[RECOMMENDATION SYSTEM]

Group Name: Group B

Group Members:

First name	Last Name	Student number
Priyanshi	Chakraborty	(C0765384)
Parth Dhirenkumar	Patel	(C0765143)
Gurpeet Singh	Virdi	(C0762178)
Manoj	Moond	(C0761024)

Submission date: ***19th Aug, 2020***

Contents

RECOMMENDATION SYSTEM

FINAL REPORT
2020

ABSTRACT:

In the age online entertainment platform, we know that most of the blockbusters be it series or movies are based on the adaption of books. We wanted to give users a platform to author, direct, produce and choose good content for their entertainment. Imagine, how great it would be watch all the good fictions turn into a movie. Most of the upcoming authors don't find a platform to publish their work and vice versa many directors and producers don't always get the good content to make movies on and sometimes even after making movie they are not aware of the story would be perceived by the audience.

Using the recommendation system, we created movie recommendation system and book recommendation system.

Movie recommendation System would help the movie makers to perceive the genre of the movie based on the ratings.

Book Recommendation System would help categorize the book genre and based on the user ratings would help movie makers to decide on the content.

Also, it would act as platform for newbie writers and authors to show their talent.

INTRODUCTION

"A recommender system, or a recommendation system (sometimes replacing 'system' with a synonym such as platform or engine), is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. They are primarily used in commercial applications."

- The explosive growth in the amount of available digital information and the number of visitors to the Internet have created a potential challenge of information overload which hinders timely access to items of interest on the Internet. Information retrieval systems, such as Google, DevilFinder and Altavista have partially solved this problem but prioritization and personalization (where a system maps available content to user's interests and preferences) of information were absent. This has increased the demand for recommender systems more than ever before. Recommender systems are information filtering systems that deal with the problem of information overload by filtering vital information fragment out of large amount of dynamically generated information according to user's preferences, interest, or observed behavior about item. Recommender system has the ability to predict whether a particular user would prefer an item or not based on the user's profile.

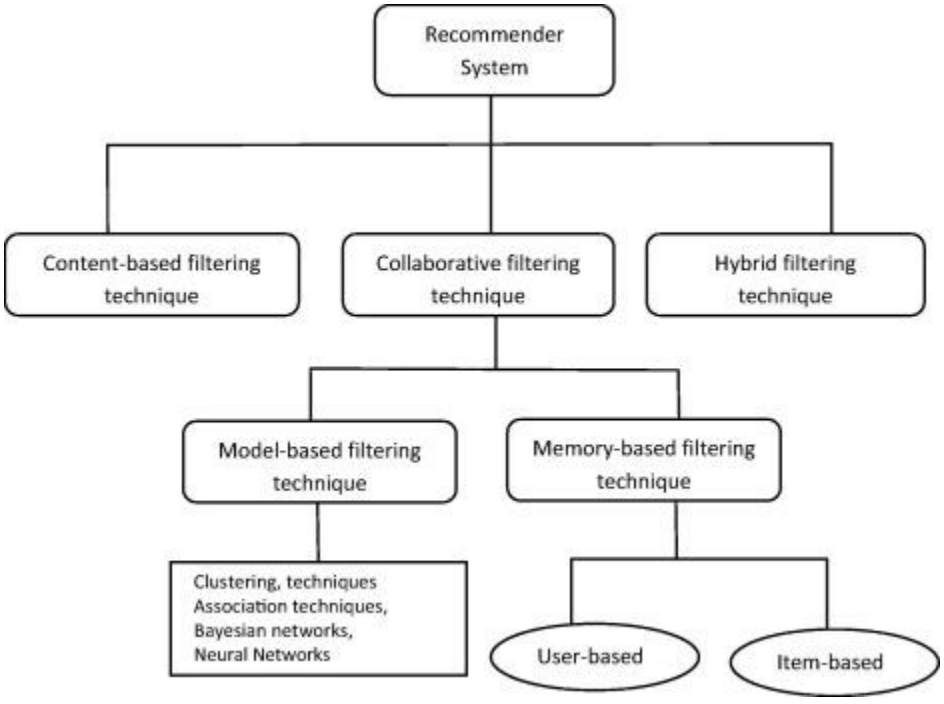
- Recommender systems are beneficial to both service providers and users. They reduce transaction costs of finding and selecting items in an online shopping environment. Recommendation systems have also proved to improve decision making process and quality. In e-commerce setting, recommender systems enhance revenues, for the fact that they are effective means of selling more products. In scientific libraries, recommender systems support users by allowing them to move beyond catalog searches. Therefore, the need to use efficient and accurate recommendation techniques within a system that will provide relevant and dependable recommendations for users cannot be over-emphasized.

