CS-410 Text Information System

Book Recommendation System

Team: Voltron

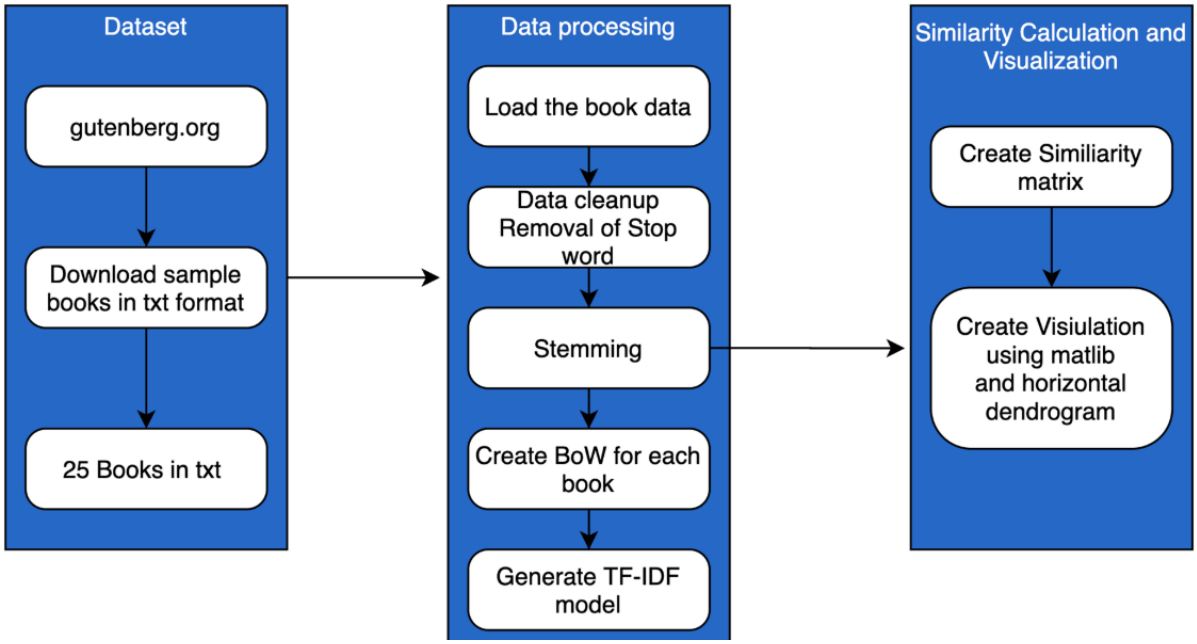
Team Member	Email	Captain
Adeeti Kaushal	*adeetik2@illinois.edu*	Yes
Vivek Bansal	*vivekb3@illinois.edu*	

Table of Contents

Introduction

The Book recommendation system aims to provide a selection to user based on the user's taste. Generally, a system would rely on user's metadata (for ex. Author of the book, theme etc.) to determine which book would user enjoy the most. However, when we take the same approach with full text search or full content, it becomes a very heavy dataset. In this project, we will try to demonstrate the same by providing recommendation based on the content.

Flow Diagram



Above picture shows the entire flow of the project. In dataset layer, we first downloaded the books (public) in text form from glutenberg.org. For our sample, we took 25 books and processed.

In the data processing layer, we loaded the books as text and titles and performed clean up. We went through removal of stop words, grouped them together (created stem). Using the stem words and dictionary, BagofWords were created for each book. Finally, in data processing layer, we built the inverted Index model.

In the end, we did the analysis using similarity calculations by creating the matrix and using the matlib, created the visualization and horizontal dendrogram.

Overview of Tasks

Following tasks were performed to complete this project.

Research/Build dataset
Code for tokenization, Stemming on corpus
Build bag of words, find stop words
Build tf-idf
Build Similarity matrix
Full end to end integration, tuning
Testing
Visual representation of results (matplotlib)
Other use case – Content based Similarity
Report

In this report, we will walk through the code step by step and also showcase the output of each step.

