return lda

    def get_theta(self):
        lda_model = self.lda_engine()
        THETA = pd.DataFrame(lda_model, index=self.DOCS.index)
        THETA.columns.name = 'topic_id'
        THETA.columns = self.TNAMES
        self.THETA = THETA.T
        return self.THETA

    def get_phi(self):
        PHI = pd.DataFrame(self.lda_components, columns=self.TERMS, index=self.TNAM
        PHI.index.name = 'topic_id'
        PHI.columns.name = 'term_str'
        self.PHI = PHI.T
```

```
        return self.PHI

    def get_topics(self, n_top_terms=10):
        self.TOPICS = self.PHI.stack().groupby('topic_id')\
        .apply(lambda x: ' '.join(x.sort_values(ascending=False).head(n_top_terms).
        .to_frame('top_terms')
        return self.TOPICS

    def sort_doc_weight(self):
        self.TOPICS['doc_weight_sum'] = self.THETA.sum()
        self.TOPICS['term_freq'] = self.PHI.sum(1) / self.PHI.sum(1).sum()
        return self.TOPICS.sort_values('doc_weight_sum', ascending=False)
```

In [8]:
```
OHCO = list(CORPUS.columns)[:3]
PARA = OHCO[:3]
CHAP = OHCO[:2]
BOOK = OHCO[:1]
```

Topic Model 1: BAG=PARA

In [9]:
```
topic = Topic(CORPUS)
topic.get_doc(bag=PARA)
theta1 = topic.get_theta()
theta1.sample(10).style.background_gradient(cmap=colors, axis=None)
```

Out[9]:

book_id								
chap_id								
para_num	1	2	3	4	5	6	7	8
T19	0.050000	0.001563	0.223385	0.264679	0.182088	0.016667	0.025000	0.025000
T15	0.050000	0.001563	0.001064	0.135098	0.004167	0.683333	0.025000	0.025000
T07	0.050000	0.001563	0.001064	0.001471	0.004167	0.016667	0.025000	0.525000
T05	0.050000	0.614620	0.001064	0.001471	0.004167	0.016667	0.025000	0.025000
T03	0.050000	0.001563	0.001064	0.001471	0.386463	0.016667	0.525000	0.025000
T17	0.050000	0.001563	0.001064	0.001471	0.004167	0.016667	0.025000	0.025000
T10	0.050000	0.001563	0.001064	0.001471	0.004167	0.016667	0.025000	0.025000
T00	0.050000	0.001563	0.001064	0.083195	0.004167	0.016667	0.025000	0.025000
T16	0.050000	0.001563	0.001064	0.001471	0.004167	0.016667	0.025000	0.025000
T01	0.050000	0.266616	0.484450	0.001471	0.004167	0.016667	0.025000	0.025000

In [10]:
```
phi1 = topic.get_phi()
phi1.sample(10).style.background_gradient(cmap=colors, axis=None)
```

Out[10]:

topic_id	T00	T01	T02	T03	T04	T05	T06	
term_str								
quarrel	0.050458	0.050000	0.051628	6.298785	0.050000	0.050000	1.274552	23.013
tear	4.385259	0.050004	0.050000	0.050000	0.050000	5.410084	0.050000	7.354
principles	3.463560	0.050000	0.050000	0.050000	0.050000	0.389003	0.050000	6.492
task	2.067276	4.156591	0.056003	0.920358	0.185036	0.050000	0.050017	0.050
close	0.050000	0.050000	0.050011	17.769274	0.050000	2.090710	0.050000	0.050
pink	2.707859	0.050000	0.050000	0.050000	0.050000	0.050000	0.050000	0.050
isnt	0.050000	0.061232	3.340762	0.050000	0.050000	0.050000	30.594248	0.050
train	0.050000	0.237536	0.050000	10.097949	0.050714	0.489362	7.055574	4.664
low	0.050000	0.050000	0.050000	0.050000	0.050000	0.050000	3.485489	0.098
number	0.050000	1.744235	0.050000	2.056428	23.147216	13.298957	0.050000	2.508

In [11]:
```
topic1 = topic.get_topics(n_top_terms=7)
topic1.head()
```

Out[11]:

topic_id	top_terms
T00	eyes head girl mother face paper silence
T01	yes oh father good ah man child
T02	sir man coffee dinner moor thank time
T03	case word course time room son papers
T04	door room apartment night chamber hand man

Topic Model 2: BAG=CHAP

In [12]:
```
topic_2 = Topic(CORPUS)
topic_2.get_doc(bag=CHAP)
theta2 = topic_2.get_theta()
theta2.sample(10).style.background_gradient(cmap=colors, axis=None)
```

Out[12]:

book_id									
chap_id	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	
T17	0.000041	0.000038	0.000052	0.000035	0.000045	0.000034	0.000042	0.000030	0.
T00	0.000041	0.000038	0.000052	0.067080	0.000045	0.000034	0.005242	0.000030	0.
T14	0.000041	0.000038	0.000052	0.000035	0.000045	0.000034	0.000042	0.000030	0.
T12	0.000041	0.000038	0.000052	0.000035	0.217812	0.000034	0.000042	0.000030	0.
T01	0.453161	0.000038	0.337203	0.337080	0.111001	0.078458	0.073031	0.101275	0.
T11	0.000041	0.000038	0.000052	0.000035	0.000045	0.000034	0.000042	0.000030	0.
T13	0.000041	0.000038	0.000052	0.000035	0.000045	0.000034	0.000042	0.000030	0.
T02	0.513976	0.999271	0.346489	0.449709	0.576448	0.920931	0.814449	0.748876	0.
T07	0.012536	0.000038	0.315419	0.099572	0.065585	0.000034	0.094079	0.000030	0.
T15	0.000041	0.000038	0.000052	0.016240	0.000045	0.000034	0.000042	0.117656	0.

In [13]:
```
phi2 = topic_2.get_phi()
phi2.sample(10).style.background_gradient(cmap=colors, axis=None)
```

Out[13]:

topic_id	T00	T01	T02	T03	T04	T05	T06	T07
term_str								
rate	0.050000	19.275675	12.565009	0.050000	2.115230	3.229982	0.050000	25.84759
opinions	0.050000	0.050000	0.840718	0.050000	0.050000	0.050000	0.050000	0.05993
directly	0.127493	0.050000	1.052698	0.050000	0.050000	0.050000	1.050000	13.43250
defiance	1.054681	1.050000	0.631161	0.050000	0.050000	0.050000	0.050000	5.93292
criminals	3.106668	4.425669	12.041647	0.050000	0.127666	0.050000	0.183489	0.14019
portico	0.050000	0.050000	0.050000	0.050000	0.050000	0.050000	0.050000	0.05000
tm works	0.050000	0.050000	0.050000	0.050000	0.050000	0.050000	0.050000	0.05000
disorder	3.503737	4.224959	0.053968	0.050000	4.499481	0.236699	0.050000	0.05000
musician	0.050000	0.050000	1.050000	0.050000	0.050000	0.050000	0.050000	0.05000
arches	0.050000	0.050000	0.050000	0.050000	0.050000	0.050000	0.050000	0.05000

In [14]:
```
topic2 = topic_2.get_topics(n_top_terms=7)
topic2.head()
```

Out[14]:

topic_id	top_terms
T00	son father man heart time hand chamber
T01	room yes man time door face way
T02	man time way hand room day night
T03	corpse body murder evidence period thicket river
T04	eyes heart hand moment bosom oh voice

Once you are done with these tasks, answer the following questions:

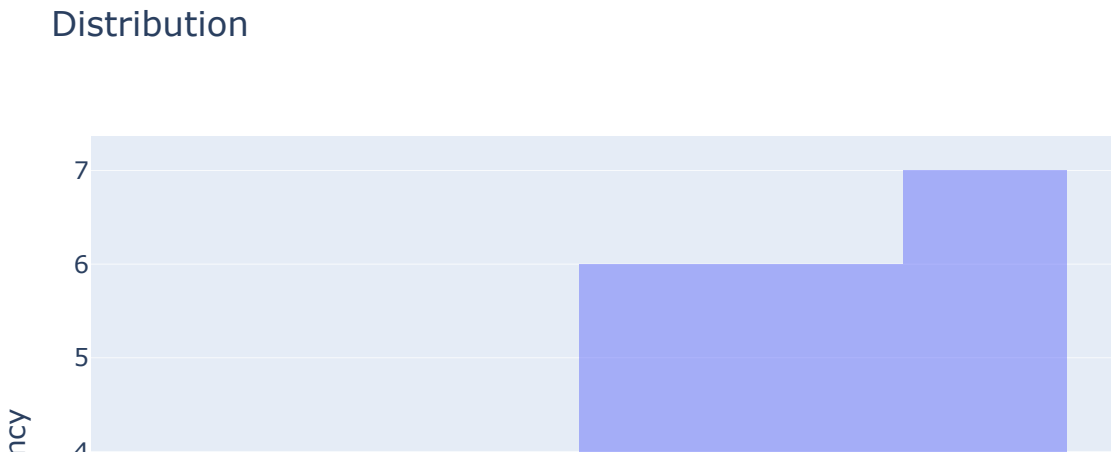
Question 1

Use the PHI table from each model to compute the entropy H of the distribution over topics. Which bag generates a lower entropy distribution? Hint: To get H work with the L1 normalized vector of word weight sums by topic in the PHI table.

In [15]:
```
word_weight_sums = phi1.sum(axis=1)
normalized_weights = phi1.div(word_weight_sums, axis=0)
entropy1 = -np.sum(normalized_weights * np.log2(normalized_weights))

