A PROJECT REPORT

on

“Movie Recommendation System”

Submitted to

KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of

BACHELOR’S DEGREE IN COMPUTER SCIENCE AND TECHNOLOGY

BY

Adil Ahmad	21051451
Sanu Verma	21051509
Tanvi Ananya	21051523
Kappala Amiya Kumari	2015100
Saujanya Chandrakar	2105153

UNDER THE GUIDANCE OF
Dr. T. Kar

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

May 2025

A PROJECT REPORT

on

"Movie Recommendation System"

Submitted to

KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of

BACHELOR'S DEGREE IN COMPUTER SCIENCE AND TECHNOLOGY

BY

Adil Ahmad	21051451
Sanu Verma	21051509
Tanvi Ananya Kappala	21051523
Amiya Kumari	2105100
Saujanya Chandrakar	2105153

UNDER THE GUIDANCE OF

Dr. T. Kar

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

BHUBANESWAE, ODISHA -751024

May 2025

KIIT Deemed to be University

School of Computer Engineering

Bhubaneswar, ODISHA 751024

CERTIFICATE

This is certify that the project entitled

“Movie Recommendation System“submitted by

Adil Ahmad	21051451
Sanu Verma	21051509
Tanvi Ananya Kappala	21051523
Amiya Kumari	2105100
Saujanya Chandrakar	2105153

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 2023-2024, under our guidance.

Date: 08/04/2024

(Dr. T. Kar)

Project Guide

Acknowledgements

We are profoundly grateful to **Dr. T. Kar** of **Affiliation** for his expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion. ....

Adil Ahmad

Sanu Verma

Tanvi Ananya Kappala

Amiya Kumari

Saujanya Chandrakar

ABSTRACT

This project aims to implement a machine learning model to recommend movies to a person. Movies are subjective in nature and a movie liked by one entity may not necessarily be liked by another. This subjectivity makes the task of recommendation very complicated because even a block-buster movie may not be liked by a specific set of individuals. Hence, mapping a movie to the taste of an individual is very important. In this regard, we have decided to use similarity as our objective function. The model takes a movie's name as a user input string and tries to generate a list of movies that are most similar to the given input. This model uses a distance based learning algorithm as its engine to plot and predict movies that are similar to each other.

There are multiple distance based algorithms to find similarity between two objects/entities. We have decided to use cosine similarity for this project. Cosine similarity is especially advantageous in this situation because a movie can have multiple attributes, this means that even similar movies can have drastic separations, this phenomenon will render conventional measures of distance obsolete, hence a vector based distance like cosine distance is preferrable.

In the 21st century, user experience is as valuable as the core product itself. Keeping this in consideration, we have implemented a modern front-end application to increase user interaction and make the experience memorable. The front-end is minimal and modern in nature. It is easy to implement and equally easy to host because its computational complexity has been kept low by design.

Keywords: Distance based machine learning, Streamlit, Python, Cosine Similarity, SciKit Learn, Recommendation System, Minimal and modern front-end

Contents

Chapter 1

Introduction

Online streaming platforms have seen a massive rise in the recent past, products such as Netflix, Hulu and Disney Hotstar have become household names. This happened because of the ease of use accompanied by the sheer quantity and variety of services provided by these services. One such service is movie recommendation. A strong movie recommendation system is important for user retention as well as user satisfaction.

This project aims to implement a full stack application which recommends new movies to people based on the movies they already like. The model was created in 3 phases. The first phase includes data pre-processing. This is followed by the second phase which includes model creation and testing. The third and final phase is the front end development and deployment phase.

Movie recommendation is a sophisticated task because movies are of various types. They have multiple features like runtime, language, actors, crew, genres, background and the script. This implies that we must come up with a multi-dimensional learning model to recommend movies on the basis of similarity.

The report has been divided into multiple sections such as introduction, problem statement specification, analysis, implementation, etc. Each section provides an abstract of the project.

Chapter 2

Literature Review

2.1 Scikit-Learn Library

Scikit-Learn is often abbreviated as Sklearn, it is an open source python library for machine learning. It offers inbuilt algorithms for classification, regression, clustering and dimensionality reduction. It interoperates with other python libraries like numpy and pandas to make implementation easier. Sklearn also provides tools for visualisation, pre-processing, evaluation and model fitting.
Source: scikit-learn: machine learning in Python — scikit-learn 1.2.2 documentation

2.2 Stemming

Stemming is the process of producing morphological variants of a root/base word. Stemming is important for NLP as it gets rid of grammatical jargon in the dataset and reduces each word to its base form. For the sake of this project, we will be using the Porter stemmer.
Source: sklearn-porter · PyPI

2.3 Countvectorizer

Countvectorizer is used to transform a given string of text to a vector on the basis of frequency of each word. Countvectorizer creates a sparse matrix in which each unique word is represented by a separate column and each text sample is a separate row in the matrix.
Source: sklearn.feature_extraction.text.CountVectorizer — scikit-learn 1.2.2 documentation

2.4 Cosine Similarity

Similarity is measured as distance between 2 vectors. Dimensions represent the features of the vector. The degree of similarity is inversely proportional to the distance between two vectors. Cosine similarity is independent of the size of the involved vectors. Cosine similarity is especially used in scenarios where dimensionality is high because in such situations even similar objects can have a huge Euclidean distance between themselves. It is important to convert the text to a vector before using cosine similarity. The formula for cosine similarity is given below: -

$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

where,

A: vector A
B: vector B
||A||: length of vector A
||B||: length of vector B
||Ai||: ith bit of the vector A
||Bi||: ith bit of vector B
A.B: Dot product of vector A with vector B

Source: sklearn.metrics.pairwise.cosine_similarity — scikit-learn 1.2.2 documentation

