Cosmos via Natural Language Processing, by Sung Been Lee

Cosmos is a popular science television series written and presented by astronomer and astrophysicist Carl Sagan in which he explored a wide range of scientific subjects, including the origins of the universe, the development of life on Earth, and the search for extraterrestrial life. In addition to its educational value, it is significant for its cultural impact, as the series was widely watched and became a cultural phenomenon with Sagan's distinctive presentation style becoming widely recognized and imitated. The series has been credited with helping to popularize the study of science and inspiring many people to pursue careers in science and related fields.

In this project, I will be exploring what insights a computer might be able to gain about *Cosmos*, via machine learning. The book is saved in a .txt file on my local disk as cosmos_by_carl_sagan.txt, containing the contents of the book.

The Data

To begin, cosmos_by_carl_sagan.txt will be brought into the workspace.

In [1]:
```
file = open("cosmos_by_carl_sagan.txt", "r") # using open() to read in the book
s = ''.join(file.readlines()) # concatenate each line of the book into one string called read
s = s.replace("\n", "") # remove line breaks
len(s)
```

Out[1]:
```
750807
```

In [2]:
```
s[:200] # show the first 200 characters of s
```

Out[2]:
```
'Spacecraft missions to nearby Planets The Library of ancient Alexandria The human brain Egyptian Hieroglyphics The or
igin of life The death of the Sun The evolution of galaxies The origins of matter, '
```

Split into chapters

Since the chapters are demaracted by the all-caps word "CHAPTER", I will split on this word to break the book up into chapters. I will be excluding contents of the book before the first chapter as this part of the book is not directly relevant to the main contents of the book.

In [3]:
```
chapters = s.split("CHAPTER")[1:] # get rid of text before chapter 1
len(chapters)
```

Out[3]:
```
13
```

As seen in the output above, there are a total of 13 chapters. The texts of each chapter will be organized in a `pandas` data frame. The data frame will have two columns: `Chapters` and `Texts`.

In [4]:
```
import pandas as pd

df = pd.DataFrame({
    "Chapters":range(1, len(chapters)+1),
    "Texts":chapters
})

df
```

Out[4]:

	Chapters	Texts
0	1	I The Shores of the Cosmic Ocean The first me...
1	2	II One Voice in the Cosmic Fugue I am bidden ...
2	3	III The Harmony of Worlds Do you know the ord...
3	4	IV Heaven and Hell Nine worlds I remember. - ...
4	5	V Blues for a Red Planet In the orchards of t...
5	6	VI Travelers' Tales Do there exist many world...
6	7	VII The Backbone of Night They came to a roun...
7	8	VIII Travels in Space and Time No one has liv...
8	9	IX The Lives of the Stars Opening his two eye...
9	10	X The Edge of Forever There is a thing confus...
10	11	XI The Persistence of Memory Now that the des...
11	12	XII Encyclopaedia Galactica 'What are you? Fr...
12	13	XIII Who Speaks for Earth ? To what purpose s...

Next, I am going to use the `CountVectorizer` function from the `sklearn.feature_extraction.text` module. This function will turn the unstructured text into quantitative numbers that can be fed into algorithms.

The `CountVectorizer` object will be applied on `df` and stored into `vec`. I've specified the `stop_words` argument to `"english"` (a stop word is a word that is considered uninteresting for the purposes of natural language processing; for instance, "she," "can", and "the").

In [5]:
```
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np

vec = CountVectorizer(stop_words = "english")
counts = vec.fit_transform(df['Texts']) # create a term-document matrix on the text data

type(counts)
```

Out[5]:
```
scipy.sparse.csr.csr_matrix
```

`counts` represents the term-document matrix. However, as shown in the code above, its type is `scipy.sparse.csr.csr_matrix`, which makes it difficult to work with. For this reason, I will be converting it to a `numpy` array with the `toarray()` method.