About the Dataset

The dataset is collection of books that is manually downloaded from Project Gutenberg. For this project, around 20 books were downloaded and used to find the content by searching the Text using similarity matrix.

Code

The code is written in Jupyter notebook and python3. Following are the 2 components:

Recommender.ipynb - The ipynb is iPythonNotebook file (Jupyter file) that contains the code. In the step 1, we will start with setting up the search criteria. The criteria for searching book of interest is set here.

Library – Downloaded data from Project Gutenberg

SETUP

Install the following tools/lang:

- Jupyter
- Python 3

Install following python/machine learning libraries

- Glob: This is used fir filename and pattern matching.
- Re: This is used for regular expression matching.
- Nltk: Natural Language toolkit
- Os: IT consists of functions interacting with operating system
- Genism: It is the natural language processing library used for unsupervised topic modeling
- Pandas: most important library used by data scientist for data analysis.
- Matplotlib: Used for visualization
- Scipy: used for numerical integration and optimization.

Step by Step Code details

SET SEARCH CRITERIA:

bookInterstedIn = RelativityandGravitation

LOAD DATASET AND SORT

- Download the dataset and store it in a folder "library".
- Read the .txt files from the library folder and load it into the memory using glob library.
- Sort the files using sort().

```
# The folder created below
    folder = "library/"

# List all the .txt files
    files = glob.glob(folder + "*.txt")
```

Following **Output** was obtained after reading the files from the "library" folder

```
['library/Relativity.txt',
 'library/ExperimentalMechanics.txt',
 'library/ThePoetryofScience.txt',
 'library/TheEinsteinTheoryofRelativityAConciseStatement.txt',
 'library/TheGravityBusiness.txt',
 'library/FromNewtontoEinstein.txt',
 'library/The BoyPlaybookofScience.txt',
 'library/SidelightsonRelativity.txt',
 'library/RelativityTheSpecialandGeneralTheory.txt',
 'library/TheEinsteinSeeSaw.txt',
 'library/AetherandGravitation.txt',
 'library/The EarthBeginning.txt',
 'library/specialtheoryRelativity.txt',
```

```
 'library/TheTheoriesof DarwinandTheirRelationtoPhilosophyRelig
ionandMorality.txt',
 'library/RelativityandGravitation.txt',
 'library/ThoughtsonArt.txt',
 'library/EinsteinTheoriesofRelativityandGravitation.txt',
 'library/TheJuniorClassics.txt']
```

LOAD TITLES AND TEXT IN OBJECTS

- Next step requires converting the data as information instead of string. For that purpose, open the files and encode them with UTF-8 signature (utg-8-sig). When reading the file using utf-8-sig, it will treat BOM as file info.
- Further clean up the file and remove the non-alphanumeric characters.
- After reading the files, store the text and tiles of the books in two lists and save them as titles and txts.
- To remove the folder name and .txt extension from the file name, use the os.path.basename() and replace() functions.

```
#define objects to hold text and titles
content_txts = []
book_titles = []

#loop through each, read, encode, remove txt extension
for n in files:
    f = open(n, encoding='utf-8-sig')
    val = re.sub('[\W_]+', ' ', f.read())
    content_txts.append(val)
    book_titles.append(os.path.basename(n).replace(".txt", ""))

[len(t) for t in content_txts]
```

Here is the **Output** of above code:

```
[24297,
 519663,
 875023,
 59083,
 56364,
 169246,
 953601,
 69853,
 24297,
 69011,
 941247,
 600351,
 197572,
 669416,
 548935,
 273437,
 548935,
 716050]
```

LOAD INDEXES

- In the next step, we need to store the index of the interested title from the “titles” list to a variable “typeofBook”
- To verify, print the content of the “typeofBook” variable.

```
# the list contains all the book titles

for i in range(len(book_titles)):
    if(book_titles[i]==bookInterstedIn):
        typeOfBook = i

print(str(typeOfBook))
```

