A PROJECT REPORT

on

"A Comparative Study of Similarity Measures Used in Collaborative Filtering and Incorporating Sentiment Analysis to Enhance Recommendations"

Submitted to
KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of

BACHELOR'S DEGREE IN
INFORMATION TECHNOLOGY

BY

BISWAJEET MOHANTY	2005158
DHYANAM ATTREYA	2005160
HAR SWARANSH BAHADUR SINHA	2005162
ANCHIT KUMAR SHARMA	2005215

UNDER THE GUIDANCE OF
Dr. RABI SHAW

SCHOOL OF COMPUTER ENGINEERING
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA - 751024

KIIT Deemed to be University

School of Computer Engineering
Bhubaneswar, ODISHA 751024

CERTIFICATE

This is certify that the project entitled

"Comparative Study of Similarity Measures Used in Collaborative Filtering and Incorporating Sentiment Analysis to Enhance Recommendations"

submitted by

BISWAJEET MOHANTY	2005158
DHYANAM ATTREYA	2005160
HAR SWARANSH	2005162
BAHADUR SINHA ANCHIT KUMAR SHARMA	2005215

is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & Engineering OR Information Technology) at KIIT Deemed to be university, Bhubaneswar. This work is done during year 2022-2023, under our guidance.

Date:5 / 5 / 2023

Acknowledgements

We are profoundly grateful to **Dr. Rabi Shaw** of **School of Computer Engineering** for his expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion which enabled our team to do the project with much ease and helped us to learn many new things while carrying out this project.

BISWAJEET MOHANTY
DHYANAM ATTREYA
HAR SWARANSH BAHADUR SINHA
ANCHIT KUMAR SHARMA

ABSTRACT

In recent years, there has been a significant growth in the number of users who have shown an inclination towards cinema and OTT platforms. According to a report published by **Statista**, the number of cinema-goers worldwide has increased from around **3.6 billion** in 2010 to **4.2 billion** in 2019. Additionally, the global revenue generated by the film industry has grown from around **88.3 billion** U.S. dollars in 2010 to **100 billion** U.S. dollars in 2019. A large part of the boom is an end result of highly advanced recommendation algorithms running in the backend that enhance the user experience by providing personalized recommendations.

The study aims to develop a movie recommendation system that utilizes collaborative filtering techniques and natural language processing to provide personalized movie recommendations to users. The study will compare the effectiveness of four different similarity measures, including cosine similarity, jaccard similarity, pearson correlation coefficient, and chebyshev similarity.

User reviews from **IMDb** will be scraped for movies in the dataset and analyzed using sentiment analysis to calculate the leniency score of a user. The leniency score will be used to determine if a user has any biasness while rating movies. By assigning weightage to the leniency score and similarity measure, the system will generate movie recommendations for the users which will be displayed on our website.

The primary objective of the study is to evaluate the effectiveness of different similarity measures and to determine if users exhibit any biasness while rating movies. Overall, this project aims to enhance the accuracy of movie recommendations and improve user satisfaction.

Keywords: collaborative filtering, natural language processing, cosine similarity,jaccard similarity,pearson correlation coefficient,chebyshev similarity,leniency score,weightage,biasness,user satisfaction.

Contents

List of Figures

Chapter 1

Introduction

COVID-19 pandemic has led to a surge in the number of users opting for movie streaming platforms, with a 19% increase in subscribers for Netflix alone in 2020 (CNBC). This trend is expected to continue in the coming years, with the global video streaming market projected to grow from 50.11 billion U.S. dollars in 2020 to 108.31 billion by 2026 (Business Wire).

However due to the large volume of data that is flowing in continuously , it becomes seemingly difficult to find relevant movies for users to watch. This is where various recommendation techniques come into play.To make more user specific and personalized recommendations, users past activity(**Content Based**) and similarity with other users/items(**Collaborative Filtering**)are taken into account.However,the challenges that the current systems face is the heavy reliance on past data for making recommendations(can cause **cold start problem**), without taking into account change in user's preference and sentiments over time, which would lead to inaccurate recommendations.

Our study is centered towards Collaborative Filtering, analyzing the similarity measures used and addressing the current problems by using a customized approach to make recommendations that would target both **rating** and **reviews** to filter out movies.
We devise a new method to approach Sentiment Analysis which involves understanding the leniency of user while rating the movies based on the entire set of movies he/she has watched and try to derive conclusions based on our findings.

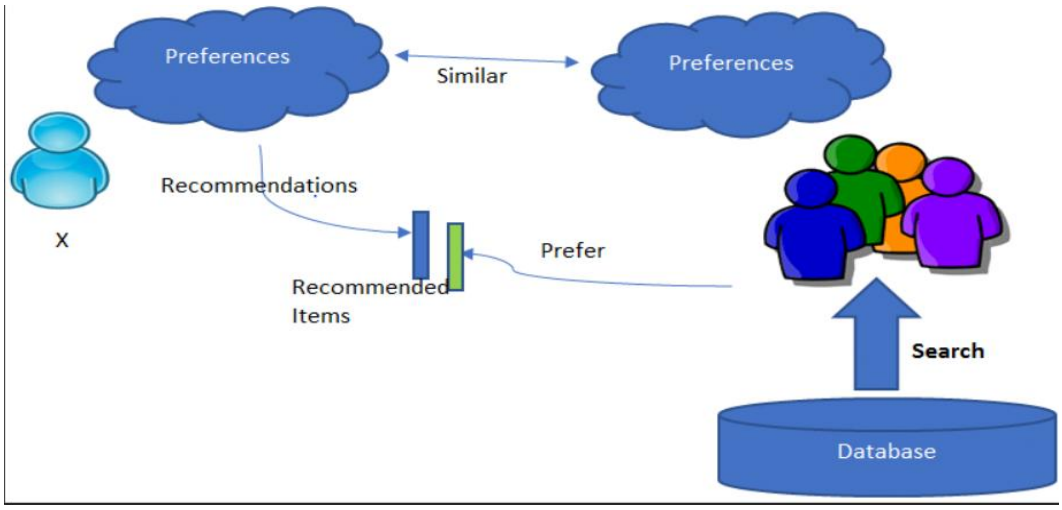


Diagram illustrating how Collaborative Filtering works

Chapter 2

Literature Review

Numerous researches have been conducted and there are still ongoing researches that tend to incorporate sentiment based analysis for better recommendations. Various techniques for sentiment analysis such as lexicon-based,machine learning based,hybrid based and aspect based have emerged over the course of time and are being heavily used in a variety of recommender systems.Here are the list of related works that address movie recommendation and sentiment analysis:

2.1 Related Work:

2.1.1 “A hybrid movie recommendation system based on sentiment analysis and matrix factorization” by Zhang and Li (2017):

The authors propose a hybrid movie recommendation system that combines sentiment analysis with matrix factorization. They use a sentiment-based approach to generate a feature matrix that captures the user's sentiment toward different movie aspects.