In [6]:
```
counts = counts.toarray()
type(counts)
```

Out[6]:
```
numpy.ndarray
```

In [7]:
```
counts.shape # dimension of counts
```

Out[7]:
```
(13, 11286)
```

Now that the term-document matrix is synthesized, I will be converting `counts` into a data frame with labeled columns, each column name representing a word that appears in the book. This will be done using the `get_feature_names_out()` method.

In [8]:
```
count_df = pd.DataFrame(counts, columns = vec.get_feature_names_out())
count_df
```

Out[8]:

	000	000a	01	018	0329	04	10	100	1000	1000a	...	zest	zeus	zhou	zodiac	zodiacal	zone	zones	zoo	zoroaster	zoroastrian
0	12	0	0	0	0	0	7	0	0	0	...	0	0	0	0	0	1	0	1	0	0
1	1	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	4	0	0	0	0	0	1	1	0	0	...	0	0	0	2	0	0	0	0	0	1
3	10	1	0	0	0	0	20	3	0	1	...	0	0	1	0	0	0	0	0	0	0
4	1	0	0	0	0	0	5	2	0	0	...	0	0	0	0	0	0	0	0	0	0
5	2	0	0	0	0	1	0	0	0	0	...	1	0	0	0	0	0	0	0	0	0
6	5	0	0	0	0	0	0	0	1	0	...	1	4	0	0	0	0	0	0	1	0
7	9	0	1	0	0	0	8	0	0	0	...	0	0	0	1	1	0	0	1	0	0
8	39	0	0	0	1	0	16	3	2	0	...	0	0	0	0	0	1	0	0	0	0
9	0	0	0	0	0	0	0	2	0	0	...	0	0	0	0	0	0	0	0	0	0
10	5	0	0	0	0	0	20	2	0	0	...	0	0	0	0	0	0	1	0	0	0
11	5	0	0	0	0	0	8	2	0	0	...	0	0	0	0	0	0	0	0	0	0
12	6	0	0	1	0	0	8	1	0	0	...	0	0	0	0	0	0	0	0	0	0

13 rows × 11286 columns

Now that a term-document matrix in the form of a data frame is created, it will be concatenated with the original data frame with chapter and text data.

In [9]:
```
df = pd.concat((df,count_df), axis = 1)

# drop columns 2 to 242 to remove non-english words, e.g. (000, 000a01, etc.)
df = df.drop(columns = df.columns[2:243])
df.head(5) # show first 5 observations
```

Out[9]:

	Chapters	Texts	abalone	abandon	abandoned	abbott	abdera	aberration	abide	abiding	...	zest	zeus	zhou	zodiac	zodiacal	zo
0	1	I The Shores of the Cosmic Ocean The first me...	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
1	2	II One Voice in the Cosmic Fugue I am bidden ...	0	0	0	0	0	0	1	0	...	0	0	0	0	0	
2	3	III The Harmony of Worlds Do you know the ord...	0	2	1	0	0	0	0	0	...	0	0	0	2	0	
3	4	IV Heaven and Hell Nine worlds I remember. - ...	0	0	1	0	0	0	0	0	...	0	0	1	0	0	
4	5	V Blues for a Red Planet In the orchards of t...	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

5 rows × 11047 columns

Interpreting the Term-Document Matrix

Now that the data frames are concatenated and cleaned, the term-document matrix can be used to check how frequently a given term appears in each chapter of the novel by indexing the term as a column. For instance, the code below displays how many times the word 'human' appeared in each capter of the book.

In [10]:
```
df[['Chapters','human']]
```

Out[10]:

