INDIAN INSTITUTE OF TECHNOLOGY (INDIAN SCHOOL OF MINES), DHANBAD-826004

Department of Computer Science and Engineering

A Project Report on

"MOVIE RECOMMENDER SYSTEM"

Submitted by

SAPIREDDY SWAMI SHANKAR

ADMN NO. 14JE000436

7TH Semester B.Tech. CSE

Under the Guidance of

Dr. A.Chandra Sekhara rao

Assistant Professor

Department of CSE

IIT(ISM) Dhanbad

INDIAN INSTITUTE OF TECHNOLOGY (INDIAN SCHOOL OF MINES)

DHANBAD-826004

Department of Computer Science and Engineering

TO WHOMSOEVER IT MAY CONCERN

This is to certify that SAPIREDDY SWAMI SHANKAR (14JE000436) has successfully completed the project titled **"MOVIE RECOMMENDER SYSTEM**" under my supervision and guidance in the fulfilment of requirements of vacational training, bonafide student of Bachelor of Technology (Computer Science and Engineering) of Indian Institute of Technology (Indian School of Mines), Dhanbad

Dr. A.Chandra Sekhara rao

Assistant Professor

Department of CSE

IIT(ISM) Dhanbad

ABSTRACT

In this project, I have implemented a recommender system for movie recommendations. It allows a user to select his choices from a given set of dataset and then recommend him a movie list based on the similarities between the test movie and all the other movies present in the dataset using the cosine simulator function. By the nature of our system, it is not an easy task to evaluate the performance since there is no right or wrong recommendation; it is just a matter of opinions. I would like to have a larger data set that will enable more meaningful results using the system. Additionally I would like to incorporate different machine learning and clustering algorithms and study the comparative results.

Keywords: K-means, recommendation system, recommender system, data mining, clustering, movies, Collaborative filtering, content based filtering

ACKNOWLEDGEMENT

A project work is a job of great enormity and it can’t be accomplished by an individual all by themselves. Eventually I am grateful to a number of individuals whose professional guidance, assistance and encouragement have made it a pleasant endeavor to undertake this project.

Guidance and deadlines play a very important role in successful completion of the project on time. I am grateful to **Dr. A.Chandra Sekhara rao**, assistant professor, Department of CSE, for having constantly monitored the development of the project.

Finally a note of thanks to the Department of Computer Science Engineering, both teaching and non-teaching staff for their co-operation extended to me.

Sapireddy Swami Shankar

14JE000436

CONTENTS:

1. INTRODUCTION:

Now a day's recommendation system has changed the style of searching the things of our interest. This is information filtering approach that is used to predict the preference of that user. The most popular areas where recommender system is applied are books, news, articles, music, videos, movies etc. In this paper, I have implemented a recommender system for movie recommendations. It allows a user to select his choices from a given set of dataset and then recommend him a movie list based on the similarities between the test movie and all the other movies present in the dataset using the cosine simulator function. It is based on collaborative filtering approach that makes use of the information provided by users, analyses them and then recommends the movies that is best suited to the user at that time. The recommended movie list is sorted according to the ratings given to these movies by previous users and it uses K-means algorithm for this purpose. Help users to find the movies of their choices based on the movie experience of other users in efficient and effective manner without wasting much time in useless browsing. The presented recommender system generates recommendations using various types of knowledge and data about users, the available items, and previous transactions stored in customized databases. The user can then browse the recommendations easily and find a movie of their choice.

What is a recommender system?

A **recommender system** or a **recommendation system** is a subclass of information filtering system that seeks to predict the "rating" or "preference" that a user would give to an item.

Recommender systems have become increasingly popular in recent years, and are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general. There are also recommender systems for experts, collaborators, jokes, restaurants, garments, financial services, life insurance, and Twitter pages.

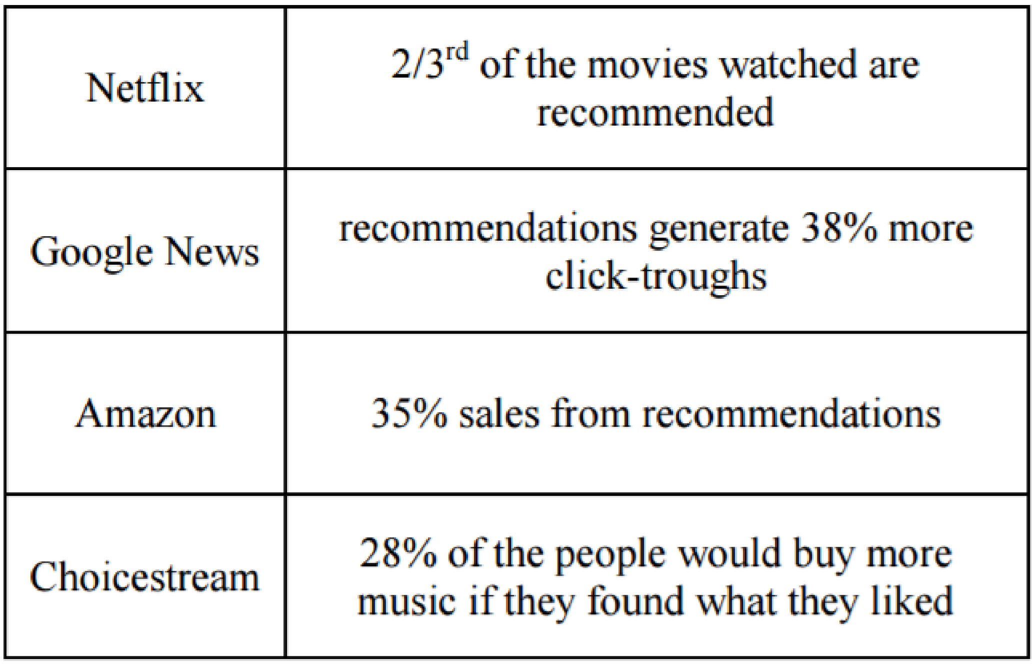
In today's world where internet has become an important part of human life, users often face the problem of too much choice. Right from looking for a motel to looking for good investment options, there is too much information available. To help the users cope with this information explosion, companies have deployed recommendation systems to guide their users. The research in the area of recommendation systems has been going on for several decades now, but the interest still remains high because of the abundance of practical applications and the problem rich domain.