TYPES OF RECOMMMENDATION

Recommender system is defined as a decision-making strategy for users under complex information environments. Also, recommender system was defined from the perspective of E-commerce as a tool that helps users search through records of knowledge which is related to users' interest and preference. Recommender system was defined as a means of assisting and augmenting the social process of using recommendations of others to make choices when there is no sufficient personal knowledge or experience of the alternatives. Recommender systems handle the problem of information overload that users normally encounter by providing them with personalized, exclusive content and service recommendations. Recently, various approaches for building recommendation systems have been developed, which can utilize either collaborative filtering, content-based filtering or hybrid filtering. Collaborative filtering technique is the most mature and the most commonly implemented. Collaborative filtering recommends items by identifying other users with similar taste; it uses their opinion to recommend items to the active user. Collaborative recommender systems have been implemented in different application areas. Group Lens is a news-based architecture which employed collaborative methods in assisting users to locate articles from massive news database. Ringo is an online social

information filtering system that uses collaborative filtering to build users profile based on their ratings on music albums. Amazon uses topic diversification algorithms to improve its recommendation. The system uses collaborative filtering method to overcome scalability issue by generating a table of similar items offline through the use of item-to-item matrix. The system then recommends other products which are

similar online according to the users' purchase history. On the other hand, content-based techniques match content resources to user characteristics. Content-based filtering techniques normally base their predictions on user's information, and they ignore contributions from other users as with the case of collaborative techniques. Fab relies heavily on the ratings of different users in order to create a training set and it is an example of content-based recommender system. Some other systems that use content-based filtering to help users find information on the Internet include Letizia. The system makes use of a user interface that assists users in browsing the Internet; it is able to track the browsing pattern of a user to predict the pages that they may be interested in. Pazzani designed an intelligent agent that attempts to predict which web pages will interest a user by using naive Bayesian classifier. The agent allows a user to provide training instances by rating different pages as either hot or cold. Jennings and Higuchi describe a neural network that models the interests of a user in a Usenet news environment.



LIMITATIONS & OVERCOMING OF LIMITATIONS

Despite the success of these two filtering techniques, several limitations have been identified. Some of the problems associated with content-based filtering techniques are limited content analysis, overspecialization and sparsity of data. Also, collaborative approaches exhibit cold-start, sparsity and scalability problems. These problems usually reduce the quality of recommendations. In order to mitigate some of the problems identified, Hybrid filtering, which combines two or more filtering techniques in different ways in order to increase the accuracy and performance of recommender systems has been proposed. These techniques combine two or more filtering approaches in order to harness their strengths while leveling out their corresponding weaknesses. They can be classified based on their operations into

weighted hybrid, mixed hybrid, switching hybrid, feature-combination hybrid, cascade hybrid, feature-augmented hybrid and meta-level hybrid. Collaborative filtering and content-based filtering approaches are widely used today by implementing content-based and collaborative techniques differently and the results of their prediction later combined or adding the characteristics of content-based to collaborative filtering and vice versa. Finally, a general unified model which incorporates both content-based and collaborative filtering properties could be developed. The problem of sparsity of data and cold-start was

addressed by combining the ratings, features and demographic information about items in a cascade hybrid recommendation technique in. In Ziegler et al., a hybrid collaborative filtering approach was proposed to exploit bulk taxonomic information designed for exacting product classification to address the data sparsity problem of CF recommendations, based on the generation of profiles via inference of super-topic score and topic diversification. A hybrid recommendation technique is also proposed in Ghazantar and Pragel-Benett, and this uses the content-based profile of individual user to find similar users which are used to make predictions. In Sarwar et al., collaborative filtering was combined with an information filtering agent. Here, the authors proposed a framework for integrating the content-based filtering agents and collaborative filtering. A hybrid recommender algorithm is employed by many applications as a result of new user problem of content-based filtering techniques and average user problem of collaborative filtering. A simple and straightforward method for combining content-based and collaborative filtering was proposed by Cunningham. A music recommendation system which combined tagging information, play counts and social relations was proposed in Konstas. In order to determine the number of neighbors that can be automatically connected on a social platform, Lee and Brusilovsky embedded social information into collaborative filtering algorithm. A Bayesian mixed-effects model that integrates user ratings, user and item features in a single unified framework was proposed by Condiff.

PHASES OF RECOMMENDATION PROCESS

INFORMATION COLLECTION PHASE

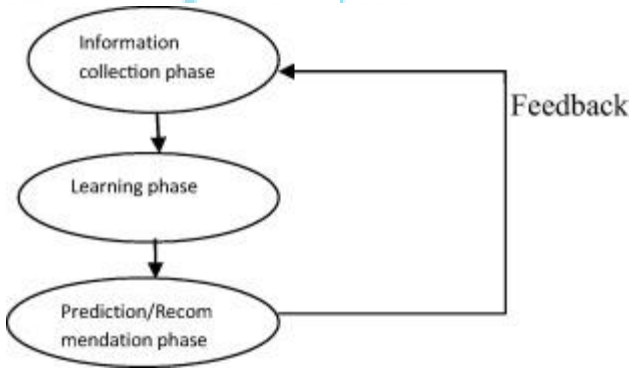
This collects relevant information of users to generate a user profile or model for the prediction tasks including user's attribute, behaviors or content of the resources the user accesses. A recommendation agent cannot function accurately until the user profile/model has been well constructed. The system needs to know as much as possible from the user in order to provide reasonable recommendation right from the onset.

LEARNING PHASE

It applies a learning algorithm to filter and exploit the user's features from the feedback gathered in information collection phase.

PREDICTION/RECOMMENDATION PHASE

It recommends or predicts what kind of items the user may prefer. This can be made either directly based on the dataset collected in information collection phase which could be memory based or model based or through the system's observed activities of the user.



FEEDBACK

Recommender systems rely on different types of input such as:

1) EXPLICIT FEEDBACK

The system normally prompts the user through the system interface to provide ratings for items in order to construct and improve his model. The accuracy of recommendation depends on the quantity of ratings provided by the user. The only shortcoming of this method is, it requires effort from the users and also, users are not always ready to supply enough information. Despite the fact that explicit feedback requires more effort from user, it is still seen as providing more reliable data, since it does not involve extracting preferences from actions, and it also provides transparency into the recommendation process that results in a slightly higher perceived recommendation quality and more confidence in the recommendations.

