**NATIONAL ECONOMICS UNIVERSITY**

**SCHOOL OF ADVANCED EDUCATION PROGRAMS**

**🙟🕮🙝**

A logo with a map and a red and blue circle

Description automatically generated

**DATA SCIENCE PROJECT**

**Building a Movie/Show Recommendation System for Hutu TV**

**Student:** Phan Thuy Anh

**Student ID:** 11204501

**Class:** Business Analytics 62

**Course:** Big Data

**Lecturer:** Oluwatosin Ogundare

**Hanoi, November 2023**

**TABLE OF CONTENTS**

[I. INTRODUCTION 3](#_Toc150296085)

[1. Project overview 3](#_Toc150296086)

[2. Dataset overview 3](#_Toc150296087)

[II. DATA PREPROCESSING 4](#_Toc150296088)

[1. Loading data 4](#_Toc150296089)

[2. Cleaning data 5](#_Toc150296090)

[a) Handling missing values and duplicates 5](#_Toc150296091)

[b) Encoding the categorical data 7](#_Toc150296092)

[III. EXPLORATORY DATA ANALYSIS 7](#_Toc150296093)

[1. Distribution of attributes 7](#_Toc150296094)

[2. Correlation between numeric attributes 9](#_Toc150296095)

[3. Top 10 movies/shows have highest IMDB score 10](#_Toc150296096)

[4. Maximum popularity by release year 11](#_Toc150296097)

[IV. BUILDING RECOMMENDATION SYSTEM FOR HULUTV 11](#_Toc150296098)

[1. Content-based Recommendation System 11](#_Toc150296099)

[a) Building content-based recommendation system 12](#_Toc150296100)

[b) Testing content-based recommendation system 13](#_Toc150296101)

[2. Collaborative Filtering Recommendation System with Singular Value Decomposition model 13](#_Toc150296102)

[a) Building collaborative filtering recommendation system with SVD 14](#_Toc150296103)

[b) Testing collaborative filtering recommendation system with SVD 15](#_Toc150296104)

[3. Evaluation 15](#_Toc150296105)

[V. CONCLUSION 16](#_Toc150296106)

[References 16](#_Toc150296107)

# **I. INTRODUCTION**

## ***1. Project overview***

My project aims to develop a recommendation system for Hutu TV using a dataset consisting of TV show information. By leveraging this comprehensive dataset and applying advanced machine learning techniques using Tensorflow and Sciki-learn, the objective is to create a highly accurate recommendation system tailored to the specific content offerings of Hutu TV. This system will enhance the viewing experience for users by providing personalized content suggestions based on their preferences and interests, ultimately leading to increased user engagement and satisfaction.

## ***2. Dataset overview***

The dataset is available in Kaggle, including ID, title, type, description, release year, age certification, runtime, genres, production countries, seasons, IMDb ID, IMDb score, IMDb votes, TMDB popularity, and TMDB score. The dataset contains a diverse range of TV shows with details spanning various genres and release years.

The data dictionary is:

|  |  |
| --- | --- |
| **Attributes** | **Description of attributes** |
| id | Unique identifier for each movie/show |
| title | The title of the movie or show. |
| type | Whether the entry is a movie or a show. |
| description | A brief summary or synopsis of the movie or show's plot or content. |
| release\_year | The year the movie or show was released. |
| age\_certification | The age group for which the content is deemed appropriate. |
| runtime | The duration of the movie or show in minutes. |
| genres | The categories or genres that the movie or show belongs to. |
| production\_countries | The countries where the movie or show was produced. |
| seasons | The number of seasons available. |
| imdb\_id | The unique identifier for the movie or show on IMDb. |
| imdb\_score | The rating score given to the movie or show on IMDb. |
| imdb\_votes | The number of votes/ratings received on IMDb. |
| tmdb\_popularity | A popularity score for the movie or show on TMDb |
| tmdb\_score | The rating score given to the movie or show on TMDb. |

The dataset information is:

#Explore data

df.shape



#Explore data

df.info()

A screenshot of a computer

Description automatically generated

# **II. DATA PREPROCESSING**

## ***1. Loading data***

After downloading the data file in .csv, I load the data using the code pd.read\_csv(“path”):

#Load data

df = pd.read\_csv("C:\\Users\\phnth\\Downloads\\hulutv.csv")

In every steps of the project, I import necessary libraries in Python (mostly using pandas, numpy, matplotlib, scikit-learn and tensorflow) and put all the imports in the head of codes.

#Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics.pairwise import linear\_kernel

import tensorflow as tf

from tensorflow.keras.models import Model

from keras.models import Sequential

from tensorflow.keras.layers import Input, Embedding, Flatten, Dense, Concatenate

## ***2. Cleaning data***

### *a) Handling missing values and duplicates*

In handling missing values, first, I check for the number of null values that available in dataset and also the proportion of null values to decide whether the attribute could be drop off:

#Check for missing values

missing\_values = df.isnull().sum()

print("Missing Values:")

missing\_values

def missing\_values\_table(df):

  mis\_val = df.isnull().sum()

  mis\_val\_percent = 100\*df.isnull().sum() / len(df)

  mis\_val\_table = pd.concat([mis\_val, mis\_val\_percent], axis=1)

  mis\_val\_table\_ren\_columns = mis\_val\_table.rename(columns = {0:'Missing Values' , 1:'% of Total Values'})

  return mis\_val\_table\_ren\_columns.round(1)

missing\_values\_table(df)

The result is:

A black and white table with white text

Description automatically generated

The missing value table is shown that we have to deal with most of null values in attributes: ‘seasons’ and ‘age\_certification’.

Dealing with null values in attributes that have small proportions: 'description', 'imdb\_score', 'imdb\_score', 'imdb\_votes', 'tmdb\_popularity', 'tmdb\_score', I drop all rows that have null values in those attributes:

#Drop the null values

null = ['description','imdb\_score', 'imdb\_score', 'imdb\_votes', 'tmdb\_popularity', 'tmdb\_score']

df.dropna(subset=null , inplace=True)

For the attribute which have largest null values, I decide to:

- ‘season’: I replace the null values with value “0”:

#Deal with "season"

null\_season\_rows = df[df['seasons'].isnull()]

null\_season\_rows

A screenshot of a computer

Description automatically generated

We can see that only type "MOVIE" has null value in "seasons"

df['seasons'].fillna(0, inplace=True)

- ‘age certification’: Because I don’t find any formula or calculation of ‘age certification’, so I can not see if the missing values are not known or did not have the requirement of age. Since I do not use this attribute, I will drop out of ‘age certification’:

#Deal with "age certification"

df.drop(['age\_certification'], axis=1,inplace=True)

After all coding to deal with null values, I double check the number of null values to ensure there are no missing values in my dataset:

#Check missing value again

df.isnull().sum()

A black rectangle with white text

Description automatically generated

Next, I check the number of duplicate rows and see clearly that there are no duplicates in my dataset:

#Check for Duplicates

df.duplicated().sum()



### *b) Encoding the categorical data*

Encoding categorical variables is essential to convert non-numeric categories into numerical format for algorithms to effectively process. There are 2 necessary categorical attributes that I decide to encode:

#Encoding the neccessary categorical variables

encoder = LabelEncoder()

df["title\_encoded"] = encoder.fit\_transform(df["title"])

df["type\_encoded"] = encoder.fit\_transform(df["type"])

# **III. EXPLORATORY DATA ANALYSIS**

Because this project does not concentrate in data mining, I only show some interesting point in exploring the dataset.

## ***1. Distribution of attributes***

# Distribution of attributes

features = ['release\_year','runtime', 'imdb\_score','imdb\_votes', 'tmdb\_popularity','tmdb\_score']

for feat in features:

    plt.figure()

    sns.distplot(df[feat], kde = False)

|  |  |
| --- | --- |
| **Shape of distribution:** Negative skew  **Insight:**  - The amount of released movies and shows are increasing over time from 1970 to 2020.  - The highest year that have largest movies/shows is 2018.   * It can be seen the tendency movies/shows is improved, showing the growing of movie industry | **Shape of distribution:** Bimodal  **Insight:**  - There are two distinct peaks or modes in the data.   * There are two common durations that appear more frequently than others.   (about 25 minutes of runtime and 100 minutes of runtimes) |
| **Shape of distribution:** Negative skew  **Insight:**  - The majority of the scores are concentrated towards the higher end of the scale.   * There are more movies or shows with higher IMDb scores, and fewer with lower scores. * A positive sentiment towards the content, indicating that viewers tend to rate movies and shows more favorably. | **Shape of distribution:** Positive skew  **Insight:**  - The majority of movies or shows have a lower number of votes, while there are a few with a significantly higher number of votes.   * Most content may not have received widespread attention or popularity, but there are a few outliers that garnered a large number of votes. |
| **Shape of distribution:** Positive skew  **Insight:**  - There are a small number of highly popular movies or shows that stand out from the rest | **Shape of distribution:** Normal  **Insight:**  - Data points are symmetrically distributed around a central value.   * The scoring system is well-balanced |