2.1.2 "An efficient movie recommendation system using sentiment analysis and cosine similarity measure" by Bansal et al. (2019):

The authors propose a movie recommendation system that uses sentiment analysis and cosine similarity measure. They use a lexicon-based approach for sentiment analysis and calculate the cosine similarity between the user's rating vector and the movie's feature vector to generate recommendations.

2.2 Limitations of Existing Work:

Some studies fail to address the cold start system which problem limit the ability of a system to generate accurate recommendations for new users whereas some studies are not able to identify the weightage that needs to be assigned to the user bias while designing recommendation systems which reflects how heavily a user’s rating pattern is influenced by his/her own bias.

Chapter 3

Problem Statement / Requirement Specifications

The primary objective of the study is to create an effective recommendation system by incorporating similarity methods and user leniency score.Based on their assigned weightage, we find a combined score which will be utilized to recommend movies.The aim is to see whether inclusion of leniency scores brings a significant change in the recommended movies .A sub study also involves finding the best suited similarity measure (cosine,jaccard,pearson,chebyshev) that will be used in calculating the combined score.

3.1 Project Planning

The project will be developed using Python programming language.For dealing with the dataset we'll be using libraries like **NumPy** and **Pandas** . For better visulaization we'll be having graphical representations of end result generated using **Matplotlib** and **Seaborn.** The project will be divided into various stages such as data collection,data preprocessing,model evaluation, testing and finally making recommendations.We'll be scrapping reviews from IMDb corresponding to the movies that we'll be having in our dataset using the **BeautifulSoup** library.The last stage of the project would involve building a user friendly interface displaying recommended movies whose frontend will be designed using **HTML** and **CSS** and backend implementation will be taken care by **Flask** framework.The additional details of the recommended movies(release date,poster etc.) would be retrieved by using **OMDB API**.

3.2 Project Analysis

The project involves a comparative study of four different types of similarity measures used in collaborative filtering: Cosine similarity, Jaccard similarity, Pearson correlation coefficient, and Chebyshev similarity. User reviews from IMDB are scrapped and natural language processing is applied to get a rating based on the reviews. A new technique is used to do sentiment analysis and calculate the leniency score of users. Recommendations are then made based on a combination of leniency score and similarity measure. The project aims to investigate if users have an element of biasness while rating movies by incorporating reviews and seeing if recommended movies differ from those which were recommended by taking only similarity measures. The project analysis will involve an in-depth study of the implementation details, data preprocessing, results obtained, and analysis of the findings .

3.3 System Design

3.3.1 Design Constraints

The system is capable of processing a large dataset with considerable amount of speed.The implementation part was carried out in jupyter notebook which offers a variety of tools to deal effectively with data.

The harder part was getting considerable number of reviews from IMDb for movies in our dataset.Some movies were very old and data availability for them was sparse.Finally we narrowed it down to 100 reviews per movie.

The other problem we faced while designing was that some movie titles were in there original language and the API(used in the backend) could retrieve information for english titles only.To deal with this we had to make some changes in movie titles.