2) IMPLICIT FEEDBACK

The system automatically infers the user's preferences by monitoring the different actions of users such as the history of purchases, navigation history, and time spent on some web pages, links followed by the user, content of e-mail and button clicks among others. Implicit feedback reduces the burden on users by inferring their user's preferences from their behavior with the system. The method though does not require effort from the user, but it is less accurate. Also, it has also been argued that implicit preference data might in actuality be more objective, as there is no bias arising from users responding in a socially desirable way and there are no self-image issues or any need for maintaining an image for others.

3) HYBRID FEEDBACK

The strengths of both implicit and explicit feedback can be combined in a hybrid system in order to minimize their weaknesses and get a best performing system. This can be achieved by using an implicit data as a check on explicit rating or allowing user to give explicit feedback only when he chooses to express explicit interest.

CONTENT-BASED FILTERING

In our project we have chosen to go with Content-based technique as it is a domain-dependent algorithm and it emphasizes more on the analysis of the attributes of items in order to generate predictions. When documents such as web pages, publications and news are to be recommended, content-based filtering technique is the most successful. In content-based filtering technique, recommendation is made based on the user profiles using features extracted from the content of the items the user has evaluated in the past.

Items that are mostly related to the positively rated items are recommended to the user. CBF uses different types of models to find similarity between documents in order to generate meaningful recommendations. It could use Vector Space Model such as Term Frequency Inverse Document Frequency (TF/IDF) or Probabilistic models such as Naïve Bayes Classifier, Decision Trees or Neural Networks to model the relationship between different documents within a corpus. These techniques make recommendations by learning the underlying model with either statistical analysis or machine learning techniques. Content-based filtering technique does not need the profile of other users since they do not influence recommendation.

Also, if the user profile changes, CBF technique still has the potential to adjust its recommendations within a very short period of time. The major disadvantage of this technique is the need to have an in-depth knowledge and description of the features of the items in the profile.

PROS AND CONS OF CONTENT-BASED FILTERING TECHNIQUES

CB filtering techniques overcome the challenges of CF. They have the ability to recommend new items even if there are no ratings provided by users. So even if the database does not contain user preferences, recommendation accuracy is not affected. Also, if the user preferences change, it has the capacity to adjust its recommendations in a short span of time. They can manage situations where different users do not share the same items, but only identical items according to their intrinsic features. Users can get recommendations without sharing their profile, and this ensures privacy. CBF technique can also provide explanations on how recommendations are generated to users. However, the techniques suffer from various problems as discussed in the literature. Content based filtering techniques are dependent on items' metadata. That is, they require rich description of items and very well-organized user profile before recommendation can be made to users. This is called limited content analysis. So, the effectiveness of CBF depends on the availability of descriptive data. Content overspecialization is another serious problem of CBF technique. Users are restricted to getting recommendations similar to items already defined in their profiles.

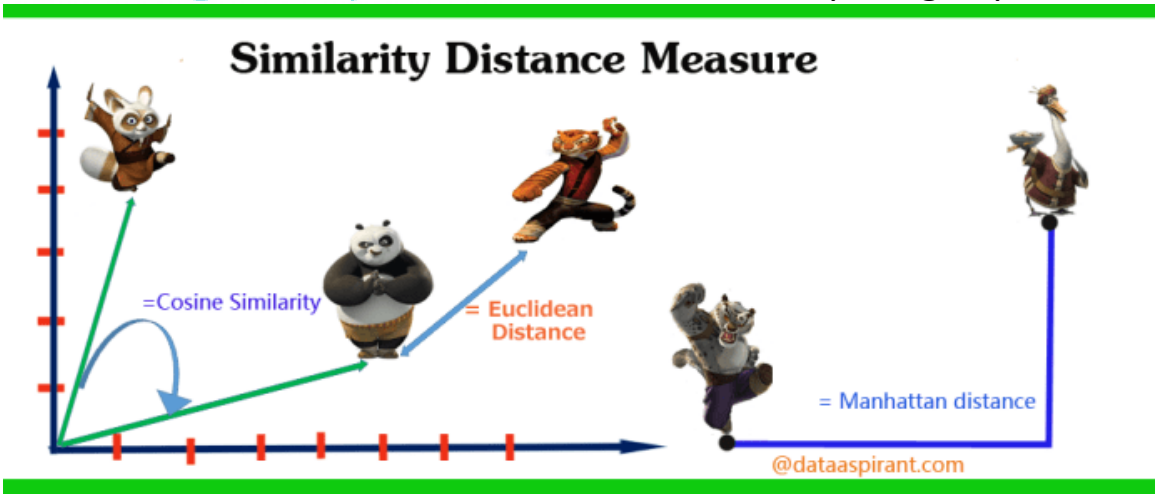
BOOK RECOMMENDATION SYSTEM

As discussed above, Content-based recommendation system recommends items to a user by taking similarity of items. This recommender system recommends products or items based on the description or features. It identifies the similarity between the products based on its description. It also considers the user previous history in order to recommend a similar product.

Example: If a user likes novel "Tell me your dreams" by Sidney Sheldon, then the recommender system recommends the user to read other Sidney Sheldon's novels or it recommends novel with the genre "Non-fiction". (Sidney Sheldon novels belong to Non-fiction genre).

We are using goodreads.com data and don't have users reading history. Hence, we have used a simple content-based recommendation system. We are going to build two recommendation system by using a book title and book description.