A number of such online recommendation systems implemented and used are the recommendation system for books at Amazon.com , for movies at MovieLens.org, CDs at CDNow.com (from Amazon.com), etc.

Recommender systems identify recommendations autonomously for individual users based on past purchases and searches, and on other users' behaviour.

A Recommender System normally focuses on a specific type of item (e.g., CDs, or news) and accordingly its design, its graphical user interface, and the core recommendation technique used to generate the recommendations are all customized to provide useful and effective suggestions for that specific type of item.

Recommender Systems have added to the economy of the some of the e-commerce websites (like Amazon.com) and Netflix which have made these systems a salient parts of their websites. A glimpse of the profit of some websites is shown in table below:



Netflix	2/3rd of the movies watched are recommended
Google News	recommendations generate 38% more click-troughs
Amazon	35% sales from recommendations
Choicestream	28% of the people would buy more music if they found what they liked

Table1. Companies benefit through recommendation system

Recommender Systems generate recommendations; the user may accept them according to their choice and may also provide, immediately or at a next stage, an implicit or explicit feedback. The actions of the users and their feedbacks can be stored in the recommender database and may be used for generating new recommendations in the next user-system interactions. The economic potential of theses recommender systems have led some of the biggest e-commerce websites (like Amazon.com, snapdeal.com) and the online movie rental company Netflix to make these systems a salient part of their websites. High quality personalized recommendations add another dimension to user experience. The web personalized recommendation systems are recently applied to provide different types of customized information to their respective users. These systems can be applied in various types of applications and are very common now a day.

We can classify the recommender systems in two broad categories:

- Collaborative Filtering Approach
- Content based Filtering Approach

2. COLLABORATIVE FILTERING APPROACH:

Collaborative filtering methods are based on collecting and analysing a large amount of information on users' behaviours, activities or preferences and predicting what users will like based on their similarity to other users. A key advantage of the collaborative filtering approach is that it does not rely on machine analysable content and therefore it is capable of accurately recommending complex items such as movies without requiring an "understanding" of the item itself.

Collaborative filtering is based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past.

When building a model from a user's behaviour, a distinction is often made between explicit and implicit forms of data collection.

Examples of explicit data collection include the following:

- Asking a user to rate an item on a sliding scale.
- Asking a user to search.
- Asking a user to rank a collection of items from favourite to least favourite.
- Presenting two items to a user and asking him/her to choose the better one of them.

- Asking a user to create a list of items that he/she likes.

Examples of implicit data collection include the following:

- Observing the items that a user views in an online store.
- Analysing item/user viewing times.
- Keeping a record of the items that a user purchases online.
- Obtaining a list of items that a user has listened to or watched on his/her computer.
- Analysing the user's social network and discovering similar likes and dislikes.
-

Collaborative filtering system recommends items based on similarity measures between users and/or items. The system recommends those items that are preferred by similar kind of users.

Collaborative filtering has many advantages:

1. It is content-independent i.e. it relies on connections only
2. Since in CF people makes explicit ratings so real quality assessment of items are done.
3. It provides serendipitous recommendations because recommendations are based on user’s similarity rather than item’s similarity.

Collaborative filtering approaches often suffer from three problems: cold start, scalability, and sparsity.

- Cold start: These systems often require a large amount of existing data on a user in order to make accurate recommendations.
- Scalability: In many of the environments in which these systems make recommendations, there are millions of users and products. Thus, a large amount of computation power is often necessary to calculate recommendations.
- Sparsity: The number of items sold on major e-commerce sites is extremely large. The most active users will only have rated a small subset of the overall database. Thus, even the most popular items have very few ratings.

A particular type of collaborative filtering algorithm uses matrix factorisation, a low rank matrix approximation technique.

Collaborative filtering methods are classified as memory-based and model based collaborative filtering. A well-known example of memory-based approaches is user-based algorithm and that of model-based approaches is Kernel-Mapping Recommender.

EXAMPLE:

Suppose you're building a website to recommend blogs. By using the information from many users who subscribe to and read blogs, you can group those users based on their preferences. For example, you can group together users who read several of the same blogs. From this information, you identify the most popular blogs that are read by that group. Then — for a particular user in the group — you recommend the most popular blog that he or she neither reads nor subscribes to.