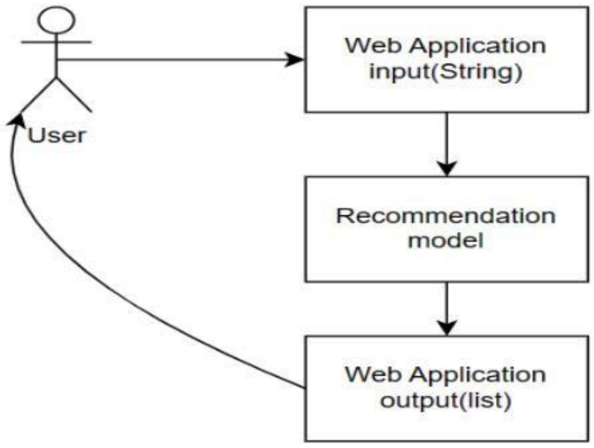
2.5 Streamlit

Streamlit is an open source application framework to convert data scripts into modern, minimal, interactive applications. It is mainly used for data applications or ML/DL applications where the team is not necessarily experienced in the field of development, rather their expertise lies in the domain of data science.
The applications built using streamlit are computationally inexpensive and can be deployed to the cloud instantly. This reduces development time, devops time and specialised manpower cost.
Source: Streamlit • The fastest way to build and share data apps

Chapter 3

The Problem
People are often confused as to which movies would suit their taste and which won't. This is a major problem for OTT platforms which are trying to increase user retention and view-time. Recommending movies specifically tailored to someone's liking is a good way to solve this problem as this will increase watch-time of people.

Problem Solution



The user will input the name of a movie he already likes as a string, the recommendation engine will refer to the database and find the IDs of movies that are similar to the input string, it will create a sorted list of 10 movies that are the most similar to the input string and return them as a list. The recommendation engine we have designed uses cosine distance as a measure of similarity.

Method of Implementation

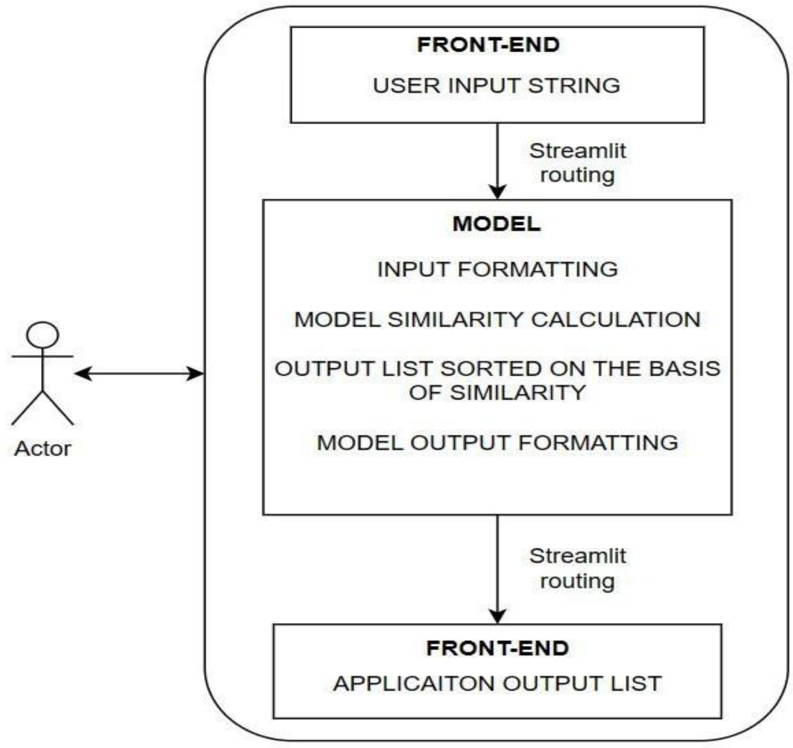
The entire project can be broken down into 3 major modules which are – data pre-processing, model creation and application development and deployment. The data pre-processing involves libraries such as numpy and pandas which make our job easier by providing us with inbuilt python libraries. The model creation phase involves tools like SKLearn which provides multiple inbuilt ML algorithms to reduce coding time, this means the development team can spend more time on design and implementation, rather than the syntactual jargon of Python.
All these steps will be discussed in depth in the next chapter.

Chapter 4

Implementation

In this section, we will briefly walkthrough the entire system using diagrams and textual explanation. In further subsections, we will go through the specific implementation of each module.

1. System overview



The user end of the application has been created using streamlit which is a data application framework. The model has been built using the Sklearn library which provides inbuilt functions and algorithms for model pre-processing and development. The application has been hosted using the Streamlit cloud.

2. Model Development Phase

The model building phase can be divided into 3 phases, we will discuss all three phases in detail.

2.1 Data cleaning and Pre-processing

This phase is perhaps the most time-taking phase of our project, it involves formatting the dataset which can be used to train the model.

.1.1 Dataset selection

For this project we will be using the TMDB movies data set which is publically available on Kaggle.

Source: TMDB 5000 Movie Dataset | Kaggle

2.1.2 Dataset analysis

Let's analyse our chosen dataset.
First, we need to create a pandas dataframe to analyse our CSV files using the pandas library.
Code:

```
# Read raw CSV files into a dataframe.
movies = pd.read_csv("Minor-Project/raw/tmdb_5000_movies.csv")
credits =
pd.read_csv("Minor-Project/raw/tmdb_5000_credits.csv")
```

Now, lets merge these data-frames into one singular dataframe:
Code:

```
movies = movies.merge(credits,on='title')
```

Now, let's take a glimpse of this dataframe:

```
movies.head()
```

Now, lets remove the unnecessary features from our dataframe:

```
# Drop the unnecessary attributes
movies = movies[['movie_id','title','overview','genres','keywords','cast','crew', 'popularity',
'vote_average']]
```

Now, we must write some functions to standardise our feature domain. Standardisation of the feature domain is necessary because human readable data-structures are not suitable for a Python program.
We have to remove spaces from the names because the stemming algorithm will treat space as a delimiter. We must also drop some crew names which are not that necessary. We must also get rid of null values.