word_weight_sums = phi2.sum(axis=1)
normalized_weights = phi2.div(word_weight_sums, axis=0)
entropy2 = -np.sum(normalized_weights * np.log2(normalized_weights))

combined_entropies = {'BAG=PARA': entropy1, 'BAG=CHAP': entropy2}
df = pd.DataFrame(combined_entropies)
fig = px.histogram(df, nbins=20, barmode='overlay',
                   title='Distribution')
fig.update_layout(xaxis_title='Entropy', yaxis_title='Frequency')
fig.show()
```



Answer 1: The model with the chapters as the bag shows a lower entropy distribution over topics compared to the model with paragraphs as the bag.

Question 2

Sort the topics in each model's PHI table by topic entropy in descending order. Are the first topics in the two models about the same? In other words, do they yield similar interpretations?

In [16]:
```
entropy1.nlargest(5)
```

Out[16]:
```
topic_id
T12    605.552674
T10    596.199161
T04    564.113079
T18    561.727184
T09    558.800462
dtype: float64
```

In [17]:
```
entropy2.nlargest(5)
```

Out[17]:
```
topic_id
T02    1110.904105
T07     945.111782
T01     907.878571
T16     837.335306
T10     800.641031
dtype: float64
```

Answer 2: No, the topics are not the same, except for maybe 1 or 2.

Question 3

What topic from each model is most strongly associated with each genre? Note that your answer have four parts.

In [18]:
```
LIB.shape
LIB.index = LIB.book_id
```

In [19]:
```
grouped1 = theta1.T.groupby(level=0).mean()
# grouped1.shape
merged1 = pd.concat([grouped1, LIB], axis=1)
```

In [20]:
```
merged1.drop(columns=['book_id', 'author_id'], inplace=True)
```

In [21]:
```
grouped2 = theta2.T.groupby(level=0).mean()
# grouped1.shape
merged2 = pd.concat([grouped2, LIB], axis=1)
merged2.drop(columns=['book_id', 'author_id'], inplace=True)
```

In [22]:
```
merged1.groupby('genre_id').mean().T.nlargest(1, 'd')
```

Out[22]:

genre_id	d	g
T14	0.077512	0.045398

In [23]:
```
merged1.groupby('genre_id').mean().T.nlargest(1, 'g')
```

Out[23]:

genre_id	d	g
T09	0.042309	0.118012

In [24]:
```
merged2.groupby('genre_id').mean().T.nlargest(1, 'd')
```

Out[24]:

genre_id	d	g
T02	0.283792	0.090618

In [25]:
```
merged2.groupby('genre_id').mean().T.nlargest(1, 'g')
```

Out[25]:

genre_id	d	g
T15	0.123243	0.195961

Answer 3: For bag=PARA model, the topic T14 is most strongly associated with genre 'd' while the topic T09 is most strongly associated with genre 'g'. For bag=CHAP model, the topic T02 is most strongly associated with genre 'd' while the topic T15 is most strongly associated with genre 'g'.

Question 4

Using the THETA table from the Chapters model, get the mean topic weights for each book. Which book is most strongly associated with the gothic genre g, based on the weight of that genre's most representative topic (as discovered in the previous question)?

In [26]:
```
merged1[merged1.genre_id=='g'].T15.nlargest(1)