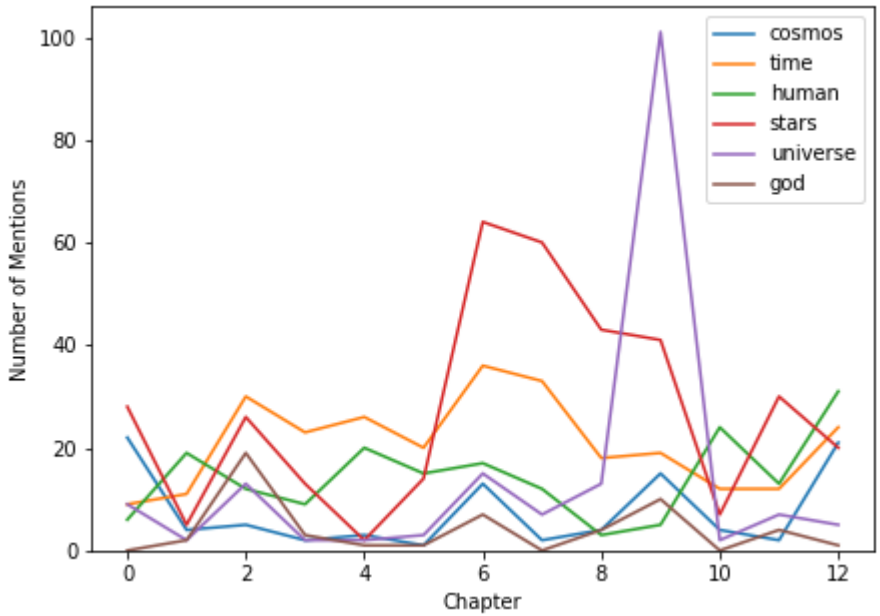
	Chapters	human
0	1	6
1	2	19
2	3	12
3	4	9
4	5	20
5	6	15
6	7	17
7	8	12
8	9	3
9	10	5
10	11	24
11	12	13
12	13	31

Now, I will use `matplotlib` to to see how words like 'cosmos', 'time', 'human', 'stars', 'universe', and 'god' appear throughout the book. These are just words that I find personally interesting in relation to the book.

In [11]:
```
from matplotlib import pyplot as plt

words = ['cosmos','time','human','stars','universe','god']
fig, ax = plt.subplots(1, figsize = (7,5))

# plot each word on the figure
for word in words:
    ax.plot(df[word], label = word)
```

```
ax.set(ylim = (0, None),
       ylabel = "Number of Mentions",
       xlabel = "Chapter")

ax.legend()
```

Out[11]:
```
<matplotlib.legend.Legend at 0x7faf8420ed90>
```



Topic Modeling

In this section, I will be using `scikit-learn` to perform topic modeling. Topic modeling is a type of unsuprivised machine learning technique that automatically extracts the main topics from a collection of documents. The overall aim of this section is to get a coarse, topic-level summary of the plot of *Cosmos*.

In [12]:
```
# redefining df to contain just the chapter and text data
df = pd.DataFrame({
    "Chapters":range(1,len(chapters)+1),
     "Texts":chapters
})
```

```
df
```

Out[12]:

	Chapters	Texts
0	1	I The Shores of the Cosmic Ocean The first me...
1	2	II One Voice in the Cosmic Fugue I am bidden ...
2	3	III The Harmony of Worlds Do you know the ord...
3	4	IV Heaven and Hell Nine worlds I remember. - ...
4	5	V Blues for a Red Planet In the orchards of t...
5	6	VI Travelers' Tales Do there exist many world...
6	7	VII The Backbone of Night They came to a roun...
7	8	VIII Travels in Space and Time No one has liv...
8	9	IX The Lives of the Stars Opening his two eye...
9	10	X The Edge of Forever There is a thing confus...
10	11	XI The Persistence of Memory Now that the des...
11	12	XII Encyclopaedia Galactica 'What are you? Fr...
12	13	XIII Who Speaks for Earth ? To what purpose s...

Next, I will again use `CountVectorizer` to create a term-document matrix; but in this case, because I'd like to eventually be able to see how topics evolve between chapters, I use the `max_df` argument to specify that I'd like like to include words that appear in at most 70% of the chapters, expressed as `max_df = .7`.

In [13]:
```
from sklearn.feature_extraction.text import CountVectorizer

vec = CountVectorizer(max_df = .7, min_df = 0, stop_words = "english")
```

Now, `vec` will be used to create the term-document matrix and collect it all as a data frame. Because of the `max_df = .7` argument of the `CountVectorizer` method, the number of words (columns) to remove from the data frame is now `df.columns[2:241]`, as the first two columns (`Chapters` and `Texts`) need to be preserved, and the other non-english words need to be removed.