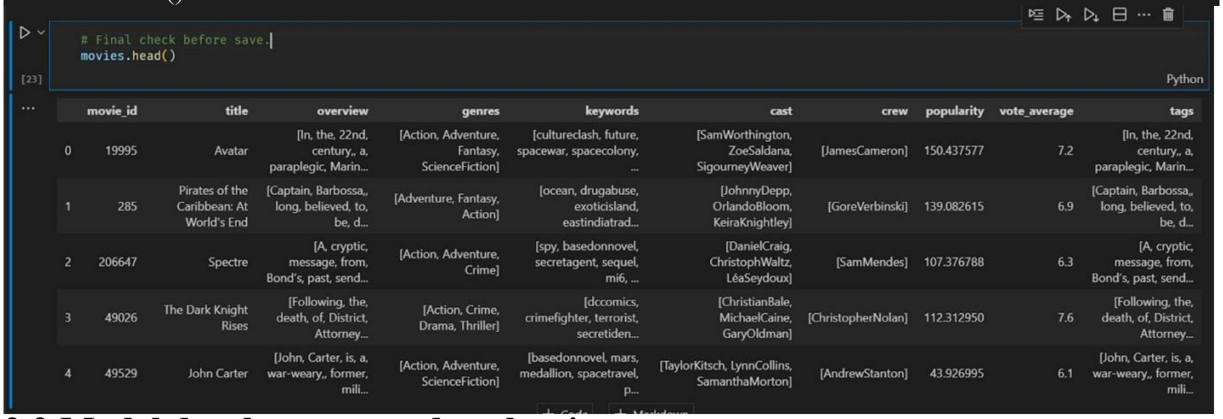
```
# Conversion functions to handle data-structure formatting.
def convformat(text):
    L = []
    for x in ast.literal_eval(text):
        L.append(x['name'])
    return L

def crewdirector(text):
    L = []
    for x in ast.literal_eval(text):
        if x['job'] == 'Director':
            L.append(x['name'])
    return L

def collapse(L):
    L1 = []
    for i in L:
        L1.append(i.replace(" ",""))
    return L1

movies.dropna(inplace=True)
movies['overview'] = movies['overview'].apply(lambda x:x.split())
```

Now let's merge the relevant features into a new attribute called 'tags'.

```
# Merge the crew, cast, overview and keywords attributes
movies['tags'] = movies['overview'] + movies['genres'] + movies['keywords'] + movies['cast']
+ movies['crew']
```

Now, let's get a final glimpse of our dataset before we start working on the model development and evaluation phase.

```
movies.head()
```



```
# Final check before save.
movies.head()
```

[23]

	movie_id	title	overview	genres	keywords	cast	crew	popularity	vote_average	tags
0	19995	Avatar	[In, the, 22nd, century,, a, paraplegic, Marin...	[Action, Adventure, Fantasy, ScienceFiction]	[cultureclash, future, spacewar, spacecolony, ...	[SamWorthington, ZoeSaldana, SigourneyWeaver]	[JamesCameron]	150.437577	7.2	[In, the, 22nd, century,, a, paraplegic, Marin...
1	285	Pirates of the Caribbean: At World's End	[Captain, Barbossa,, long, believed, to, be, d...	[Adventure, Fantasy, Action]	[ocean, drugabuse, exoticisland, eastindiatrad...	[JohnnyDepp, OrlandoBloom, KeiraKnightley]	[GoreVerbinski]	139.082615	6.9	[Captain, Barbossa,, long, believed, to, be, d...
2	206647	Spectre	[A, cryptic, message, from, Bond's, past, send...	[Action, Adventure, Crime]	[spy, basedonnovel, secretagent, sequel, mi6, ...	[DanielCraig, ChristophWaltz, LéaSeydoux]	[SamMendes]	107.376788	6.3	[A, cryptic, message, from, Bond's, past, send...
3	49026	The Dark Knight Rises	[Following, the, death, of, District, Attorney...	[Action, Crime, Drama, Thriller]	[dccomics, crimefighter, terrorist, secretiden...	[ChristianBale, MichaelCaine, GaryOldman]	[ChristopherNolan]	112.312950	7.6	[Following, the, death, of, District, Attorney...
4	49529	John Carter	[John, Carter, is, a, war-weary,, former, mili...	[Action, Adventure, ScienceFiction]	[basedonnovel, mars, medallion, spacetravel, p...	[TaylorKitsch, LynnCollins, SamanthaMorton]	[AndrewStanton]	43.926995	6.1	[John, Carter, is, a, war-weary,, former, mili...

2.2 Model development and evaluation

This phase involves the development of our core engine. It can be divided into 3 phases. The 1st subsection includes the stemming algorithm which gets rid of the grammatical jargon in our feature set. The

2nd subsection involves the countvectorizer algorithm which vectorizes our strings to create a machine readable vector. The 3rd subsection involves the cosine similarity algorithm which finds similarity of each data-point corresponding to the given input. The nearest or most similar data-points are sorted and used as our output list.
Now, let's go through each step in detail.