## ***2. Correlation between numeric attributes***

# Correlation between numeric attributes

plt.figure(figsize = (12, 6))

sns.heatmap(df[features].corr(), annot = True, cmap = 'rocket\_r')

A screenshot of a computer screen

Description automatically generated

From the heatmap, It can see that there are no variables that have extremely strong correlation. The highest is belong to positive relationship of tmdb\_score and imdb\_score (0.63). There is some degree of agreement between the scores assigned on both platforms. Some interesting point is the negative relationship of release\_year and imdb\_scores and imdb\_votes: indicate that as the release year of a movie or show increases, there tends to be a decrease in both IMDb scores and the number of IMDb votes. This could indicate that older content may not receive as much attention or may not be rated as highly on IMDb compared to more recent releases.

## ***3. Top 10 movies/shows have highest IMDB score***

#Highest IMDB score

topRatedMovies = df.nlargest(10, 'imdb\_score')

topRatedMoviesPlot = sns.barplot(x=topRatedMovies['title'], y=topRatedMovies['imdb\_score'], palette='viridis')

topRatedMoviesPlot.set\_xlabel('Movie Titles')

topRatedMoviesPlot.set\_ylabel('IMDb Score')

topRatedMoviesPlot.set\_title('Top 10 Movies with Highest IMDb Scores')

plt.xticks(rotation=45, ha='right')

sns.despine(left=True, bottom=True)

plt.show()

A chart of different colors

Description automatically generated

The top 10 movies/shows has highest IMDB score don't have a significant difference in score between them.

## ***4. Maximum popularity by release year***

# Maximum popularity by release year

popByYear = df.groupby('release\_year')['tmdb\_popularity'].max()

sns.lineplot(x = popByYear.index, y = popByYear)

A graph of a number of years

Description automatically generated

During 2000-2005, there were movies with particularly high levels of popularity compared to other years.

# **IV. BUILDING RECOMMENDATION SYSTEM FOR HULUTV**

A recommendation system allows us to learn from user or song data and build models that predict other songs that a user might like.

There are 2 common types of recommendation systems:

* Content-Based: Focus on attributes of items and recommends based on similarity
* Collaborative Filtering: Focus on wisdom of the crowd, recommends based on other users

In this project, I will conduct 2 types of recommendation system, but note that my dataset does not have the user information so that I need to assume and create a user-item matrix in collaborative filtering.

## ***1. Content-based Recommendation System***

I will use a distance metric known as cosine similarity in building content-based recommendation system.

Cosine similarity is defined as:

A math equation with black text

Description automatically generated with medium confidence

***How it works:*** Let's consider two vectors, A and B, where Ai and Bi are components representing the various features associated with each movie or show. By calculating the cosine similarity between these two vectors, we obtain a measure of their similarity. A higher cosine similarity value indicates a greater resemblance, which translates to a higher position on the recommendation list.

***How I apply:*** This method allows us to quantitatively assess how closely related two movies or shows are in terms of their descriptions. Those with higher cosine similarity scores are considered more similar and are therefore more likely to be recommended to viewers who have shown interest in one of them.

### *a) Building content-based recommendation system*

I use the Term Frequency-Inverse Document Frequency (TF-IDF) to count occurrences of words in ‘description’ and weighing the importance of words to calculate a score.

#Recommendation system with Content-based: Cosine Similarity Model

#Remove all english stop words such as 'the', 'a'

tfidf = TfidfVectorizer(stop\_words='english')

#Fit and transform the data

tfidf\_matrix = tfidf.fit\_transform(df['description'])

tfidf\_matrix.shape



I begin by calculating the cosine similarity between all pairs of items, which quantifies their similarity. I then establish a mapping between titles and dataset indices for quick retrieval. Next, I modify the get\_recommendations function to efficiently return the top 10 similar titles for a given input. The function utilizes the precomputed cosine similarity matrix to streamline the recommendation process.

#Calculate Cosine Similarity

cosine\_sim = linear\_kernel(tfidf\_matrix, tfidf\_matrix)

indices = pd.Series(df.index, index=df['title']).drop\_duplicates()

#Function

def get\_recommendations(title, cosine\_sim=cosine\_sim):

    idx = indices[title]

    sim\_scores = list(enumerate(cosine\_sim[idx]))

    sim\_scores = sorted(sim\_scores, key=lambda x: x[1], reverse=True)

    sim\_scores = sim\_scores[1:11]

    movie\_indices = [i[0] for i in sim\_scores]

    similarity\_scores = [i[1] for i in sim\_scores]

    results = pd.DataFrame({'title': df['title'].iloc[movie\_indices], 'cosine\_similarity': similarity\_scores})

    return results

### *b) Testing content-based recommendation system*

#Test content-based recommendation system

get\_recommendations('Battleground')

A screenshot of a computer

Description automatically generated

After seeing the result, specially the cosine scores, we can see that they are small (often below 0,1). Given this observation, it is imperative for me to explore alternative approaches in constructing a more effective recommendation system. Let's take a look at some other models and see if they yield better results.