We need to find similar books to a given book and then recommend those similar books to the user. Similarity measure was used to find the same. There are different similarity measures are available. Cosine Similarity was used in our recommender system to recommend the books.



CODE

In [13]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from sklearn.metrics.pairwise import linear_kernel
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import RegexpTokenizer
import re
import string
import random
from PIL import Image
import requests
from io import BytesIO
import matplotlib.pyplot as plt
%matplotlib inline
```

In [14]:
```
# Reading the file
df = pd.read_csv("goodreads.csv", encoding = "ISO-8859-1", engine='python')
```

In [15]:
```
#Reading the first five records
df.head()
```

Out[15]:

	Id	title	genre	Unnamed: 3	authors	Rating	Desc	Unnamed: 7	Unnamed: 8	Unnamed: 9
0	1	Harry Potter and the Half-Blood Prince (Harry ...	Fantasy Fiction	NaN	J.K. Rowling	4.57	The war against Voldemort is not going well; e...	NaN	NaN	NaN
1	2	Harry Potter and the Order of the Phoenix (Har...	Fantasy Fiction	NaN	J.K. Rowling	4.50	There is a door at the end of a silent corrido...	NaN	NaN	NaN
2	3	Harry Potter and the Sorcerer's Stone (Harry P...	Fantasy Fiction	NaN	J.K. Rowling	4.47	Harry□s perfectly normal life at number 4 priv...	NaN	NaN	NaN
3	4	Harry Potter and the Chamber of Secrets (Harry...	Fantasy Fiction	NaN	J.K. Rowling	4.42	Harry Potter is about to start his second year...	NaN	NaN	NaN
4	5	Harry Potter and the Prisoner of Azkaban (Harr...	Fantasy Fiction	NaN	J.K. Rowling	4.57	Harry Potter and the Prisoner of Azkaban is th...	NaN	NaN	NaN

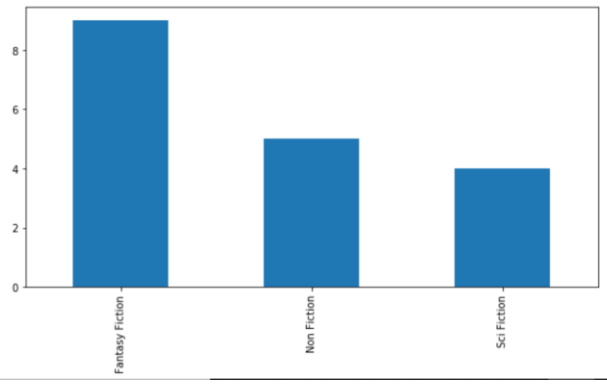
In [16]:
```
#Checking the shape of the file
df.shape
```

Out[16]: (18, 10)

In [17]:
```
#Exploratory Data Analysis
```

In [18]:
```
# Genre distribution
df['genre'].value_counts().plot(x = 'genre', y ='count', kind = 'bar', figsize = (10,5)  )
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x228da9ce488>
```



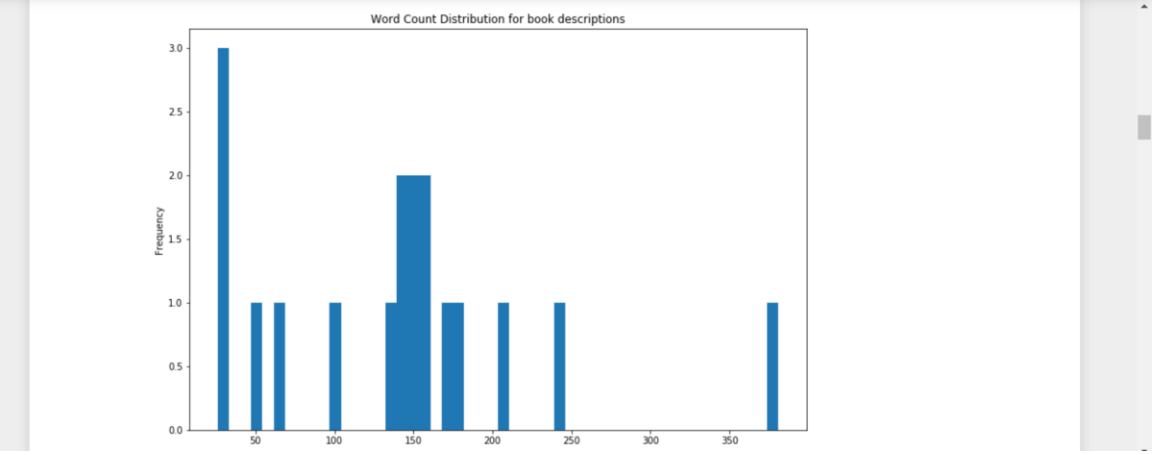
In [63]:
```
# Printing the book title and description randomly
df['title']
df['Desc']
```

Out[63]:
```
0     The war against Voldemort is not going well; e...
1     There is a door at the end of a silent corrido...
2     Harry□s perfectly normal life at number 4 priv...
3     Harry Potter is about to start his second year...
4     Harry Potter and the Prisoner of Azkaban is th...
5     Schools come together for an ultimate competit...
6     Author J. K. Rowling does something amazing wi...
7     The war against Voldemort is not going well; e...
8     Author J. K. Rowling does something amazing wi...
9     Seconds before the Earth is demolished to make...
10    ""The Hitchhiker's Guide to the Galaxy"\n\nSec...
11    "Seconds before the Earth is demolished to mak...
12    "Suppose a good friend calmly told you over a ...
13    Bryson's biggest book, he confronts his greate...
14    In the early fall of 2002, famed travel Writer...
15    A revised and updated edition of a humorous pr...
16    Despite the fact that Australia harbors more t...
17    After living in Britain for two decades, Bill ...
Name: Desc, dtype: object
```

In [64]:
```
#Book description – Word count distribution
```

In [107]:
```
# Calculating the word count for book description
df['word_count'] = df['Desc'].apply(lambda x: len(str(x).split()))
# Plotting the word count
df['word_count'].plot(
    kind='hist',
    bins = 50,
    figsize = (12,8),title='Word Count Distribution for book descriptions')
```

Out[107]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x228e20bf608>
```



Natural Language Processing (NLP)

Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language. The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable.

