DSBDAL Mini Project

Guide: Prof. S. Shintre

Name: Prasad Dhondge (31120)

Viraag Borundiya (31112)

Title: Movie Recommendation System

Problem Statement:

Develop a movie recommendation model using the scikit-learn library in python.

# Learning Objectives:

1. Understand the principles of content-based filtering in recommendation systems.
2. Learn how to preprocess and analyze movie data using Python libraries like pandas and scikit-learn.
3. Gain hands-on experience in building a movie recommendation system using scikit-learn.
4. Evaluate the performance of the recommendation system using relevant metrics.

# Learning Outcomes:

By the end of this project, participants will:

1. Understand the theory behind content-based filtering and cosine similarity.
2. Implement a movie recommendation system using Python and scikit-learn.
3. Evaluate the effectiveness of the recommendation system using test cases and metrics.
4. Analyze the results and draw conclusions about the performance of the system.

# Theory:

1. Collaborative Filtering:

* Collaborative filtering is a popular technique used in recommendation systems to provide personalized recommendations based on the preferences of similar users.
* It assumes that users who have agreed in the past tend to agree again in the future.
* Collaborative filtering methods can be categorized into two types:

1. User-based collaborative filtering: Recommends items by finding users who are similar to the target user and recommending items that they have liked.
2. Item-based collaborative filtering: Recommends items by finding items that are similar to the items liked by the target user.
3. Content-Based Filtering:

* Content-based filtering recommends items based on the features of the items themselves and the user's past interactions with similar items.
* Content-based filtering recommends items based on the features of the items themselves and the user's past interactions with similar items.

1. Cosine Similarity:

* Cosine similarity is a measure of similarity between two vectors in an n-dimensional space.
* In the context of recommendation systems, cosine similarity is used to measure the similarity between items or users based on their feature vectors.
* It calculates the cosine of the angle between two vectors, where a value of 1 indicates perfect similarity and 0 indicates no similarity.

1. CountVectorizer:

* CountVectorizer is a text preprocessing technique used to convert text data into numerical feature vectors.
* It tokenizes the text into words and counts the frequency of each word in the document.
* The result is a sparse matrix where each row represents a document and each column represents a unique word in the corpus.

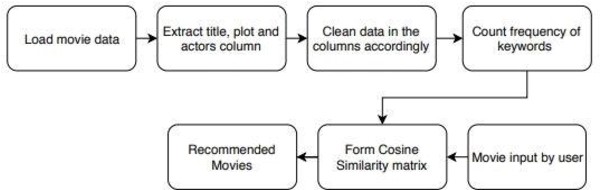
1. Data Preprocessing:

* Data preprocessing involves cleaning and transforming raw data into a format suitable for analysis.
* In the case of movie recommendation, preprocessing may involve handling missing values, combining relevant features, and encoding categorical variables.

1. Building the Recommendation Model:

* To build the recommendation model, we first combine relevant features such as keywords, cast, genres, and director into a single feature vector for each movie.
* We then use CountVectorizer to convert the text data into a matrix of token counts.
* Cosine similarity is computed between the count matrix to measure the similarity between pairs of movies based on their combined features.
* Finally, for a given input movie liked by the user, similar movies are identified based on their cosine similarity scores, and the top similar movies are recommended.

# System Architecture:



The system architecture for the movie recommendation model using the scikit-learn library involves several components working together to provide personalized movie recommendations. Here's an overview of the system architecture:

1. Data Ingestion:

* Data ingestion is the process of collecting and importing data from various sources into the system.
* In this case, the movie dataset is typically stored in a CSV file or a database and is ingested into the system for processing.

1. Data Preprocessing:

* Data preprocessing involves cleaning, transforming, and preparing the raw data for analysis.
* In the movie recommendation model, data preprocessing may involve handling missing values, combining relevant features, and encoding categorical variables.

1. Feature Extraction:

* Feature extraction is the process of extracting relevant features from the raw data to represent each movie.
* In this model, features such as keywords, cast, genres, and director are extracted from the movie dataset and combined into a single feature vector for each movie.

1. Model Training:

* Model training involves building a recommendation model using machine learning algorithms.
* In this case, the recommendation model is trained using the features extracted from the movie dataset and cosine similarity as the similarity metric.

1. Recommendation Engine:

* The recommendation engine is responsible for generating personalized movie recommendations for users based on their preferences.
* It takes as input the movie liked by the user and computes the similarity scores between the input movie and other movies in the dataset.
* Based on the similarity scores, the recommendation engine selects the top similar movies and presents them as recommendations to the user.

# Dataset Description:

The dataset used for the movie recommendation system contains information about various movies, including their titles, genres, keywords, cast members, directors, and other relevant attributes. Here's a brief description of the dataset columns:

1. title: The title of the movie.
2. genres: The genre(s) of the movie.
3. keywords: Keywords or tags associated with the movie.
4. cast: The cast members starring in the movie.
5. director: The director(s) of the movie.
6. Other attributes such as release year, budget, revenue, etc., may also be present depending on the dataset's specific details.

# Methodology/Algorithm Details:

1. Data Preprocessing:

* Missing values handling: Fill missing values in relevant columns (e.g., keywords, cast, genres) with empty strings to avoid errors during feature extraction.
* Feature engineering: Combine relevant features (e.g., keywords, cast, genres, director) into a single feature vector for each movie to represent its content comprehensively.

1. Feature Extraction:

* Combine features: Use a function to combine relevant features into a single feature vector for each movie. This combined feature vector will be used to compute similarity between movies.
* CountVectorizer: Convert the combined feature vector into a matrix of token counts using CountVectorizer from scikit-learn. This step converts text data into numerical feature vectors suitable for analysis.

1. Cosine Similarity:

* Cosine similarity is a popular metric used to measure the similarity between two vectors.
* In this context, cosine similarity is calculated between the feature vectors of pairs of movies to determine their similarity based on content.
* Cosine similarity ranges from -1 to 1, where 1 indicates perfect similarity, 0 indicates no similarity, and -1 indicates dissimilarity.

1. Recommendation Generation:

* Given a movie liked by the user, the recommendation system identifies similar movies based on their cosine similarity scores.
* It sorts the similar movies in descending order of similarity scores and presents the top-n similar movies as recommendations to the user.
* The number of recommendations (n) can be adjusted based on the desired output.

# Results:

A screenshot of a computer

Description automatically generated

# Analysis Conclusion:

Recommendation systems have become an important part of everyone’s lives. With the enormous number of movies releasing worldwide every year, people often miss out on some amazing work of arts due to the lack of correct suggestion. Putting machine learning based Recommendation systems into work is thus very important to get the right recommendations. We saw content-based recommendation systems that although may not seem very effective on its own, but when combined with collaborative techniques can solve the cold start problems that collaborative filtering methods face when run independently.