University of Essex
Department of Mathematical Sciences

MA981:DISSERTATION

CONTENT-BASED MOTIVATIONAL BOOK RECOMMENDATION USING NATURAL LANGUAGE PROCESSING AND CLUSTERING

Sudheena Misra KS
Register Number: 2010655
Word Count: 10,884

Supervisor: Yanchun Bao

November 26, 2021

Colchester

Contents

List of Figures

List of Tables

CHAPTER 1

Abstract

The objective of this paper is to build a content-based book recommendation for some motivational books with the help of Natural Language Processing (NLP) and Clustering. This paper explores whether there exists any similarity between the books I have selected based on the words in each book. This paper is not a typical recommendation system built based on existing customer ratings and recommends books to a new customer with the same patterns as the old customers. In contrast, this paper is purely focused on the text mining techniques and implemented using Natural Language Toolkit (NLTK) package. Books are recommended based on how similar the discussed contents are. In this project, the methods I have used are some of the most commonly used text mining techniques like Stemming, Bag of Words Model (BOW), Term Frequency - Inverse Document Frequency (TF-IDF) Model, etc. To compute the similarity of books, I performed Clustering to classify and group the books to find the closest relative of each book, and I have used the Cosine similarity distance. Then generated a distance matrix and compared the pair-wise similarity of each book with one another based on their cosine distance. Finally, visualize the results of Clustering with the help of a dendrogram.

Keywords: NLP, natural language processing, clustering, term frequency inverse document frequency , recommendation, books, text mining

CHAPTER 2

Introduction

Natural language processing (NLP) is a high-demand technology nowadays. Most of the top companies like Google, Amazon, Facebook, Twitter, etc., use NLP to improve their business to provide customer-specific recommendations. To extract the contents in a piece of texts plays a vital role in developing marketing strategies. As human beings, while reading a paper on social media content, we could easily understand the context of that texts. But for a business, it may not be a single paper or single piece of text they have to analyze, and they might have to go through thousands and thousands of texts per day to decide to expand their business. Going through all the texts manually and understanding the context, and making a decision is really a time-consuming and challenging task. So there must be a need to automate these tasks. The contribution of NLP is a milestone in these industries. The main applications of NLP include speech recognition systems, search engines, information retrieval, text classification, social media sentiment analysis, etc.

Wikipedia states that “Natural language processing is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data”[Wikipedia]. We know that the computers understand only the binary digits 0 and 1. NLP is a process of making the computer understand the human language and the emotions and sentiments in a piece of text. In this paper, I am attempting to analyze the texts of five motivational books that I downloaded from a website to explore whether there is any similar content in those books to find out whether they convey the same idea. I selected five books randomly, but they all belong to the same genre. They all are motivational books. From my analysis, if we found some books are closely related to each other or have some similar contents, then it is evident that we can recommend these books to the reader. I have implemented the standard text mining techniques in NLP for the preprocessing of these books to organize the contents of the books and extract the significance of the words used in those books then. I implemented Hierarchical clustering analysis employing Cosine similarity distance to group the books into groups. Then the result of this Clustering is evaluated with the help of a dendrogram.

In this paper, I am trying to implement a book recommendation system. There are other book recommendation systems are available based on the customer ratings towards a specific book. How these systems works are that they have the data of existing customers and their preferences. When a new customer arrives, the system will learn the pattern of the new customers and check whether their preferences also match with an old customer. If yes recommendation system will start to recommend those matched results. But in my paper, I am trying to suggest books based on the contents of the book. This project is wholly implemented in python utilizing a jupyter notebook.

The methodology of this paper is organized as follows. First is the data collection. In this part, I have explained how I chose the books and where I got those books. In the second part, I explained some preprocessing procedures I did to transform these books to perform the analysis. This stage describes the data life cycle, and this is a crucial step since our results depend on this step. Before going through any investigation, it is necessary to prepare the data for analysis. In this step, I have used some most common efforts in document analysis projects, such as converting the texts into lowercase, tokenization, removing stop words, stemming, generating a bag of words model and counting the most common words and, building a Term Frequency- Inverse Document Frequency model. Next part, I have explained how to calculate the similarity between the books using a similarity distance called Cosine similarity. I also explained why I chose this distance. The next part of the methodology focuses on how to group the books so that the books with similar contents will be in the same group. I perform this task using hierarchical Clustering. The result section of this paper mainly focuses on visualizing the output of the methodology sections.

CHAPTER 3

Literature Review

There are so many text mining and data mining researches are conducted in the past to extract the sentiments in the texts and classify the texts. Some similar works are described as follows. 1) “Document Clustering Using Concept Space and Cosine Similarity Measurement," in this paper, the researchers are extracting the pieces of information from 20 newsgroups; they have used document clustering to perform the data mining. They used cosine distance as their similarity matrix for the Clustering [17]. 2) “Similarity Measures for Text Document Clustering," In this paper, the researcher has used partitional Clustering for text documents. This study utilized the standard K-means algorithm, tested the results on seven document datasets, and implemented five similarity distance measures commonly used in text clustering[18].

The motivation behind my work is some guided projects I found in Datacamp. Datacamp is an online learning platform with lots of data science-related online certification courses, case studies, and guided projects. In these guided projects, they have a set of instructions and hints provided on how to do those projects. The code of this study is more similar to a guided project called "The book reccomention from charle's Darwin." There are some other guided projects on this site I found. Those are also implemented using natural language processing; they also give me motivation and ideas to understand the most widely used techniques in NLP. The projects such as "Hottest topic in machine learning," "A text analysis of Trump's tweets," "Real-time insight from social media data," etc., are also some other related works I found. These projects help me implement the codes, and for the writing of my findings, I mostly rely on google resources, google scholarly articles, and youtube videos.

CHAPTER 4

Methodology

4.1 Data collection

The first step I followed in conducting my analysis was data collection. I wanted to do text mining for some motivational books. When I searched the books online, I found most of the sites provide paid downloads only. So I searched about the free sites on YouTube to download books. I found the following site https://z-lib.org/ which provides free access to books. When we go through this link, we can find three other links, "Books," "Articles," and "sign in/up," on the page. For analyzing the text first, it is better to convert the books into .txt format. The books on this site, by default, are in .epub format; most of the system doesn't support this format. But this site has an option to convert to a .txt file and download if we sign up. So I signed up to this site first, went to the "Books" section, and searched the keyword "Motivation." The result shows lots of motivational books. I randomly selected five books from 5 different authors from the result page and downloaded them into .txt format for my analysis. The books I selected are

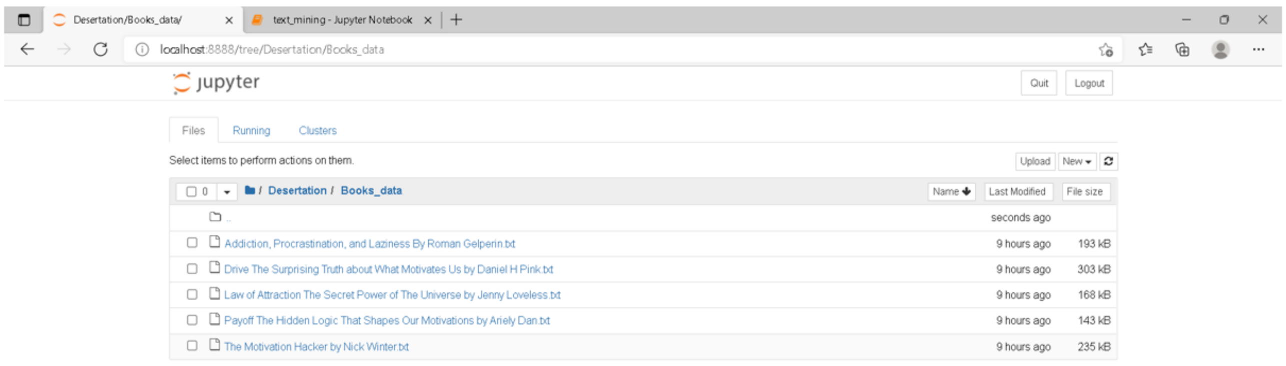
i) “Addiction, Procrastination, and Laziness A Proactive Guide to the Psychology of Motivation” by Roman Gelperin
ii) “Drive The Surprising Truth About What Motivates Us" by Daniel H. Pink
iii) “The Law of Attraction, The Secret Power Of The Universe” by Jenny Loveless
iv) “Payoff The Hidden Logic That Shapes Our Motivations” by Ariely Dan
v) “The Motivation Hacker by Nick Winter

After I downloaded these books, I went through all these text files quickly without reading just a quick scroll down; I found Some of the texts, in the beginning, stating the copyright permissions of each book and some acknowledgments and some recommendations at the end of each book. I think these texts are insignificant for my analysis, so I manually removed all those texts before I uploaded these .txt files into miniconda software for my analysis.

4.2 Data Cleaning and Pre-processing

As stated earlier, I will be using the Jupyter notebook in the miniconda environment to implement the code for this project. I have uploaded all my books in a folder named "Books_data" in the miniconda environment. Refer the fig1.

Fig 1. Books as .txt files



Select items to perform actions on them.

/ Desertation / Books_data

Name	Last Modified	File size
..	seconds ago	
Addiction, Procrastination, and Laziness By Roman Gelperin.txt	9 hours ago	193 kB
Drive The Surprising Truth about What Motivates Us by Daniel H Pink.txt	9 hours ago	303 kB
Law of Attraction The Secret Power of The Universe by Jenny Loveless.txt	9 hours ago	168 kB
Payoff The Hidden Logic That Shapes Our Motivations by Ariely Dan.txt	9 hours ago	143 kB
The Motivation Hacker by Nick Winter.txt	9 hours ago	235 kB

Python provides a built-in module called "glob" to retrieve all the files contained in a folder that matches a specific pattern. Since this module is built-in, there is no requirement for it to install externally. Here, in this case, all of my files are in .txt format, so I used the pattern "*.txt" to retrieve all the files contained in the folder "Books_data" and sort them alphabetically according to the file name[1]. When we go through the text, we can find so many non-alphanumeric characters, and those include all the characters other than alphabets and numbers. The punctuation characters such as an exclamation mark(!), commas(,), question mark(?), colon(:), semicolon(;), etc. and the special characters such as dollar sign($), at symbol(@), plus sign(+), etc. are examples of non-alphanumeric characters[2]. To make the texts clean, it is required to remove all these characters present in the text. I used the regular expression(re) library for this purpose. Regular expressions are commonly known as Regex, are used to detect a specific string. In other words, it is a pattern that we can use to identify certain characters or words inside a collection of words or texts.

To facilitate preprocessing, it is required to load the contents of these books into python; for that, I have created two list objects; one is to store all the texts of the five books and the other for storing titles of all the books. They are named "books" and "titles," respectively. So the "books" list contains all the texts in 5 books in such a way that the first index contains the first book, the second index contains the second book, and so on. Since the list “books” is a collection of words, it is called a Corpus in natural language processing. Likewise, the "titles" list contains all the titles of five books, with the first index containing the name of the first book and the second index, the title of the second book, and so on.

I have computed the count of all the words and characters contained in the texts in each book separately. From table 1, we can analyze how many words and characters are present in each book.

Table 1. Total number of characters(letters) and words present in each book

Books	**Number of words**	**Number of characters(letters)**
Addiction, Procrastination, and Laziness	33069	187725
Drive, The Surprising Truth About What Motivates	51310	294961
Law of Attraction, The Secret Power of The Universe	31692	161866
The payoff, The Hidden Logic That Shapes Our Motivations	24643	136953
The Motivation Hacker	42668	224212

Fig 2. Bar plot showing Word Count of each book

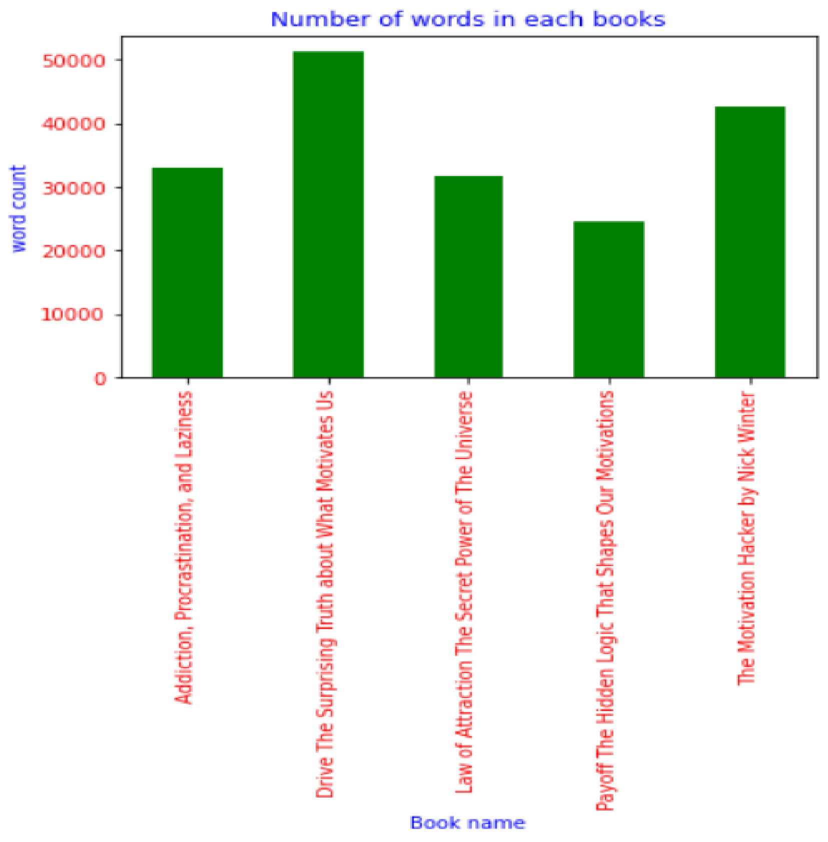


Fig 2. Shows the number of words present in each book. From this bar plot, we can see that the book "Drive, The surprising truth about what motivate us" has more words, and the book "Payoff The Hidden Logic That Shapes Our Motivations" is the shortest book among these five books.

4.3 Text Mining

I started my text mining by converting all the texts into lowercase. The remaining steps I followed are listed below.

4.3.1 Tokenization

Tokenization is the process of splitting the texts into individual tokens. These tokens can be a sentence, word, or character. That means each line of the text either contains one sentence if it is a sentence token, or each line contains only one word if it is a word token or each line contains only one character if it is a character token. If we want to do sentence tokens, we can split the texts with respect to a full stop("."). So that each line contains one sentence, or if we want to do the words token, we can split with respect to "space" So that each line contains only one word. Likewise, we can split the text so that each line contains only one letter, which is called character tokens. In most natural language applications, the word tokens are considered most valuable because we could explore and extract each word's sentiment individually. But every token is beneficial depending on the tasks we want to perform with the texts. For my analysis, I am converting all these texts into words tokens. Tokenization also removes the punctuations in the texts.

