

---

## CS-410 Text Information System

Fall 2020

---

### *Book Recommendation System*



### Team: Voltron

Team Member	Email	Captain
Adeeti Kaushal	<a href="mailto:adeetik2@illinois.edu">adeetik2@illinois.edu</a>	Yes
Vivek Bansal	<a href="mailto:vivekb3@illinois.edu">vivekb3@illinois.edu</a>	

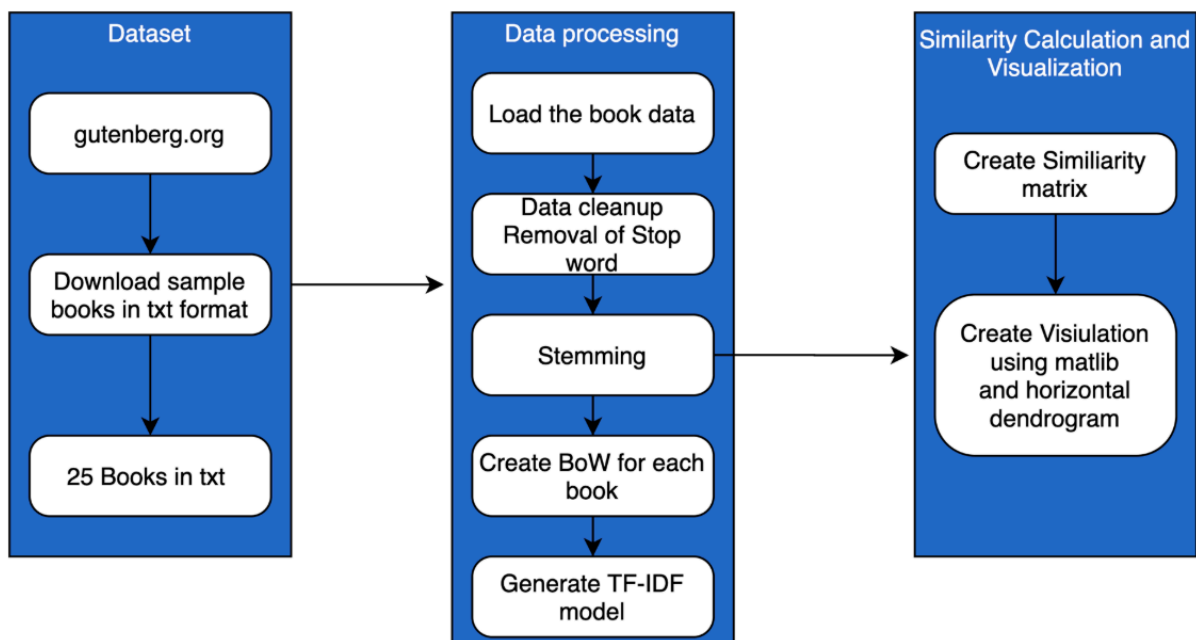
## Table of Contents

<b>INTRODUCTION</b> .....	3
<b>FLOW DIAGRAM</b> .....	3
<b>OVERVIEW OF TASKS</b> .....	4
<b>ABOUT THE DATASET</b> .....	4
<b>CODE</b> .....	4
<b>SETUP</b> .....	4
<b>Step by Step Code details</b> .....	5
<b>SET SEARCH CRITERIA:</b> .....	5
<b>LOAD DATASET AND SORT</b> .....	5
<b>LOAD TITLES AND TEXT IN OBJECTS</b> .....	6
<b>LOAD INDEXES</b> .....	7
<b>TOKENIZE THE CORPUS</b> .....	8
<b>PICKLING THE TOKENIZED CORPUS</b> .....	9
<b>STEMMING OF TOKENIZED CORPUS</b> .....	9
<b>BUILDING A BAG-OF-WORDS MODEL</b> .....	10
<b>VISUALIZE THE MOST COMMON WORDS</b> .....	11
<b>BUILD A TF-IDF MODEL</b> .....	12
<b>RESULTS OF THE TF-IDF MODEL</b> .....	13
<b>COMPUTE DISTANCE BETWEEN TEXTS</b> .....	14
<b>SIMILAR BOOKS</b> .....	15
<b>BOOKS WITH SIMILAR CONTENT</b> .....	16

## 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 [gutenberg.org](http://gutenberg.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 matlab, 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 25 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.