Output:
14

TOKENIZE THE CORPUS

Now that the information has been collected, we will tokenize the corpus and transform into list of individual words. This is important step as we will now perform following steps in this:

- To filter our words for processing, we need to define the stop words
- Use lower() method to convert the contents in “content_txts”
- Breakdown the lower case text into individual words and python provides a method “split()” for the same.
- Store the split word into another variable “txts_split”.
- Remote the list of stop words in “stoplist”.
- Store the resulting list into another variable “texts”
- Print first few tokens for the searched books.

```
# Define a list of stop words

stoplist = set('for w a of the and to in to be which some is at that we i who
whom show via may my our might as well project by gutenberg
ebook'.split())

# Convert the text to lower case
```

```
txts_lower_case = [txt.lower() for txt in content_txts]

# Transform the text into tokens
txts_split = [txt.split() for txt in txts_lower_case]

# Remove tokens which are part of the list of stop words
texts = [[word for word in txt if word not in stoplist] for txt in txts_split]

# Print the first 20 tokens
texts[typeOfBook][0:20]
```

The output containing 20 tokens is below:

```
['einstein',
 'theories',
 'relativity',
 'gravitation',
 'malcolm',
 'united',
 'states',
 'other',
 'parts',
 'world',
 'cost',
 'restrictions',
 'whatsoever',
 'away',
 'or',
 're',
 'under',
 'terms',
 'license',
 'included']
```

PICKLING THE TOKENIZED CORPUS

- In this step, we will generate a stem for each token.
- Further, we used the pickle library of python for serializing. Python pickle module is used for serializing and de-serializing a Python object structure. Pickling is a way to convert a python object (list, dict, etc.) into a character stream.

```
# Create an instance of a PorterStemmer object
porter = PorterStemmer()

# For each token of each text, we generated its stem
texts_stem = [[porter.stem(token) for token in text] for text in texts]

# Save to pickle file
pickle.dump( texts_stem, open( "library/porterstem.p", "wb" ) )
```

STEMMING OF TOKENIZED CORPUS

Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma. Generally, it is also a part of queries and Internet search engines. In our use case, the words related to the concept of selection would be gathered under the *select* stem.

As we are analyzing 25 full books, the stemming algorithm can take several minutes to run. We will then load the final results from a pickle file and review the method used to generate it.

```
# Load the stemmed tokens list from the pre generated pickle file
texts_stem = pickle.load(open("library/porterstem.p", "rb" ) )

# Print the 20 first stemmed tokens from
texts_stem[typeOfBook][0:20]
```

Output:

```
['einstein',
 'theori',
 'rel',
 'gravit',
 'malcolm',
 'unit',
 'state',
 'other',
 'part',
 'world',
 'cost',
 'restrict',
 'whatsoev',
 'away',
 'or',
 're',
 'under',
 'term',
 'licens',
 'includ']
```

BUILDING A BAG-OF-WORDS MODEL

Now, we need to build the model using the stemmed tokens, it will be used by algorithms in next part.

- We created a dictionary that contains universe of all words in our corpus of books.
- Then, using the stemmed tokens and the dictionary, we will create **bag-of-words models** (BoW) of each of our texts.
- The BoW models will represent our books as a list of all unique tokens they contain associated with their respective number of occurrences

```
# Create a dictionary from the stemmed tokens
```

```
dictionary = corpora.Dictionary(texts_stem)

# Create a bag-of-words model for each book, using the previously
generated dictionary
bows = [dictionary.doc2bow(text) for text in texts_stem]

# Print the first five elements using BoW model
bows[typeOfBook][0:5]
```

Output:

```
[(0, 1), (1, 1), (5, 51), (6, 5), (13, 3)]
```

VISUALIZE THE MOST COMMON WORDS

For better understanding and interpret the results returned by BoW model, there is a need for visualization. This will help understand which stemmed tokens are present in given book and how many occurrences are found.