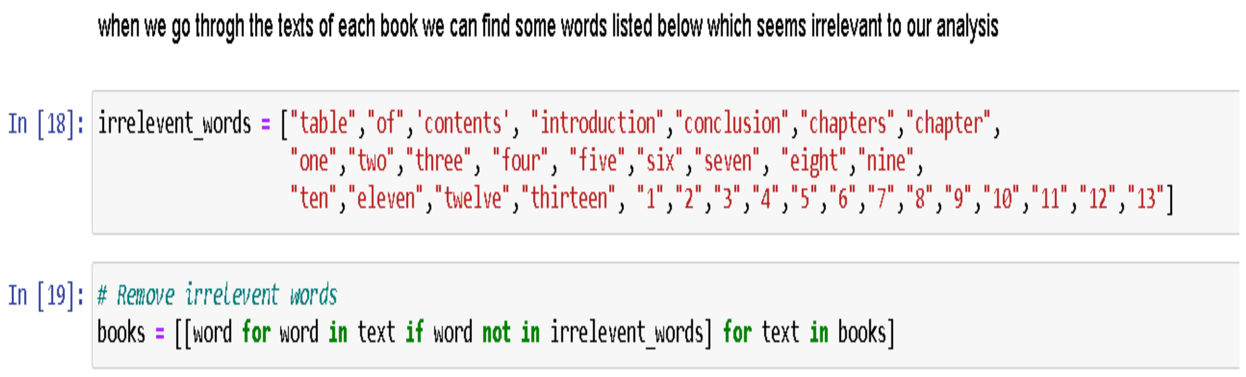
4.3.2 Defining and Removing Stop Words

If we go through any piece of text, these texts can be from any books, articles, opinions, conversations, and whatnot, we could find some of the words dominate in every text. The words such as "the," "a," "as," "which," "has," etc., are the most common words we could find in every text. The collection of these words are called stop words. Since these words don't add much significance to the context of our text analysis, it is recommended to remove all these words before the analysis, which will reduce the size of the texts and also facilitate less running time, thereby enhancing the quality of the code. By removing the stop words, we are helping our analysis focus on the significant pieces of information rather than the low-level pieces of information. So our analysis will be in point to point rather than including all the useless pieces of information, thereby facilitating the model training easier.[3]

To remove these stop words, I have searched for the lists of stop words in google. I found a couple of links. Some of them contain more number stop words, and others contain less number of stop words; I choose the one with more number stop words. I copied all these stop words and created a list of stop words in python. I named this variable stop_words.

Since I already split the texts into word tokens in the previous step, finding and removing the stop words will be easy. With the help of the list comprehension function, I created a loop that will find and remove the words which match the words in the 'stop_words' variable and the book texts. Now the stop words are removed, So to verify my function works well, I printed the first 15 word tokens of each book separately. When I get to see the results which facilitates me to dig into my lists more deeply. I found some of the words such as "table", "contents", "introduction", "chapter" etc. Then I go through my text files again, noticing some other insignificant words such as "conclusion" and the chapter numbers. So I realized before going further it is necessary to remove all these words too. Since I am analyzing five books, I am not planning to analyze each book by chapter wise it will be time-consuming, and I did a previous work in R where I analyzed two books by its chapter-wise also I am not planning to repeat those, So I decide to remove all the chapter specifications in each book. I found each book has a different number of chapters, and one book doesn't have any chapters, and one of the books contains 13 chapters. That book has the maximum number of chapters. All the other books contain chapters less than thirteen. I also notice that in most of the books, the chapter number is denoted numerically(e.g., "chapter 1"), but in some others, the chapter number is denoted alphabetically(e.g., "chapter one”). So I created a list of unnecessary words like I created the stop word list to match and remove these words from my book's texts. Refer the fig3. To see how they look alike.

Fig 3. Removing irrelevant words



when we go throgh the texts of each book we can find some words listed below which seems irrelevant to our analysis

```
In [18]: irrelevent_words = ["table","of",'contents', "introduction","conclusion","chapters","chapter",
                              "one","two","three", "four", "five","six","seven", "eight","nine",
                              "ten","eleven","twelve","thirteen", "1","2","3","4","5","6","7","8","9","10","11","12","13"]
```

```
In [19]: # Remove irrelevent words
         books = [[word for word in text if word not in irrelevent_words] for text in books]
```

4.3.3 Stemming

Stemming is transforming a word to its base or root form called a "stem" by reducing the inflections in those words. Wikipedia states the definition of inflection as, “In linguistic morphology, inflection (or inflexion) is a process of word formation, in which a word is modified to express different grammatical categories such as tense , case, voice person, number, gender, mood, animacy, and definiteness. An inflection expresses grammatical categories with affixation (such as prefix, suffix, infix, circumfix, and transfix), apophony ((as Indo-European ablaut), or other modifications”[4].

Stemming is the most common practice of text mining in natural language processing, search engines, and information retrieval applications. This algorithm unifies the different forms of a single word to its root form. The authors sometimes use different words to explain the same concept. So identifying these words and converting them to their base form plays a vital role in text mining.

There are three main types of stemming algorithms. They are truncating methods (affix removal), statistical methods, and mixed methods. All these three algorithms have several sub-categories as well. Since this is a vast topic to explore, I will only address that one stemming algorithm that I will be using in this project, the Porters Stemmer algorithm.[5]

Porters Stemmer algorithm is a popular stemming algorithm with less error rate and produces the best output compared to other stemmers. This belongs to the sub-category of truncating methods or affix removal. Let's understand the function of this algorithm with the help of a diagram.

Fig 4. Porters Stemmer

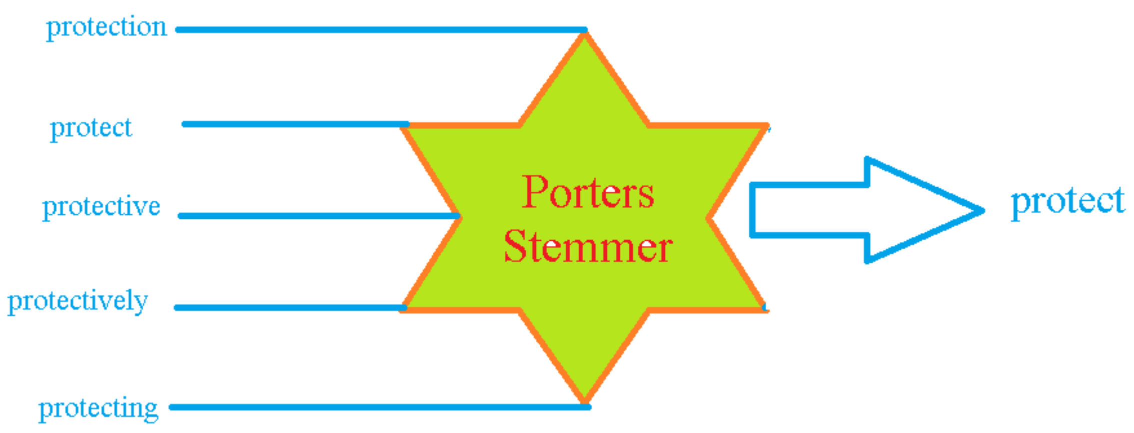


Fig 4 explains how this algorithm works. Consider the example in the figure, suppose we have a piece of texts with some sentences and some of these sentences the word "protect" is used in different forms such as "protection," "protect," "protective," "protectively," and "protecting" and we know that all these five words have the same meaning even though it is in different forms. So if we input our texts into our porters' stemmer algorithm, it will remove its affixes and transform all these words to its base form "protect." Here the word "protect" is our stem, and this process is called stemming.

I have passed all my book texts as an input of this stemmer and obtained the results (refer to Fig 5.). We need to do stemming because, for the upcoming step, we will be counting the most common words in the texts. Hence, we see earlier that the word “protect” can be represented in 5 forms if we don't do stemming when we count the repetencies of each word, all these words will be treated as different words and counted separately, but after stem, it will be treated as a single word that repeats five times(in this case).

4.3.4 Creating a Bag of Words (BOW) Model

Fortunately, python provides a most helpful library to deal with the texts. This library is not other than the "gensim." Gensim is a widely used Python library in Natural Language Processing for topic modeling, document indexing, and similarity retrieval [7]. Let's understand some standard terms in gensim such as "document," "corpus," "vector," and "model." According to gensim, a "document" is any piece of text; it can be a sentence, an article, a short paragraph, a tweet, or a book. "A corpus is a collection of documents" [6]. The texts or the document can be represented mathematically to perform the text mining called the Vectors. A model is an algorithm that can transform the vectors from one form of representation to another [6].

So next step is to build a bag of words model. A bag of words model consists of the collection of words in a document regardless of its order and grammar, like a key-value pair, where each word is a key, and its value is the number of occurrences of that word in the whole document(texts).

Let’s consider an example;

Document = “I started to realize my power. My power is endless”

Bag of words model = {“I”:1, “started”:1, “to”:1, “realize”:1, “my”:2, “power”:2, “is”:1, “endless”:1 }

From the above example, it has been clear that how a bag of words model works. If we input a document like an example above that has the words "my" and "power" repeated two times and the remaining only appear once so BOW creates a key-value pair as shown. But there is a slightly different way it is represented in python; that is, each of these words has a unique id, and this unique id is used in place of the words as a key, and its value remains the same.

4.3.5 Generating a Term Frequency - Inverse Document Frequency (TF- IDF) model

In the previous step, we have counted the most common words in each book. Even though some words occur more times in a book or document, it doesn't mean that is the word that is more significant in that book. If we shouldn't have removed the stop words, they might have been repeated in every book and may provide us with the wrong results. Even though we removed the stop words, we couldn't say that we obtained the most important words. Term Frequency – Inverse Document Frequency(TF-IDF) algorithm helps us identify the most important words of a particular document by comparing that specific words in all other documents. This algorithm is advantageous in search engines, classifying texts, and finding keywords. This significant word may be the keyword for that particular book. Suppose if we searched a word in google, for example, how the search algorithm work is that they will return the results of the documents which have the high TF-IDF scores for that particular word. So what I am going to do is calculate the TF-IDF scores of each word in each book. Before that, I will explain this algorithm in more detail.
Mathematically, TF-IDF scores are calculated as follows [13];
TF_IDF = TF * IDF
Where TF is the Term Frequency and IDF is the inverse document frequency. Let’s consider three documents d1, d2, and d3, in a corpus (as we know, the corpus is a collection of documents, and the document is a collection of words) for an example.
d1 = “Life will change when you change."
d2 = “If you can dream it you can achieve it."
d3 = “Life, life , life everyone have only one life.”

Let's consider the word "life" in the three documents and calculate its term frequency, inverse document frequency, and TF-IDF scores.

Term Frequency(TF)

Term frequency is calculated per document. ‘Term frequency is defined as the number of times a word or term appears in a document[10]. The word “life” has appeared once in d1, zero times in d2, and four times in d3. So the term frequency is as follows [11].

TF(“life) of d1 = 1
TF(“life”) of d2 = 0
TF(“life”) of d3 = 4

Inverse Document Frequency (IDF)
IDF is calculated as follows;

$$\text{IDF} = log\left(\frac{\text{Total number of documents}}{\text{number of documents in which the word is present}}\right)$$

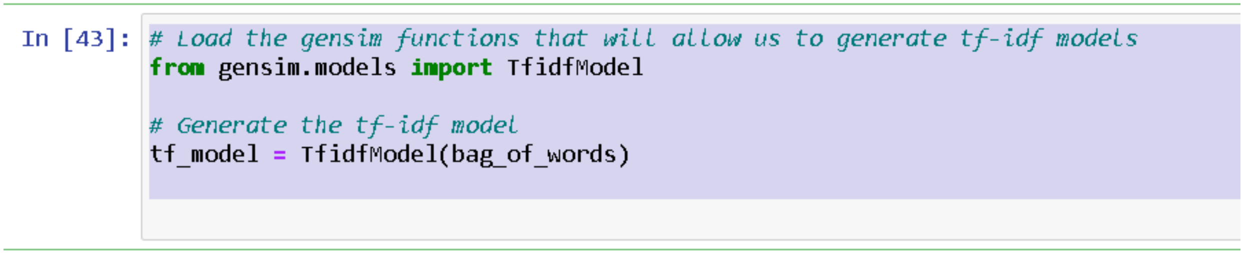
Here we have three documents in our corpus and among which 2 of the documents contain the word "life."
IDF(“life”) of d1 = log (3/2) = 0.176
IDF(“life”) of d2 = log (3/2) = 0.176
IDF(“life”) of d3 = log (3/2) = 0.176
From the above example, we can see that the IDF is constant per corpus.

Calculation of TF_IDF (“life”) [12]
TF-IDF = TF * IDF
TF_IDF (“life”) of d1 = 1 * 0.176 = 0.176
TF_IDF (“life”) of d2 = 0 * 0.176 = 0
TF_IDF (“life”) of d3 = 4 * 0.176 =0.704

The above calculation shows that document 3 has the highest TF-IDF scores, which suggests that the word "life" is more significant to d3 than d1.

So we now understand how this algorithm works. In this project, I have my five books as documents, and the collection of all these books is the corpus here. In the previous step, I have my bag of words model, which is stored in the variable 'bag_of_words". This variable is a 2-dimensional list object with five indexes. Its first index contains my first book with its unique words and frequencies, the second index contains the second book, and so on. Gensim library has the inbuilt function to perform this algorithm. We just need to pass our variable into it, as shown in the figure below.