Datasets – Downloaded data from Project Gutenberg

## SETUP

Install the following tools/lang:

- [Jupyter](#)
- Python 3

## Install following python/machine learning libraries

- [Glob](#): This is used for 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:

bookInterestedIn = Relativity

### LOAD DATASET AND SORT

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

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

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

#sort alphabetically
files.sort()
```

Following **Output** was obtained after reading the files from the “datasets” folder

```
['datasets/CoralReefs.txt',
 'datasets/DescentofMan.txt',
 'datasets/DifferentFormsofFlowers.txt',
 'datasets/EffectsCrossSelfFertilization.txt',
 'datasets/ExpressionofEmotionManAnimals.txt',
```

```
'datasets/FormationVegetableMould.txt',  
'datasets/FoundationsOriginofSpecies.txt',  
'datasets/GeologicalObservationsSouthAmerica.txt',  
'datasets/InsectivorousPlants.txt',  
'datasets/International May Day.txt',  
'datasets/LifeandLettersVoll1.txt',  
'datasets/LifeandLettersVol2.txt',  
'datasets/MonographCirripedia.txt',  
'datasets/MonographCirripediaVol2.txt',  
'datasets/MovementClimbingPlants.txt',  
'datasets/OriginofSpecies.txt',  
'datasets/PowerMovementPlants.txt',  
'datasets/RelativityandGravitation.txt',  
'datasets/ThoughtsonArt.txt',  
'datasets/VariationPlantsAnimalsDomestication.txt',  
'datasets/VolcanicIslands.txt',  
'datasets/VoyageBeagle.txt',  
'datasets/specialtheoryRelativity.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 (utf-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 titles 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:

```
[496068,  
 1776539,  
 617088,  
 913713,  
 624232,  
 335920,  
 523021,  
 797401,  
 901406,  
 68928,  
 1047518,  
 1010643,  
 767492,  
 1660866,  
 298319,  
 916267,  
 1093567,  
 548935,  
 273437,  
 1043499,  
 341447,  
 1149574,  
 197572]
```

## 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:**

17

## 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 gutenber
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:**

```
['relativity',
 'special',
 'general',
 'theory',
 'albert',
 'einstein',
 'translated',
 'robert',
 'lawson',
 'this',
```



```
'use',  
'anyone',  
'anywhere',  
'no',  
'cost',  
'with',  
'almost',  
'no',  
'restrictions',  
'whatsoever']
```

## 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( "datasets/text_stem.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 pregenerated pickle file  
texts_stem = pickle.load(open("datasets/text_stem.p", "rb" ) )  
  