In [14]:

```
counts = vec.fit_transform(df['Texts'])
counts = counts.toarray()
count_df = pd.DataFrame(counts, columns = vec.get_feature_names_out())
df = pd.concat((df,count_df), axis = 1)
df = df.drop(columns = df.columns[2:241])

df.head(5)
```

Out[14]:

	Chapters	Texts	abalone	abandon	abandoned	abbott	abdera	aberration	abide	abiding	...	zest	zeus	zhou	zodiac	zodiacal	zo
0	1	I The Shores of the Cosmic Ocean The first me...	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
1	2	II One Voice in the Cosmic Fugue I am bidden ...	0	0	0	0	0	0	1	0	...	0	0	0	0	0	
2	3	III The Harmony of Worlds Do you know the ord...	0	2	1	0	0	0	0	0	...	0	0	0	2	0	
3	4	IV Heaven and Hell Nine worlds I remember. - ...	0	0	1	0	0	0	0	0	...	0	0	1	0	0	
4	5	V Blues for a Red Planet In the orchards of t...	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

5 rows × 10693 columns

Since topic modeling is a form of unsupervised machine learning, I will only need to define `X`, the predictor data, as shown below. Columns `['Chapters','Texts']` will be removed as only the term-document matrix portion of the data franme will be fed into the algorithm.

In [15]:
```
X = df.copy()
X = X.drop(['Chapters','Texts'], axis = 1)
X.head(5)
```

Out[15]:

	abalone	abandon	abandoned	abbott	abdera	aberration	abide	abiding	abilities	ability	...	zest	zeus	zhou	zodiac	zodiacal	zone	z
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1	
1	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	
2	0	2	1	0	0	0	0	0	0	2	...	0	0	0	2	0	0	
3	0	0	1	0	0	0	0	0	0	1	...	0	0	1	0	0	0	
4	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	

5 rows × 10691 columns

There are many algorithms for topic modeling but in this project I will use nonnegative matrix factorization, or NMF.

NMF decomposes the term-document matrix into topics. It starts with a matrix $X \sim \text{documents} \times \text{words}$. Then it factors $X = WH$ where $W = \text{documents} \times \text{topics}$ and $H = \text{topics} \times \text{words}$.

In other words, before the factorization, the frequency of each word in each document is observed. After the factorization, how strongly a word is associated with a given topic and how strongly associated a topic is with a given document.

In summary, this process can be broken down into three steps:

1. Import the desired model.
2. Initialize an instance of the model.
3. Fit the model on data.

NMF requires the `n_components` argument to be defined, which is the number of topics to find:

In [16]:
```
from sklearn.decomposition import NMF

topic_count = 6
```

```
model = NMF(n_components = topic_count, init = "random", random_state = 0)
model.fit(X)
```

Out[16]:
```
NMF(init='random', n_components=6, random_state=0)
```

The topics are stored in the `components_` attribute of the model.

In [17]:
```
model.components_
```

Out[17]:
```
array([[0.00000000e+00, 4.38327998e-04, 5.87630311e-01, ...,
        5.69921708e-03, 0.00000000e+00, 8.12735934e-03],
       [2.57911015e-01, 0.00000000e+00, 1.17641114e-01, ...,
        5.35679973e-02, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        4.23148239e-02, 0.00000000e+00, 2.47897842e-03],
       [0.00000000e+00, 8.23434995e-03, 0.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 8.50726610e-03],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        4.49517105e-02, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 1.38439934e+00, 1.70545305e+00, ...,
        1.10542798e-01, 4.20720050e-01, 4.81282563e-01]])
```

In [18]:
```
model.components_.shape
```

Out[18]:
```
(6, 10691)
```

Each row is a different component, 0,1,2,3,. Each component can be thought of as a collection of weights for each word.

The most important words in each component can be found by identifying the words with the highest weights within that component. This can be accomplished using the `np.argsort()` function, which indicates the entries of an array that are the largest, second largest, and so on.