Fig. 5 TF-IDF model



```
In [43]: # Load the gensim functions that will allow us to generate tf-idf models
         from gensim.models import TfidfModel

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

4.4 Measuring the similarity between the texts in books

Now we have our TF-IDF model results which give the TF-IDF scores of the most specific stemmed words in each book. The next step is to measure the distance between the texts or, in other words finding out how similar each book is. In order to perform this, we need to select the best choice of similarity. There are lots of options available when considering measuring the similarity, but all those may not lead to accurate results since all similarity measures available serve a different purpose and are only applicable in specific scenarios. Let's consider the three most popular similarity distance measures in machine learning, and they are Manhattan distance, Euclidean distance, and Cosine Similarity. Before deciding how to choose the similarity measure, it’s important to know how we can represent the documents(books) and words mathematically to calculate it.

In NLP, the documents are represented in multi-dimensional space as vectors where the words are their coordinates, and they will decide where to place these documents in space depending on some features of these words in each document (In most scenarios, these features may be the number of occurrences of these words and the TF-IDF scores). Suppose if we have two documents that consist of only two words, it can be visualized in a two-dimensional space or if we have more than two documents, say 100 documents, but if each document contains only two words, still it can be represented in 2-dimensional space. But if the words are 3 in each document, they can be represented in 3-dimensional space. Here the dimension of the space where the documents are placed will be determined by the unique words in each document, not the number of documents we have. We can visualize up to 3-dimensional space, i.e., up to 3 words. But we all know that might not be the ideal case. Every document that can be any piece of text like books, articles, essays, etc., contains more words, most probably more than 100 words. We couldn't even imagine a 100-

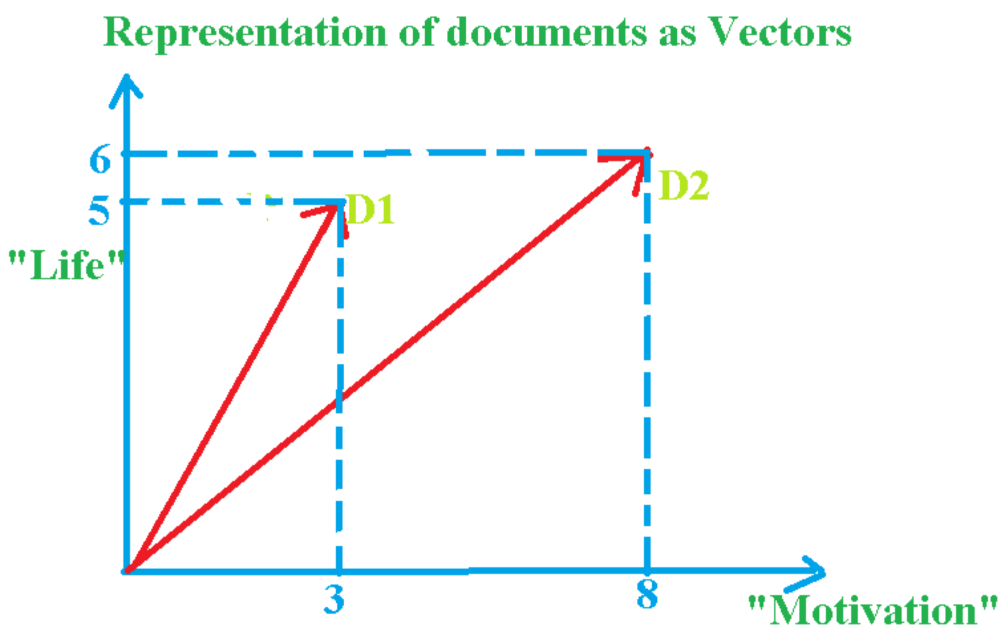
dimensional space and visualize it on paper. But still, we can employ the same methods and formulas we used in 2-D space or 3-D space to measure the distance and orientation of these documents in any multi-dimensional space. Let's see an example of how two documents that contain only two words can be visualized in a two-dimensional space.

For example, consider two documents, D1 and D2, where these two documents contains only two words, and they have the same words but the number of occurrences of each word are different in 2 documents.

D1 = { “Life” : 5, “Motivation”: 3}
D2 = {“Life” : 6, “Motivation”: 8 }

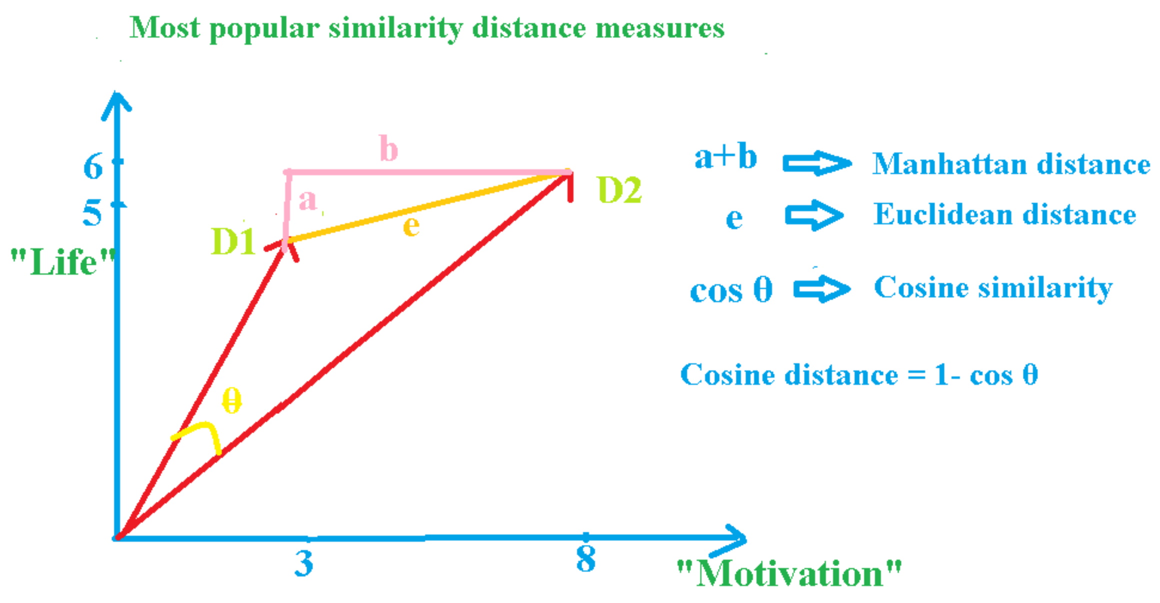
Here the word "life" has occurred five times in D1 and six times in D2, whereas the word "motivation" has occurred three times in D1 and eight times in D2. In the fig below, we can see that I represented these two documents in a 2-dimensional plane with the number of occurrences of the words "life" in the Y-axis and the number of occurrences of "Motivation" in the X-axis. So we can represents the documents D1 and D2 as vectors mathematically in terms of (x,y) coordinates as D1(x,y) = (3,5) and D2(x,y) = (8,6)

Fig 6 Documents in the 2-D plane as Vectors



Now we understand how the documents are represented in space. The next question will be how we calculate the similarity between the documents. As at the beginning of this section, I have mentioned about three popular distance measures. Let's understand these distance measures with the help of a diagram.

Fig 7. Three Popular similarity distance measures


Most popular similarity distance measures
6
5
"Life"
b
a
D1
e
D2
θ
3
8
"Motivation"
a+b ⇨ Manhattan distance
e ⇨ Euclidean distance
cos θ ⇨ Cosine similarity
Cosine distance = 1- cos θ

From the fig. 6, we need to calculate the distance between the document vectors D1 and D2, and we can calculate these distances in three ways; the First one is the Manhattan distance, the second Euclidean distance, and the third Cosine distance.

Manhattan distance: This distance is also called city block distance as it is calculated by means of horizontal and vertical distances from each document vector.

As per the diagram, Manhattan distance = a + b

Where ‘a’ is the vertical distance, and ‘be is the horizontal distance.

Euclidean distance: As we can see from the figure, this is the shortest distance between the two document vectors. Pythagoras theorem is used to calculate this distance.

Both the Manhattan distance and Euclidean distance are the special cases of Minkowski Distance. I am not going to explain more about these distances because it is not relevant to this project.

Both these distances are useful in scenarios where the magnitude of the vectors makes more sense than their orientations. In other words, when dealing with numerical quantities, these distances are more useful. But if we use these distances in our project to measure the similarities of books, these algorithms may classify the books like all the shorter books are similar or all the lengthy books are similar, because we know that when analyzing texts, some books are shorter, some others are longer. If a book is lengthy, some words may repeat more times than they repeat in short books. So, we don't need to arrive at the wrong conclusion with this. We need to compare the contents regardless of their length in general. So, the **Cosine distance** will be the better choice. It is calculated by means of the angle between the two document vectors. It focuses on the orientations of the vectors in space regardless of their magnitude.

4.4.1 Calculation of Cosine distance

Cosine distance can be explained mathematically as.

Cosine similarity, $\cos \boldsymbol{\theta}$

$$\cos \boldsymbol{\theta} = \frac{\vec{\boldsymbol{D}}_1 \cdot \vec{\boldsymbol{D}}_2}{|\vec{\boldsymbol{D}}_1||\vec{\boldsymbol{D}}_2|}$$

Cosine distance = 1- cosine similarity = 1- cos(θ)

Let's consider an example with more words. This cannot visualize because it is in multi-dimensional space.

D1 = “ I like chocolate, but I like spices more.”

D2 = "She also like chocolate."

So we have two documents D1 and D2, with more words. Here we have eight unique words in total, and some words occur in both documents. So these documents can be represented as vectors in 8-dimensional space. Before measuring the cosine distances, we will do preprocessing so any punctuations they have all get removed, and also the texts will be in lowercase as well. This is just an example of how it works.

Table 2. An example of documents as vectors

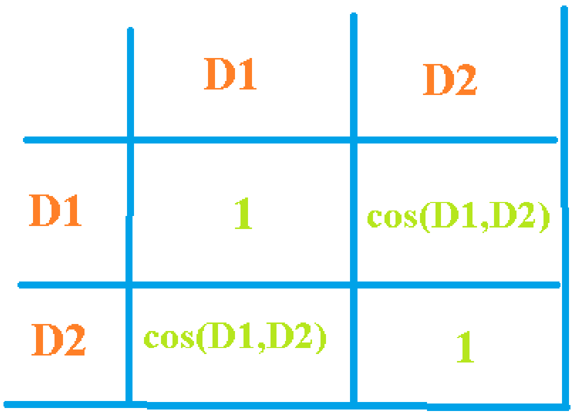
Documents	**“I”**	**“like”**	**“chocolate”**	**“but”**	**“spices”**	**“more”**	**“she”**	**“also”**
D1	2	2	1	1	1	1	0	0
D1	0	1	1	0	0	0	1	1

So D1 and D2 can be represented as vectors, as shown below.
D1 = [2,2,1,1,1,1,0,0]
D2 = [0,1,1,0,0,0,1,1]
and the cosine similarity can be calculated from above formula as

$$\cos\boldsymbol{\theta} = \frac{(2*0 + 2*1 + 1*1 + 1*0 + 1*0 + 1*0 + 0*1 + 0*1)}{sqrt(2^2+2^2+1^2+1^2+1^2+1^2+0^2+0^2)*sqrt(0^2+1^2+1^2+0^2+0^2+0^2+1^2+1^2)} = \mathbf{0.375}$$

i.e., cosine similarity (D1, D2) = 0.375
cosine distance = 1- 0.375 = 0.625
The pair-wise distance can be calculated, and it can be represented in a matrix called the similarity matrix by applying the cosine similarity formula. The similarity matrix is represented in a square form where the rows and columns will be the same. Refer to fig; below.

Fig. 8 Similarity matrix for the example



	D1	D2
D1	1	cos(D1,D2)
D2	cos(D1,D2)	1

From the figure, we can see the diagonal elements of the matrix are 1. This is because the similarity is calculated between the documents themselves, that is, between D1 and D1, the D2 and D2. Since they are the same and the angle between D1 and D1 is zero, likewise for D2. Hence cos 0 = 1. If cos 0 is 1, which implies that they are the same in orientation but may be or may not be different magnitude, but we can say that they are similar. The cosine function is equal to 0 when the angle between them is 90 degrees. This indicates that both the document vectors are dissimilar. Cos (D1, D2) represents the cosine distance between the documents D1 and D2; this will be calculated using the formula I have mentioned. We see that the elements of the matrix above and below the diagonal elements are the same; this is how a similarity matrix is represented. When we calculate the cos (D1, D2), if we get a value close to 1, it means that these documents are similar, and if we get a value close to zero, it means they have more dissimilar contents. Here in this example, we only have two documents; hence the dimension of this matrix is 2x2, and it has four elements (i.e., $\mathbf{2^2}$). When the number of documents changes, the dimension of this matrix changes accordingly. In this project, our five books are the documents, so our similarity matrix will have a dimension of 5x5 (i.e., $5^{\mathbf{2}} = 25$ elements in total).

In the above example, I represent the vectors in space based on the word count for simplicity. But in this project and most of the cases, it is represented, and the cosine distance is calculated based on the word's TF-IDF scores instead of its word count because that makes more sense.

4.5 Clustering to group similar books together

Clustering is the most commonly used algorithm in Unsupervised Machine Learning. Clustering is the process of identifying similar groups in the data points. According to Clustering, each observation or element in one group exhibits stronger similarities between the members of the same group and share dissimilarities between the members of the other group [1]. Clustering is not just a single algorithm but it is a collection of several algorithms. There are different types of Clustering. I will be using Hierarchical Clustering for this project.

“Hierarchical clustering (also called hierarchical cluster analysis or HCA) is a method of cluster analysis which seeks to build a hierarchy of clusters”[Wikipedia]. In order to achieve hierarchical Clustering, we need to apply certain criteria to this algorithm, and these criteria will decide how to group the data observation. These criteria are called distance similarity or dissimilarity measures. Usually, any of the similarity distance matrices such as Euclidean distance, Manhattan distance, Cosine distance, etc., will be used in this scenario. The shape and outcome of the hierarchical Clustering depend on the choice of the matrix. In the previous section, we already selected the appropriate similarity distance that is our cosine similarity distance, and we already have our cosine similarity matrix with us. So in this step, we just need to pass this matrix to the clustering algorithm.

The results of hierarchical Clustering are usually represented in the form of the dendrogram. A Dendrogram is a pictorial representation of data points or the documents arranged in a bottom-up approach which shows the hierarchical relationship between the data points in that cluster[19].

4.5.1 Comparing Hierarchical Clustering based on different linkage Algorithms

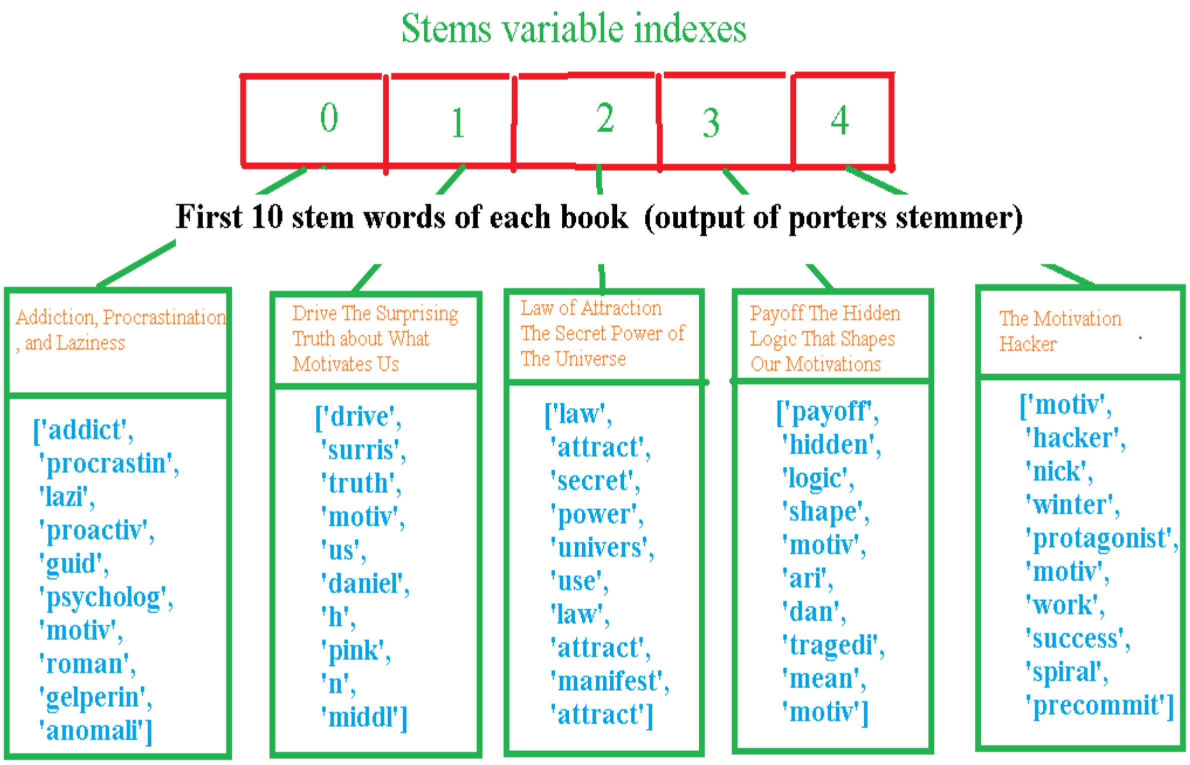
In the previous step, we generated the similarity matrix (Refer results section). So, in this step, we can see how the clustering algorithms work and will perform Clustering based on single, complete and average linkage methods. This clustering algorithm is performed in several iterations, and it will terminate when all the data points or here documents form a big cluster. I am discussing three main types of linkage criteria to group the clusters and to generate the dendrograms. They are the single linkage, complete linkage, and average linkage. Single linkage is the minimum inter-cluster similarity or dissimilarity; it is also known as the nearest neighbor. Complete linkage is the maximum inter-cluster similarity or dissimilarity, which is also called the farthest neighbor. At the same time, the Average linkage is the mean inter-cluster similarity or dissimilarity [university Essex, lecture notes]. The steps followed in all these algorithms are the same and are different in the way the distance is selected. The first iteration is the same in all three, that is, to find out the minimum pair-wise distance from the similarity matrix. These algorithms are explained detailed in result section with its associated dendrograms.

CHAPTER 5

Results

5.1 Stemming results

Fig 9. Stemming output


Stems variable indexes
0
1
2
3
4
First 10 stem words of each book (output of porters stemmer)
Addiction, Procrastination, and Laziness
['addict', 'procrastin', 'lazi', 'proactiv', 'guid', 'psycholog', 'motiv', 'roman', 'gelperin', 'anomali']
Drive The Surprising Truth about What Motivates Us
['drive', 'surris', 'truth', 'motiv', 'us', 'daniel', 'h', 'pink', 'n', 'middl']
Law of Attraction The Secret Power of The Universe
['law', 'attract', 'secret', 'power', 'univers', 'use', 'law', 'attract', 'manifest', 'attract']
Payoff The Hidden Logic That Shapes Our Motivations
['payoff', 'hidden', 'logic', 'shape', 'motiv', 'ari', 'dan', 'tragedi', 'mean', 'motiv']
The Motivation Hacker
['motiv', 'hacker', 'nick', 'winter', 'protagonist', 'motiv', 'work', 'success', 'spiral', 'precommit']

In fig 13, I am trying to visualize how it looks like after the stemming. I have assigned the output of the stemming into a variable called "stems." This "stems" has five indexes, and each index consists of the list of stem words of each book, in Fig 13. I only marked the first ten stems of each book. I draw this figure manually to communicate better.

5.2 Bag of words model (BOW) output

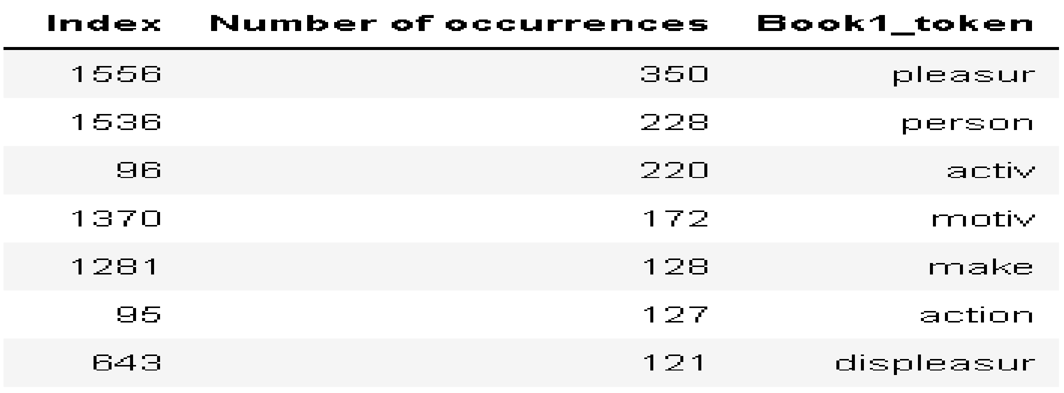
As discussed in the methodology section, I have generated a Bag of words model using these stemmed words. And we have come a long way with the texts. Let's have a look at the number of words in this bag of words model and compare it with the words we have before. When I started my analysis, The books (book1 to book 5) contained 33048, 51310, 31692, 24618, and 42594 words, respectively. As we see, I have done some preprocessing for these books; first, I removed the stop words, then I removed some irrelevant words and did stemming. Stemming technically doesn't remove any words; it just replaces the word form. After all these steps, the count of the words becomes 15595, 26737, 13319, 11989, and 20339, respectively.

Table 3. Comparison of the count of words along the text mining process

Books	**The actual number of words in books initially**	**Number of words after Pre-processing (includes stop words, irrelevant words removal, and stemming)**	**Number of workout BOW model(unique words)**
'Addiction, Procrasti nation, and Laziness'	33048	15595	2359
'Drive The Surprising Truth about What Motivates Us	51310	26737	4459
'Law of Attraction The Secret Power of The Universe'	31692	13319	1702
'Payoff The Hidden Logic That Shapes Our Motivations	24618	11989	2676
'The Motivation Hacker by Nick Winter'	42594	20339	3519
Total words	**183262**	**87979**	**14715**

It has been seen that almost half of the words get removed after the preprocessing. Now let's have a look at the unique words, the bag of words output will only contain unique words since it is a dictionary of words with its number of occurrences. Now we only left with 2359, 4459, 1702, 2676, and 3519 words in the books, respectively. This shows that only 12% to 22% of the preprocessed words remain for us to analyze further. Refer to table 2.
From the table 2, we can we that at the beginning, there are 183262 words in total of all the books after preprocessing only 87979 words left, among which only 14715 words are only unique.
So let's have a look at the most common words in each book.

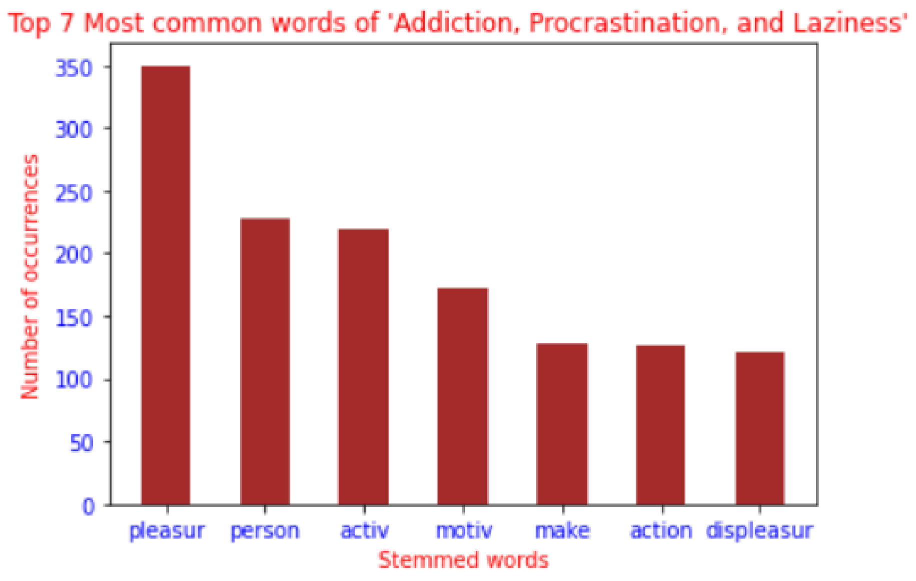
Table 4. Top 7 word frequencies of the book 'Addiction, Procrastination, and Laziness



Index	Number of occurrences	Book1_token
1556	350	pleasur
1536	228	person
96	220	activ
1370	172	motiv
1281	128	make
95	127	action
643	121	displeasur

From table 3 it represents the number of times a word is repeated in book 1. I sorted the table in descending order with respect to the column "number of occurrences." Fig 14. I have plotted this in a bar plot to better understand the most common words in book 1.

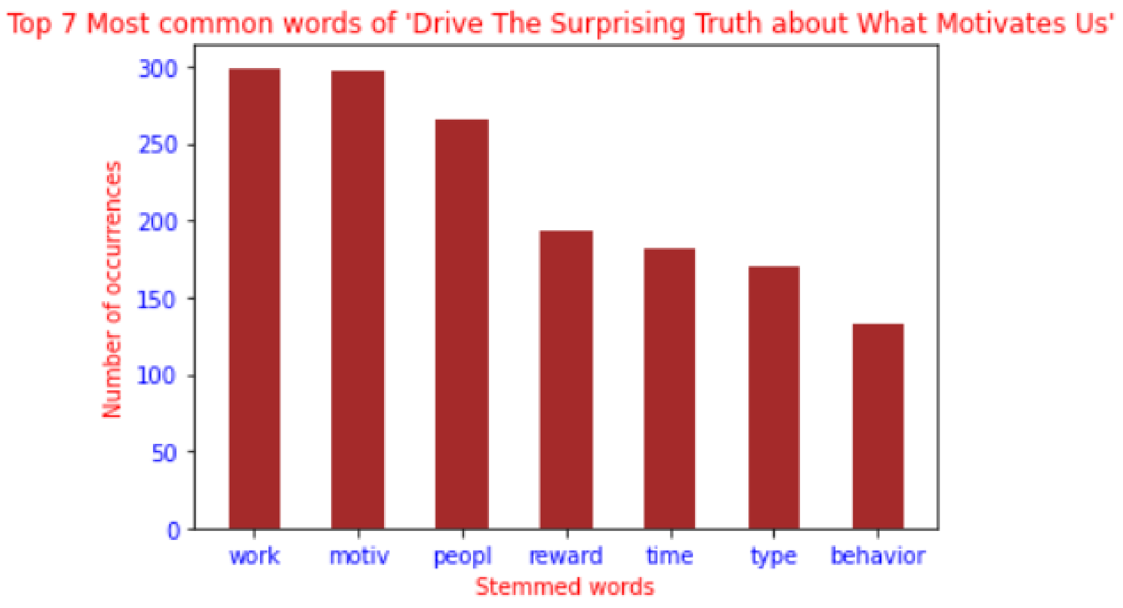
Fig 10. Most common words of book 1



From the figure 10, we can see that the word "pleasur"(stemmed word) and its original form may be "pleasure" is the most common word in book 1. This word has occurred 350 times in the book.

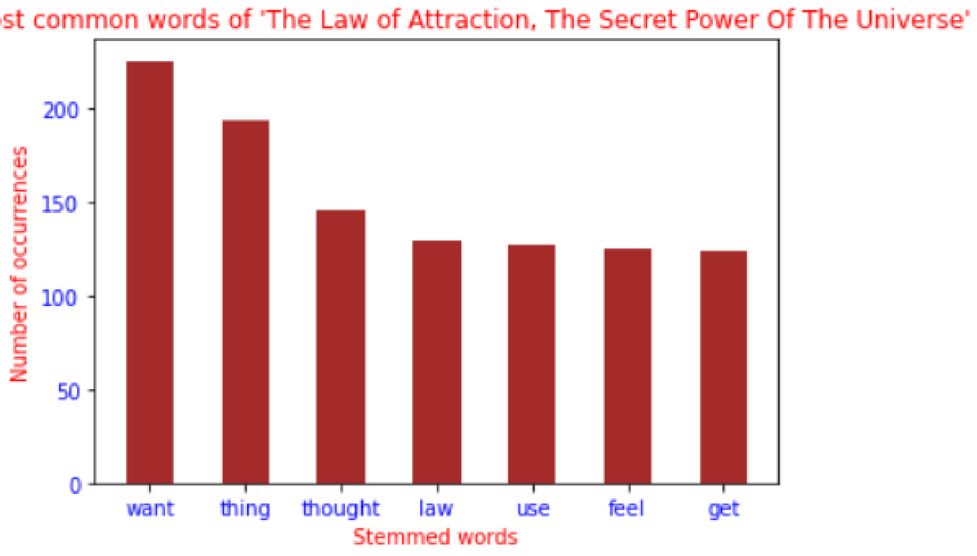
Now Let's also use the most common words in other books also.

Fig 11. Most common words of book 2



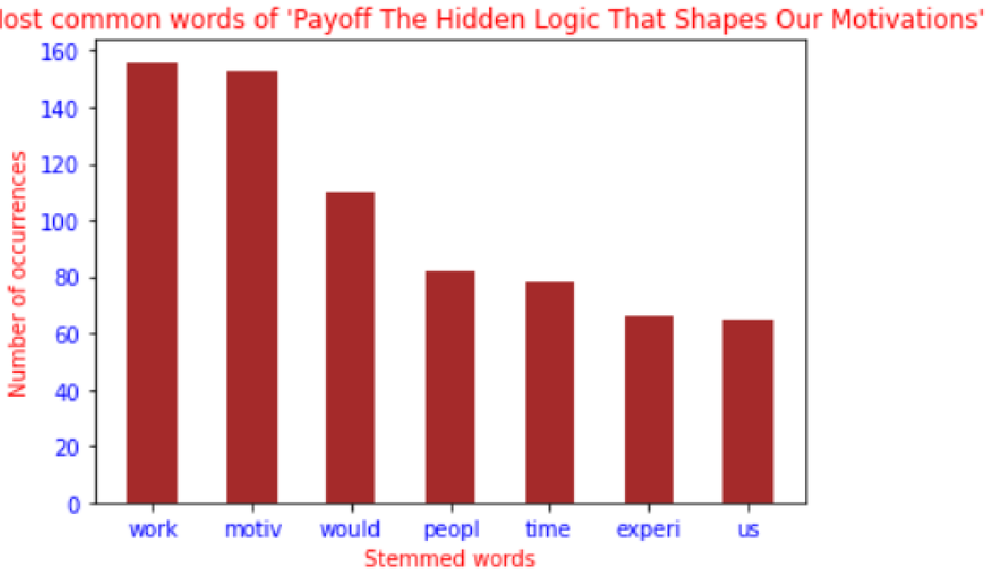
For Book 2, the word “work” and “motivation” is the most common words where “work” is repeated 156 times and “motivation” is repeated 153 times in the entire book.

Fig 12 Most common words of book 3



For book 3, the word "want" is the most common word and is repeated 225 times.

Fig 13 Most common words of book 4



For book 4, the word "work" and "motivation" is the most common words, where "work" is repeated 156 times, and "motivation" is repeated 153 times. From these results, I notice that Book 4 that is 'Payoff The Hidden Logic That Shapes Our Motivations' and book two that is 'Drive The Surprising Truth about What Motivates Us,' have "work" and "motivation" repeated the same number of times. This may indicate that maybe these two books will have some similar contents. But we can't conclude this based on this BOW model since it has some limitations. So we will discover this as we go further.

Fig 14 Most common words of book 5

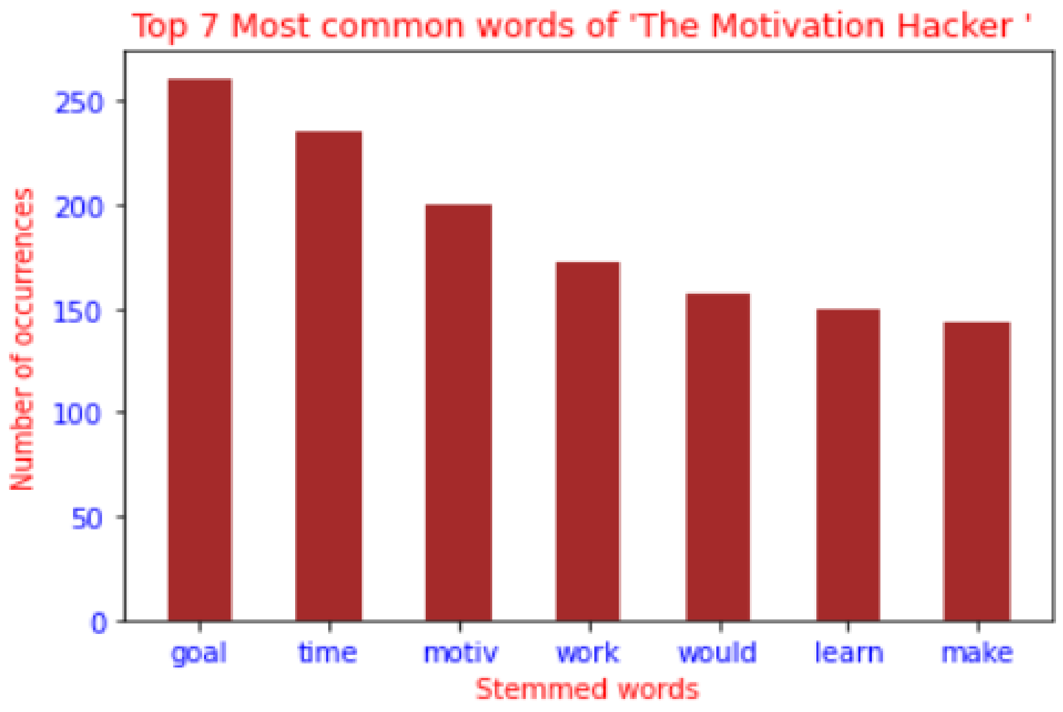
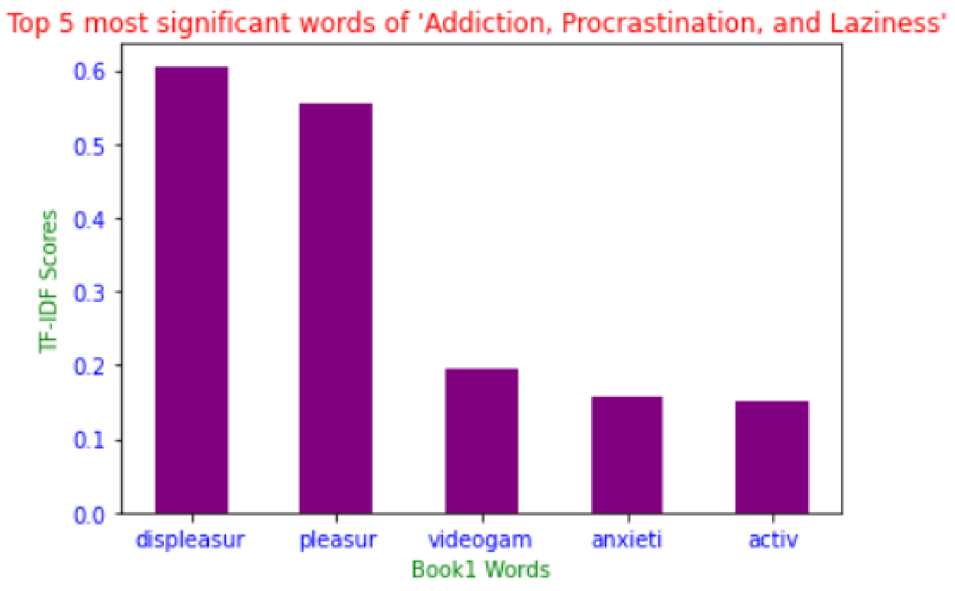


Fig 14 shows that the word "goal" has been repeated more times in book five, and it is repeated 261 times in the entire book.

5.3 TF-IDF model results

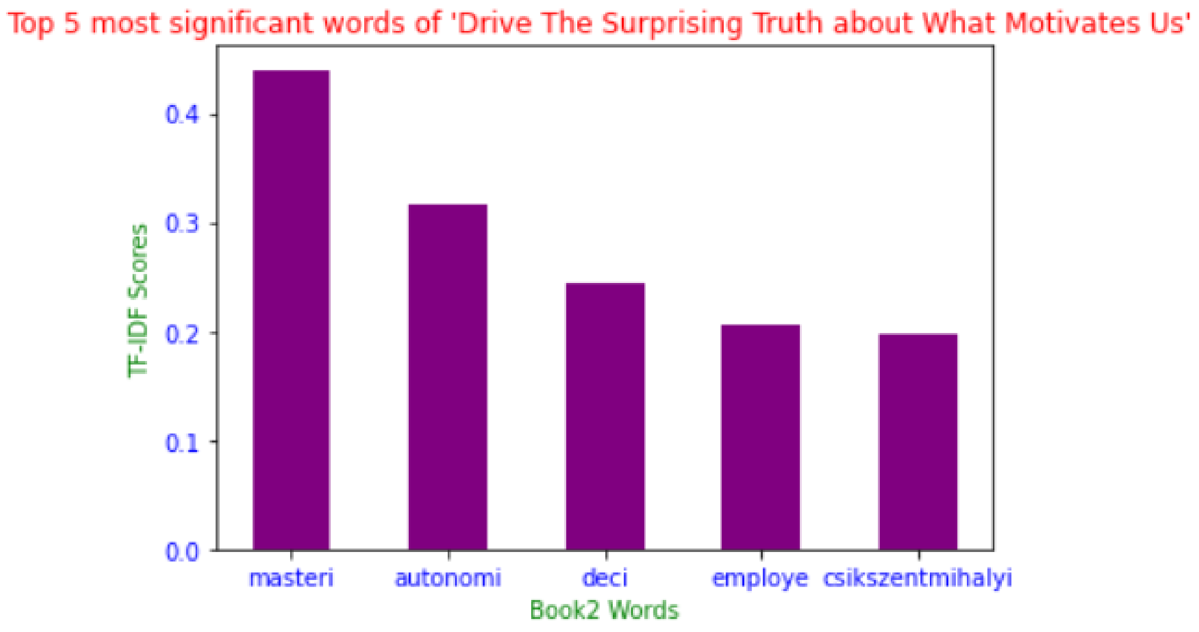
From the BOW, we see that the top 5 most common words in book 1 are "pleasur," "person," "activ," "motiv," "make," and when we analyze the most significant words of book one according to TF-IDF model the results were very surprised that even though these words are mostly used throughout the book all of these doesn't have much significance in book1. In other words, some of these words are also repeated in other books, and we couldn't say that these have more importance in this book compared to other books. With the help of the TF-IDF model, we can find those particular words which are repeated more times in this book than it repeated in other books. “displeasure”, “pleasur”, “videogam”, “anxiety”, “activ” these are the top 5 significant words of book1. We can see that the words “pleasur” and “active” are also pop up in the most common words as well as the most significant words, but other words are not the same.

Fig 15. Most Significant words of book one based on TF_IDF scores



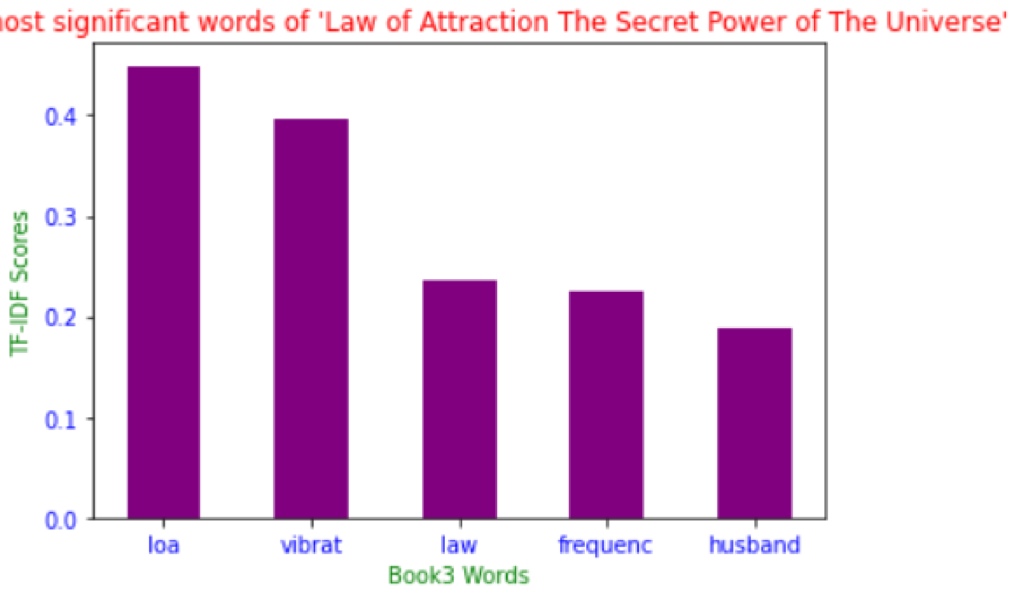
We calculate the TF-IDF scores of every word according to each document separately. And if the TF-IDF score of a word for book 1 is less than the TF-IDF score of that same word for book 2 (if that word is also present in book two, of course), then we can say that particular word is more significant to book two than book 1. The highest the TF-IDF score, the more significant that word for that document or book.

Fig 16. Most Significant words of book two based on TF_IDF scores



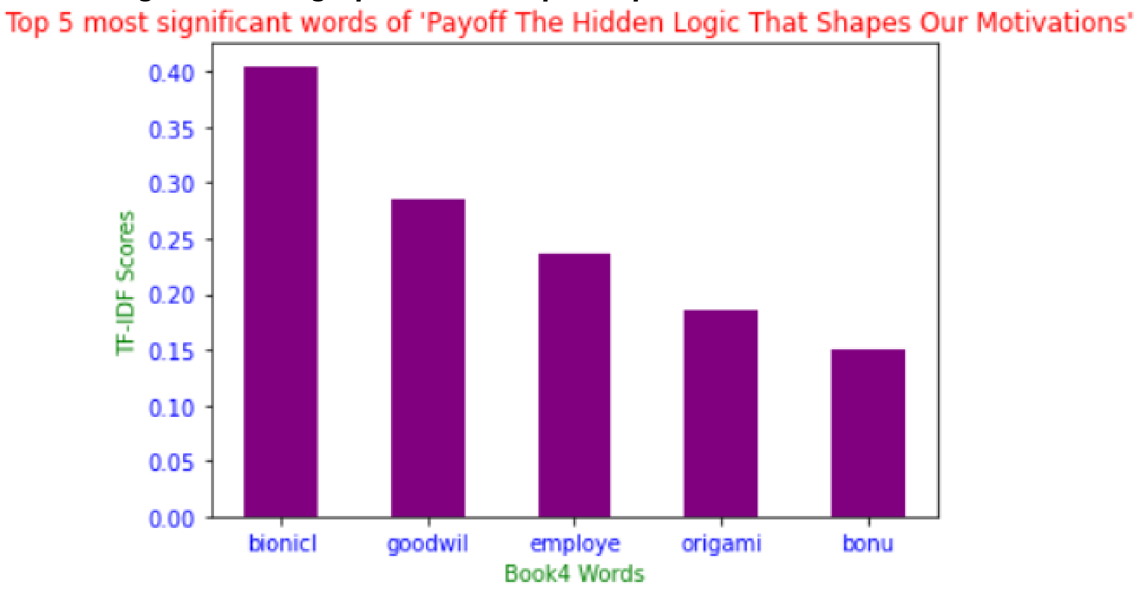
For Book2, "work," "motiv," "people," "reward," and "time" were the most common words according to the BOW model, but from the bar plot below, we can we that none of these words are the significant words of this book.

Fig 17. Most Significant words of book three based on TF_IDF scores



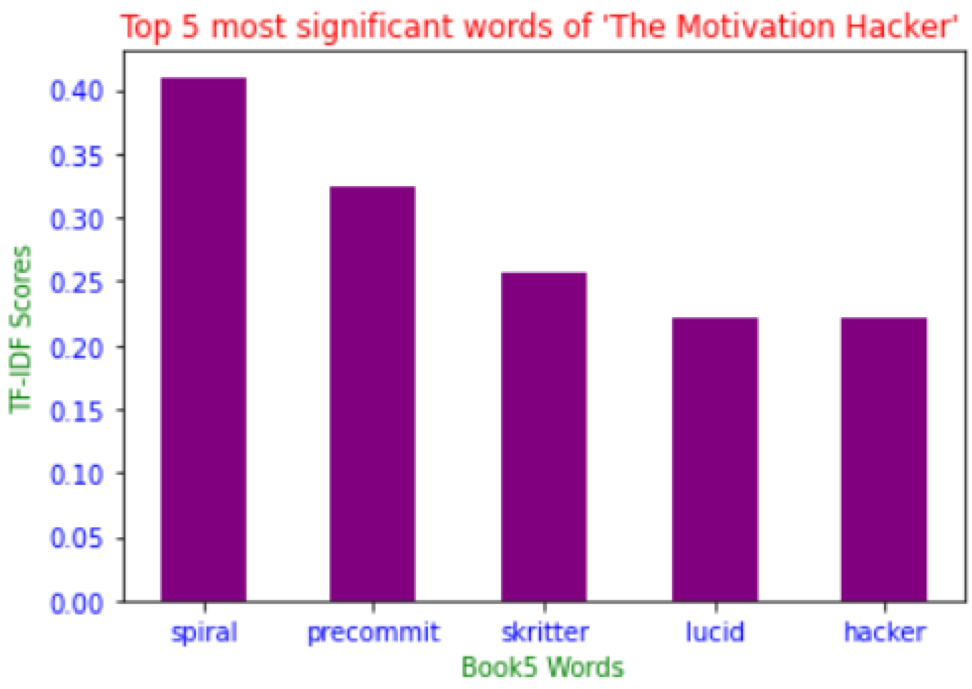
For book 3, "want," "thing," "thought," "law," and "use" were the most common words according to the Bow model, but according to TF-IDF scores, the words. "loa," "vibrat," "law," "frequenc," and "husband" are the words that are more significant.

Fig 18. Most Significant words of book four based on TF_IDF scores



According to TF-IDF scores, the top 5 most significant words of book4 are "bionici," "goodwill," "emplloye," "Origami," and "bonu," whereas the most common word based on the BOW model were "work," "motiv," "would," "peol" and "time."

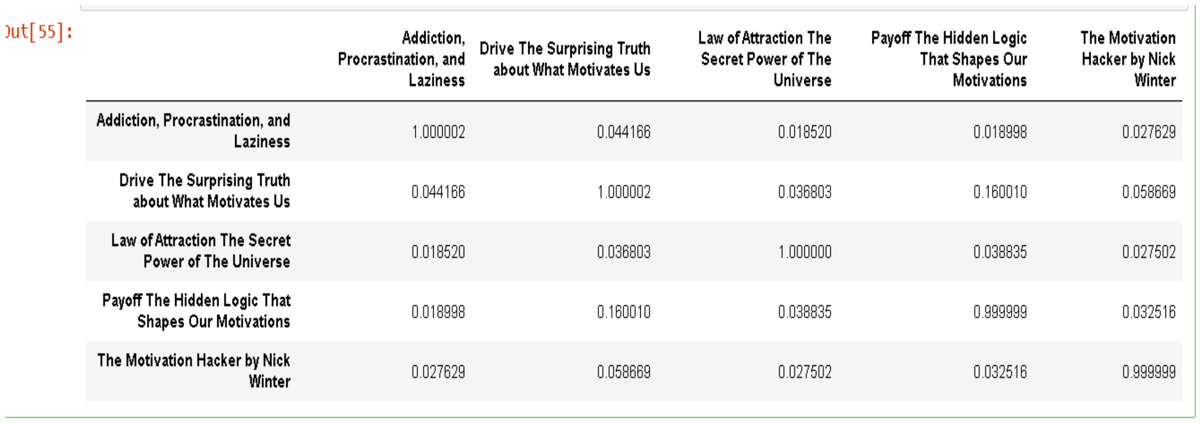
Fig 19. Most Significant words of book five based on TF_IDF scores



5.4 Cosine Similarity matrix of the 5 Books

In the methodology section, I have explained how we can calculate the cosine similarity matrix. Gensim library also has the inbuilt functions to calculate this matrix. The similarity matrix of our five books is calculated as shown below.

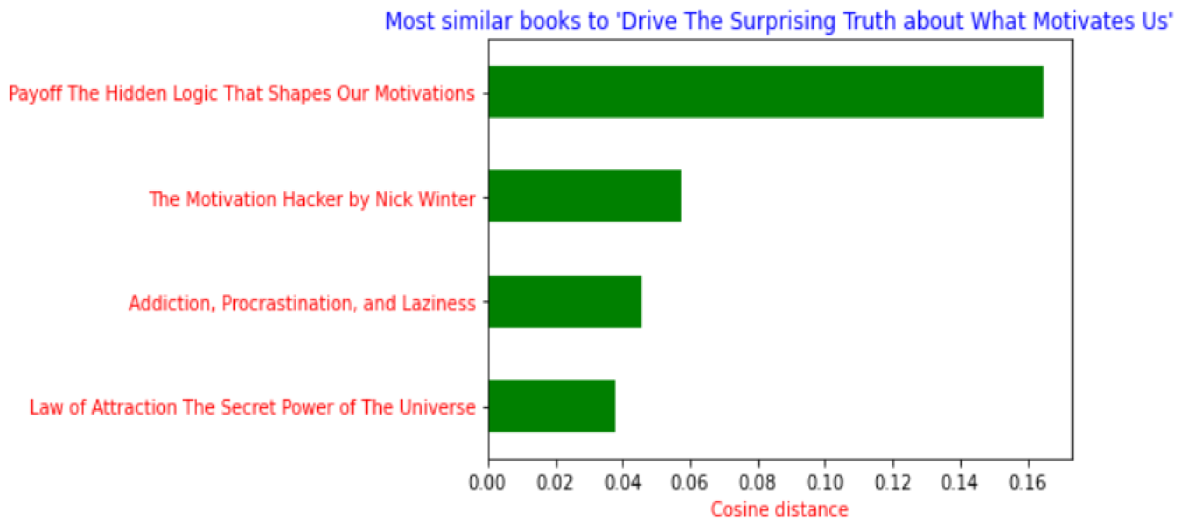
Fig 20. Similarity matrix of the five books