Natural Language Processing is the driving force behind the following common applications:

- Language translation applications such as Google Translate
- Word Processors such as Microsoft Word and Grammarly that employ NLP to check grammatical accuracy of texts.
- Interactive Voice Response (IVR) applications used in call centers to respond to certain users' requests.
- Personal assistant applications such as OK Google, Siri, Cortana, and Alexa.

Natural language toolkit (NLTK) is the most popular library for natural language processing (NLP) which was written in Python and has a big community behind it.

In linguistics, a corpus (plural corpora) or text corpus is a large and structured set of texts. In corpus linguistics, they are used to do statistical analysis and hypothesis testing, checking occurrences or validating linguistic rules within a specific language territory.

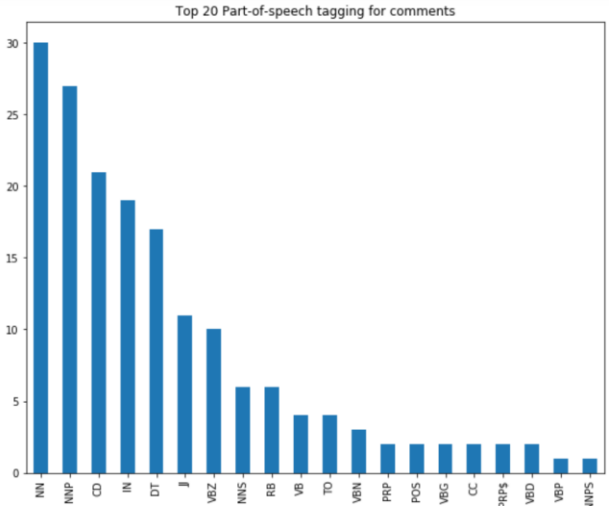
Each corpus reader class is specialized to handle a specific corpus format. In addition, the nltk.corpus package automatically creates a set of corpus reader instances that can be used to access the corpora in the NLTK data package.

When we import nltk a separate window opens, we have to download all the packages before using nltk package.

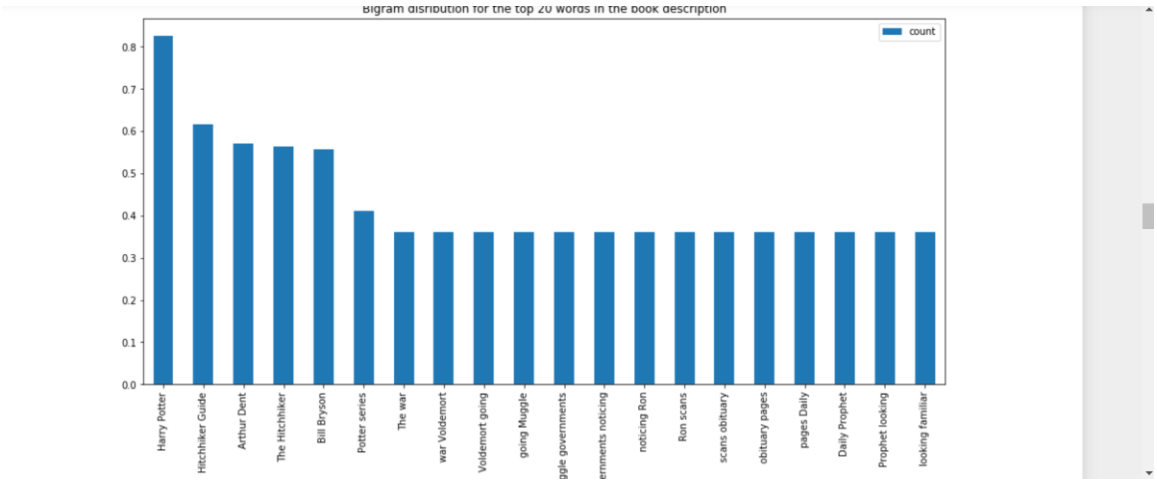

```
In [66]: #import nltk
         #nltk.download()
         #It will open a page for corpus download.Download all the packages
```

```
In [67]: #The distribution of top part-of-speech tags in the book descriptions
         from textblob import TextBlob
         blob = TextBlob(str(df['Desc']))
         pos_df = pd.DataFrame(blob.tags, columns = ['word' , 'pos'])
         pos_df = pos_df.pos.value_counts()[:20]
         pos_df.plot(kind = 'bar', figsize=(10, 8), title = "Top 20 Part-of-speech tagging for comments")
```

```
Out[67]: <matplotlib.axes._subplots.AxesSubplot at 0x228e15f6ac8>
```