In [19]:
```
orders = np.argsort(model.components_, axis = 1)
orders
```

Out[19]:
```
array([[    0,  5743,  5742, ...,  1744,  1742, 10300],
       [ 5345,  5723,  5722, ..., 10424,  6406,  1594],
       [    0,  5895,  5894, ..., 10393,  4669,  5281],
       [    0,  6253,  6252, ..., 10333,  5770,  5767],
       [    0,  6187,  6186, ...,  9327,  7513,   666],
       [    0,  4577,  4573, ...,   897,  3773,  6118]])
```

`numpy` indexing is used to arrange the words in the required orders.

In [20]:
```
important_words = np.array(X.columns)[orders]
important_words
```

Out[20]:
```
array([['abalone', 'manuscript', 'manufactures', ..., 'comets', 'comet',
        'venus'],
       ['konigsberg', 'maneuver', 'manetho', ..., 'war', 'nuclear',
        'civilization'],
       ['abalone', 'messages', 'message', ..., 'voyager', 'huygens',
        'jupiter'],
       ['abalone', 'navigators', 'navigations', ..., 'viking', 'martian',
        'mars'],
       ['abalone', 'mutating', 'mustering', ..., 'supernova', 'protons',
        'atoms'],
       ['abalone', 'honors', 'hominid', ..., 'believed', 'flame',
        'motion']], dtype=object)
```

In [21]:
```
def top_words(X, model, component, num_words):
    orders = np.argsort(model.components_, axis = 1)
    important_words = np.array(X.columns)[orders]

    return important_words[component][-num_words:]
```

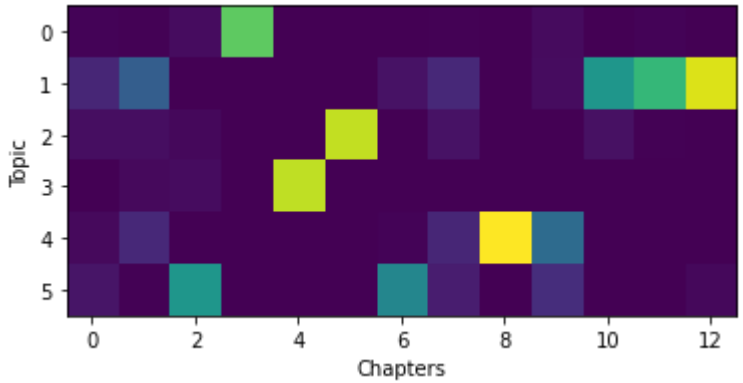
Assigning topics to each document based on weights is an important aspect of topic modeling. This can be done using the `transform()` method of the model.

In [22]:
```
weights = model.transform(X)
weights[:3]
```

Out[22]:
```
array([[0.01850533, 0.22878405, 0.07659417, 0.        , 0.05508537,
        0.119826  ],
       [0.00883732, 0.63032278, 0.07774356, 0.05489606, 0.2476392 ,
        0.01076627],
       [0.06640471, 0.        , 0.04856012, 0.06798911, 0.        ,
        1.11583166]])
```

In [23]:
```
fig,ax = plt.subplots(1)
ax.imshow(weights.T)
ax.set(xlabel = "Chapters",
       ylabel = "Topic")
```

Out[23]: [Text(0.5, 0, 'Chapters'), Text(0, 0.5, 'Topic')]

