**SINGULAR VALUE DECOMPOSITION (SVD) & ITS APPLICATION IN RECOMMENDER SYSTEM**

Ref: <https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-system/>

Singular Value Decomposition (SVD), a classical method from linear algebra is getting popular in the field of data science and machine learning. This popularity is because of its application in developing recommender systems. There are a lot of online user-centric applications such as video players, music players, e-commerce applications, etc., where users are recommended with further items to engage with. Finding and recommending many suitable items that would be liked and selected by users is always a challenge. There are many techniques used for this task and SVD is one of those techniques. This article presents a brief introduction to recommender systems, an introduction to singular value decomposition and its implementation in movie recommendation.

### **What is a Recommender System?**

A recommender system is an intelligent system that predicts the rating and preferences of users on products. The primary application of recommender systems is finding a relationship between user and products in order to maximise the user-product engagement. The major application of recommender systems is in suggesting related video or music for generating a playlist for the user when they are engaged with a related item. A similar application is in the field of e-commerce where customers are recommended with the related products, but this application involves some other techniques such as association rule learning. It is also used to recommend contents based on user behaviours on social media platforms and news websites.

There are two popular approaches used in recommender systems to suggest items to the users:

1. **Collaborative Filtering:** The assumption of this approach is that people who have liked an item in the past will also like the same in future. This approach builds a model based on the past behaviour of users. The user behaviour may include previously watched videos, purchased items, given ratings on items. In this way, the model finds an association between the users and the items. The model is then used to predict the item or a rating for the item in which the user may be interested. Singular value decomposition is used as a collaborative filtering approach in recommender systems.
2. **Content-Based Filtering:** This approach is based on a description of the item and a record of the user’s preferences. It employs a sequence of discrete, pre-tagged characteristics of an item in order to recommend additional items with similar properties. This approach is best suited when there is sufficient information available on the items but not on the users. Content-based recommender systems also include the opinion-based recommender system.

Apart from the above two approaches, there are few more approaches to build recommender systems such as multi-criteria recommender systems, risk-aware recommender systems, mobile recommender systems, and hybrid recommender systems (combining collaborative filtering and content-based filtering).

### **Singular Value Decomposition**

The Singular Value Decomposition (SVD), a method from linear algebra that has been generally used as a dimensionality reduction technique in machine learning. SVD is a matrix factorisation technique, which reduces the number of features of a dataset by reducing the space dimension from N-dimension to K-dimension (where K<N). In the context of the recommender system, the SVD is used as a collaborative filtering technique. It uses a matrix structure where each row represents a user, and each column represents an item. The elements of this matrix are the ratings that are given to items by users.

The factorisation of this matrix is done by the singular value decomposition. It finds factors of matrices from the factorisation of a high-level (user-item-rating) matrix. The singular value decomposition is a method of decomposing a matrix into three other matrices as given below: Where *A*is a *m x n*utility matrix, *U*is a *m x r* orthogonal left singular matrix, which represents the relationship between users and latent factors, *S*is a *r x r*diagonal matrix, which describes the strength of each latent factor and *V*is a *r x n*diagonal right singular matrix, which indicates the similarity between items and latent factors. The latent factors here are the characteristics of the items, for example, the genre of the music. The SVD decreases the dimension of the utility matrix *A*by extracting its latent factors. It maps each user and each item into a *r*-dimensional latent space. This mapping facilitates a clear representation of relationships between users and items.

Let each item be represented by a vector *xi*and each user is represented by a vector *yu*. The expected rating by a user on an item  can be given as:Here,  is a form of factorisation in singular value decomposition. The *xi* and *yu* can be obtained in a manner that the square error difference between their dot product and the expected rating in the user-item matrix is minimum. It can be expressed as:In order to let the model generalise well and not overfit the training data, a regularisation term is added as a penalty to the above formula. singular value decomposition machine learningIn order to reduce the error between the value predicted by the model and the actual value, the algorithm uses a bias term. Let for a user-item pair *(u, i)*, *μ* is the average rating of all items, *bi* is the average rating of item *i*minus *μ* and *bu* is the average rating given by user *u*minus *μ*, the final equation after adding the regularisation term and bias can be given as:singular value decomposition algorithm

The above equation is the main component of the algorithm which works for singular value decomposition based recommendation system.

### **Singular Value Decomposition (SVD) based Movie Recommendation**

Below is an implementation of singular value decomposition (SVD) based on collaborative filtering in the task of movie recommendation. This task is implemented in Python. For simplicity, the [MovieLens 1M Dataset](https://grouplens.org/datasets/movielens/1m/) has been used. This dataset has been chosen because it does not require any preprocessing as the main focus of this article is on SVD and recommender systems.