2.2.1 Stemming phase

In this case, we are using the Porter stemmer algorithm
Code

```
ps = PorterStemmer()
def stem(text):
   y = []
   for i in text.split():
      y.append(ps.stem(i))
   return " ".join(y)
```

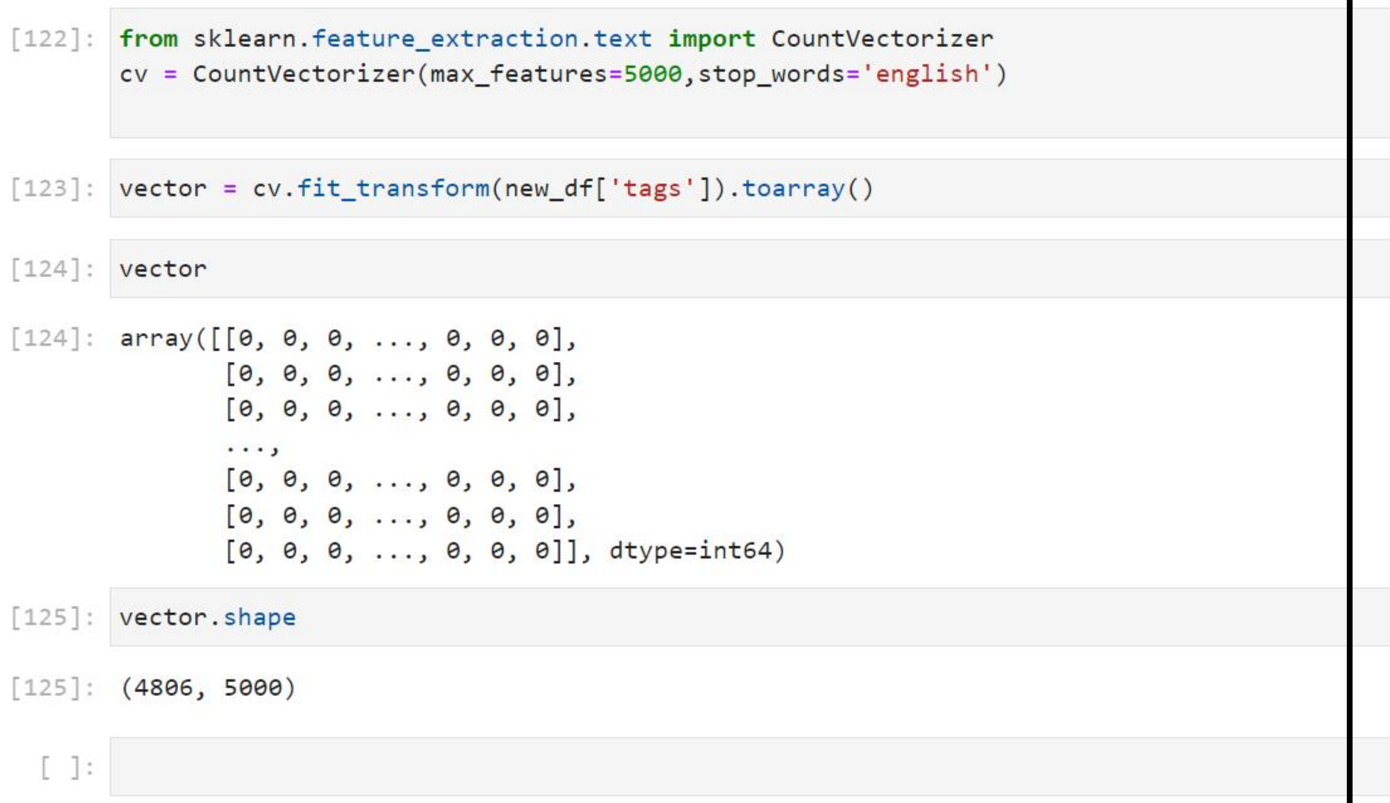
The porter stemmer algorithm essentially takes the input string and converts each word into its root word. It is a technique to get rid of the grammatical jargon in our feature-set.

2.2.2 CountVectorizer

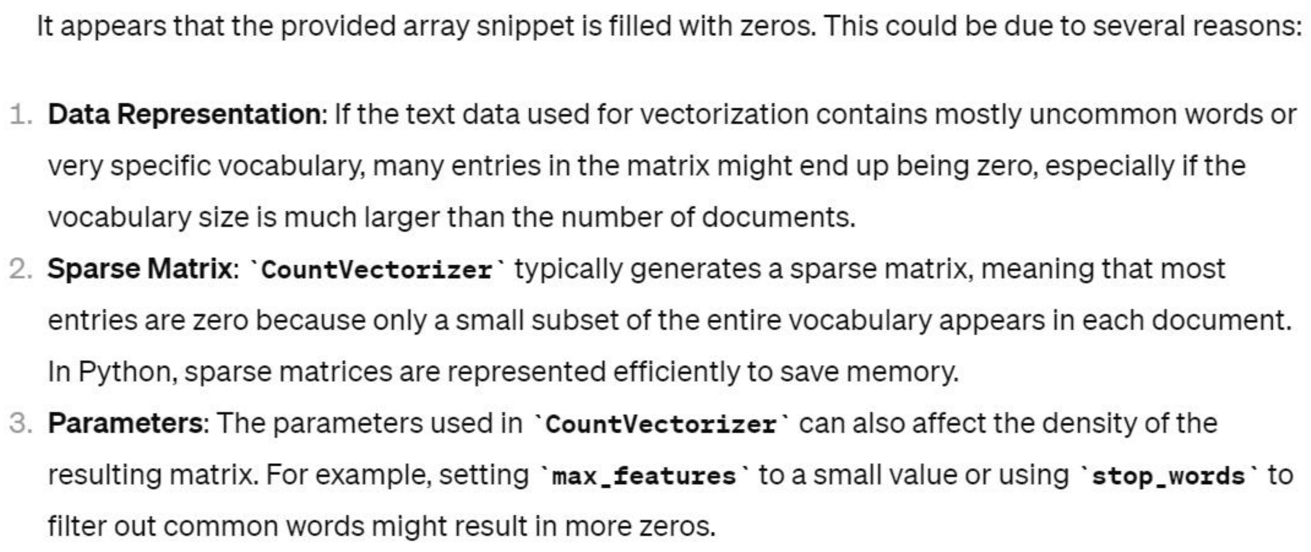
In this case, we will vectorize our stemmed feature-set using word count or frequency of unique elements.
Code

```
cv = CountVectorizer(max_features=5000, stop_words='english')
vector = cv.fit_transform(movies['tags']).toarray()
```