To visualize the content, we need to transform the content into DataFrame using the libraries and display 10 most common stems for the book as searched book “search criteria”.

```
# Convert the BoW model into a DataFrame
df_bow_origin = pd.DataFrame(bows[typeOfBook])

# Add the column names to the DataFrame
df_bow_origin.columns = ["index", "occurrences"]

# Add a column containing the token corresponding to the dictionary
index
df_bow_origin["token"] = [dictionary[index] for index in
df_bow_origin["index"]]

# Sort the DataFrame by descending number of occurrences and print the
first 10 values
df_bow_origin.sort_values(by="occurrences", ascending=False).head(10)
```

Output:

	index	occurrences	token
85	105	656	are
417	483	651	not
575	649	614	space
614	693	606	time
437	504	592	or
1618	2712	537	observ
297	345	480	have
114	140	471	but
276	322	466	from
316	368	464	if

BUILD A TF-IDF MODEL

Next, we need to generate the TF-IDF (term frequency–inverse document frequency) model from BoW model using the library function gensim's (TfidfModel()). TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.

This model defines the importance of each word depending on how frequent it is in this text and how infrequent it is in all the other documents. As a result, a high tf-idf score for a word will indicate that this word is specific to this text.

We will compute the score and print the results from model.

```
# Generate the tf-idf model
model = TfidfModel(bows)

# Print  the model
model[bows[typeOfBook]]
```

Output:

```
[(0, 0.0009610037857046851),
 (1, 0.0009610037857046851),
 (13, 0.0009188839366695132),
 (25, 0.0006125892911130088),
 (29, 0.0009188839366695132),
 (48, 0.0018786203238710653),
 (60, 0.0009188839366695132),
 (74, 0.0009188839366695132),
 (108, 0.0025048270984947 54),
 (121, 0.0012251785822260176),
 (131, 0.0003062946455565044),
 (139, 0.0019220075714093702),
 (144, 0.0006125892911130088),
 (156, 0.0009188839366695132),
 (161, 0.0019220075714093702),
 (167, 0.0006262067746236885),
 (173, 0.0012524135492473 77),
 (179, 0.0012251785822260176),
 (183, 0.0026242773755418033),
```

RESULTS OF THE TF-IDF MODEL

In order to interpret the results of TF-IDF model, we will display the 10 most specific words for a book. In our case, we used "*Relativity*" book.

```
# Convert the tf-idf model into a DataFrame
df_tfidf = pd.DataFrame(model[bows[typeOfBook]])

# Name the columns of the DataFrame id and score
df_tfidf.columns=["id", "score"]

# Add the tokens corresponding to the numerical indices for better
readability
df_tfidf['token'] = [dictionary[i] for i in list(df_tfidf["id"])]

# Sort the DataFrame by descending tf-idf score and print the first 10
rows.
df_tfidf.sort_values(by="score", ascending=False).head(10)
```

Output:

	id	score	token
2490	8693	0.349934	coordin
50	277	0.293775	euclidean
47	260	0.257377	einstein
1910	5513	0.240128	geometri
1843	5187	0.186218	essay
45	229	0.162138	dimension
42	189	0.126821	continuum
1046	2712	0.125209	observ
3018	13392	0.125055	contest
1764	4746	0.108441	curvatur

COMPUTE DISTANCE BETWEEN TEXTS

Stemmed token that are specific to each book are returned by TF-IDF model. The topics defined on the book "Relativity" can be seen now (like, gravitation etc). With this, we have a model associating tokens to how specific they are to each book, we can measure how related to books are between each other.

```
#Compute similarity matrix
sims = similarities.MatrixSimilarity(model[bows])

# Transform results to DF
sim_df = pd.DataFrame(list(sims))

# Add book_titles of the books as columns and index of DF
sim_df.columns = book_titles
sim_df.index = book_titles

# Print matrix
sim_df
```