3. CONTENT-BASED FILTERING APPROACH:

Content-based filtering methods are based on a description of the item and a profile of the user’s preference. In a content-based recommender system, keywords are used to describe the items and a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past (or is examining in the present). In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended. This approach has its roots in information retrieval and information filtering research.

Content-based filtering is based on the profile of the user’s preference and the item’s description. In CBF to describe items we use keywords apart from user’s profile to indicate user’s preferred liked or dislikes. In other words CBF algorithms recommend those items or similar to those items that were liked in the past. It examines previously rated items and recommends best matching item.

To abstract the features of the items in the system, an item presentation algorithm is applied. A widely used algorithm is the tf-idf representation (also called vector space representation).

To create a user profile, the system mostly focuses on two types of information: 1. A model of the user's preference. 2. A history of the user's interaction with the recommender system.

Basically, these methods use an item profile (i.e., a set of discrete attributes and features) characterizing the item within the system. The system creates a content-based profile of users based on a weighted vector of item features. The weights denote the importance of each feature to the user and can be computed from individually rated content vectors using a variety of techniques. Simple approaches use the average values of the rated item vector while other sophisticated methods use machine learning techniques such as Bayesian Classifiers, cluster analysis, decision trees, and artificial neural networks in order to estimate the probability that the user is going to like the item.[32]

Direct feedback from a user, usually in the form of a like or dislike button, can be used to assign higher or lower weights on the importance of certain attributes (using Rocchio classification or other similar techniques).

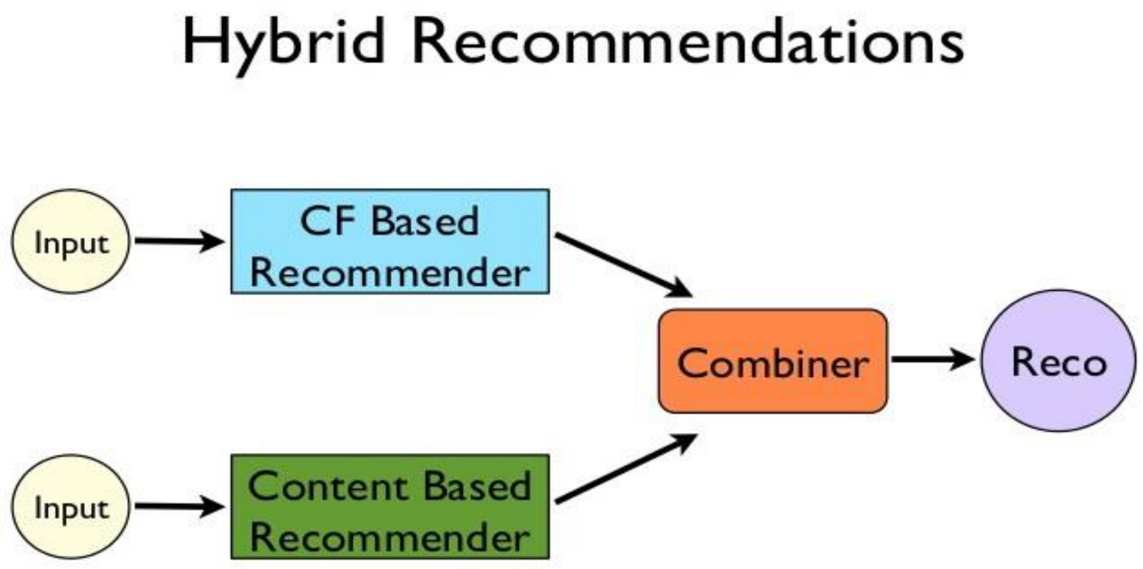
A key issue with content-based filtering is whether the system is able to learn user preferences from users' actions regarding one content source and use them across other content types. When the system is limited to recommending content of the same type as the user is already using, the value from the recommendation system is significantly less than when other content types from other services can be recommended. For example, recommending news articles based on browsing of news is useful, but would be much more useful when music, videos, products, discussions etc. from different services can be recommended based on news browsing.

4. HYBRID RECOMMENDER SYSTEM:

Recent research has demonstrated that a hybrid approach, combining collaborative filtering and content-based filtering could be more effective in some cases. Hybrid approaches can be implemented in several ways: by making content-based and collaborative-based predictions separately and then combining them; by adding content-based capabilities to a collaborative-based approach (and vice versa); or by unifying the approaches into one model (see for a complete review of recommender systems). Several studies empirically compare the performance of the hybrid with the pure collaborative and content-based methods and demonstrate that the hybrid methods can provide more accurate recommendations than pure approaches. These methods can also be used to overcome some of the common problems in recommender systems such as cold start and the sparsity problem.

Netflix is a good example of the use of hybrid recommender systems. The website makes recommendations by comparing the watching and searching habits of similar users (i.e., collaborative filtering) as well as by offering movies that share characteristics with films that a user has rated highly (content-based filtering).