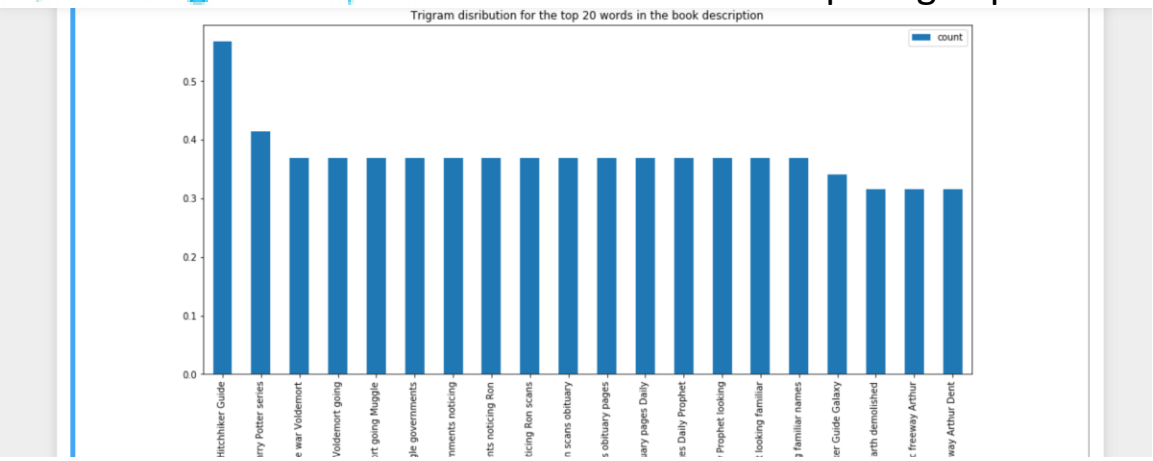
```
In [68]: #Converting text descriptions into vectors using TF-IDF using Bigram
tf = TfidfVectorizer(ngram_range=(2, 2), stop_words='english', lowercase = False)
tfidf_matrix = tf.fit_transform(df['Desc'])
total_words = tfidf_matrix.sum(axis=0)
#Finding the word frequency
freq = [(word, total_words[0, idx]) for word, idx in tf.vocabulary_.items()]
freq =sorted(freq, key = lambda x: x[1], reverse=True)
#converting into dataframe
bigram = pd.DataFrame(freq)
bigram.rename(columns = {0:'bigram', 1: 'count'}, inplace = True)
#Taking first 20 records
bigram = bigram.head(20)

In [69]: #Plotting the bigram distribution
bigram.plot(x ='bigram', y='count', kind = 'bar', title = "Bigram disribution for the top 20 words in the book description", figs

Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x228e1c601c8>
```



TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. ... It has many uses, most importantly in automated text analysis, and is very useful for scoring words in machine learning algorithms for Natural Language Processing (NLP).

TF-IDF, which stands for term frequency — inverse document frequency, is a scoring measure widely used in information retrieval (IR) or summarization. TF-IDF is intended to reflect how relevant a term is in a given document.

```
In [70]: #Converting text descriptions into vectors using TF-IDF using Trigram
tf = TfidfVectorizer(ngram_range=(3, 3), stop_words='english', lowercase = False)
tfidf_matrix = tf.fit_transform(df['Desc'])
total_words = tfidf_matrix.sum(axis=0)
#Finding the word frequency
freq = [(word, total_words[0, idx]) for word, idx in tf.vocabulary_.items()]
freq =sorted(freq, key = lambda x: x[1], reverse=True)
#converting into dataframe
trigram = pd.DataFrame(freq)
trigram.rename(columns = {0:'trigram', 1: 'count'}, inplace = True)
#Taking first 20 records
trigram = trigram.head(20)
#Plotting the trigramn distribution
trigram.plot(x ='trigram', y='count', kind = 'bar', title = "Trigram disribution for the top 20 words in the book description", f

Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x228e1c86648>
```




```
In [71]: #Text Preprocessing
         #Cleaning the book description.
         # Function for removing NonAscii characters
         def _removeNonAscii(s):
             return "".join(i for i in s if  ord(i)<128)
         # Function for converting into lower case
         def make_lower_case(text):
             return text.lower()
         # Function for removing stop words
         def remove_stop_words(text):
             text = text.split()
             stops = set(stopwords.words("english"))
             text = [w for w in text if not w in stops]
             text = " ".join(text)
             return text
         # Function for removing punctuation
         def remove_punctuation(text):
             tokenizer = RegexpTokenizer(r'\w+')
             text = tokenizer.tokenize(text)
             text = " ".join(text)
             return text
         #Function for removing the html tags
         def remove_html(text):
             html_pattern = re.compile('<.*?>')
             return html_pattern.sub(r'', text)
         # Applying all the functions in description and storing as a cleaned_desc
         df['cleaned_desc'] = df['Desc'].apply(_removeNonAscii)
         df['cleaned_desc'] = df.cleaned_desc.apply(func = make_lower_case)
         df['cleaned_desc'] = df.cleaned_desc.apply(func = remove_stop_words)
```

Recommendation engine

We are going to build two recommendation engine using book title and description.

Convert each book title and description into vectors using TF-IDF and bigram. For more details on TF-IDF

We are building two recommendation engines, one with a book title and another one with a book description. The model recommends a similar book based on title and description.

Calculate the similarity between all the books using cosine similarity.

Cosine Similarity

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis. A document can be represented by thousands of attributes, each recording the frequency of a particular word (such as a keyword) or phrase in the document. Thus, each document is an object represented by what is called a term-frequency vector.