# Print the 20 first stemmed tokens from  
texts_stem[typeOfBook][0:20]
```

## Output:

```
['rel',  
'special',  
'gener',  
'theori',  
'albert',  
'einstein',  
'translat',  
'robert',  
'lawson',  
'thi',  
'use',  
'anyon',  
'anywher',  
'no',  
'cost',  
'with',  
'almost',  
'no',  
'restrict',  
'whatsoev']
```

## 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
464	4013	88	work
298	2665	85	or
469	4047	70	you
706	18407	57	tm
2	5	51	1
416	3649	50	thi
460	4004	49	with
62	482	40	ani
357	3067	36	rel
668	18224	29	electron

## 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, 0.002504827098494754),
 (121, 0.0012251785822260176),
 (131, 0.0003062946455565044),
 (139, 0.0019220075714093702),
 (144, 0.0006125892911130088),
 (156, 0.0009188839366695132),
 (161, 0.0019220075714093702),
 (167, 0.0006262067746236885),
 (173, 0.001252413549247377),
 (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
445	18407	0.568685	tm
407	18224	0.289331	electron
403	18217	0.209516	donat
208	4047	0.173723	you
422	18294	0.169608	licens
398	18194	0.168483	copyright
377	15896	0.164619	pt
341	12619	0.158047	00
430	18329	0.139677	org
507	25767	0.137232	linda

## 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:

	CoralReefs	DescentofMan	DifferentFormsofFlowers	EffectsCrossSelfFertilization	ExpressionofEmotionManAnimals	Formation
CoralReefs	1.000000	0.014908	0.003737	0.003364	0.009012	
DescentofMan	0.014908	1.000000	0.090203	0.035275	0.144963	
DifferentFormsofFlowers	0.003737	0.090203	1.000000	0.488335	0.011225	
EffectsCrossSelfFertilization	0.003364	0.035275	0.488335	1.000000	0.009606	
ExpressionofEmotionManAnimals	0.009012	0.144963	0.011225	0.009606	1.000000	
FormationVegetableMould	0.033720	0.032641	0.016951	0.039802	0.030667	
FoundationsOriginofSpecies	0.033511	0.164525	0.074177	0.071711	0.065112	
GeologicalObservationsSouthAmerica	0.074636	0.017288	0.007343	0.005574	0.012180	
InsectivorousPlants	0.003697	0.013060	0.014787	0.014320	0.018646	
International May Day	0.001968	0.010270	0.002260	0.001712	0.011889	
LifeandLettersVol1	0.050843	0.074423	0.035711	0.038292	0.087648	
LifeandLettersVol2	0.027911	0.107696	0.080678	0.081071	0.069958	
MonographCirripedia	0.009756	0.080298	0.015736	0.006082	0.026079	
MonographCirripediaVol2	0.016391	0.055502	0.010221	0.005653	0.040816	
MovementClimbingPlants	0.002947	0.008642	0.019817	0.030464	0.008996	
OriginofSpecies	0.051610	0.297719	0.192191	0.195983	0.079755	
PowerMovementPlants	0.004479	0.014369	0.031004	0.056963	0.015683	
Relativity	0.001368	0.002044	0.000589	0.004034	0.003088	
RelativityandGravitation	0.008028	0.015651	0.005251	0.004542	0.019290	
ThoughtsonArt	0.004683	0.026837	0.004511	0.005550	0.048270	
VariationPlantsAnimalsDomestication	0.018364	0.246918	0.083459	0.084561	0.094771	
VolcanicIslands	0.075066	0.014910	0.006284	0.005080	0.011474	
VoyageBeagle	0.294442	0.133827	0.029688	0.033318	0.113205	
specialtheoryRelativity	0.004368	0.009911	0.004499	0.003204	0.013041	

24 rows x 24 columns

## 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

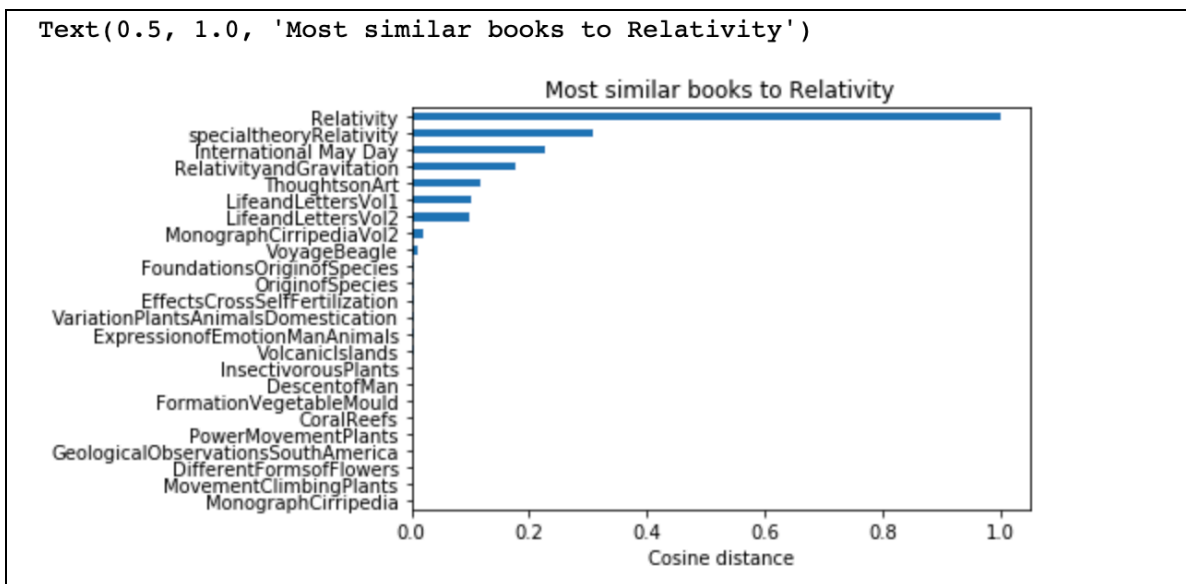
# Select the column corresponding to
v = sim_df[bookInterstedIn]

# 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



## 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=8, labels=sim_df.index, orientation="left")
```

**Output:**