A variety of techniques have been proposed as the basis for recommender systems: collaborative, content-based, knowledge-based, and demographic techniques. Each of these techniques has known shortcomings, such as the well-known cold-start problem for collaborative and content-based systems (what to do with new users with few ratings) and the knowledge engineering bottleneck in knowledge-based approaches. A hybrid recommender system is one that combines multiple techniques together to achieve some synergy between them.



- Collaborative: The system generates recommendations using only information about rating profiles for different users or items. Collaborative systems locate peer users / items with a rating history similar to the current user or item and generate recommendations using this neighbourhood. The user based and the item based nearest neighbour algorithms can be combined to deal with the cold start problem and improve recommendation results.[36]
- Content-based: The system generates recommendations from two sources: the features associated with products and the ratings that a user has given them. Content-based recommenders treat recommendation as a user-specific classification problem and learn a classifier for the user's likes and dislikes based on product features.
- Demographic: A demographic recommender provides recommendations based on a demographic profile of the user. Recommended products can be produced for different demographic niches, by combining the ratings of users in those niches.

- Knowledge-based: A knowledge-based recommender suggests products based on inferences about a user's needs and preferences. This knowledge will sometimes contain explicit functional knowledge about how certain product features meet user needs.

The term hybrid recommender system is used here to describe any recommender system that combines multiple recommendation techniques together to produce its output. There is no reason why several different techniques of the same type could not be hybridized, for example, two different content-based recommenders could work together, and a number of projects have investigated this type of hybrid: NewsDude, which uses both naive Bayes and kNN classifiers in its news recommendations is just one example.

Seven hybridization techniques:

- Weighted: The score of different recommendation components are combined numerically.
- Switching: The system chooses among recommendation components and applies the selected one.
- Mixed: Recommendations from different recommenders are presented together.
- Feature Combination: Features derived from different knowledge sources are combined together and given to a single recommendation algorithm.
- Feature Augmentation: One recommendation technique is used to compute a feature or set of features, which is then part of the input to the next technique.
- Cascade: Recommenders are given strict priority, with the lower priority ones breaking ties in the scoring of the higher ones.
- Meta-level: One recommendation technique is applied and produces some sort of model, which is then the input used by the next technique.

5. IMPLEMENTATION OF A MOVIE RECOMMENDER SYSTEM:

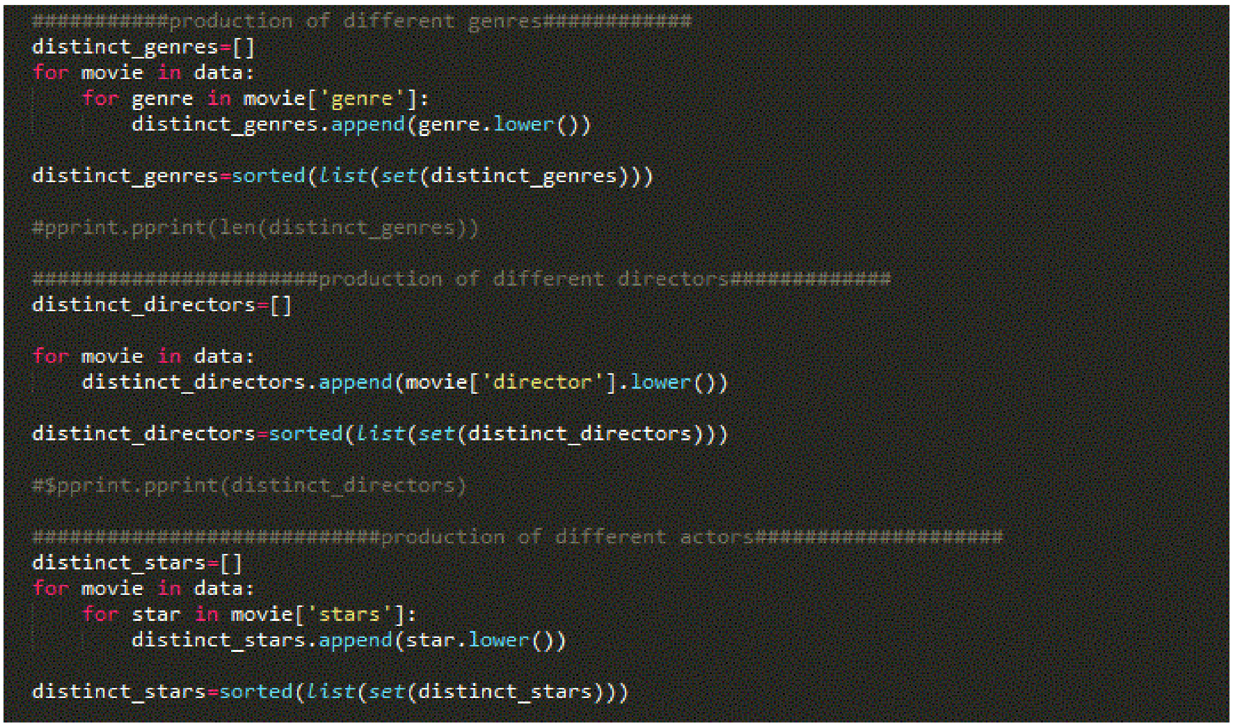
Here, I am using Cosine Simulator as the similarity function to find the recommended movie list for a particular test movie.