3.3.2 System Architecture/Block Diagram

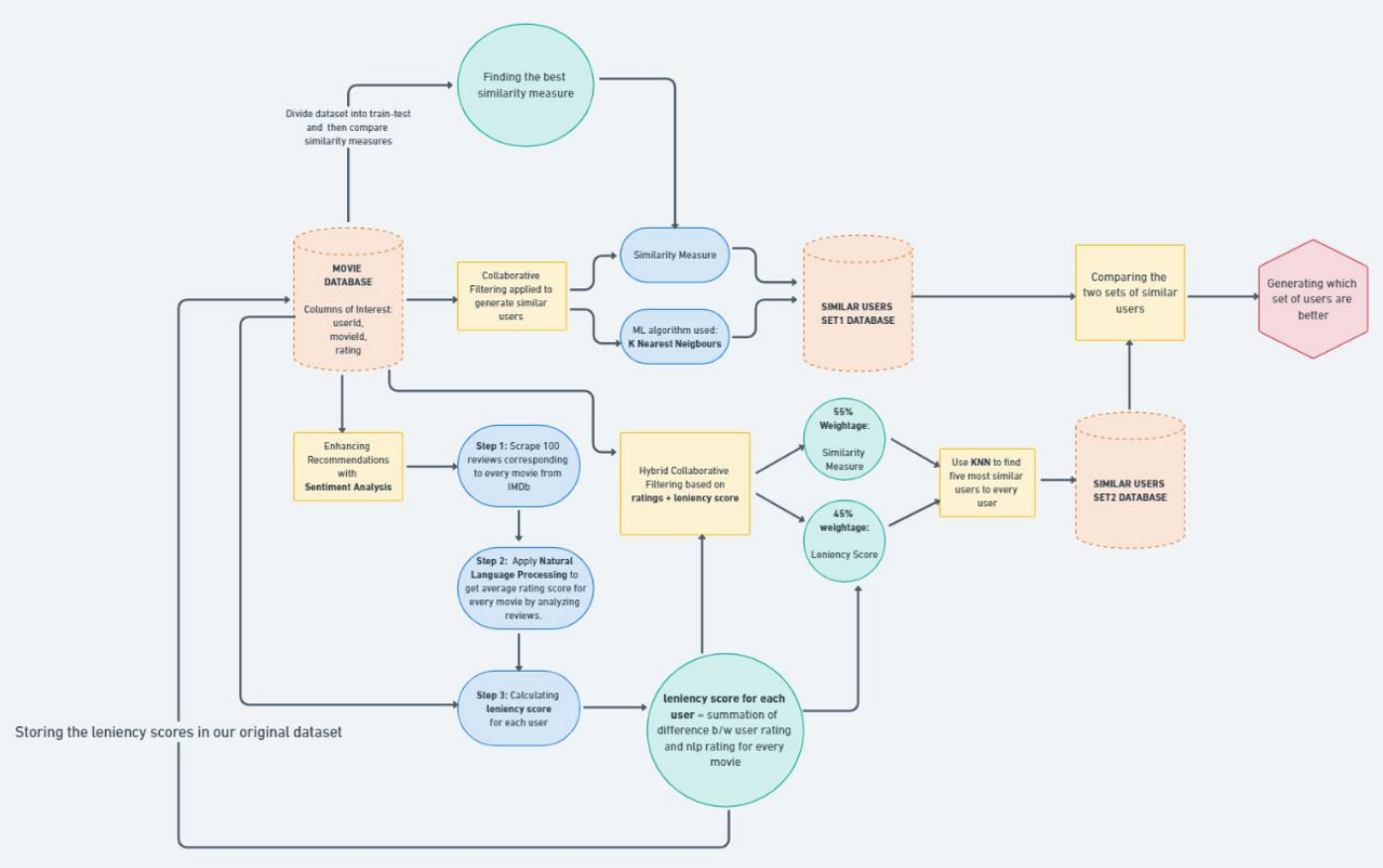


Diagram representing the workflow of model

The system architecture will consist of the following components:

Data Collection: This component will involve scraping user reviews from IMDb using web scraping tools like BeautifulSoup and obtaining the rating dataset from Kaggle.

Sentiment Analysis: This component will involve processing and analyzing the scrapped data to do sentiment analysis using NLP technique: Vader.Vader will be tokenizing the individual reviews into words and generating sentiment score for every review which will be equal to the aggreagte sentiment score of all words in the review.

Recommendation System: This component will involve building a recommendation system that utilizes the outputs of the similarity measures and sentiment analysis algorithms to provide personalized movie recommendations to users.

Webpage : This will be utilized to display top 5 recommended movies corresponding to every user present in the dataset.

Chapter 4

Implementation

4.1 Methodology

The proposed study aims to develop a effective recommender system that will be recommending movies for the users based on their rating similarity with other users and their own scale of leniency while rating movies.

1) Preprocessing involves compressing the dataset to have only 1219 users , assigning the nan values 0 rating and changing movie titles in other languages to English.

2) From the original dataset , unique movies titles are extracted & reviews for them are scrapped from IMDb for Sentiment Analysis and stored in another database.

3) Natural Language Processing is used to generate an average rating per movie by analyzing the reviews.We had used Vader from NLTK that assigns each word in a review a sentiment score based on its context and final score is aggregated score of all the words in the review.

4) We then find the optimal measure for similarity.First we creating a rating_matrix with user_id as rows and movie_id as columns and ratings as the value at that particular row and column.

5) We then randomly mask indices in the ratings_matrix(assigning 0 rating to some rated movies).The four similarity measures were used to predict the masked ratings and by comparing the predicted ratings with the actual, the overall accuracy,precision,mean squared error,f1 score and recall was calculated.We then analyzed these metrics to decide the final similarity measure.

6) Next step involves calculating the leniency score for each user.Leniency score was the summation of differences of user rating and rating generated through nlp for each movie that the user has watched, normalized on a scale of 0-1.

7) We then tried to find out if these leniency scores impacted our recommendations or not . Two dataframes are created one for storing the set of 5 similar users corresponding to every user based on solely the similarity measure and the other by assigning weightage to both leniency score and similarity measure and there differences are compared.

8) Then we observe difference in movies recommended to each user both by inclusion & exclusion of leniency score.

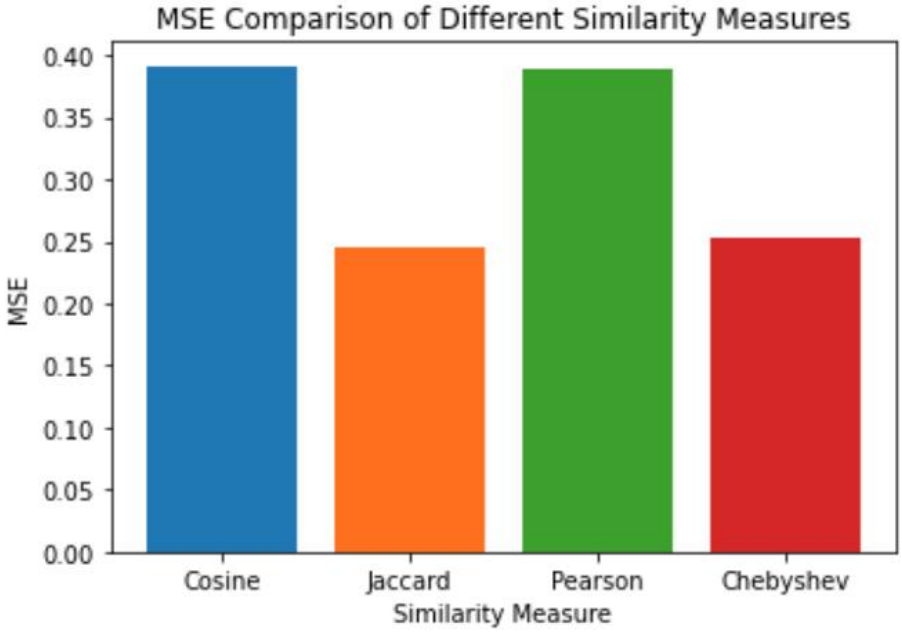
9) Finally we developed a website for recommending movies wherein the backend was developed using Flask, to provide an API for the frontend to interact with & integrated the

backend with the movie recommendation model to provide personalized recommendations to users.The frontend was created using HTML and CSS and JavaScript.

4.2 Testing/Verification Plan

SIMILARITY MEASURE VS PERFORMANCE METRICS

	PRECISION	ACCURACY	F1 SCORE	RECALL	MSE
JACCARD	0.875780	0.974051	0.927307	0.985277	0.244787
COSINE	0.967417	0.973244	0.891819	0.827180	0.391326
PEARSON	0.966043	0.973607	0.893596	0.831257	0.389072
CHEBYSHEV	0.876272	0.980552	0.931678	0.994563	0.253686



Based on the above findings , we decided to use both cosine and jaccard and compare the results obtained by inclusion and exclusion of leniency score in both cases. Reasons for choosing cosine was the high precision in case of cosine and since cosine has been used quite a lot of time in sentiment analysis based recommendations, we wanted to compare it's result with the result we'll obtain by using other similarity measures(Jaccard in this case). Jaccard

was used because it seemed an ideal fit, due to the balance between it's recall and precision scores and comparatively high accuracy and low mean squared error.