We have limited our vectorizer to 5000 unique words only. Increasing the word count will lead to a much more sparse array. Sparse arrays consume more memory and time, this will increase our computational expenses while training the model.
In our project we have used the countvectorizer algorithm provided by the Sklearn feature-selection library.



```
[122]: from sklearn.feature_extraction.text import CountVectorizer
       cv = CountVectorizer(max_features=5000,stop_words='english')

[123]: vector = cv.fit_transform(new_df['tags']).toarray()

[124]: vector

[124]: array([[0, 0, 0, ..., 0, 0, 0],
              [0, 0, 0, ..., 0, 0, 0],
              [0, 0, 0, ..., 0, 0, 0],
              ...,
              [0, 0, 0, ..., 0, 0, 0],
              [0, 0, 0, ..., 0, 0, 0],
              [0, 0, 0, ..., 0, 0, 0]], dtype=int64)

[125]: vector.shape

[125]: (4806, 5000)

[ ]:
```



It appears that the provided array snippet is filled with zeros. This could be due to several reasons:

1. **Data Representation**: If the text data used for vectorization contains mostly uncommon words or very specific vocabulary, many entries in the matrix might end up being zero, especially if the vocabulary size is much larger than the number of documents.
2. **Sparse Matrix**: `CountVectorizer` typically generates a sparse matrix, meaning that most entries are zero because only a small subset of the entire vocabulary appears in each document. In Python, sparse matrices are represented efficiently to save memory.
3. **Parameters**: The parameters used in `CountVectorizer` can also affect the density of the resulting matrix. For example, setting `max_features` to a small value or using `stop_words` to filter out common words might result in more zeros.

2.2.3 Cosine-similarity engine

Cosine similarity is used to find the distance between two vectors. In our case, each vector tuple represents a unique feature-set.
The formula for cosine similarity is given as:

$$\cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$

Cosine similarity is just one of the many distance based algorithms used to measure similarity. We could have used something like the Euclidean distance but it won't suffice because in a model with high dimensionality the distance and similarity not always go hand in had, i.e. sometimes, you may find a case where two tuples far apart end up being similar to each other, this happens because each attribute has its own domain, this results in disproportional linear calculations.
In python, the Sklearn library provides an inbuilt implementation of the cosine similarity algorithm.

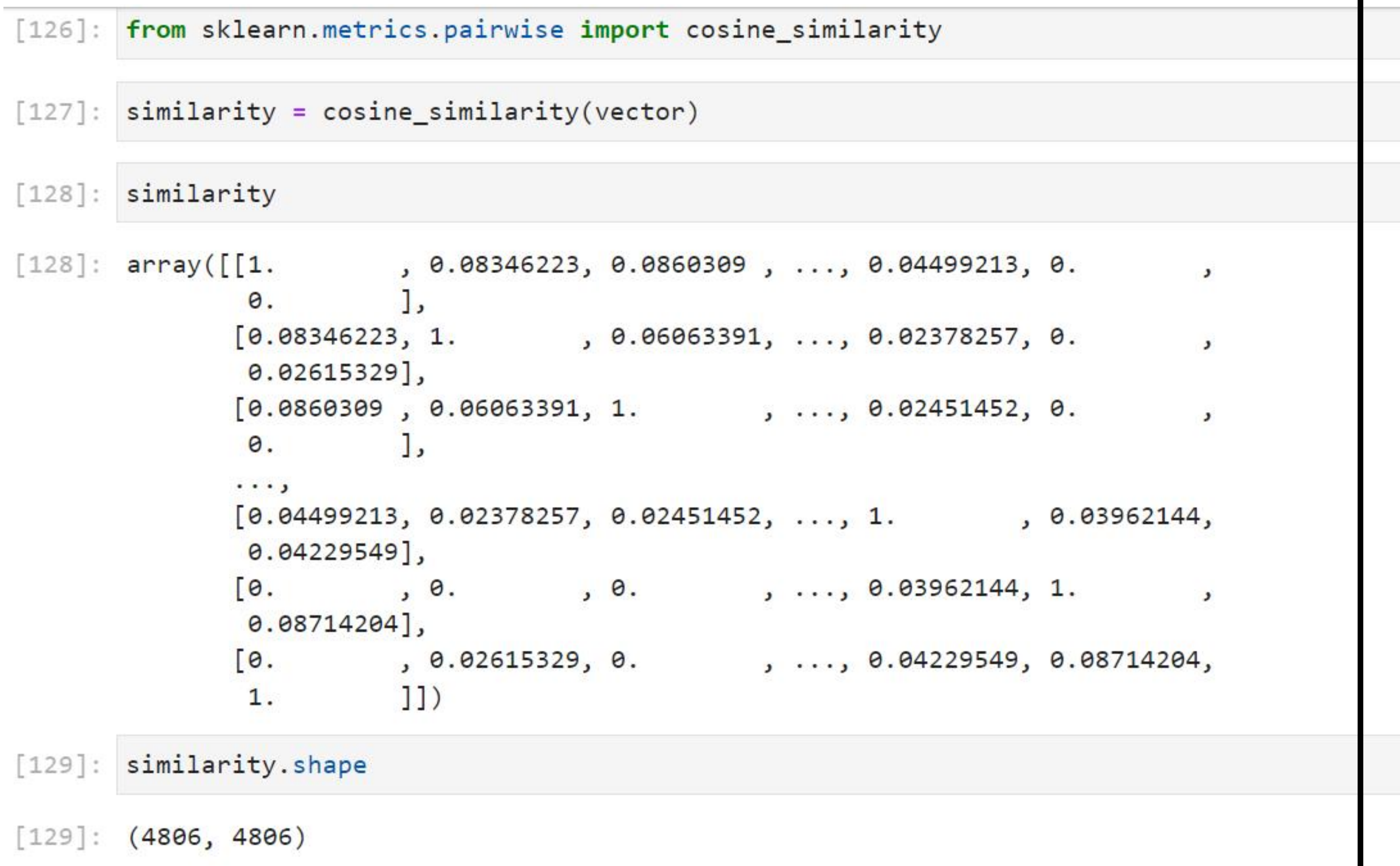
Code

```
similarity = cosine_similarity(vector)
```