APPROACH:

1. Find the attributes which are sparse and fill the sparse places with the median of all the similar kind of attributes. Though there are many efficient ways to fill them I opted median for simplicity.

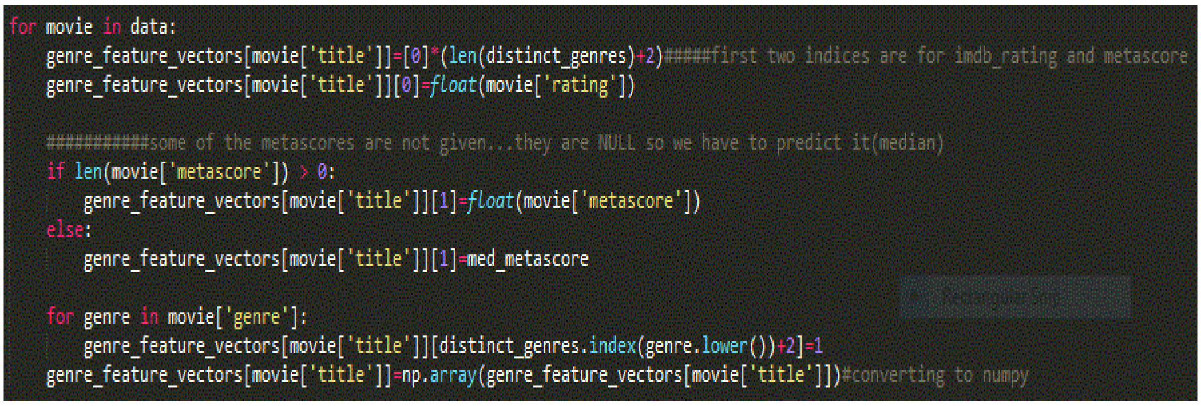
2. Find distinct number of actors, directors, genres to make their feature vectors for all the films.

3. Feature Vector: It is a vector where 1 is stored in every place corresponding to the attributes which are related to them i.e., for "DARK KNIGHT" movies the column corresponding to Christopher Nolan is '1' in director feature vectors.



```
###########production of different genres############
distinct_genres=[]
for movie in data:
    for genre in movie['genre']:
        distinct_genres.append(genre.lower())

distinct_genres=sorted(list(set(distinct_genres)))

#pprint.pprint(len(distinct_genres))

#######################production of different directors#############
distinct_directors=[]

for movie in data:
    distinct_directors.append(movie['director'].lower())

distinct_directors=sorted(list(set(distinct_directors)))

#$pprint.pprint(distinct_directors)

############################production of different actors####################
distinct_stars=[]
for movie in data:
    for star in movie['stars']:
        distinct_stars.append(star.lower())

distinct_stars=sorted(list(set(distinct_stars)))
```

4. Combine all the feature vectors into single array and add IMDB rating, run length etc. All the factors needed to the feature vector



```
for movie in data:
    genre_feature_vectors[movie['title']]=[0]*(len(distinct_genres)+2)#####first two indices are for imdb_rating and metascore
    genre_feature_vectors[movie['title']][0]=float(movie['rating'])

    ###########some of the metascores are not given...they are NULL so we have to predict it(median)
    if len(movie['metascore']) > 0:
        genre_feature_vectors[movie['title']][1]=float(movie['metascore'])
    else:
        genre_feature_vectors[movie['title']][1]=med_metascore

    for genre in movie['genre']:
        genre_feature_vectors[movie['title']][distinct_genres.index(genre.lower())+2]=1
    genre_feature_vectors[movie['title']]=np.array(genre_feature_vectors[movie['title']])#converting to numpy
```

5. Now we have a dictionary with movie titles as keys and feature vectors as values.

6. **Mean Normalization.....i.e..; subtract mean of all the feature vectors from every feature vector and divide it by standard deviation of the same.**

What does the normalization mean?

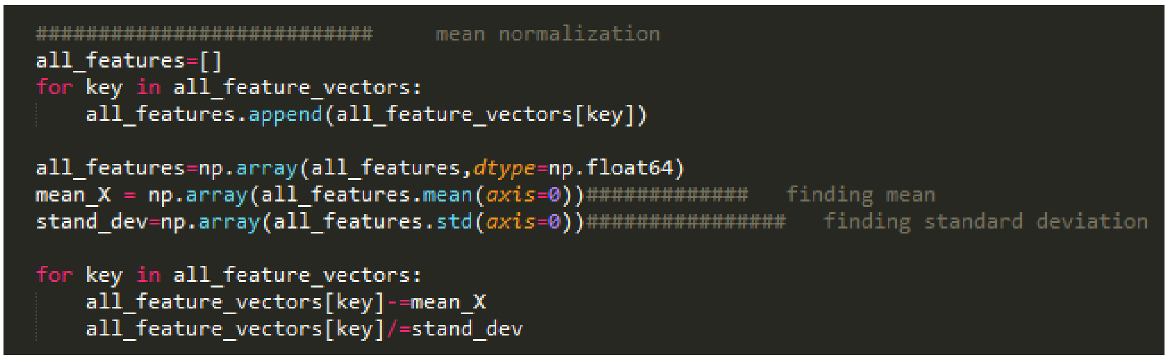
Database **normalization**, or simply **normalization**, is the process of organizing the columns (attributes) and tables (relations) of a relational database to reduce data redundancy and improve data integrity.