```
Out[55]:
```

	Addiction, Procrastination, and Laziness	Drive The Surprising Truth about What Motivates Us	Law of Attraction The Secret Power of The Universe	Payoff The Hidden Logic That Shapes Our Motivations	The Motivation Hacker by Nick Winter
Addiction, Procrastination, and Laziness	1.000002	0.044166	0.018520	0.018998	0.027629
Drive The Surprising Truth about What Motivates Us	0.044166	1.000002	0.036803	0.160010	0.058669
Law of Attraction The Secret Power of The Universe	0.018520	0.036803	1.000000	0.038835	0.027502
Payoff The Hidden Logic That Shapes Our Motivations	0.018998	0.160010	0.038835	0.999999	0.032516
The Motivation Hacker by Nick Winter	0.027629	0.058669	0.027502	0.032516	0.999999

Fig 21. Books most similar to "Drive The Surprising Truth about What Motivates Us"



With the help of the cosine similarity matrix, I have plotted all other books against the content of the book "Drive The Surprising Truth about What Motivates Us." I just picked this book as a reference. We can choose any other book as a reference and plot it against all other books, which gives the same conclusion.

From the graph, we can see that "Drive, the Surprising Truth about What Motivates Us" and "Payoff The Hidden Logic That Shapes Our Motivations" are more similar compared to other books, and this books is more dissimilar to book

"Law of Attraction, The Secret Power of The Universe." The above graph only gives an overall idea of the similarities of these books. For better understanding, that is, which book is the closest relative of other books, we can use Clustering in the next step and group each book with respect to its closer one and visualize using Dendrogram[1].

5.5 Hierarchical Clustering output

We see that the elements above and below the diagonal elements are the same for a similarity matrix. For simplicity, we can represent our similarity matrix as shown below. The figure 24 is the same cosine similarity matrix we obtained from the previous section. I just replaced the name of the books with D1, D2, D3, D4, and D5 for simplicity.

Where, D1: 'Addiction, Procrastination, and Laziness'

D2: 'Drive, the Surprising Truth about What Motivates Us'

D3: 'The Law of Attraction, The Secret Power of The Universe'

D4: 'Payoff, The Hidden Logic That Shapes Our Motivations'

D5: 'The Motivation Hacker'

Since we are using the cosine similarity distance, the distances which are closer to 1 are the minimum distance.

Iteration 1: First iteration is same for all the 3 linkage algorithms

Identifying the minimum pair-wise similarity distance(highlighted in yellow color)

Fig 22. Clustering algorithm iteration 1

	D1	D2	D3	D4	D5
D1	1				
D2	0.045	1			
D3	0.018	0.038	1		
D4	0.019	0.165	0.039	1	
D5	0.028	0.057	0.027	0.032	1

In the figure 22, I have highlighted a distance in yellow color, which is the value closer to 1 when compared to other values. From matrix we already identified the minimum distance as 0.165.

So, we can see that this value is actually the cosine distance between D2 and D4(i.e., 'Drive, the Surprising Truth about What Motivates Us' and 'Payoff, the Hidden Logic That Shapes Our Motivations.' So, these two books form a cluster in the first Iteration of each linkage.
Let's see how the complete linkage algorithm works.

5.5.1 Clustering based on Complete Linkage

Complete Linkage calculation steps

Iteration 1
D2 and D4 form a cluster in the first Iteration. So, the similarity matrix initially was a 5x5 matrix which now converges into a 4x4 matrix. We can denote the newly formed cluster as (D2, D4). (Refer fig 22)
Iteration 2
we need to update the similarity matrix based on the complete linkage considering the new clusters formed. Only the values affected by the new cluster (D2, D4) need to be re-calculated; other values remain the same.

Calculating the maximum distance between (D2, D4) and D1

Max {(D2, D4), D1} = Max {(D2, D1), (D4, D1)}
= Max {0.045, 0.019} (from fig. 22)
= 0.019
Remark: 0.045 is the minimum distance because it is cosine similarity (value close to one is minimum (not close to zero) for cosine similarity) So we choose 0.019 as the maximum distance between the two.

Calculating the maximum distance between (D2, D4) and D3

Max {(D2, D4), D1} = Max {(D2, D3), (D4, D3)}
= Max {0.038, 0.039} (from fig. 22)
= 0.038

Calculating the maximum distance between (D2, D4) and D5

Max {(D2, D4), D1} = Max {(D2, D5), (D4, D5)}
= Max {0.057, 0.032} (from fig. 22)
= 0.032

The similarity matrix can be updated as follow

Fig 23. Complete linkage iteration 2

	D1	(D2, D4)	D3	D5
D1	1			
(D2,D4)	0.019	1		
D3	0.018	0.038	1	
D5	0.028	0.032	0.027	1

Iteration3

Repeating the above steps until the algorithms converge
Here the least distance is identified as 0.038, So D3 ('Law of Attraction The Secret Power of The Universe') joins the cluster (D2, D4) in this Iteration. So the new cluster formed can be denoted as ((D2, D4), D3).

Calculate the maximum distance between ((D2, D4), D3) and D1.

Max {((D2, D4), D3), D1} = Max {((D2, D4), D1), (D3, D1)}
= Max {0.019, 0.018} (from fig. 23)
= 0.018

Calculate the maximum distance between ((D2, D4), D3) and D5.

Max {((D2, D4), D3), D5} = Max {(((D2, D4), D5)), (D3, D5)}
= Max {0.032, 0.027} (from fig. 23)
= 0.027

According to the re-calculated values, the similarity matrix can be updated as follows;

Fig 24. Complete linkage iteration 3

	D1	((D2, D4), D3)	D5
D1	1		
((D2,D4), D3)	0.018	1	
D5	0.028	0.027	1

Iteration 4

The minimum distance is identified as 0.028 and which is the distance between D1 and D5, So these two books form another cluster. It can be denoted as (D1, D5)

Calculating the maximum distance between ((D2, D4), D3) and (D1, D5)

Max {((D2, D4), D3), (D1, D5)} = Max {((D2, D4), D3), D1), ((D2, D4), D3), D5)}

= Max {0.018, 0.027} (from fig. 24)

= 0.018

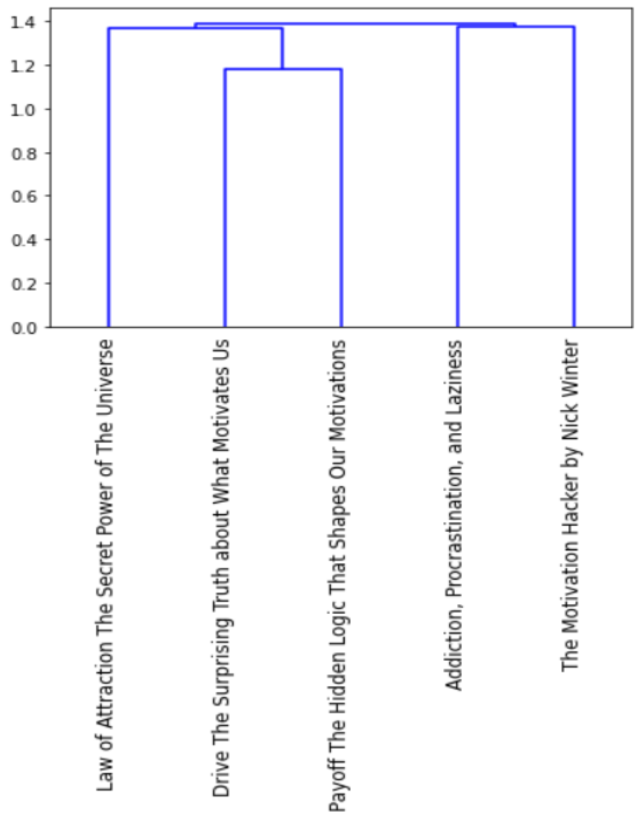
According to the re-calculated values, the similarity matrix can be updated as follows;

Fig 25. Complete linkage iteration 4

	(D1, D5)	((D2, D4), D3)
(D1, D5)	1	
((D2,D4), D3)	0.018	1

Now the minimum distance is identified as 0.018, and at this point, the algorithm terminates, and another big cluster will be formed with all these books. The results of this algorithm can be represented as a dendrogram as shown below.

Fig 26. Complete linkage Dendrogram



As we can see, the dendrogram follows a bottom-up approach. In the first iteration books, "Drive" and "Payoff" forms a cluster, then in the second Iteration, "law of attraction" Joins these cluster, then, in the third Iteration, another cluster of the other two remaining books are formed. Then in the last Iteration, a cluster is formed with all books. As we may notice that the books "Payoff" and "Drive" have more similar contents and "law of attraction contains more unique ideas than all other books, but it has more similarity towards "payoff" and "Drive" than the other two books. While the other books, "Addiction" and "motivation hacker" have somewhat similar contents, so they belong to the same cluster.

5.5.2 Clustering based on Single Linkage

Single Linkage calculation steps

Single linkage is also calculated like the complete linkage, but instead of taking the Max{distance} distance, we take the Min{distance}

Iteration 1

D2 and D4 form a cluster in the first Iteration (Refer fig 22).

Iteration 2

we need to update the similarity matrix based on the single linkage considering the new clusters formed. Only the values affected by the new cluster (D2, D4) need to be re-calculated; other values remain the same.

Calculating the minimum distance between (D2, D4) and D1

Min {(D2, D4), D1} = Min {(D2, D1), (D4, D1)}
= Min {0.045, 0.019} (from fig. 22)
= 0.045

Remark: 0.045 is the minimum distance because it is cosine similarity.

Calculating the maximum distance between (D2, D4) and D3

Min {(D2, D4), D1} = Min {(D2, D3), (D4, D3)}
= Min {0.038, 0.039} (from fig. 22)
= 0.039

Calculating the minimum distance between (D2, D4) and D5

Min {(D2, D4), D1} = Max {(D2, D5), (D4, D5)}
= Max {0.057, 0.032} (from fig. 22)
= 0.057

The similarity matrix can be updated as follow

Fig 27. Single linkage iteration 2

	D1	(D2, D4)	D3	D5
D1	1			
(D2,D4)	0.045	1		
D3	0.018	0.039	1	
D5	0.028	0.057	0.027	1

Iteration3

Repeating the above steps until the algorithms converge

Here the least distance is identified as 0.057, So D5 (‘The Motivation Hacker') joins the cluster (D2, D4) in this Iteration. So, the new cluster formed can be denoted as ((D2, D4), D5).

Calculate the minimum distance between ((D2, D4), D5) and D1.

Min {((D2, D4), D5), D1} = Min {((D2, D4), D1), (D5, D1)}
= Min {0.045, 0.028} (from fig. 27)
= 0.045s

Calculate the minimum distance between ((D2, D4), D5) and D3.

Min {((D2, D4), D3), D5} = Min {(((D2, D4), D5)), (D5, D3)}
= Min {0.039, 0.027} (from fig. 27)
= 0.039

According to the re-calculated values, the similarity matrix can be updated as follows;

Fig 28. Single linkage iteration 3

	D1	((D2, D4), D5)	D3
D1	1		
((D2,D4), D5)	0.045	1	
D3	0.018	0.039	1

Iteration 4

The minimum distance is identified as 0.045 and which is the distance between D1 and ((D2, D4), D5), So these books form another cluster. It can be denoted as **(((D2, D4), D5), D1**)

Calculating the minimum distance between (((D2, D4), D5), D1) and D3

Min {((D2, D4), D3), (D1, D5)} = Min {((D2, D4), D5), D3), (D1, D3)}
= Min {0.039, 0.018} (from fig. 28)
= 0.039

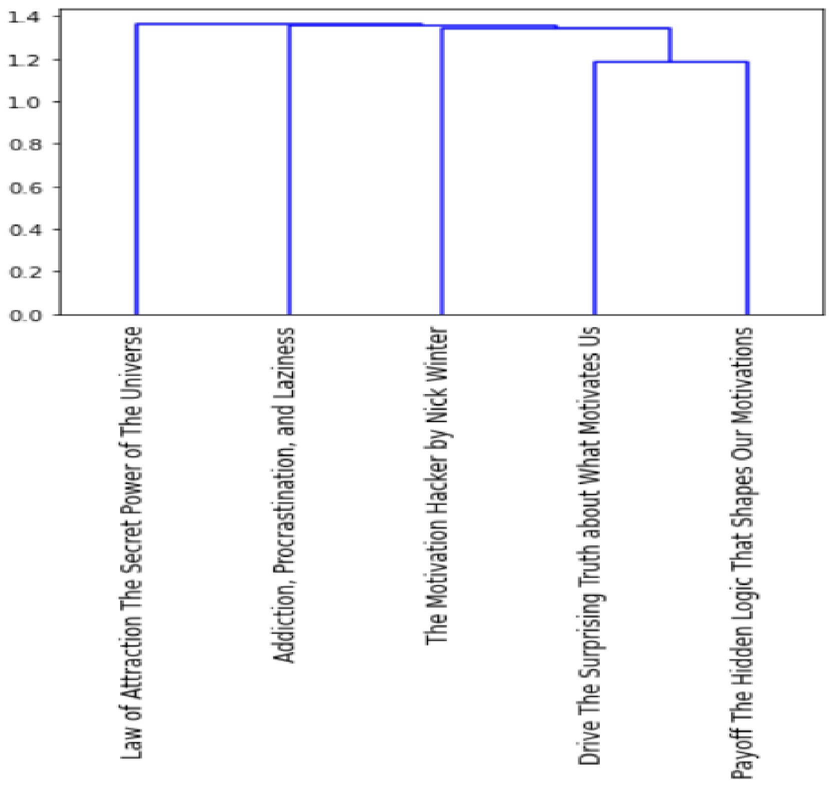
According to the re-calculated values, the similarity matrix can be updated as follows;

Fig 29. Single linkage iteration 4

	(((D2, D4), D5), D1)	D3
(((D2, D4), D5), D1)	1	
D3	0.039	1

Now the minimum distance is identified as 0.039, and at this point, the algorithm terminates, and another big cluster will be formed with all these books. The results of this algorithm can be represented as a dendrogram as shown below.

Fig 30. Single linkage Dendrogram



Like the complete linkage, the first cluster is formed with the books "Payoff" and "Drive" in the first Iteration, and in the second Iteration, "motivation hacker" joins this cluster, and in the 3rd Iteration, "addiction" joins and in the 4th iteration "law of attraction" joins these clusters.

5.5.3 Clustering based on Average Linkage

Average Linkage calculation steps

Average linkage is also calculated like the single and complete linkage only difference is that it take the average, Avg{distance}of the distances instead of Min{} or Max{}

Iteration 1

D2 and D4 form a cluster in the first Iteration (Refer fig 22).

Iteration 2

we need to update the similarity matrix based on the average linkage considering the new clusters formed. Only the values affected by the new cluster (D2, D4) need to be re-calculated; other values remain the same.

Calculating the average distance between (D2, D4) and D1

Avg {(D2, D4), D1} = Avg {(D2, D1), (D4, D1)}
= Avg {0.045, 0.019} (from fig. 22)
$= \frac{0.045+0.019}{2} = 0.032$

Calculating the average distance between (D2, D4) and D3

Avg {(D2, D4), D1} = Avg {(D2, D3), (D4, D3)}
= Avg {0.038, 0.039} (from fig. 22)
= 0.0385

Calculating the average distance between (D2, D4) and D5

Avg {(D2, D4), D1} = Avg {(D2, D5), (D4, D5)}
= Avg {0.057, 0.032} (from fig. 22)
= 0.0445

The similarity matrix can be updated as follow

Fig 31. Average linkage iteration 2

	D1	(D2, D4)	D3	D5
D1	1			
(D2,D4)	0.032	1		
D3	0.018	0.0385	1	
D5	0.028	0.0445	0.027	1

Iteration3
Repeating the above steps until the algorithms converge
Here the least distance is identified as 0.0445, So D5 (‘The Motivation Hacker') joins the cluster (D2, D4) in this Iteration. So, the new cluster formed can be denoted as ((D2, D4), D5).

Calculate the average distance between ((D2, D4), D5) and D1.

Avg {((D2, D4), D5), D1} = Avg {((D2, D4), D1), (D5, D1)}
= Avg {0.032, 0.028} (from fig. 31)
= 0.033

Calculate the average distance between ((D2, D4), D5) and D3.

Avg {((D2, D4), D3), D5} = Avg {(((D2, D4), D5)), (D5, D3)}
= Avg {0.0385, 0.027} (from fig. 31)
= 0.032

According to the re-calculated values, the similarity matrix can be updated as follows;

Fig 32. Average linkage iteration 3

	D1	((D2, D4), D5)	D3
D1	1		
((D2,D4), D5)	0.033	1	
D3	0.018	0.032	1

Iteration 4

The minimum distance is identified as 0.033 and which is the distance between D1 and ((D2, D4), D5), So these books form another cluster. It can be denoted as **(((D2, D4), D5), D1**)

Calculating the average distance between (((D2, D4), D5), D1) and D3

Avg {((D2, D4), D3), (D1, D5)} = Avg {((D2, D4), D5), D3), (D1, D3)}
= Avg {0.032, 0.018} (from fig. 34)
= 0.025

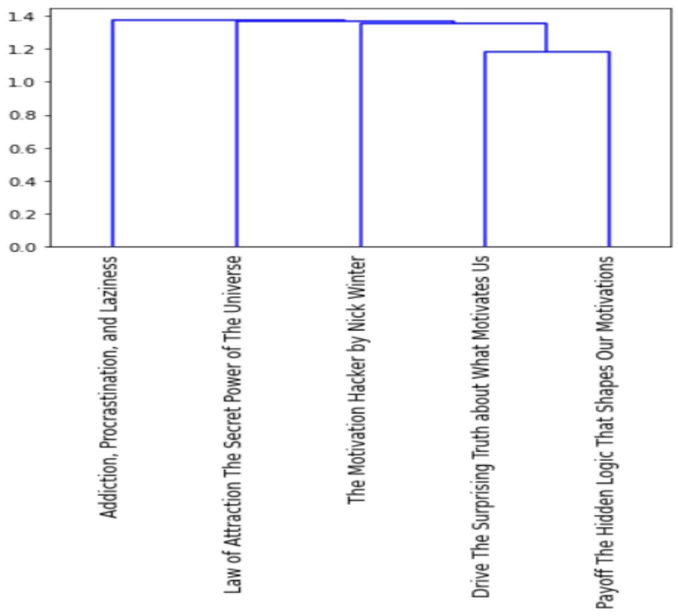
According to the re-calculated values, the similarity matrix can be updated as follows;

Fig 33. Average linkage iteration 4

	(((D2, D4), D5), D1)	D3
(((D2, D4), D5), D1)	1	
D3	0.025	1

Now the minimum distance is identified as 0.025, and at this point, the algorithm terminates, and another big cluster will be formed with all these books. The results of this algorithm can be represented as a dendrogram as shown below.

Fig 34. Average linkage Dendrogram



As we can see we the Dendrogram for Average linkage is similar to the dendrogram we get from the single linkage.

From the above three linkage algorithms we have analyzed, we can conclude that the books "Drive" and "payoff" have more similar contents. Those who like the book "Drive" may also be interested to read the book "Payoff." All other books have less similarity in general. Even though some books form a cluster in complete linkage, they are not that similar. They are grouped because this algorithm is the farthest neighbor; this we can understand from the single and average linkage.

CHAPTER 6

Discussion and Conclusion

To conclude, in this paper, I have investigated and explored some most popular techniques in NLP and Clustering. To conduct my study, I have downloaded five books from an online site and performed some preprocessing to transform the data for analysis. I have created a BOW to only keep the unique words in each book and generated a TF-IDF model using this BOW model. According to the generated TF-IDF scores, cosine similarities between the books are calculated. Using this cosine similarity, I have performed hierarchical Clustering. Then I generated the dendrograms with three different linkage algorithms in hierarchical Clustering. Single linkage and average linkage have the same dendrogram as output, but for complete linkage it was quite different since it explains in a wide perspective.

From the output from the Clustering, we can conclude that the books "Payoff" and "Drive" are the closest relatives also the books "Addiction" and "Motivation hacker" also have a little similarity. The book "Law of attraction" has is more dissimilar to all other books, but it has a little lean towards "Drive" or "Payoff" compared to other books. In general, we can conclude that even though all the books I have selected belong to the same category of motivation books. They all have unique content than similar content. Some books have expressed some similar ideas, but most of the contents are dissimilar. All these books are from different authors. So, all of them express the view towards how to motivate people from a different perspective. Some similarities have occurred because all of their targets are the same, that is, to give motivation. In future works, I would like to consider analyzing the books from the same author and also like to do some research about how the google search queries or any other search engines recommend the result page.

Acknowledgment

I got this project idea and inspiration from some guided projects I found in datacamp(https://app.datacamp.com/learn) and also from previous coursework I implemented in R language, But here I used python to implement the code. I have used some codes and functions from the guided project "Book recommendation from Charles Darwin," "Hottest topic in Machine learning," etc., to implement this project. The clustering algorithms I explained in the result section, I used the ideas from the lecture notes of Applied statistics module taught by prof. Fanlin. His lecture notes explains how to perform these algorithms using Euclidian distance but I modified those for performing this algorithms using cosine distance.

The codes of my dissertation project along with .txt files will be also available in https://github.com/SudheenaMisra/Dissertation

APPENDIX A

Appendix