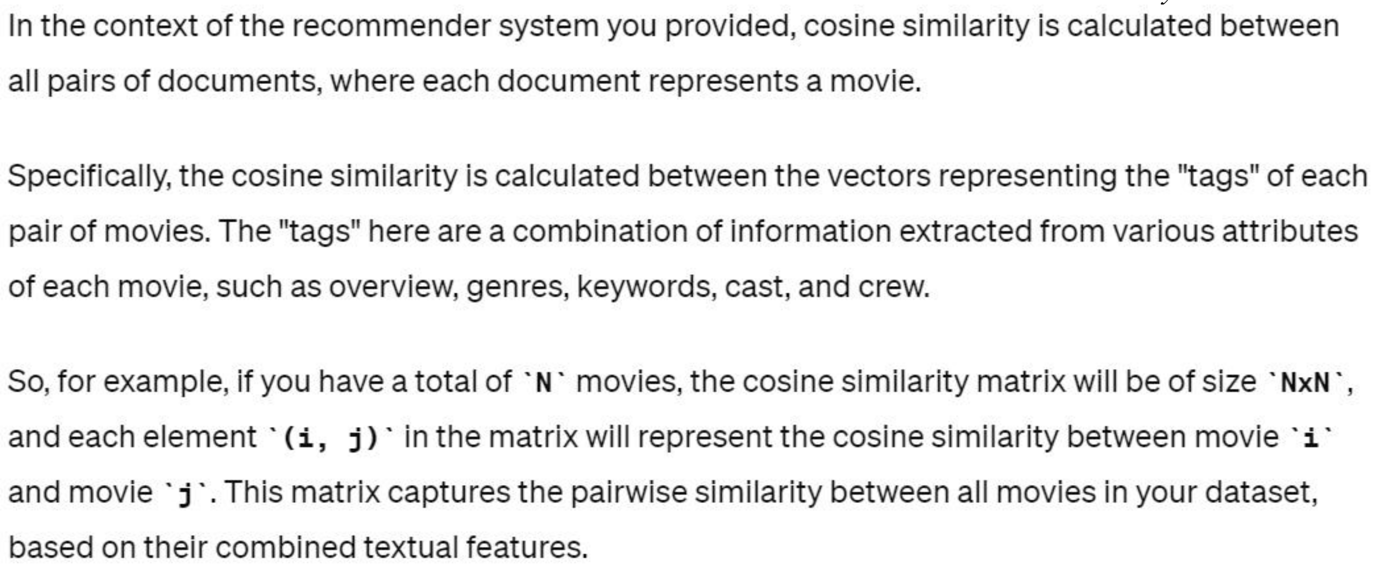
Now, let's return the most similar vectors by sorting the output list.
Code

```
sorted(list(enumerate(similarity[0])), reverse=True, key=lambda x:x[1])[1:6]
```

As this is a recommendation algorithm which uses distance based learning, model evaluation is not really necessary because the output for every input is unique. This is not a numerical prediction model hence we will skip the evaluation phase.

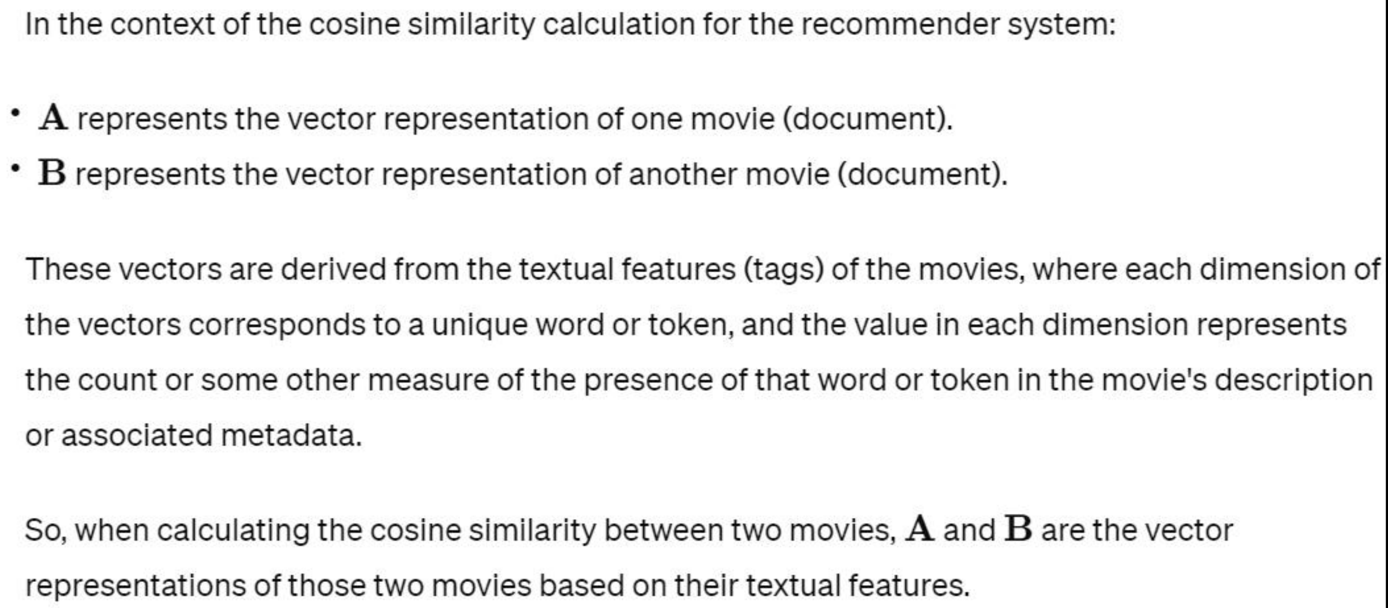


```
[126]: from sklearn.metrics.pairwise import cosine_similarity

[127]: similarity = cosine_similarity(vector)

[128]: similarity

[128]: array([[1.        , 0.08346223, 0.0860309 , ..., 0.04499213, 0.        ,
                0.        ],
               [0.08346223, 1.        , 0.06063391, ..., 0.02378257, 0.        ,
                0.02615329],
               [0.0860309 , 0.06063391, 1.        , ..., 0.02451452, 0.        ,
                0.        ],
               ...,
               [0.04499213, 0.02378257, 0.02451452, ..., 1.        , 0.03962144,
                0.04229549],
               [0.        , 0.        , 0.        , ..., 0.03962144, 1.        ,
                0.08714204],
               [0.        , 0.02615329, 0.        , ..., 0.04229549, 0.08714204,
                1.        ]])

[129]: similarity.shape

[129]: (4806, 4806)
```