Term-frequency vectors are typically very long and sparse (i.e., they have many 0 values). Applications using such structures include information retrieval, text document clustering, biological taxonomy, and gene feature mapping. The traditional distance measures that we have studied in this chapter do not work well for such sparse numeric data. For example, two term-frequency vectors may have many 0 values in common, meaning that the corresponding documents do not share many words, but this does not make them similar. We need a measure that will focus on the words that the two documents do have in common, and the occurrence frequency of such words. In other words, we need a measure for numeric data that ignores zero-matches.

Cosine similarity is a measure of similarity that can be used to compare documents or, say, give a ranking of documents with respect to a given vector of query words. Let x and y be two vectors for compariso|n. Using the cosine measure as a similarity function, we have

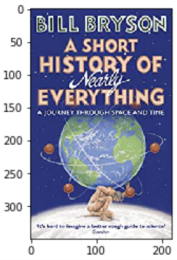
sin(x,y) = x.y/||x||y||

where ||x|| is the Euclidean norm of vector , defined as . Conceptually, it is the length of the vector. Similarly, ||y|| is the Euclidean norm of vector y. The measure computes the cosine of the angle between vectors x and y. A cosine value of 0 means that the two vectors are at 90 degrees to each other (orthogonal) and have no match. The closer the cosine value to 1, the smaller the angle and the greater the match between vectors. Note that because the cosine similarity measure does not obey all of the properties of Section 2.4.4 defining metric measures, it is referred to as a nonmetric measure.

```
In [72]: from sklearn.metrics.pairwise import cosine_similarity

In [73]: #Recommendation based on book title
         # Function for recommending books based on Book title. It takes book title and genre as an input.
         def recommend(title, genre):

             # Matching the genre with the dataset and reset the index
             data = df.loc[df['genre'] == genre]
             data.reset_index(level = 0, inplace = True)

             # Convert the index into series
             indices = pd.Series(data.index, index = data['title'])

             #Converting the book title into vectors and used bigram
             tf = TfidfVectorizer(analyzer='word', ngram_range=(2, 2), min_df = 1, stop_words='english')
             tfidf_matrix = tf.fit_transform(data['title'])

             # Calculating the similarity measures based on Cosine Similarity
             sg = cosine_similarity(tfidf_matrix, tfidf_matrix)

             # Get the index corresponding to original_title

             idx = indices[title]
         # Get the pairwsie similarity scores
             sig = list(enumerate(sg[idx]))
         # Sort the books
             sig = sorted(sig, key=lambda x: x[1], reverse=True)
         # Scores of the 5 most similar books
             sig = sig[1:6]
         # Book indicies
             movie_indices = [i[0] for i in sig]

             # Top 5 book recommendation
             rec = data[['title', 'url']].iloc[movie_indices]

             # It reads the top 5 recommend book url and print the images

             for i in rec['url']:
                 response = requests.get(i)
                 img = Image.open(BytesIO(response.content))
                 plt.figure()
                 print(plt.imshow(img))

In [80]: recommend("In a Sunburned Country", "Non Fiction")
```

Define a function that takes book title and genre as an input and returns the top five similar recommended books based on the title and description.

```
9Ps63/Du7f7NPbdttsfX1k28eXH/Qp27t9f3nICNH58fQ3p9nXjx3dv8AZJ2xu23r+8c+PLu1SdfVunpOQkhH5sXQ3p9nVv46v7Ig/c4/nA2eONTnYV/7rH+uc3ItKvHx9Ddn2f/Z'
```



We are using the same above function by converting book description into vectors.

In [103]:
```
#Recommendation based on book description
#We are using the same above function by converting book description into vectors.
# Function for recommending books based on Book title. It takes book title and genre as an input.
def recommend(title, genre):

    global rec
    # Matching the genre with the dataset and reset the index
    data = df.loc[df['genre'] == genre]
    data.reset_index(level = 0, inplace = True)
# Convert the index into series
    indices = pd.Series(data.index, index = data['title'])
    #Converting the book description into vectors and used bigram
    tf = TfidfVectorizer(analyzer='word', ngram_range=(2, 2), min_df = 1, stop_words='english')
    tfidf_matrix = tf.fit_transform(data['cleaned_desc'])
     # Calculating the similarity measures based on Cosine Similarity
    sg = cosine_similarity(tfidf_matrix, tfidf_matrix)
    # Get the index corresponding to original_title

    idx = indices[title]
    # Get the pairwsie similarity scores
    sig = list(enumerate(sg[idx]))
    # Sort the books
    sig = sorted(sig, key=lambda x: x[1], reverse=True)
# Scores of the 5 most similar books
    sig = sig[1:6]
# Book indicies
    movie_indices = [i[0] for i in sig]
```

```
    # Top 5 book recommendation
    rec = data[['title']].iloc[movie_indices]

    # It reads the top 5 recommend book url and print the images

    for i in rec['url']:
        response = requests.get(i)
        img = Image.open(BytesIO(response.content))
        plt.figure()
        print(plt.imshow(img))
```

In [105]:
```
recommend("Harry Potter and the Prisoner of Azkaban","Fantasy Fiction")
```

```
Harry Potter and the Goblet of Fire
Harry Potter and the Deathly Hallows
Harry Potter and the Chamber of Secrets
Harry Potter and the Sorcerer's Stone
Harry Potter and the Order of the Phoenix
```