```
#!/usr/bin/env python
# coding: utf-8
#importing the glob library for Retrieving a list of all text files present in a folder
import glob
#The book files are contained in the folder Desertation/Books_data
data_folder = "Books_data/"
#List all the .txt files and sort them alphabetically
files = glob.glob(data_folder + "*.txt")
files.sort()
files
# Import libraries
import re, os
# Initialize the object that will contain the texts and titles
books = []
titles = []
for p in files:
    # Open each file
    files = open(p, encoding='utf-8-sig')
    # Remove all non-alpha-numeric characters
    data = re.sub('[\W_]+', ' ', files.read())
    # Store the texts and titles of the books in two separate lists
    books.append(data)
    #Using os.path.basename() and replace() functions to remove the folder name and .txt extension from the file name.
    titles.append(os.path.basename(p).replace(".txt", ""))
# Counting characters in each book
characters = [len(t) for t in books]
characters
#counting the number of words in each book
words = [len(words.split()) for words in books]
authors = ["Roman Gelperin","Daniel H Pink", "Jenny Loveless","Ariely Dan", "Nick Winter" ]
# Creating a dictinary object to store index, titles, authors and book texts
Book_dict = {"Titles":titles, "authors": authors, "book_texts":books}
import pandas as pd
# Creating a dataframe using dictory object
Books_dataFrame = pd.DataFrame(Book_dict)
Books_dataFrame
# Books_dataFrame.to_csv(r'C:\Users\Sudheena Sona\Desktop\Desertation\my_books.csv', index = False)
#Bar plot
```