4.3 Result Analysis or Screenshots

4.3.1 : Loading and Preprocessing of Data

```
complete_df = pd.read_csv(r'final_dataset(2).csv')
# dropping unnecessary columns
complete_df = complete_df.drop(columns='Unnamed: 0',axis=1)
# reducing the number of users we'll be dealing with(compressing dataset):
complete_df = complete_df[complete_df['userId']<=1219]
# replacing titles in original language to english
complete_df['original_title']=complete_df['original_title'].replace({'Trois couleurs : Rouge':'Three Colors: Red','Les Quatre
complete_df
```

Scrapping Code:

```
# Importing required python libraries,pandas: for data manipulation,requests: for making HTTP
# requests to the IMDb website and BeautifulSoup: for parsing HTML content.

import pandas as pd
import requests
from bs4 import BeautifulSoup

# this is a fnc that takes a single argument 'movie_id' as function parameter which is actually
# the unique id of the movie provided in the IMDb url.

def scrape_reviews(movie_id):

    # initializing an empty list 'reviews' , which will later be used to store the scraped reviews.
    reviews = []

    # If we observe the IMDb url structure,on searching for different movies the movie_id is the
    # part of the url that continuously changes and therefore we declare a variable 'url' that
    # will contain the IMDb url for the movie reviews page,with the {} placeholder replaced by
    # the movie_id argument.
    url = 'https://www.imdb.com/title/{}/reviews/_ajax?ref_=undefined&paginationKey='.format(movie_id)

    # pagination_key is a variable that will be used to retrieve addtional pages of reviews because
    # by default only 25 reviews are displayed per page in the IMDb review section and we want
    # 100 reviews per movie for our dataset.
    pagination_key = ''

    # So we start a while loop that will run until our review list length is less than 100.
    while len(reviews) < 100:

        # Here, we make an HTTP get request to the url variable to get the HTML content of the url
        # and the 'pagination_key' variable is appended to the end of the url to retrieve the content
        # of the specific pagination key, which is further used to access additional pages of reviews
        # beyond the first page.
        response = requests.get(url + pagination_key)

        # creating BeautifulSoup object from the HTML content retrieved from the IMDb website.
        soup = BeautifulSoup(response.text, 'html.parser')

        # This loop iterates through all the 'div' elements with the 'text' class,which contain
```

```
        for review in soup.find_all('div', {'class': 'text'}):
            review_text = review.get_text().strip()
            reviews.append(review_text)
            if len(reviews) == 100:
                break

        # This finds the div element with the load-more-data class, which contains a data-key
        # attribute that can be used to retrieve additional pages of reviews. If this element
        # exists, its data-key value is assigned to the pagination_key variable, which is then
        # appended to the URL for the next page of reviews. If the element does not exist, the loop
        # is exited with a break statement.
        pagination_key_element = soup.find('div', {'class': 'load-more-data'})
        if pagination_key_element:
            pagination_key = pagination_key_element.get('data-key')
        else:
            break

    return reviews

# Create a dictionary to store the reviews for each movie
reviews_dict = {}

# Loop through each movie in the DataFrame and call the scrape_reviews() function to get the review
# The iterrows() function allows us to iterate over the rows of a Pandas dataframe as (index, Series) pairs.
for index, row in movies.iterrows():
    movie_id = row['unique_id']
    reviews = scrape_reviews(movie_id)
    reviews_dict[movie_id] = reviews

# Convert the dictionary to a new DataFrame and export it to a new Excel file
reviews_df = pd.DataFrame.from_dict(reviews_dict, orient='index')
reviews_df.to_excel('reviews_list.xlsx')
```

Out[5]:

	0	1	2	3	4	5	6	7	8	
tt0111495	It's almost impossible for me to sit down and ...	This is the last film of Krzysztof Kieslowski ...	The last film in the Three Colors trilogy, RED...	The final and most haunting of Polish director...	Valentine, a model in France is separated from...	Trilogies are very interesting. Some go out wi...	One of my favorite films of all time. With bea...	The final part of Kieslowski's trilogy based o...	In my analysis of "Trois couleurs: Blanc" I wr...	I found "TI Colors: F to l strange
tt0053198	Every day life, however 'real' and gritty it m...	So here it is, the landmark film which ushered...	THE FOUR HUNDRED BLOWS (François Truffaut - Fr...	The memorable story of young, troubled Antoine...	As the seminal work of the French New Wave, th...	I first saw this film around three or four yea...	The title of the film is The 400 Blows but it ...	This classic and unforgettable movie with amaz...	Les quatre-cents coups is the film that opens ...	Good mo but ha brilliant class
tt0108160	Sleepless in Seattle is manipulative and unash...	I will not lie to you, I will admit right from...	Another lovely film from Nora Ephron in the tr...	The movies are full of alternate universes and...	Annie Reed (Meg Ryan) hears, on a late night t...	Regardless of how cyberspace has seemingly dim...	I won't lie to you-this movie is a CHICK FLICK...	This is a movie with characters and performanc...	After his wife's funeral, Sam (Tom Hanks), an ...	Sleeples Seattle is a film fo

4.3.2) Utilizing Natural Language Processing to generate leniency score for users:

Sentiment Score Generator Code:

```
import pandas as pd
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Create a new column to hold the average sentiment score for each movie
movie_reviews['average_rating'] = 0.0

# Create an instance of the VADER sentiment analyzer
sid = SentimentIntensityAnalyzer()

# Loop over the rows of the dataframe.'i' is the index of the row, and 'row' is the data in that row.
for i, row in movie_reviews.iterrows():

    # Initialize a list to hold the individual review scores for this movie
    review_scores = []

    # This line starts a nested loop that iterates over the first 100 columns in the current row.
    # Each column represents a movie review.
    for j in range(100):

        # Get the text of the review
        review_text = str(row[j+1])

        # Calculate the sentiment score for the review using VADER.
        # The 'compound' score is used, which is a normalized score between -1 (most negative) and +1 (most positive).
        review_score = sid.polarity_scores(review_text)['compound']

        # Append the score to the list of review scores
        review_scores.append(review_score)

    # Calculate the average of the review scores for this movie
    average_score = sum(review_scores) / len(review_scores)

    # Assign the average score to the new column for this row.The average sentiment score multiplied by 2.5
    # and shifted by 1 to map the score to the [0, 5] range.
    movie_reviews.at[i, 'average_rating'] = (average_score + 1) * 2.5  # map score to [0,5]
```

```
movie_reviews
```