Movie Recommendation System

Code

```
# Libraries
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from ast import literal_eval

#top rows of data
print('Top Rows of Data: ','\n', data.head())

# Dataset
data = pd.read_csv("C:/Users/Vinay Moond/PycharmProjects/TermProject/movies_metadata.csv",

# Mean of average vote
mean_vote = data['vote_average'].mean()
print('The mean vote across the whole report: ', mean_vote)

# Minimun vote required
min_vote = data['vote_count'].quantile(0.90)
print('The minimum votes required to be listed in the chart: ',min_vote)
```

```
# new dataset with qualified movies
movies = data.copy().loc[data['vote_count'] >= min_vote]
print('Size of Movies dataset: ','\n',movies.shape)
print('Size of Original dataset: ','\n',data.shape)
```

```
#Weighted Rating Fumction
def WR_function(x, min_vote=min_vote, mean_vote=mean_vote):
    vote_count = x['vote_count']
    vote_avg = x['vote_average']
    return (vote_count/(vote_count+min_vote) * vote_avg) + (min_vote/(min_vote+vote_count
```

```
# New column defing the weighted rating
movies['WR'] = movies.apply(WR_function, axis=1)
print('Top Rows of Data: ','\n',movies.head())

#Sorting according to WR
movies = movies.sort_values('WR', ascending=False)

#top 15 movies
print('Top 15 Rows of Data: ','\n',movies[['title', 'vote_count', 'vote_average', 'WR']].h

#overview from data
print('Overview from Data: ','\n',movies['overview'].head())
```

```
#Replacing NaN with an empty string
movies['overview'] = movies['overview'].fillna('')

#term frequency-inverse document frequency vectorizer object
tfidf = TfidfVectorizer(stop_words='english')

#term frequency-inverse document frequency matrix
t_matrix = tfidf.fit_transform(movies['overview'])
print('Term Frequency-Inverse Document Frequency Matrix: ','\n',t_matrix)

#cosine similarity matrix
c_matrix = linear_kernel(t_matrix, t_matrix)
print('Cosine Similarity Matrix: ','\n',c_matrix)
```

```
#reverse map of indices and movie titles
indices = pd.Series(movies.index, index=movies['title']).drop_duplicates()
print('Reverse Map of Indices and Movie Titles: ','\n',indices[:10])
```

```
# Recommendation function
def recommendations(title, c_matrix=c_matrix):
    index = indices[title]

    similarity_scores = list(enumerate(c_matrix[index]))

    similarity_scores = sorted(similarity_scores, key=lambda x: x[1], reverse=True)

    similarity_scores = similarity_scores[1:11]

    movies_title = [i[0] for i in similarity_scores]

    return movies['title'].iloc[movies_title]
```

```
#trying if recommendation system works
print('Recommendations based on the movie are: ','\n',recommendations('The Godfather'))
```

```
Recommendations based on the movie are:
 5345          K-19: The Widowmaker
1189                      Das Boot
2965       The World Is Not Enough
5743                        Solaris
4238     Atlantis: The Lost Empire
40024            Deepwater Horizon
461           Hot Shots! Part Deux
897          2001: A Space Odyssey
7101          The Butterfly Effect
10384                    Flightplan
```

Conclusion:

We see both the book and movie recommendation model recommends a similar books and movie based on title and description. Though both the models are only prototype as due to lack of funds we were not able to transfer the model to cloud and our normal servers were not able to process the big data required for these models.

Future Possibilities:

As discussed earlier, when the project will be moved to cloud and we would be working with Big Data. We can merge movie recommendation system and book recommendation system.

Movie recommendation System would help the movie makers to perceive the genre of the movie based on the ratings.

Book Recommendation System would help categorize the book genre and based on the user ratings would help movie makers to decide on the content.

Also, it would act as platform for newbie writers and authors to show their talent.

References:

https://www.google.com/search?rlz=1C1CHBF_enIN862IN862&sxsrf=ALeKk031EDWXV2KJ13UINsJyxR1fUzhMoA%3A1597891607499&ei=F-Q9X66JHubB_QaQlrfoCQ&q=natural+language+processing+using+nltk&oq=natural+language+processing+nltk&gs_lcp=CgZwc3ktYWIQARgBMgIIADIGCAAQFhAeMgYIABAWEB4yBggAEBYQHjIGCAAQFhAeMgYIABAWEB4yBggAEBYQHjIGCAAQFhAeMgYIABAWEB4yBggAEBYQHjoECAAQRzoECAAQQzoHCAAQFBCHAjoICAAQFhAKEB5QsRVYrSBgszVoAHABeACAAZEBiAHoA5IBAzMuMpgBAKABAaoBB2d3cy13aXrAAQE&sclient=psy-ab

https://www.sciencedirect.com/science/article/pii/S1110866515000341

https://www.sciencedirect.com/topics/computer-science/cosine-similarity

https://www.codecademy.com/learn/natural-language-processing?g_acctid=243-039-7011&g_keywordid=kwd-35559750&g_adid=435749871838&g_keyword=natural%20language%20processing&g_campaign=CA+Language%3A+Pro+-+Exact&g_adtype=search&g_network=g&g_adgroupid=102650148953&g_campaignid=10074200777&utm_id=t_kwd-35559750:ag_102650148953:cp_10074200777:n_g:d_c&utm_term=natural%20language%20processing&utm_campaign=&utm_source=google&utm_medium=paid-search&utm_content=435749871838&hsa_acc=2430397011&hsa_cam=10074200777&hsa_grp=102650148953&hsa_ad=435749871838&hsa_src=g&hsa_tgt=kwd-35559750&hsa_kw=natural%20language%20processing&hsa_mt=e&hsa_net=adwords&hsa_ver=3&gcl