Output:

	Relativity	ExperimentalMechanics	ThePoetryofScience	TheEinsteinTheoryofRelativityAConciseStatement	TheG
Relativity	1.000000	0.008280	0.003962	0.053577	
ExperimentalMechanics	0.008280	1.000000	0.091683	0.055434	
ThePoetryofScience	0.003962	0.091683	1.000000	0.102918	
TheEinsteinTheoryofRelativityAConciseStatement	0.053577	0.055434	0.102918	1.000000	
TheGravityBusiness	0.000423	0.004846	0.017416	0.006886	
FromNewtontoEinstein	0.022279	0.078886	0.147916	0.453587	
The BoyPlaybookofScience	0.003726	0.200147	0.484822	0.074519	
SidelightsonRelativity	0.027098	0.023681	0.066549	0.159785	
RelativityTheSpecialandGeneralTheory	1.000000	0.008280	0.003962	0.053577	
TheEinsteinSeeSaw	0.001548	0.020880	0.064418	0.026472	
AetherandGravitation	0.001311	0.032216	0.098532	0.045713	
The EarthBeginning	0.002820	0.096258	0.241800	0.103897	
specialtheoryRelativity	0.056891	0.063486	0.073009	0.154376	
TheTheoriesof DarwinandTheirRelationtoPhilosophyReligionandMorality	0.002888	0.027802	0.126757	0.053501	
RelativityandGravitation	0.044164	0.088884	0.133446	0.344666	
ThoughtsonArt	0.003139	0.042188	0.121210	0.038498	
EinsteinTheoriesofRelativityandGravitation	0.044164	0.088884	0.133446	0.344666	
TheJuniorClassics	0.003569	0.044035	0.067292	0.036086	

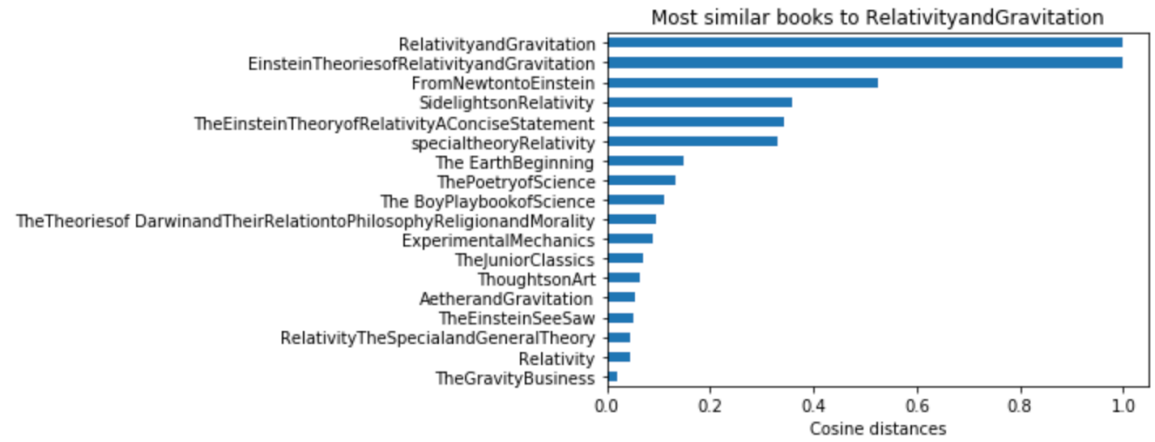
SIMILAR BOOKS

The output we have contains the matrix containing all the similarity measures between any pair of books from the library. This matrix will be useful to quickly extract the information like distance between two books or more. This is needed to display plots in a notebook

```
%matplotlib inline

# Select the column corresponding to
v = sim_df[bookInterstedIn]

# Sort by ascending scores
v_sorted = v.sort_values(ascending=True)

# Plot this data has a horizontal bar plot
v_sorted.plot.barh(x='lab', y='val', rot=0).plot()

# Modify the axes labels and plot title for better readability
plt.xlabel("Cosine distance")
plt.ylabel("")
plt.title("Most similar books to "+ bookInterstedIn)
```