Calculating leniency score and normalizing it on scale of (0-1):

```
# Calculate the difference between the rating given by the user and the average rating for each movie
merged_df_new['diff'] = merged_df_new['rating'] - merged_df_new['nltk_rating']

# Group the data by user and sum up the differences to get the total leniency score for each user
leniency_scores_df = merged_df_new.groupby('userId')['diff'].sum().reset_index(name='leniency_score')

# Normalize the scores using Min-Max scaling
min_score = leniency_scores_df['leniency_score'].min()
max_score = leniency_scores_df['leniency_score'].max()
leniency_scores_df['normalized_score'] = (leniency_scores_df['leniency_score'] - min_score) / (max_score - min_score)

# Print the leniency scores for each user
leniency_scores_df
```

]:

	userId	leniency_score	normalized_score
0	1	3.823808	0.791561
1	2	-9.841387	0.413206
2	3	-6.866605	0.495571
3	4	3.649467	0.786734
4	5	0.332933	0.694907
...	...	...	...
890	1213	-3.395695	0.591671
891	1214	7.204382	0.885160
892	1215	0.561880	0.701246
893	1216	1.658765	0.731616
894	1219	-2.338365	0.620946

895 rows × 3 columns

Combined Score Calculation :

(Cosine Similarity Matrix)

	1	2	3	4	5	7	8	9	11	12	...	1207	1208	1209	1211	
1	1.0	0.0	0.2561	0.25102	0.360141	0.477315	0.419518	0.563	0.367229	0.375382	...	0.806617	0.037753	0.360141	0.288113	
2	0.0	1.0	0.0	0.115153	0.0	0.0	0.0	0.0	0.07854	0.139638	...	0.0	0.149374	0.0	0.0	0.65
3	0.2561	0.0	1.0	0.0	0.444444	0.0	0.27735	0.0	0.372556	0.356348	...	0.185025	0.326133	0.444444	0.0	
4	0.25102	0.115153	0.0	1.0	0.0	0.0	0.280166	0.447192	0.069597	0.31497	...	0.294372	0.0	0.0	0.0	0.16
5	0.360141	0.0	0.444444	0.0	1.0	0.0	0.0	0.0	0.0	0.356348	...	0.416305	0.0	1.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1213	0.141446	0.37814	0.106149	0.128672	0.0	0.220477	0.0	0.0	0.458305	0.18913	...	0.154666	0.38946	0.0	0.371521	0.43
1214	0.0	0.480272	0.132175	0.160221	0.0	0.0	0.0	0.0	0.127491	0.0	...	0.0	0.207836	0.0	0.0	0.22
1215	0.346706	0.112749	0.498199	0.139876	0.593442	0.0	0.0	0.255418	0.433095	0.622668	...	0.439206	0.276487	0.593442	0.0	0.15
1216	0.056245	0.0	0.277642	0.0	0.0	0.0	0.0	0.0	0.509304	0.278261	...	0.0	0.736716	0.0	0.0	
1219	0.297306	0.053469	0.238247	0.09747	0.0	0.318119	0.0	0.177983	0.738656	0.286534	...	0.357061	0.56194	0.0	0.0	0.07

895 rows × 895 columns

(User Leniency Matrix)

	1	2	3	4	5	7	8	9	11	12	...	1207	1208	1209	1211
1	0.791561	0.791561	0.791561	0.791561	0.791561	0.791561	0.791561	0.791561	0.791561	0.791561	...	0.791561	0.791561	0.791561	0.791561
2	0.413206	0.413206	0.413206	0.413206	0.413206	0.413206	0.413206	0.413206	0.413206	0.413206	...	0.413206	0.413206	0.413206	0.413206
3	0.495571	0.495571	0.495571	0.495571	0.495571	0.495571	0.495571	0.495571	0.495571	0.495571	...	0.495571	0.495571	0.495571	0.495571
4	0.786734	0.786734	0.786734	0.786734	0.786734	0.786734	0.786734	0.786734	0.786734	0.786734	...	0.786734	0.786734	0.786734	0.786734
5	0.694907	0.694907	0.694907	0.694907	0.694907	0.694907	0.694907	0.694907	0.694907	0.694907	...	0.694907	0.694907	0.694907	0.694907
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1213	0.591671	0.591671	0.591671	0.591671	0.591671	0.591671	0.591671	0.591671	0.591671	0.591671	...	0.591671	0.591671	0.591671	0.591671
1214	0.885160	0.885160	0.885160	0.885160	0.885160	0.885160	0.885160	0.885160	0.885160	0.885160	...	0.885160	0.885160	0.885160	0.885160
1215	0.701246	0.701246	0.701246	0.701246	0.701246	0.701246	0.701246	0.701246	0.701246	0.701246	...	0.701246	0.701246	0.701246	0.701246
1216	0.731616	0.731616	0.731616	0.731616	0.731616	0.731616	0.731616	0.731616	0.731616	0.731616	...	0.731616	0.731616	0.731616	0.731616
1219	0.620946	0.620946	0.620946	0.620946	0.620946	0.620946	0.620946	0.620946	0.620946	0.620946	...	0.620946	0.620946	0.620946	0.620946

(Assign weightage to Similarity Measures(**w1**))

(Assign weightage to User Leniency Scores(**w2**))

(Combined Score Matrix)

= (w1* similarity_measure_matrix) + (w2 * user_leniency_matrix)