```
import numpy as np
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
plt.bar(titles, words, color = 'green', width = 0.5)
plt.xlabel("Book name", color = "blue")
plt.ylabel("word count", color = "blue")
plt.title("Number of words in each books", color="blue" )
plt.xticks(rotation=90, color= "red")
plt.yticks(color='red')
#plt.savefig("words.pdf")
plt.show()
#Storing the texts of each books into sepearte variables for future reference
#Book_1 = Addiction procastination and laziness
book_1 = books[0]
# Book_2 = Drive The Surprising Truth about What Motivates Us
book_2 = books[1]
# Book_3 = Law of Attraction The Secret Power of The Universe
book_3 = books[2]
# Book_4 = Payoff The Hidden Logic That Shapes Our Motivations
book_4 = books[3]
# Book_5 = The Motivation Hacker
book_5 = books[4]
#Defining stop words
stop_words = ['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all', 'am', 'an', 'and', 'any', 'are', 'aren',
"aren't", 'as', 'at', 'be', 'because', 'been', 'before', 'being', 'below', 'between', 'both', 'but', 'by', 'can', 'couldn',
"couldn't", 'd', 'did', 'didn', "didn't", 'do', 'does', 'doesn', "doesn't", 'doing', 'don', "don't", 'down', 'during',
'each', 'few', 'for', 'from', 'further', 'had', 'hadn', "hadn't", 'has', 'hasn', "hasn't", 'have', 'haven', "haven't",
'having', 'he', 'her', 'here', 'hers', 'herself', 'him', 'himself', 'his', 'how', 'i', 'if', 'in', 'into', 'is', 'isn',
"isn't", 'it', "it's", 'its', 'itself', 'just', 'll', 'm', 'ma', 'me', 'mightn', "mightn't", 'more', 'most', 'mustn',
"mustn't", 'my','myself', 'needn', "needn't", 'no', 'nor', 'not', 'now', 'o', 'of', 'off', 'on', 'once', 'only', 'or',
'other', 'our', 'ours', 'ourselves', 'out', 'over', 'yours', 'yourself', 'yourselves',"you'll", "you're", "you've", 'your',
'own', 're', 's', 'same', 'shan', "shan't", 'she', "she's", 'should', "should've", 'shouldn', "shouldn't", 'so', 'some',
'such', 't', 'than', 'that', "that'll", 'the', 'their', 'theirs', 'them', 'themselves', 'then', 'there', 'these',
'they', 'this', 'those', 'through', 'to', 'too', 'under', 'until', 'up', 've', 'very', 'was', 'wasn', "wasn't",
'we', 'were', 'weren', "weren't", 'what', 'when', 'where', 'which', 're', 's', 'same', 'shan', "shan't", 'she',
"she's", 'should', "should've", 'shouldn', "shouldn't", 'so', 'some', 'such', 't', 'than', 'that', "that'll",
'the', 'their', 'theirs', 'them', 'themselves', 'then', 'there', 'these', 'they', 'this', 'those', 'through',
'to', 'too', 'under', 'until', 'up', 've', 'very', 'was', 'wasn', "wasn't", 'we', 'were', 'weren', "weren't",
'what', 'when', 'where', 'which', 'while', 'who', 'whom', 'why', 'will', 'with', 'won', "won't", 'wouldn',
"wouldn't", 'y', 'you', "you'd"]
 # Convert the texts to lower case
lower_case = [letters.lower() for letters in books]
# Transform the text into tokens
texts_tokens = [txt.split() for txt in lower_case]
# Remove tokens which are part of the list of stop words
books = [[word for word in txt if word not in stop_words] for txt in texts_tokens]
# when we go throgh the texts of each book we can find some words listed below which seems irrelevant to our analysis
irrelevent_words = ["table","of",'contents', "introduction","conclusion","chapters","chapter",
                  "one","two","three", "four", "five","six","seven", "eight","nine",
                  "ten","eleven","twelve","thirteen", "0","1","2","3","4","5","6","7","8","9","10","11","12","13"]
# Remove irrelevent words
books = [[word for word in text if word not in irrelevent_words] for text in books]
books[0:15]
#Stemming
import  nltk
from nltk.stem import PorterStemmer
ps = PorterStemmer()
#Generating stem for each words
stems = [[ps.stem(words) for words in text] for text in books]
stems
stems[0][0:10]
stems[1][0:10]
stems[2][0:10]
stems[3][0:10]
stems[4][0:10]
# #### Counting the number of words after pre-processing in each book
# Before pre-processing the number of words in each books was the following
```

