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
T16	0.050000	0.001563	0.001064	0.001471	0.004167	0.016667	0.025000	0.025000
T04	0.050000	0.001563	0.001064	0.460585	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
T12	0.050000	0.001563	0.001064	0.001471	0.004167	0.016667	0.025000	0.025000
T17	0.050000	0.001563	0.001064	0.001471	0.004167	0.016667	0.025000	0.025000
T11	0.050000	0.001563	0.001064	0.001471	0.004167	0.016667	0.025000	0.025000
T05	0.050000	0.614620	0.001064	0.001471	0.004167	0.016667	0.025000	0.025000
T13	0.050000	0.001563	0.001064	0.034384	0.004167	0.016667	0.025000	0.025000
T14	0.050000	0.092201	0.001064	0.001471	0.004167	0.016667	0.025000	0.025000