	1	2	3	4	5	7	8	9	11	12	...	1207	1208	1209	1211	
1	0.906202	0.356202	0.497057	0.494263	0.55428	0.618725	0.586937	0.665852	0.558178	0.562662	...	0.799841	0.376966	0.55428	0.514664	0.35
2	0.185943	0.735943	0.185943	0.249277	0.185943	0.185943	0.185943	0.185943	0.22914	0.262744	...	0.185943	0.268099	0.185943	0.185943	0.54
3	0.363862	0.223007	0.773007	0.223007	0.467451	0.223007	0.375549	0.223007	0.427912	0.418998	...	0.32477	0.40238	0.467451	0.223007	0.22
4	0.492091	0.417364	0.35403	0.90403	0.35403	0.35403	0.508121	0.599986	0.392308	0.527264	...	0.515935	0.35403	0.35403	0.35403	0.4
5	0.510786	0.312708	0.557153	0.312708	0.862708	0.312708	0.312708	0.312708	0.312708	0.5087	...	0.541676	0.312708	0.862708	0.312708	0.31
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1213	0.344047	0.474229	0.324634	0.337022	0.266252	0.387514	0.266252	0.266252	0.51832	0.370274	...	0.351319	0.480455	0.266252	0.470589	0.50
1214	0.398322	0.662472	0.471018	0.486443	0.398322	0.398322	0.398322	0.398322	0.468442	0.398322	...	0.398322	0.512632	0.398322	0.398322	0.51
1215	0.506249	0.377573	0.58957	0.392492	0.641954	0.315561	0.315561	0.456041	0.553763	0.658028	...	0.557124	0.467629	0.641954	0.315561	0.40
1216	0.360162	0.329227	0.48193	0.329227	0.329227	0.329227	0.329227	0.329227	0.609345	0.482271	...	0.329227	0.734421	0.329227	0.329227	0.32
1219	0.442944	0.308834	0.410462	0.333034	0.279426	0.454391	0.279426	0.377316	0.685687	0.437019	...	0.475809	0.588492	0.279426	0.279426	0.32

895 rows × 895 columns

- Final weightage given to similarity measure = 0.55
- Final weightage given to user leniency = 0.45

4.3.3) Masking indices and splitting dataset into training and testing data:

-> Masking 3 indices (ratings being assigned 0 value) per user:

```
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# creating a masked matrix.Corresponding to every user 3 movie ratings will be masked and there updated values will be stored
# in the masked matrix.
mask_matrix = ratings_matrix.copy()

# will hold the indices(y labels)of the masked ratings corresponding to every row(userId).
masked_indices_dict = {}

# loop over each user:
for user in ratings_matrix.index:

    # get the indices of non-zero ratings for the user
    non_zero_indices = np.where(ratings_matrix.loc[user] != 0)[0]

    non_zero_indices_labels = ratings_matrix.columns[non_zero_indices]

    # randomly choose the indices to mask
    if len(non_zero_indices) >= 3:
        masked_indices = np.random.choice(non_zero_indices, size=3, replace=False)
    else:
        masked_indices = np.concatenate((non_zero_indices, np.random.choice(np.where(ratings_matrix.loc[user] == 0)[0], size=

    # set the masked ratings to 0
    mask_matrix.loc[user, ratings_matrix.columns[masked_indices]] = 0
    masked_indices_dict[user]=ratings_matrix.columns[masked_indices]
```

TESTING RESULTS:

	PRECISION	ACCURACY	F1 SCORE	RECALL	MSE
JACCARD	0.872847	0.979133	0.9276559	0.9857315	0.250154
JACCARD + LENIENCY SCORE	0.873146	0.979133	0.926528	0.986862	0.250154
COSINE	0.966773	0.971885	0.885724	0.817214	0.393557
COSINE + LENIENCY SCORE	0.966319	0.972036	0.886463	0.818799	0.392776

These test results were obtained on a random sample obtained via masking 3 indices per user and predicting the ratings of masked indices.Though the model for ‘jaccard’ didn’t show any changes when combined with ‘leniency score’, the results were very slightly improved in case of cosine.

However, in both cases the p-value test showed a very less value(<0.05) suggesting that the difference in predictions was statistically significant and not likely due to chance.

4.3.4) Comparing list of Similar Users obtained with and without the inclusion of leniency score:

55% weigtage : Cosine

45% weightage: Leniency score

	userId	similar_user_1	similar_user_2	similar_user_3	similar_user_4	similar_user_5
0	1.0	1207.0	719.0	1136.0	672.0	1.0
1	2.0	341.0	191.0	518.0	900.0	2.0
2	3.0	493.0	201.0	521.0	773.0	3.0
3	4.0	53.0	77.0	999.0	692.0	4.0
4	5.0	453.0	1209.0	401.0	959.0	5.0

100% weightage: Cosine

	userId	similar_user_1	similar_user_2	similar_user_3	similar_user_4	similar_user_5
0	1.0	340.0	719.0	672.0	1136.0	1.0
11	2.0	261.0	191.0	400.0	533.0	2.0
28	3.0	24.0	346.0	201.0	150.0	3.0
36	4.0	229.0	392.0	682.0	611.0	4.0
42	5.0	37.0	461.0	727.0	661.0	5.0

```
# get the set of userIds that are present in both datasets
userIds = set(similar_users_df['userId']).intersection(set(similar_users_no_reviews_df['userId']))

# create a list to store the matching userIds
matching_userIds = []

# iterate through the userIds
for userId in userIds:
    # get the rows from both datasets that correspond to this userId
    df1_rows = similar_users_df[similar_users_df['userId'] == userId]
    df2_rows = similar_users_no_reviews_df[similar_users_no_reviews_df['userId'] == userId]

    # get the similar_user_1 and similar_user_2 values for both datasets
    df1_similar_users = set(zip(df1_rows['similar_user_1'],df1_rows['similar_user_2']))
    df2_similar_users = set(zip(df2_rows['similar_user_1'],df2_rows['similar_user_2']))

    # check if the similar_user_1 and similar_user_2 values are the same in both datasets
    if df1_similar_users == df2_similar_users:
        matching_userIds.append(userId)

# print the matching userIds
print(matching_userIds)
```

```
[24.0, 262.0, 717.0, 879.0, 887.0, 1002.0, 1061.0, 1158.0]
```

We see that for only 8 users out of a set of 1219 users the first and second similar users match which suggest that inclusion of leniency score had made a significant change in similarity based on solely similarity measure which suggests us that the similar users in case of purely cosine might have had varied opinions while rating movies.

4.3.5) Recommending Movies :

Since the variations in tested results of ‘jaccard’ were negligible the recommended movies varied for very less users even by tweaking the weightage of leniency score and jaccard similarity.