## ***2. Collaborative Filtering Recommendation System with Singular Value Decomposition model***

***How it works:*** Through SVD, the user-item matrix that I use is decomposed into three separate matrices: one for users, one for items, and a diagonal matrix representing the importance of each latent feature. This reduction of dimensionality allows the system to focus on the most significant characteristics while discarding less relevant ones. With these reduced matrices, the system can estimate missing ratings and reconstruct the original user-item matrix.

***How I apply:*** SVD-based recommendation system provides personalized suggestions by utilizing latent features from user-item interactions.

### *a) Building collaborative filtering recommendation system with SVD*

I create a user-item matrix to simulate a single user's ratings for all available items. The SVD model learns embeddings for users and items. Training focuses on minimizing mean squared error (MSE) with the Adam optimizer. Post-training, embeddings are used to compute similarities between items. Recommendations are sorted based on similarity scores. The results are then stored in a DataFrame, 'result', containing recommended titles and their similarity scores.

#Recommendation system with SVD

#Create user-item matrix

num\_users = 1

num\_items = len(df)

user\_item\_matrix = np.zeros((num\_users, num\_items))

#Assume all items have the same ratings for this user

ratings = np.ones(num\_items)

user\_item\_matrix[0, :] = ratings

#Define the SVD model

embedding\_dim = 10

SVDmodel = tf.keras.Sequential([

    tf.keras.layers.Embedding(num\_items, embedding\_dim, input\_length=num\_items),

    tf.keras.layers.Reshape((embedding\_dim, num\_items)),

    tf.keras.layers.Lambda(lambda x: tf.reduce\_mean(x, axis=1))

])

#Compile the model

SVDmodel.compile(optimizer='adam', loss='mse')

#Train the model

SVDmodel.fit(user\_item\_matrix, user\_item\_matrix, epochs=10)

#Use the learned embeddings to make recommendations

user\_embedding = SVDmodel.layers[0].get\_weights()[0][0]

similarities = np.dot(model.layers[0].get\_weights()[0], user\_embedding)

#Combine recommendations with their similarity scores

recommendations = [(df['title'].iloc[i], similarities[i]) for i in range(num\_items)]

#Sort recommendations by similarity score

recommendations.sort(key=lambda x: x[1], reverse=True)

#Create a DataFrame to store the results

SVD\_recommenddations = pd.DataFrame({'title': [rec[0] for rec in recommendations], 'similarities': [rec[1] for rec in recommendations]})

#Return the result DataFrame

SVD\_recommendations

### *b) Testing collaborative filtering recommendation system with SVD*

A screenshot of a black and white screen

Description automatically generated

The testing result based on ratings-recommendation system also have low similarity scores. This suggests that the system may not be effectively identifying and recommending content. This could be due to insufficient data for accurate recommendations.

## ***3. Evaluation***

#Calculate evaluation metric

actual\_ratings = df['imdb\_score'].values

SVDpredicted\_ratings = np.array([rec[1] for rec in recommendations])

rmse\_svd = np.sqrt(mean\_squared\_error(actual\_ratings, SVDpredicted\_ratings))

print(f'Root Mean Squared Error (RMSE) of SVD model: {rmse\_svd}')



# RMSE of 6.8053 is considered reasonably not good. This is a relatively large error, indicating that the model's predictions may not be very accurate.

# **V. CONCLUSION**

In conclusion, this project aimed to create a recommendation system for Hutu TV using a dataset containing information about TV shows. I use advanced machine learning techniques to build a system that suggests personalized content to users based on their interests. I also explored two types of recommendation systems: content-based, which looks at show descriptions, and collaborative filtering using Singular Value Decomposition (SVD) to understand user preferences.

Through analysis, I discover interesting insights about the dataset, such as trends in release years and viewer ratings. I also use techniques like TF-IDF and cosine similarity to build the content-based recommendation system. Additionally, I implemente SVD for collaborative filtering, even without explicit user data. In evaluating models, I use metrics like Root Mean Squared Error (RMSE) to assess their performance. The RMSE find that the recommendations are not very good due to data quality.

# **References**

Victor Soeiro (2022). Hulu TV Shows and Movies. Kaggle. https://www.kaggle.com/datasets/victorsoeiro/hulu-tv-shows-and-movies