The Dot Product

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

$$\vec{a} \cdot \vec{b} = \|\vec{b}\| \|\vec{a}\| \cos\theta$$

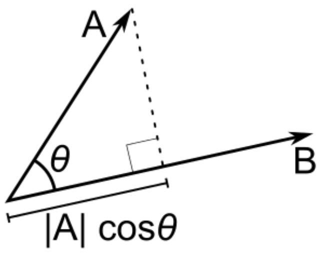
Standard deviation is a measure of the dispersion of a set of data from its **mean**. If the data points are further from the **mean**, there is higher **deviation** within the data set. **Standard deviation** is calculated as the square root of variance by determining the variation between each data point relative to the **mean**.



```
###########################     mean normalization
all_features=[]
for key in all_feature_vectors:
    all_features.append(all_feature_vectors[key])

all_features=np.array(all_features,dtype=np.float64)
mean_X = np.array(all_features.mean(axis=0))#############   finding mean
stand_dev=np.array(all_features.std(axis=0))################   finding standard deviation

for key in all_feature_vectors:
    all_feature_vectors[key]-=mean_X
    all_feature_vectors[key]/=stand_dev
```

7. <u>Use Cosine Similarity to get the top 10 similar movies as shown in the code.</u>

The Cosine Similarity:

- **Cosine similarity** is a measure of **similarity** between two non-zero vectors of an inner product space that measures the **cosine** of the angle between them. The **cosine** of 0° is 1, and it is less than 1 for any other angle.
- 

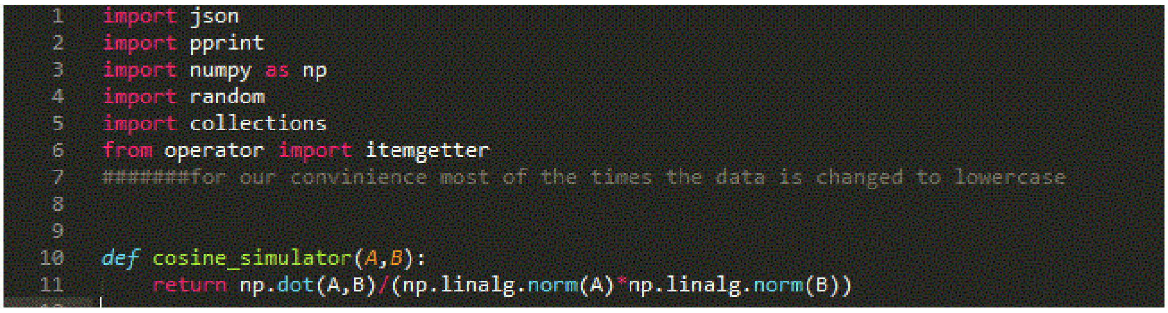
The cosine of two vectors can be derived by using the Euclidean dot product formula:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|\,\|\mathbf{b}\| \cos\theta$$

Given two vectors of attributes, *A* and *B*, the cosine similarity, *cos(θ)*, is represented using a dot product and magnitude as

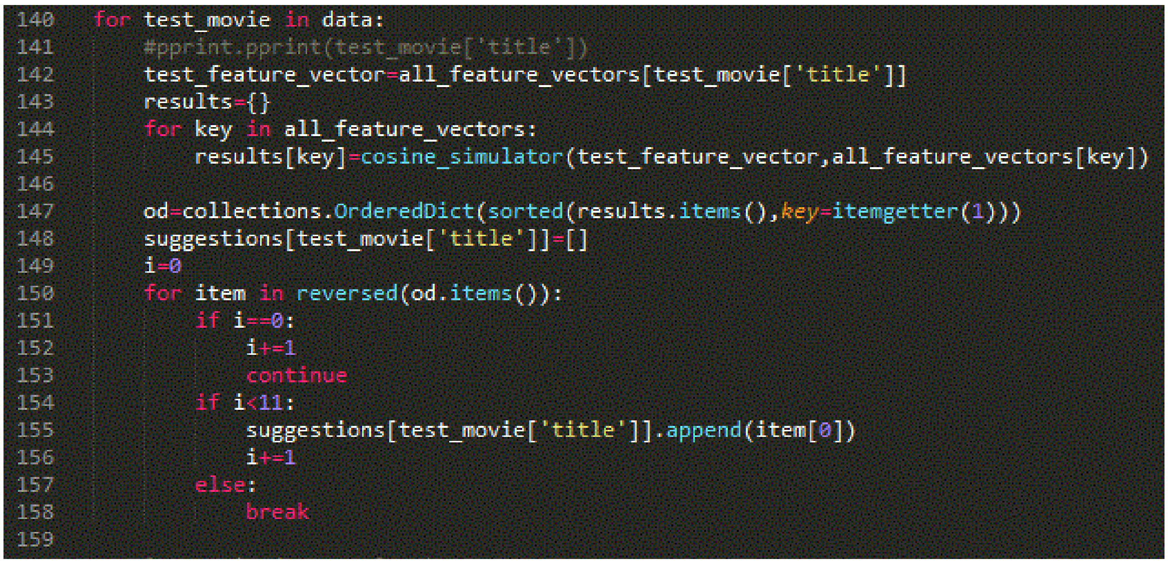
$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

-



```
import json
import pprint
import numpy as np
import random
import collections
from operator import itemgetter
#######for our convinience most of the times the data is changed to lowercase


def cosine_simulator(A,B):
    return np.dot(A,B)/(np.linalg.norm(A)*np.linalg.norm(B))
```