Recommended movies for user id = 2 using **Jaccard** (as a measure of similarity):

```
Recommended movies for user 2 are:

Young and Innocent
Trois couleurs : Rouge
Sleepless in Seattle
License to Wed
Monsoon Wedding
```

Recommended movies for user id = 2 using **Jaccard + Leniency Score**(as a measure of similarity):

```
Recommended movies for user 2 are:

Once Were Warriors
Young and Innocent
Trois couleurs : Rouge
Sleepless in Seattle
Monsoon Wedding
```

However there was considerable diversity in the recommended movies provided in case cosine making it more susceptible to leniency scores:

Recommended movies for user id = 1 using **Cosine**(as a measure of similarity):

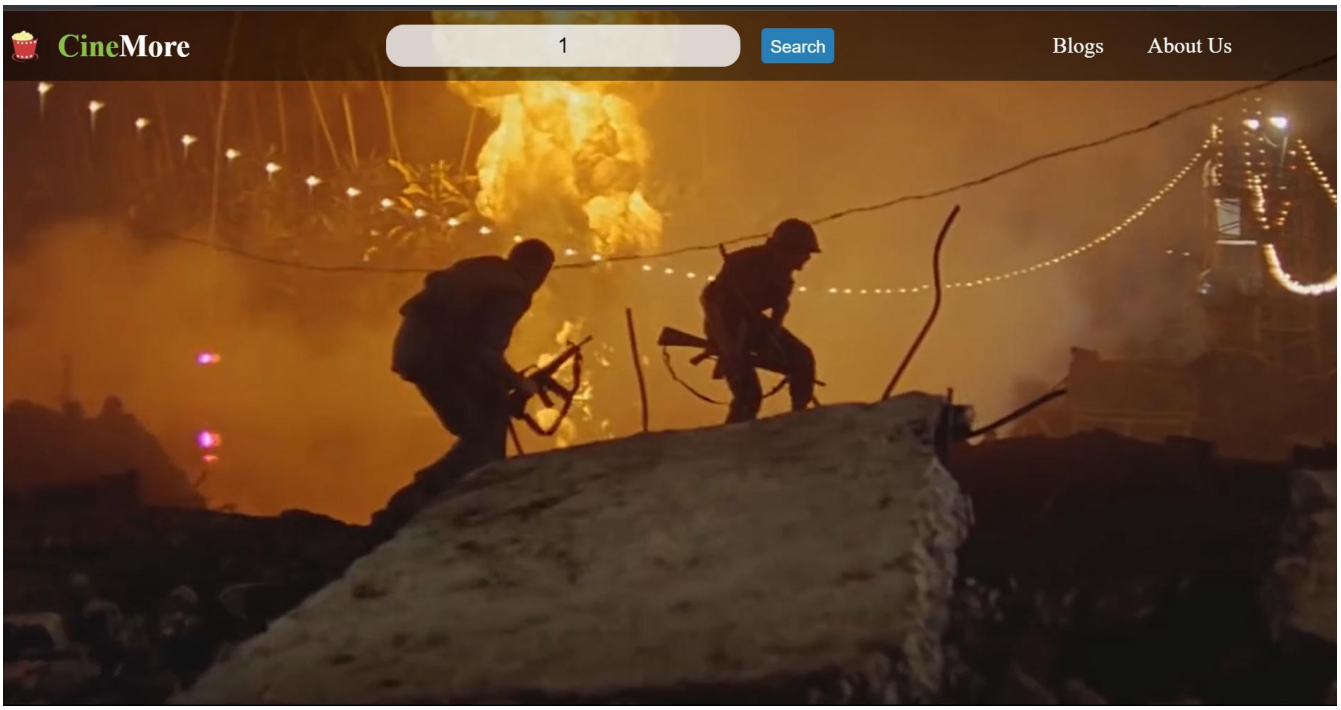
```
Recommended movies for 1 are:

Rebecca
Once Were Warriors
Four Rooms
Interview with the Vampire
The Passion of Joan of Arc
```

Recommended movies for user id = 1 using **Cosine + Leniency Score**(as a measure of similarity):

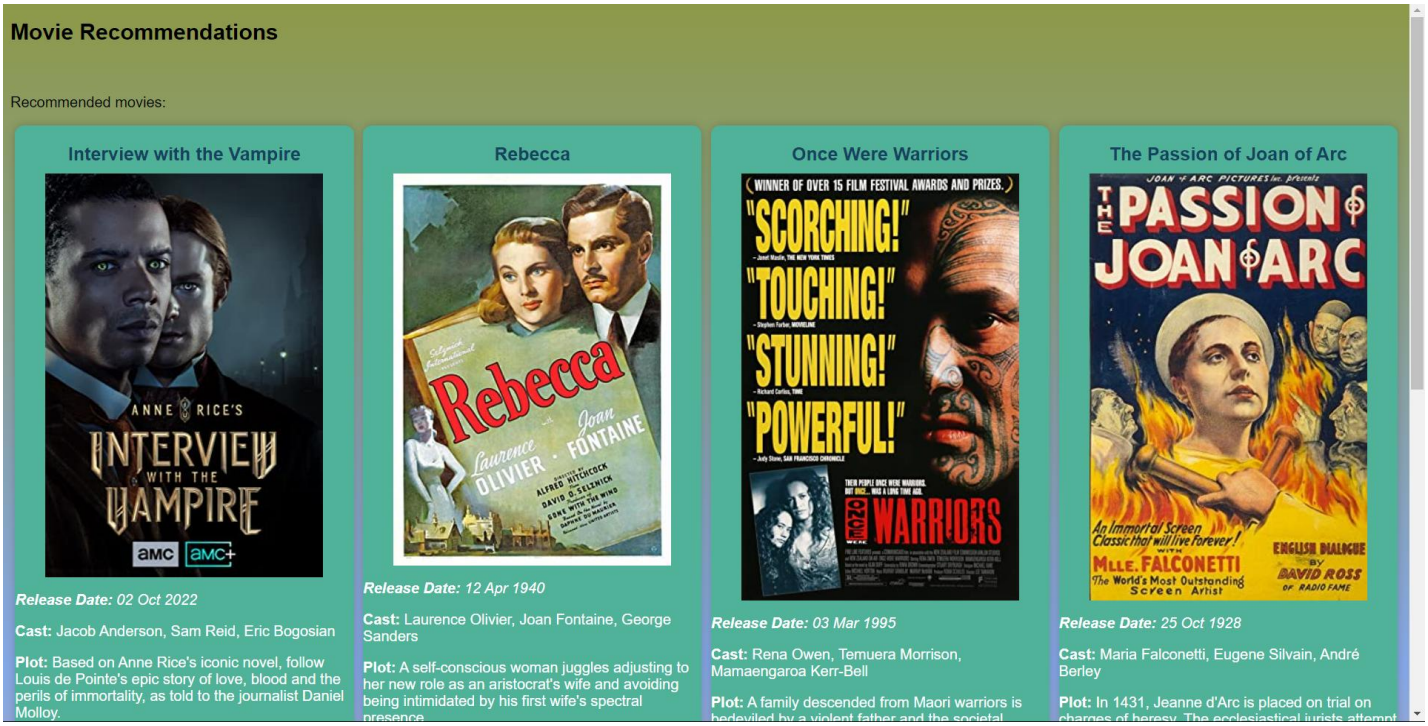
```
Recommended movies for user 1 are:

Rebecca
Reservoir Dogs
Once Were Warriors
Donnie Darko
The Passion of Joan of Arc
```