```

Out[26]:
```
book_id
dracula    0.079487
Name: T15, dtype: float64
```

Answer 4: Dracula is the book that is most strongly associated with the gothic genre based on the gothic rep. topic, T15.

Question 5

How would you characterize the subject matter of the two genres based on their topic models? Consider the words associated with the dominant topics from each model, but also the models overall.

In [27]:
```
g1 = merged1[merged1.genre_id=='g']
g1 = g1.drop(columns=['genre_id'])
g1 = g1.mean().to_frame('mean')
g1.index.name = 'topic_id'
pd.concat([g1, topic1], axis=1).nlargest(5, 'mean')
```

Out[27]:

topic_id	mean	top_terms
T09	0.118012	moment blood face eyes hand death heart
T10	0.072159	mind day night hour hope steps time
T12	0.070003	room door light hall air window hand
T04	0.057246	door room apartment night chamber hand man
T05	0.052756	country thing scene world mind day mountains

In [28]:
```
d1 = merged1[merged1.genre_id=='d']
d1 = d1.drop(columns=['genre_id'])
d1 = d1.mean().to_frame('mean')
d1.index.name = 'topic_id'
pd.concat([d1, topic1], axis=1).nlargest(5, 'mean')
```

Out[28]:

topic_id	mean	top_terms
T14	0.077512	hand face time case evidence room man
T15	0.068529	day time room way night sort thing
T11	0.064233	lady business right matter things years mind
T18	0.063951	time man morning hour way house chance
T03	0.059443	case word course time room son papers

In [29]:
```
g2 = merged2[merged2.genre_id=='g']
g2 = g2.drop(columns=['genre_id'])
g2 = g2.mean().to_frame('mean')
g2.index.name = 'topic_id'
pd.concat([g2, topic2], axis=1).nlargest(5, 'mean')
```

Out[29]:

topic_id	mean	top_terms
T15	0.195961	room door house time hand bed voice
T00	0.158952	son father man heart time hand chamber
T10	0.117973	time heart friend family brother moment room
T02	0.090618	man time way hand room day night
T13	0.078666	tm work works terms agreement tm works donations

In [30]:
```
d2 = merged2[merged2.genre_id=='d']
d2 = d2.drop(columns=['genre_id'])
d2 = d2.mean().to_frame('mean')
d2.index.name = 'topic_id'
pd.concat([d2, topic2], axis=1).nlargest(5, 'mean')
```

Out[30]:

topic_id	mean	top_terms
T02	0.283792	man time way hand room day night
T01	0.257933	room yes man time door face way
T07	0.139969	time way house man room sir place
T15	0.123243	room door house time hand bed voice
T03	0.062014	corpse body murder evidence period thicket river

Answer 5: For the model that bagged in paragraphs, there's a lot more clarity between the two genres. For the detective genre, words like business, case, and evidence are words I expect to be more prominent in a detective novel than a gothic novel. Meanwhile the model that bagged in chapters, it's difficult to make that distinction just by looking at the top 2-3 topics since there's a lot of overlap/similarities of the words between genres (i.e. man, door, house).