<u>to get the top 10 similar movies as shown in the code.</u>



```
for test_movie in data:
    #pprint.pprint(test_movie['title'])
    test_feature_vector=all_feature_vectors[test_movie['title']]
    results={}
    for key in all_feature_vectors:
        results[key]=cosine_simulator(test_feature_vector,all_feature_vectors[key])

    od=collections.OrderedDict(sorted(results.items(),key=itemgetter(1)))
    suggestions[test_movie['title']]=[]
    i=0
    for item in reversed(od.items()):
        if i==0:
            i+=1
            continue
        if i<11:
            suggestions[test_movie['title']].append(item[0])
            i+=1
        else:
            break

```

6. CODE:

import json

import pprint

import numpy as np

import random

import collections

from operator import itemgetter

#######for our convinience most of the times the data is changed to lowercase

def cosine_simulator(A,B):

```
	return np.dot(A,B)/(np.linalg.norm(A)*np.linalg.norm(B))



#####reading json file##################
with open("input.json") as main_file:
	data=json.load(main_file)



########################finding median for metascore as some metscores are missing in data


li=[]
for movie in data:
	if len(movie['metascore']) >0:
		li.append(float(movie['metascore']))
med_metascore= np.median(np.array(li))

#########################finding median of running time as the data is sparse in this ocation
li=[]
for movie in data:
	if len(movie['running_time']) >0:
		li.append(int(movie['running_time'].split(' ')[0]))
med_running_time= np.median(np.array(li))

###########production of different genres############
distinct_genres=[]
for movie in data:
        for genre in movie['genre']:
                distinct_genres.append(genre.lower())

distinct_genres=sorted(list(set(distinct_genres)))

#pprint.pprint(len(distinct_genres))

########################production of different directors##############
distinct_directors=[]

for movie in data:
        distinct_directors.append(movie['director'].lower())

distinct_directors=sorted(list(set(distinct_directors)))

#$pprint.pprint(distinct_directors)

##############################production of different
actors#####################
distinct_stars=[]
for movie in data:
        for star in movie['stars']:
                distinct_stars.append(star.lower())

distinct_stars=sorted(list(set(distinct_stars)))

########################production of feature vectors where keys are
movie titles and
```

```
#######################values are feature vectors of their genre indicating 1 at that position
genre_feature_vectors={}


for movie in data:
        genre_feature_vectors[movie['title']]=[0]*(len(distinct_genres)+2)#####first two indices are for imdb_rating and metascore
        genre_feature_vectors[movie['title']][0]=float(movie['rating'])


        ###########some of the metascores are not given...they are NULL so we have to predict it(median)
        if len(movie['metascore']) > 0:

        genre_feature_vectors[movie['title']][1]=float(movie['metascore'])
        else:
                genre_feature_vectors[movie['title']][1]=med_metascore


        for genre in movie['genre']:


        genre_feature_vectors[movie['title']][distinct_genres.index(genre.lower())+2]=1
        genre_feature_vectors[movie['title']]=np.array(genre_feature_vectors[movie['title']])#converting to numpy



#############production od feature vectors where  keys are movie titles and values are
#############feature vectors of which director directed that movie...indicated by 1 at that position
director_feature_vectors={}
```