4.3.6) Displaying results on website:

(User Id entered in the search bar)



(Movies recommended for user alongwith their details)



4.4 Quality Assurance

1. **Quality Objectives:**

    - The project aims to compare different similarity measures used in collaborative filtering and evaluate their effectiveness in movie recommendations.
    - The project also aims to incorporate user reviews through natural language processing to improve the recommendation system.
    - The project strives to provide accurate and relevant recommendations based on user preferences and leniency scores.

2. **Quality Standards and Compliance**

    - **Use of standard similarity measures:** To ensure that our project meets the industry standards, we have used four commonly used similarity measures - cosine similarity, Jaccard similarity, Pearson correlation coefficient, and Chebyshev similarity. These measures have been widely used in the field of collaborative filtering, and their effectiveness has been well established in the literature.
    - **Compliance with ethical guidelines:** Our project involves scraping user reviews from IMDb, which raises ethical concerns related to data privacy and intellectual property. To ensure compliance with ethical guidelines, we have obtained necessary permissions and made sure that the data is anonymized and used only for research purposes.
    - **Adherence to coding standards:** We have followed coding standards to ensure that the code is readable, maintainable, and scalable. We have used appropriate variable names, comments, and coding conventions to make sure that the code is easily understandable by other developers.

3. **Quality Processes:**

    - The team has established a clear plan and timeline for the project, including regular check-ins and meetings to ensure progress and identify potential issues.
    - The team has conducted thorough research on the different similarity measures and sentiment analysis techniques to ensure they are using the most appropriate and effective methods.
    - The team has developed a testing framework to ensure the accuracy and reliability of the recommendation system and conducted multiple rounds of testing to identify and fix any issues.

4. **Quality Control:**

    - The team has also implemented a peer review process for the final report to ensure that it is comprehensive, well-organized, and accurate.

- The team has taken measures to ensure that the natural language processing techniques used in sentiment analysis are producing accurate results, and has implemented error handling to detect and correct any issues.

- The team has taken steps to ensure that the data being scraped from IMDb is of high quality and accuracy, and has implemented error handling to prevent any data inconsistencies.

- Change management: Changes to the project scope or requirements will be managed through a change management process to ensure that they are properly documented and approved.

Overall, the quality assurance plan for the project aims to ensure that the recommendation system is accurate, reliable, and provides relevant recommendations to users. The team has taken a systematic and thorough approach to the project, incorporating regular testing and reviews to ensure that the final product meets the desired objectives and standards.

Chapter 5

Standards Adopted

5.1 Design Standards

- Follow a modular design approach, breaking down the project into smaller, more manageable components or modules.
- Use standard design patterns wherever applicable.
- Ensure code is maintainable and easily extendable.
- Follow best practices for data retrieval, processing, and storage.
- Ensure that the code is scalable and can handle increasing amounts of data.

5.2 Coding Standards

- Throughout the project a consistent coding style should be maintained.
- Ensure code is well-documented, with clear comments for complex code sections.
- Use variable and function names that are self-explanatory and consistent with the naming conventions.
- Version control to be used to track changes and collaborate with team members.
- Ensure code is optimized for performance.

5.3 Testing Standards

- Develop a comprehensive testing plan that includes unit tests, integration tests, and end-to-end tests.
- Testing frameworks like Pytest or Unittest should be used.
- Ensure that the testing plan covers all possible use cases and scenarios.
- Automate testing wherever possible to reduce manual testing efforts.
- Use continuous integration to ensure that code changes do not break existing functionality.
- Overall, following these design, coding, and testing standards will ensure that the project is well-organized, maintainable, and reliable.

Chapter 6

Conclusion & Future Scope

6.1 Conclusion

In conclusion, this project aims to provide insights into the effectiveness of different similarity measures and the impact of user leniency in movie recommendations, and it has the potential for further research and extension in various domains.

6.2 Future Scope

The future scope of this project includes the exploration and incorporation of more advanced techniques to improve the accuracy of movie recommendations. For instance, hybrid recommendation systems could be considered, which combine collaborative filtering with content-based filtering or other advanced techniques like matrix factorization. Additionally, further research could be conducted on incorporating user demographic information and analyzing its impact on the recommendations. This could help to tailor recommendations based on factors such as age, gender, or location, and provide more personalized recommendations.

Moreover, sentiment analysis techniques could be improved and refined to provide more accurate ratings based on user reviews. This could involve the use of deep learning models or other advanced natural language processing techniques.

Finally, this work could be expanded to other domains beyond movie recommendations, such as product recommendations in e-commerce platforms or music recommendations in streaming services. This would require adapting the similarity measures and sentiment analysis techniques to fit the specific domain.

In conclusion, this project aims to provide insights into the effectiveness of different similarity measures and the impact of user leniency in movie recommendations, and it has the potential for further research and extension in various domains.

References

[1] Hutto, Clayton J., and Eric Gilbert. "VADER: A parsimonious rule-based model for sentiment analysis of social media text." Eighth International AAAI Conference on Weblogs and Social Media. 2014.

https://www.semanticscholar.org/paper/VADER%3A-A-Parsimonious-Rule-Based-Model-for-Analysis-Hutto-Gilbert/bcdc102c04fb0e7d4652e8bcc7edd2983bb9576d

[2] Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence, 43-52.

https://www.semanticscholar.org/paper/Empirical-Analysis-of-Predictive-Algorithms-for-Breese-Heckerman/36b4a92c8eca6fd6d1b8588fc1fd0e3f89a16623

[3] Konstan, J. A., & Riedl, J. (2012). Recommender systems: from algorithms to user experience. Springer.

https://link.springer.com/article/10.1007/s11257-011-9112-x

[4] Charu C. Aggarwal, Neighborhood-Based Collaborative Filtering

https://link.springer.com/chapter/10.1007/978-3-319-29659-3_2