In the context of the recommender system you provided, cosine similarity is calculated between all pairs of documents, where each document represents a movie.

Specifically, the cosine similarity is calculated between the vectors representing the "tags" of each pair of movies. The "tags" here are a combination of information extracted from various attributes of each movie, such as overview, genres, keywords, cast, and crew.

So, for example, if you have a total of `N` movies, the cosine similarity matrix will be of size `NxN`, and each element `(i, j)` in the matrix will represent the cosine similarity between movie `i` and movie `j`. This matrix captures the pairwise similarity between all movies in your dataset, based on their combined textual features.


In the context of the cosine similarity calculation for the recommender system:

- $\mathbf{A}$ represents the vector representation of one movie (document).
- $\mathbf{B}$ represents the vector representation of another movie (document).

These vectors are derived from the textual features (tags) of the movies, where each dimension of the vectors corresponds to a unique word or token, and the value in each dimension represents the count or some other measure of the presence of that word or token in the movie's description or associated metadata.

So, when calculating the cosine similarity between two movies, $\mathbf{A}$ and $\mathbf{B}$ are the vector representations of those two movies based on their textual features.

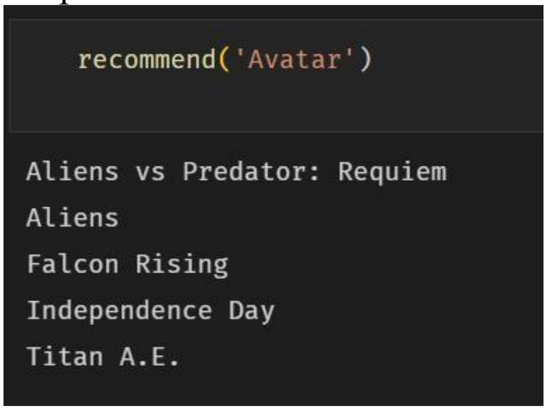
2.3 Model recommendation Phase

The recommendation must be done on a single tuple and not the entire dataset, therefore, we must format our input before feeding it into the model. Once our model has recommended similar movies then we must enumerate the list in order to sort it. After enumeration, we must return the top 5 results as they are the most similar movies.

Code

```
def recommend(movie):
    index = movies[movies['title'] == movie].index[0]
    distances = similarity[index]
    movies_list = sorted(list(enumerate(similarity[index])),reverse=True,key = lambda x:
x[1])[1:6]
    for i in movies_list:
        print(movies.iloc[i[0]].title)
```

Now, let's test the model by recommending movies for a given input string.
Ex input: Avatar
Output:



```
recommend('Avatar')
```

```
Aliens vs Predator: Requiem
Aliens
Falcon Rising
Independence Day
Titan A.E.
```