Output

```
Text(0.5, 1.0, 'Most similar books to RelativityandGravitation')
```



BOOKS WITH SIMILAR CONTENT

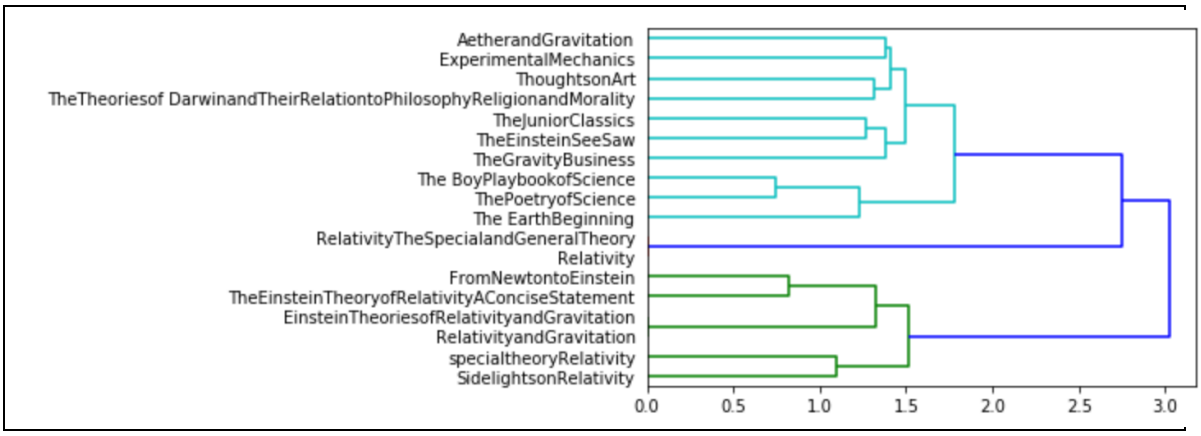
This project/approach is a good fit and of use, if we want to determine similar books that match user's interest.

For example, if user picked up the book "*Relativity*," user can read books discussing similar concepts such as "*Special Theory of Relativity*" or "*Relativity and Gravitation* If you are familiar with Einstein's work, these suggestions will likely seem natural to you.

However, we now want to have a better understanding of the big picture and see how Einstein's books are generally related to each other (in terms of topics discussed). To this purpose, we will represent the whole similarity matrix as a dendrogram, which is a standard tool to display such data. This last approach will display all the information about book similarities at once. For example, we can find a book's closest relative but, also, we can visualize which groups of books have similar topics

```
# Compute the similarity matrix the WVMA (Ward variance minimization algorithm)

Z = hierarchy.linkage(sim_df, 'ward')

# Display the results in horizontal dendrogram

a = hierarchy.dendrogram(Z,  leaf_font_size=10, labels=sim_df.index,  orientation="right")
```

Output:


AetherandGravitation
ExperimentalMechanics
ThoughtsonArt
TheTheoriesof DarwinandTheirRelationtoPhilosophyReligionandMorality
TheJuniorClassics
TheEinsteinSeeSaw
TheGravityBusiness
The BoyPlaybookofScience
ThePoetryofScience
The EarthBeginning
RelativityTheSpecialandGeneralTheory
Relativity
FromNewtontoEinstein
TheEinsteinTheoryofRelativityAConciseStatement
EinsteinTheoriesofRelativityandGravitation
RelativityandGravitation
specialtheoryRelativity
SidelightsonRelativity
0.0
0.5
1.0
1.5
2.0
2.5
3.0