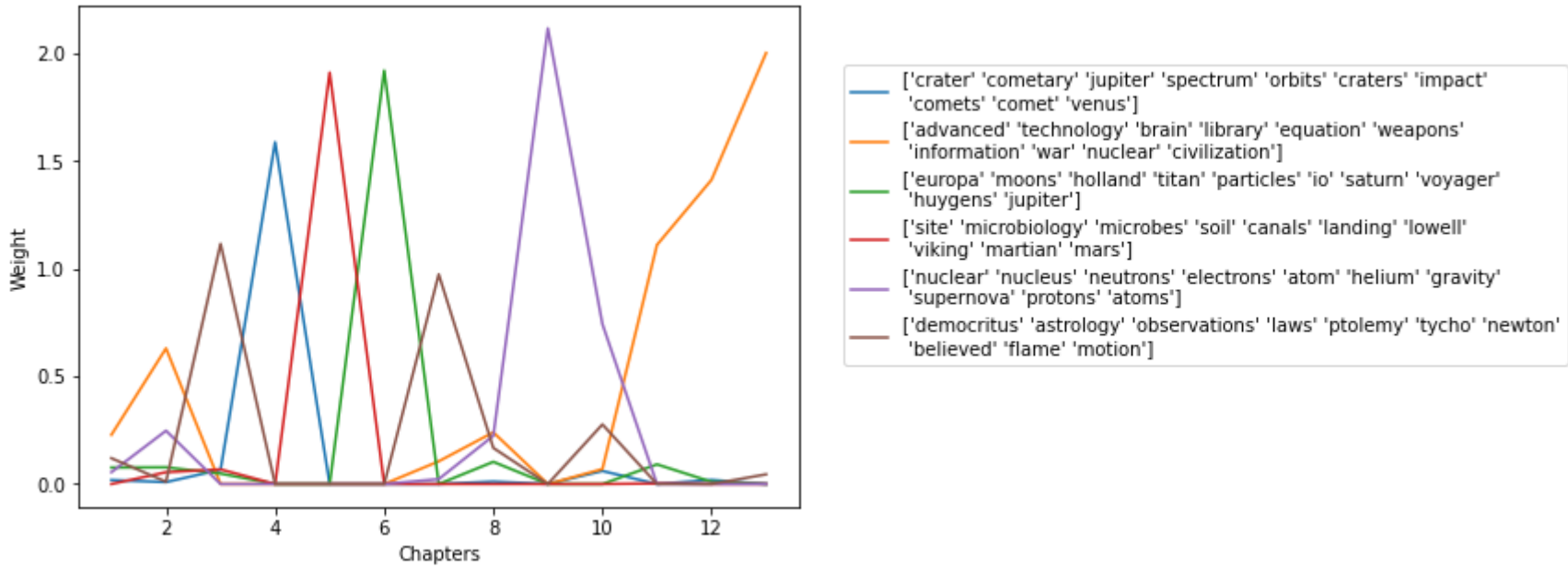


The weights indicate the relative presence of each topic in each chapter; for instance, Topics 2 and 3 are highly present in chapters 4 and 5, and topic 1 becomes increasingly more presenent throughout chapters 10 through 12.

This can also be visualized as a line chart. Labels featuring some of the top words for each topic will be added:

In [24]:
```
fig, ax = plt.subplots(1, figsize = (7,5))

for i in range(topic_count):
    ax.plot(df['Chapters'], weights[:,i], label = top_words(X,model,i,10))

ax.set(xlabel = "Chapters",
       ylabel = "Weight")
ax.legend(bbox_to_anchor=(1.05,.9),loc='upper left')
```

Out[24]: <matplotlib.legend.Legend at 0x7faf475d0100>


Weight
Chapters
0.0
0.5
1.0
1.5
2.0
2
4
6
8
10
12
['crater' 'cometary' 'jupiter' 'spectrum' 'orbits' 'craters' 'impact' 'comets' 'comet' 'venus']
['advanced' 'technology' 'brain' 'library' 'equation' 'weapons' 'information' 'war' 'nuclear' 'civilization']
['europa' 'moons' 'holland' 'titan' 'particles' 'io' 'saturn' 'voyager' 'huygens' 'jupiter']
['site' 'microbiology' 'microbes' 'soil' 'canals' 'landing' 'lowell' 'viking' 'martian' 'mars']
['nuclear' 'nucleus' 'neutrons' 'electrons' 'atom' 'helium' 'gravity' 'supernova' 'protons' 'atoms']
['democritus' 'astrology' 'observations' 'laws' 'ptolemy' 'tycho' 'newton' 'believed' 'flame' 'motion']