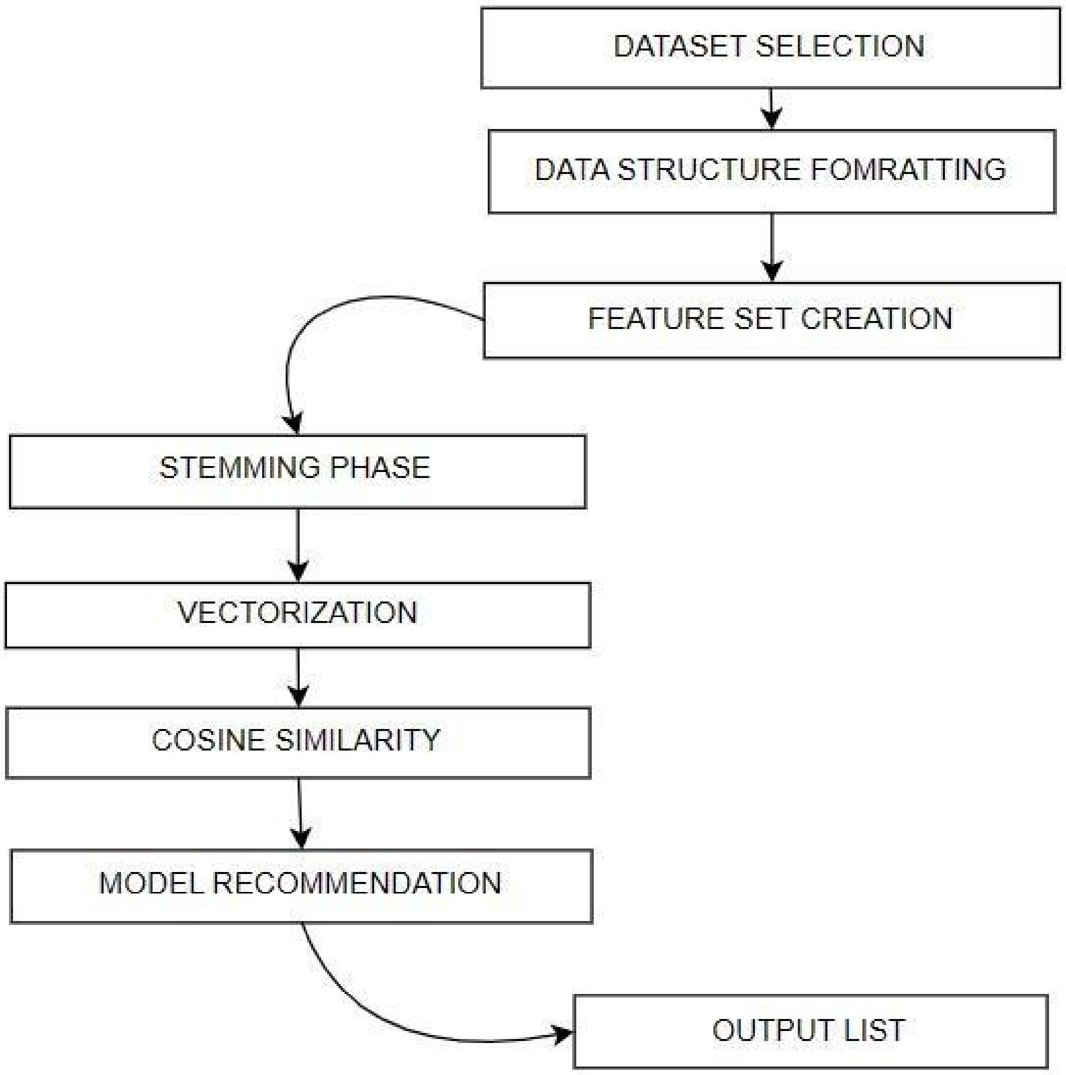
3. **Application Development Phase**

This phase involves the development of a user interactive front-end to make it easier for people to interact with this project.
In this project we have used Streamilt as our primary framework for development. This phase can be broken down into 3 phases. The first phase involves model initiation and integration phase. The second phase involves the development of an interactive UI. The final phase implements the cloud deployment of this model.

Code:

```
import streamlit as stimport
numpy as np import pandas as
pd import pickle

# Unpack data files
mvlist = pd.read_csv('df.csv')
similarity = pickle.load(open('similarity.pkl','rb'))

# Accept user input
movie = st.selectbox("A movie you already like", mvlist.title)

# Recommend function def
recommend(movie):
    index = mvlist[mvlist['title'] == movie].index[0]distances
    = similarity[index]
    movies_list = sorted(list(enumerate(similarity[index])),reverse=True,key
= lambda x: x[1])[1:6]sol =
    []
    for i in movies_list: sol.append(mvlist.iloc[i[0]].title)
    return sol

# Trigger recommend functionif
movie is not None:
    st.write(recommend(movie))
```

Flowchart of the implementation phase:



Chapter 5

Standards Adopted

Coding Guidelines
Machine learning and Data applications are a very modern phenomenon due to this the classical standards like IEEE don't perform well on these applications. Taking this into consideration, we have decided to stick to Google's ML Application rules guide as a coding guideline for this project.
Source: Rules of Machine Learning: | Google Developers

As far as the application front-end and user oriented design is concerned, we have adopted standard testing approaches to validate our project.

Testing standards used:

1. IEEE 1008-1987: IEEE Standard for Software Unit Testing

Source: 1008-1987 - IEEE Standard for Software Unit Testing | IEEE Standard | IEEE Xplore

2. IEEE 1059-1993: IEEE Standard for Software Validation and Verification

Source: 1059-1993 - IEEE Guide for Software Verification and Validation Plans | IEEE Standard | IEEE Xplore

3. IEEE 830-1984: IEEE Standard for Software Requirement Specification

Source: 830-1984 - IEEE Guide for Software Requirements Specifications | IEEE Standard | IEEE Xplore

4. ISO/IEC/IEEE 90003:2018: Guidelines for the application of ISO 9001:2015 to Computer Software

Source: ISO/IEC/IEEE 90003:2018 - Software engineering — Guidelines for the application of ISO 9001:2015 to computer software
This standard was used just for the sake of reference, implementing it in its full capacity requires a lot of manpower and time, which is unnecessary considering the scope of this project.

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

The project named "Movie Recommendation System" has been concluded. The project involved multiple phases of learning where our team learnt how to pre-process data, analyse data, perform feature engineering, perform stemming using the Porter stemmer algorithm, perform vectorization using the CountVectorizer package, creation of a user-interactive data application to enhance user experience. The team learnt about distance based learning, especially the concept of cosine similarity. The team also learnt about deploying the project onto a live cloud environment.

This project will help users to find movies based on their taste. It can also be used as an integral part of other projects as a sub-module.

We have hosted the source-code online in the spirit of open source software development.

Source Code: om22shree/Minor-Movie-Recommendation-System: This repository is specifically used to host the source code of our group minor project related to movie recommendation. (github.com)

6.2 Future scope

The current project is limited in scope, it was built for the purpose of skill demonstration and implementation of concepts learnt during various engineering semesters. The scope can be expanded by improving the core engine and coming up with a better algorithm to find similarity between 2 vectors. The scope of improvement also lies in improving the web application. A better and more robust framework can be used to achieve similar results which would offer better user interface and user experience. Not only that, improvements can also be made by adding filters to the input criteria. It is possible to create an application to recommend movies on the basis of someone's mood and taste pertaining to availability of data, which as of now, doesn' exist. In future, with increased data logging and availability, it is possible to come up with an NLP algorithm to cater to a much more refined audience.

References

1. Distance based machine learning
Source: 4 Distance Measures for Machine Learning - MachineLearningMastery.com

2. Supervised Machine Learning
Source: What is Supervised Learning? | IBM

3. Cosine Similarity - Understanding the mathematics and how it works
Source: Cosine Similarity - Understanding the math and how it works? (with python) (machinelearningplus.com)

4. Basics of countVectorizer
Source: Basics of CountVectorizer | by Pratyaksh Jain | Towards Data Science

5. Application deployment on streamlit
Source: Deploy an app - Streamlit Docs