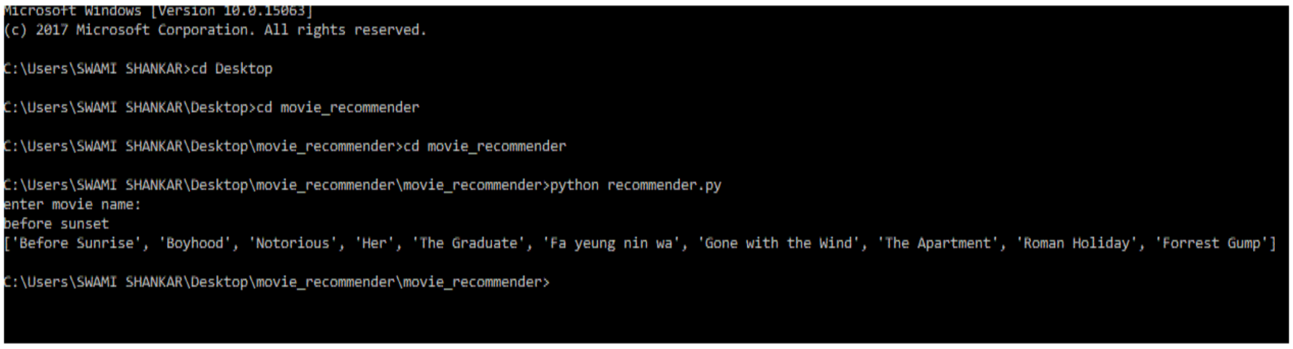
```
for movie in data:

        director_feature_vectors[movie['title']]=[0]*len(distinct_directors)

        director_feature_vectors[movie['title']][distinct_directors.index(movie[
'director'].lower())]=1

        director_feature_vectors[movie['title']]=np.array(director_feature_vect
ors[movie['title']])#converting to numpy


#pprint.pprint(director_feature_vectors)


#############production od feature vectors where  keys are movie titles
and values are

#############feature vectors of which actors acted in that movie...indicated
by 1 at that position


actor_feature_vectors={}


for movie in data:

        actor_feature_vectors[movie['title']]=[0]*len(distinct_stars)

        for star in movie['stars']:


        actor_feature_vectors[movie['title']][distinct_stars.index(star.lower())]
=1

        actor_feature_vectors[movie['title']]=np.array(actor_feature_vectors[m
ovie['title']])#converting to numpy


#pprint.pprint(actor_feature_vectors)


###############Combining all the feature vectors to make a single
feature vector as the keys in all the dicts are same ...i.e; the movies
```

```
all_feature_vectors={}

for key in genre_feature_vectors:
        all_feature_vectors[key]=np.append(np.append(genre_feature_vectors[
key],director_feature_vectors[key]),actor_feature_vectors[key])

#pprint.pprint(all_feature_vectors)

######adding votes ,, release year and run length
for movie in data:
        np.append(all_feature_vectors[movie['title']],int(movie['votes'].replace
(',', '')))
        np.append(all_feature_vectors[movie['title']],float(movie['year']))
        if len(movie['running_time']) > 0:

        np.append(all_feature_vectors[movie['title']],int(movie['running_time']
.split(' ')[0]))
        else:

        np.append(all_feature_vectors[movie['title']],med_running_time)

############################     mean normalization
all_features=[]
for key in all_feature_vectors:
        all_features.append(all_feature_vectors[key])

all_features=np.array(all_features,dtype=np.float64)
```

```
mean_X = np.array(all_features.mean(axis=0))############# finding mean
stand_dev=np.array(all_features.std(axis=0))################ finding standard deviation


for key in all_feature_vectors:
    all_feature_vectors[key]-=mean_X
    all_feature_vectors[key]/=stand_dev




############### forming output
suggestions={}
for test_movie in data:
    #pprint.pprint(test_movie['title'])
    test_feature_vector=all_feature_vectors[test_movie['title']]
    results={}
    for key in all_feature_vectors:

    results[key]=cosine_simulator(test_feature_vector,all_feature_vectors[key])


    od=collections.OrderedDict(sorted(results.items(),key=itemgetter(1)))
    suggestions[test_movie['title']]=[]
    i=0
    for item in reversed(od.items()):
        if i==0:
            i+=1
            continue
```

```
                if i<11:
                        suggestions[test_movie['title']].append(item[0])
                        i+=1
                else:
                        break

#pprint.pprint(suggestions)

######writing output to JSON file
with open('result.json','w') as f:
        json.dump(suggestions,f,indent=2)
```

SAMPLE OUTPUT:



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\SWAMI SHANKAR>cd Desktop

C:\Users\SWAMI SHANKAR\Desktop>cd movie_recommender

C:\Users\SWAMI SHANKAR\Desktop\movie_recommender>cd movie_recommender

C:\Users\SWAMI SHANKAR\Desktop\movie_recommender\movie_recommender>python recommender.py
enter movie name:
before sunset
['Before Sunrise', 'Boyhood', 'Notorious', 'Her', 'The Graduate', 'Fa yeung nin wa', 'Gone with the Wind', 'The Apartment', 'Roman Holiday', 'Forrest Gump']

C:\Users\SWAMI SHANKAR\Desktop\movie_recommender\movie_recommender>
```

```
"The Shawshank Redemption": [
   "The Green Mile",
   "Se7en",
   "Unforgiven",
   "Million Dollar Baby",
   "The Godfather: Part II",
   "The Godfather",
   "Pulp Fiction",
   "Goodfellas",
   "Fight Club",
   "12 Angry Men"
 ],
 "The Godfather": [
   "The Godfather: Part II",
   "Apocalypse Now",
   "On the Waterfront",
   "Dog Day Afternoon",
   "Heat",
   "Scarface",
   "The Shawshank Redemption",
   "Pulp Fiction",
   "Goodfellas",
   "Seven Samurai"
 ],
```

CONCLUSION:

In this project, I have implemented a recommender system for movie recommendations. It allows a user to select his choices from a given set of dataset and then recommend him a movie list based on the similarities between the test movie and all the other movies present in the dataset using the cosine simulator function. By the nature of our system, it is not an easy task to evaluate the performance since there is no right or wrong recommendation; it is just a matter of opinions. I would like to have a larger data set that will enable more meaningful results using the system. Additionally I would like to incorporate different machine learning and clustering algorithms and study the comparative results.

REFERENCES

- Choi, S.-M., Han, Y.-S.: A content recommendation system based on category correlations. In: The Fifth International Multi-Conference on Computing in the Global Information Technology, pp. 1257–1260 (2010)
- Huang, Z., Chen, H., Zeng, and D.D.: Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. ACM Trans. Inf. Syst. 22(1), 116–142 (2004)
- https://en.wikipedia.org/
- https://www.youtube.com/channel/Video Tutorials - All in One