```
#number of words = [33048, 51310, 31692, 24618, 42594]
words_after_preprocessing = [len(words) for words in stems]
words_after_preprocessing
# #### Generating a bag of words model
# In[31]:
# Load the functions allowing to create and use dictionaries
from gensim import corpora
# Create a dictionary from the stemmed tokens
dict = corpora.Dictionary(stems)
print(dict)
# Create a bag-of-words model for each book, using the previously generated dictionary
bag_of_words = [dict.doc2bow(text) for text in stems]
# Print the first five elements of the first book Bag of words model
bag_of_words[0][0:10]
# #### Counting the number of words in BOW model (unique words)
bow_words = [len(words) for words in bag_of_words]
bow_words
# #### Most common words
# Generating dataframe for each book which represent the frequency of  words in each book
# Book 1
# Import pandas to create and manipulate DataFrames
import pandas as pd
# Convert the Bag of words model for book 1 into a DataFrame
book_1_df = pd.DataFrame(bag_of_words[0])
# Add the column names to the DataFrame
book_1_df.columns = ["Index","Number of occurrences"]
# Add a column containing the token corresponding to the dictionary index
book_1_df["Book1_token"] = [dict[index] for index in book_1_df["Index"]]
# Sort the DataFrame by descending number of occurrences and print the first 7 values
book1_df_head = book_1_df.sort_values(by="Number of occurrences", ascending= False).head(7)
book1_df_head
get_ipython().run_line_magic('matplotlib', 'inline')
plt.bar(book1_df_head["Book1_token"], book1_df_head["Number of occurrences"], color = 'brown', width = 0.5)
plt.xlabel("Stemmed words", color = "red")
plt.ylabel("Number of occurrences", color = "red")
plt.title("Top 7 Most common words of 'Addiction, Procrastination, and Laziness' ", color="red" )
plt.xticks(rotation=0, color= "blue")
plt.yticks(color='blue')
#plt.savefig("words.pdf")
plt.show()
#book 2
# Convert the Bag of words model for book 2 into a DataFrame
book_2_df = pd.DataFrame(bag_of_words[1])
# Add the column names to the DataFrame
book_2_df.columns = ["Index","Number of occurrences"]
# Add a column containing the token corresponding to the dictionary index
book_2_df["Book2_token"] = [dict[index] for index in book_2_df["Index"]]
# Sort the DataFrame by descending number of occurrences and print the first 10 values
book2_df_head = book_2_df.sort_values(by="Number of occurrences", ascending=False).head(7)
book2_df_head
get_ipython().run_line_magic('matplotlib', 'inline')
plt.bar(book2_df_head["Book2_token"], book2_df_head["Number of occurrences"], color = 'brown', width = 0.5)
plt.xlabel("Stemmed words", color = "red")
plt.ylabel("Number of occurrences", color = "red")
plt.title("Top 7 Most common words of 'Drive The Surprising Truth about What Motivates Us' ", color="red" )
plt.xticks(rotation=0, color= "blue")
plt.yticks(color='blue')
#plt.savefig("words.pdf")
plt.show()
#book 3
# Convert the Bag of words model for book 3 into a DataFrame
book_3_df = pd.DataFrame(bag_of_words[2])
# Add the column names to the DataFrame
book_3_df.columns = ["Index","Number of occurrences"]
# Add a column containing the token corresponding to the dictionary index
book_3_df["Book3_token"] = [dict[index] for index in book_3_df["Index"]]
# Sort the DataFrame by descending number of occurrences and print the first 10 values
```