Import the required python libraries:

import numpy as np  
import pandas as pd

Read the dataset from where it is downloaded in the system. It consists of two files ‘ratings.dat’ and ‘movies.dat’ which need to be read.

data = pd.io.parsers.read\_csv('data/ratings.dat',   
    names=['user\_id', 'movie\_id', 'rating', 'time'],  
    engine='python', delimiter='::')  
  
movie\_data = pd.io.parsers.read\_csv('data/movies.dat',  
    names=['movie\_id', 'title', 'genre'],  
    engine='python', delimiter='::')

Create the rating matrix with rows as movies and columns as users.

ratings\_mat = np.ndarray(  
    shape=(np.max(data.movie\_id.values), np.max(data.user\_id.values)),  
    dtype=np.uint8)  
ratings\_mat[data.movie\_id.values-1, data.user\_id.values-1] = data.rating.values

Normalise the matrix.

normalised\_mat = ratings\_mat - np.asarray([(np.mean(ratings\_mat, 1))]).T

Compute the Singular Value Decomposition (SVD).

A = normalised\_mat.T / np.sqrt(ratings\_mat.shape[0] - 1)  
U, S, V = np.linalg.svd(A)

Define a function to calculate the cosine similarity. Sort by most similar and return the top N results.

def top\_cosine\_similarity(data, movie\_id, top\_n=10):  
    index = movie\_id - 1 # Movie id starts from 1 in the dataset  
    movie\_row = data[index, :]  
    magnitude = np.sqrt(np.einsum('ij, ij -> i', data, data))  
    similarity = np.dot(movie\_row, data.T) / (magnitude[index] \* magnitude)  
    sort\_indexes = np.argsort(-similarity)  
    return sort\_indexes[:top\_n]

Define a function to print top N similar movies.

def print\_similar\_movies(movie\_data, movie\_id, top\_indexes):  
    print('Recommendations for {0}: \n'.format(  
    movie\_data[movie\_data.movie\_id == movie\_id].title.values[0]))  
    for id in top\_indexes + 1:  
        print(movie\_data[movie\_data.movie\_id == id].title.values[0])

Initialise the value of *k* principal components, id of the movie as given in the dataset, and number of top elements to be printed.

k = 50  
movie\_id = 10 # (getting an id from movies.dat)  
top\_n = 10  
sliced = V.T[:, :k] # representative data  
indexes = top\_cosine\_similarity(sliced, movie\_id, top\_n)

Print the top N similar movies.

print\_similar\_movies(movie\_data, movie\_id, indexes)

Binding all together:

#Importing Libraries  
import numpy as np  
import pandas as pd  
  
  
#Reading dataset (MovieLens 1M movie ratings dataset: downloaded from https://grouplens.org/datasets/movielens/1m/)  
data = pd.io.parsers.read\_csv('data/ratings.dat',   
    names=['user\_id', 'movie\_id', 'rating', 'time'],  
    engine='python', delimiter='::')  
movie\_data = pd.io.parsers.read\_csv('data/movies.dat',  
    names=['movie\_id', 'title', 'genre'],  
    engine='python', delimiter='::')  
  
  
#Creating the rating matrix (rows as movies, columns as users)  
ratings\_mat = np.ndarray(  
    shape=(np.max(data.movie\_id.values), np.max(data.user\_id.values)),  
    dtype=np.uint8)  
ratings\_mat[data.movie\_id.values-1, data.user\_id.values-1] = data.rating.values  
  
#Normalizing the matrix(subtract mean off)  
normalised\_mat = ratings\_mat - np.asarray([(np.mean(ratings\_mat, 1))]).T  
  
#Computing the Singular Value Decomposition (SVD)  
A = normalised\_mat.T / np.sqrt(ratings\_mat.shape[0] - 1)  
U, S, V = np.linalg.svd(A)  
  
#Function to calculate the cosine similarity (sorting by most similar and returning the top N)  
def top\_cosine\_similarity(data, movie\_id, top\_n=10):  
    index = movie\_id - 1 # Movie id starts from 1 in the dataset  
    movie\_row = data[index, :]  
    magnitude = np.sqrt(np.einsum('ij, ij -> i', data, data))  
    similarity = np.dot(movie\_row, data.T) / (magnitude[index] \* magnitude)  
    sort\_indexes = np.argsort(-similarity)  
    return sort\_indexes[:top\_n]  
  
# Function to print top N similar movies  
def print\_similar\_movies(movie\_data, movie\_id, top\_indexes):  
    print('Recommendations for {0}: \n'.format(  
    movie\_data[movie\_data.movie\_id == movie\_id].title.values[0]))  
    for id in top\_indexes + 1:  
        print(movie\_data[movie\_data.movie\_id == id].title.values[0])  
  
#k-principal components to represent movies, movie\_id to find recommendations, top\_n print n results          
k = 50  
movie\_id = 10 # (getting an id from movies.dat)  
top\_n = 10  
sliced = V.T[:, :k] # representative data  
indexes = top\_cosine\_similarity(sliced, movie\_id, top\_n)  
  
#Printing the top N similar movies  
print\_similar\_movies(movie\_data, movie\_id, indexes)

Some sample outputs are given below:

A picture containing bird

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA picture containing bird

Description automatically generated