```
book3_df_head = book_3_df.sort_values(by="Number of occurrences", ascending=False).head(7)
book3_df_head
get_ipython().run_line_magic('matplotlib', 'inline')
plt.bar(book3_df_head["Book3_token"], book3_df_head["Number of occurrences"], color = 'brown', width = 0.5)
plt.xlabel("Stemmed words", color = "red")
plt.ylabel("Number of occurrences", color = "red")
plt.title("Top 7 Most common words of 'The Law of Attraction, The Secret Power Of The Universe' ", color="red" )
plt.xticks(rotation=0, color= "blue")
plt.yticks(color='blue')
#plt.savefig("words.pdf")
plt.show()
#book 4
# Convert the Bag of words model for book 4 into a DataFrame
book_4_df = pd.DataFrame(bag_of_words[3])
# Add the column names to the DataFrame
book_4_df.columns = ["Index","Number of occurrences"]
# Add a column containing the token corresponding to the dictionary index
book_4_df["Book4_token"] = [dict[index] for index in book_4_df["Index"]]
# Sort the DataFrame by descending number of occurrences and print the first 10 values
book4_df_head = book_4_df.sort_values(by="Number of occurrences", ascending=False).head(7)
book4_df_head
get_ipython().run_line_magic('matplotlib', 'inline')
plt.bar(book4_df_head["Book4_token"], book4_df_head["Number of occurrences"], color = 'brown', width = 0.5)
plt.xlabel("Stemmed words", color = "red")
plt.ylabel("Number of occurrences", color = "red")
plt.title("Top 7 Most common words of 'Payoff The Hidden Logic That Shapes Our Motivations' ", color="red" )
plt.xticks(rotation=0, color= "blue")
plt.yticks(color='blue')
#plt.savefig("words.pdf")
plt.show()
# In[42]:
#Book 5
# Convert the Bag of words model for book 5 into a DataFrame
book_5_df = pd.DataFrame(bag_of_words[4])
# Add the column names to the DataFrame
book_5_df.columns = ["Index","Number of occurrences"]
# Add a column containing the token corresponding to the dictionary index
book_5_df["Book5_token"] = [dict[index] for index in book_5_df["Index"]]
# Sort the DataFrame by descending number of occurrences and print the first 10 values
book5_df_head = book_5_df.sort_values(by="Number of occurrences", ascending=False).head(7)
book5_df_head
get_ipython().run_line_magic('matplotlib', 'inline')
plt.bar(book5_df_head["Book5_token"], book5_df_head["Number of occurrences"], color = 'brown', width = 0.5)
plt.xlabel("Stemmed words", color = "red")
plt.ylabel("Number of occurrences", color = "red")
plt.title("Top 7 Most common words of 'The Motivation Hacker ' ", color="red" )
plt.xticks(rotation=0, color= "blue")
plt.yticks(color='blue')
#plt.savefig("words.pdf")
plt.show()
# ### Building a term frequency - inverse document frequency model(tf-idf model)
# Load the gensim functions that will allow us to generate tf-idf models
from gensim.models import TfidfModel
# Generate the tf-idf model
tf_model = TfidfModel(bag_of_words)
# #### Inspecting the tf_idf words of each book seperately
# ## Book1
# Print the model for book 1
tf_model[bag_of_words[0]]
# Convert the tf-idf model for book 1 into a DataFrame
df_tf_idf = pd.DataFrame(tf_model[bag_of_words[0]])
# Name the columns of the DataFrame id and score
df_tf_idf.columns=["id", "tf_idf_score"]
# Add the tokens corresponding to the numerical indices for better readability
df_tf_idf['Book1_words'] = [dict[i] for i in list(df_tf_idf["id"])]
# Sort the DataFrame by descending tf-idf score and print the first 7 rows.
book1_tfidf_head = df_tf_idf.sort_values(by="tf_idf_score", ascending=False).head(5)
```

```
book1_tfidf_head
get_ipython().run_line_magic('matplotlib', 'inline')
plt.bar(book1_tfidf_head["Book1_words"], book1_tfidf_head["tf_idf_score"], color = 'purple', width = 0.5)
plt.xlabel("Book1 Words", color = "green")
plt.ylabel("TF-IDF Scores", color = "green")
plt.title("Top 5 most significant words of 'Addiction, Procrastination, and Laziness' ", color="red" )
plt.xticks(rotation=0, color= "blue")
plt.yticks(color='blue')
#plt.savefig("words.pdf")
plt.show()
# ### Book 2
# Print the model for book 2
#tf_model[bag_of_words[1]]
# Convert the tf-idf model for book 1 into a DataFrame
df_tf_idf = pd.DataFrame(tf_model[bag_of_words[1]])
# Name the columns of the DataFrame id and score
df_tf_idf.columns=["id", "tf_idf_score"]
# Add the tokens corresponding to the numerical indices for better readability
df_tf_idf['Book2_words'] = [dict[i] for i in list(df_tf_idf["id"])]
# Sort the DataFrame by descending tf-idf score and print the first 10 rows.
book2_tfidf_head = df_tf_idf.sort_values(by="tf_idf_score", ascending=False).head(5)
book2_tfidf_head
get_ipython().run_line_magic('matplotlib', 'inline')
plt.bar(book2_tfidf_head["Book2_words"], book2_tfidf_head["tf_idf_score"], color = 'purple', width = 0.5)
plt.xlabel("Book2 Words", color = "green")
plt.ylabel("TF-IDF Scores", color = "green")
plt.title("Top 5 most significant words of 'Drive The Surprising Truth about What Motivates Us' ", color="red" )
plt.xticks(rotation=0, color= "blue")
plt.yticks(color='blue')
#plt.savefig("words.pdf")
plt.show()
# Print the model for book 3
#tf_model[bag_of_words[2]]
# Convert the tf-idf model for book 1 into a DataFrame
df_tf_idf = pd.DataFrame(tf_model[bag_of_words[2]])
# Name the columns of the DataFrame id and score
df_tf_idf.columns=["id", "tf_idf_score"]
# Add the tokens corresponding to the numerical indices for better readability
df_tf_idf['Book3_words'] = [dict[i] for i in list(df_tf_idf["id"])]
# Sort the DataFrame by descending tf-idf score and print the first 10 rows.
book3_tfidf_head = df_tf_idf.sort_values(by="tf_idf_score", ascending=False).head(5)
book3_tfidf_head
get_ipython().run_line_magic('matplotlib', 'inline')
plt.bar(book3_tfidf_head["Book3_words"], book3_tfidf_head["tf_idf_score"], color = 'purple', width = 0.5)
plt.xlabel("Book3 Words", color = "green")
plt.ylabel("TF-IDF Scores", color = "green")
plt.title("Top 5 most significant words of 'Law of Attraction The Secret Power of The Universe' ", color="red" )
plt.xticks(rotation=0, color= "blue")
plt.yticks(color='blue')
#plt.savefig("words.pdf")
plt.show()
# Print the model for book 4
#tf_model[bag_of_words[3]]
# Convert the tf-idf model for book 1 into a DataFrame
df_tf_idf = pd.DataFrame(tf_model[bag_of_words[3]])
# Name the columns of the DataFrame id and score
df_tf_idf.columns=["id", "tf_idf_score"]
# Add the tokens corresponding to the numerical indices for better readability
df_tf_idf['Book4_words'] = [dict[i] for i in list(df_tf_idf["id"])]
# Sort the DataFrame by descending tf-idf score and print the first 10 rows.
book4_tfidf_head = df_tf_idf.sort_values(by="tf_idf_score", ascending=False).head(5)
get_ipython().run_line_magic('matplotlib', 'inline')
plt.bar(book4_tfidf_head["Book4_words"], book4_tfidf_head["tf_idf_score"], color = 'purple', width = 0.5)
plt.xlabel("Book4 Words", color = "green")
plt.ylabel("TF-IDF Scores", color = "green")
plt.title("Top 5 most significant words of 'Payoff The Hidden Logic That Shapes Our Motivations' ", color="red" )
plt.xticks(rotation=0, color= "blue")
plt.yticks(color='blue')
#plt.savefig("words.pdf")
plt.show()
# Print the model for book 5
#tf_model[bag_of_words[4]]
# Convert the tf-idf model for book 1 into a DataFrame
df_tf_idf = pd.DataFrame(tf_model[bag_of_words[4]])
# Name the columns of the DataFrame id and score
df_tf_idf.columns=["id", "tf_idf_score"]
# Add the tokens corresponding to the numerical indices for better readability
df_tf_idf['Book5_words'] = [dict[i] for i in list(df_tf_idf["id"])]
# Sort the DataFrame by descending tf-idf score and print the first 10 rows.
book5_tfidf_head = df_tf_idf.sort_values(by="tf_idf_score", ascending=False).head(5)
get_ipython().run_line_magic('matplotlib', 'inline')
plt.bar(book5_tfidf_head["Book5_words"], book5_tfidf_head["tf_idf_score"], color = 'purple', width = 0.5)
plt.xlabel("Book5 Words", color = "green")
plt.ylabel("TF-IDF Scores", color = "green")
plt.title("Top 5 most significant words of 'The Motivation Hacker' ", color="red" )
plt.xticks(rotation=0, color= "blue")
plt.yticks(color='blue')
#plt.savefig("words.pdf")
plt.show()
# #### Measuring the distance between texts
# Load the library allowing similarity computations
from gensim import similarities
# Compute the similarity matrix (pairwise distance between all texts)
similarity = similarities.MatrixSimilarity(tf_model[bag_of_words])
# Transform the resulting list into a DataFrame
similarity_df = pd.DataFrame(list(similarity))
# Add the titles of the books as columns and index of the DataFrame
similarity_df.columns = titles
similarity_df.index = titles
# Print the resulting matrix
similarity_df
# ### Comparing the contents of one book to other books
# ####  Books most similar to "Drive The Surprising Truth about What Motivates Us"
#Comparing books with respect to "Drive The Surprising Truth about What Motivates Us"
# This is needed to display plots in a notebook
get_ipython().run_line_magic('matplotlib', 'inline')
# Select the column corresponding to "Drive The Surprising Truth about What Motivates Us" and
compare = similarity_df["Drive The Surprising Truth about What Motivates Us"]
# Sort by ascending scores
compare_sorted = compare.sort_values(ascending=True)
#storing all books except "Addiction, Procrastination, and Laziness" to a variable
excluded_book = compare_sorted.iloc[[0,1,2,3]]
# Plot this data has a horizontal bar plot
excluded_book.plot.barh(x='lab', y='val', rot=0, color = "green").plot()
# Modify the axes labels and plot title for better readability
plt.xlabel("Cosine distance", color = "red")
plt.ylabel("")
plt.title("Most similar books to 'Drive The Surprising Truth about What Motivates Us'", color = "Blue")
plt.yticks(color='red')
plt.show()
# ### Performing Hierarchical clustering  based on Single, complete and average linkages.
#Single linkage
cluster = hierarchy.linkage(similarity_df, 'single')
# Display this result as a dendrogram
p = hierarchy.dendrogram(Z,  leaf_font_size=12, labels=similarity_df.index,  orientation="top", leaf_rotation=90)
# Complete linkage
cluster = hierarchy.linkage(similarity_df, 'complete')
# Display this result as a dendrogram
p = hierarchy.dendrogram(Z,  leaf_font_size=12, labels=similarity_df.index,  orientation="top", leaf_rotation=90)
# Average linkage
Z = hierarchy.linkage(similarity_df, 'average')
# Display this result as a dendrogram
a = hierarchy.dendrogram(Z,  leaf_font_size=12, labels=similarity_df.index,  orientation="top", leaf_rotation=90)
```

References

[1] Datacamp: https://app.datacamp.com/learn

[2] Prashant Srivastava, Non-alphanumeric characters, https://www.geeksforgeeks.org/how-to-remove-all-non-alphanumeric-characters-from-a-string-in-java/

[3] Chetna Khanna, Text pre-processing: Stop words removal using different libraries, https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a

[4] Inflection, https://en.wikipedia.org/wiki/Inflection

[5] Anjali Ganesh Jivani et al. A comparative study of stemming algorithms.Int. J. Comp.Tech. Appl, 2(6):1930–1938, 2011.

[6]Gensim,https://radimrehurek.com/gensim/auto_examples/core/run_core_concepts.html

[7] Gensim, https://anaconda.org/anaconda/gensim

[8] Genism, https://www.tutorialspoint.com/gensim/gensim_creating_a_dictionary.htm

[9] William Scott, TF-IDF from scratch in python on a real-world dataset. https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089

[10] Genism, TF-IDF model, https://rare-technologies.com/pivoted-document-length-normalisation/

[11] Kranti Ghag and Ketan Shah. Sentitfidf–sentiment classification using relative term frequency-inverse document frequency.International Journal of Advanced ComputerScience and Applications, 5(2), 2014.

[12] Ari Aulia Hakim, Alva Erwin, Kho I Eng, Maulahikmah Galinium, and Wahyu Muliady.Automated document classification for news articles in Bahasa Indonesia based on term frequency-inverse document frequency (tf-idf) approach. In2014 6th international conference on information technology and electrical engineering (ICEE), pages 1–4. IEEE,2014.

[13] Lukáš Havrlant and Vladik Kreinovich. A simple probabilistic explanation of term frequency-inverse document frequency (tf-IDF) heuristic (and variations motivated by this explanation).International Journal of General Systems, 46(1):27–36, 2017.

[14] Unfold data science, Euclidean Manhattan and Cosine Distance | Euclidean distance vs Cosine similarity, https://www.youtube.com/watch?v=mWmotJRSHJU

[15] scipy.cluster.hierarchy.linkage,
https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html

[16] Stephen Robertson. Understanding inverse document frequency: on theoretical argu-ments for idf.Journal of documentation, 2004

[17] Lailil Muflikhah and Baharum Baharudin. Document clustering using concept space andcosine similarity measurement. In2009 International conference on computer technologyand development, volume 1, pages 58–62. IEEE, 2009

[18] Anna Huang et al. Similarity measures for text document clustering. InProceedingsof the sixth new zealand computer science research student conference (NZCSRSC2008),Christchurch, New Zealand, volume 4, pages 9–56, 2008.

[19] Advanced analysis | cluster analysis | segmentation | using displayr, dendrogram, https://www.displayr.com/what-is-dendrogram