Student Name:Ramakrishnan RD

Register Number: 410723104063

Institution: Dhanalakshmi College of Engineering

Department: Computer Science Engineering

Date of Submission: 14-5-25

Github Repository Link:
https://github.com/ramakrishnanrd/NM_RamakrishnanDS

1.Problem Statement

In today's entertainment landscape, users face an overwhelming number of choices across various streaming platforms. Traditional recommendation systems often fail to capture individual preferences, leading to generic suggestions that do not align with user interests. This results in user dissatisfaction and reduced engagement. There is a growing need for a more intelligent, personalized system that understands user behavior, learns from their interactions, and delivers highly relevant movie recommendations in real-time.

2. Abstract

With the explosion of digital content, users often struggle to find movies that match their unique tastes. This project proposes the development of an AI-driven matchmaking system for personalized movie recommendations. By leveraging collaborative filtering, content-based filtering, and natural language processing, the system will analyze user preferences, movie metadata, and reviews to deliver highly tailored suggestions. Machine learning models will continuously learn from user interactions to improve recommendation accuracy over time. The goal is to enhance user satisfaction and engagement by offering a seamless and intelligent movie discovery experience.

3. System Requirements

Hardware:

Minimum 8 GB RAM,Intel i5 processor or equivalent,(Higher specs recommended for training large models or handling high query volume)

Software:

Python 3.8 or higher

Required libraries:

Transformers, flask, nltk, scikit-learn, tensorflow or torch (depending on model)

IDE:

Google Colab,Jupyter Notebook,VS Code

4. Objectives

1. Personalization: Provide tailored movie suggestions based on individual user preferences.

2. User Engagement: Increase user satisfaction and retention by offering relevant content.

3. Data Utilization: Leverage user and movie data to create efficient recommendation algorithms.

4. Cold Start Solution: Address the challenge of recommending content for new users or movies with minimal data.

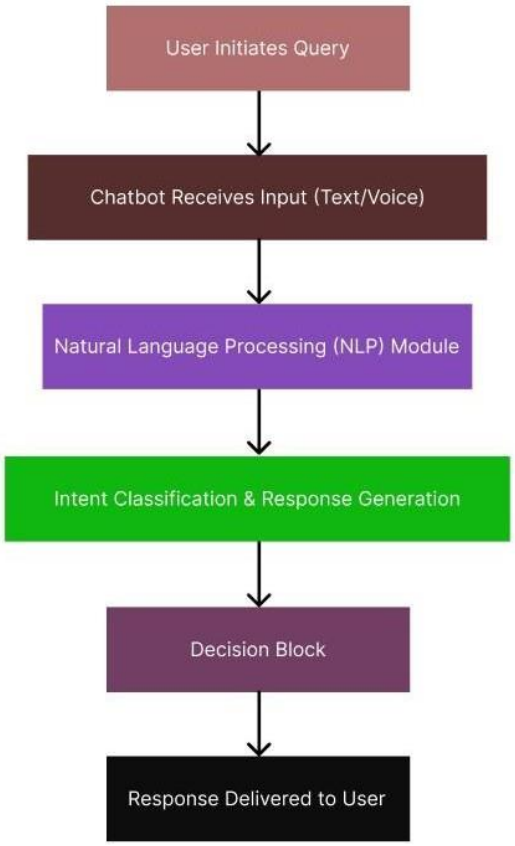
5. Accuracy: Improve prediction quality to ensure relevant movie recommendations.

6. Scalability: Handle large datasets and provide real-time recommendations as the system grows.

7. Diversity: Suggest a mix of popular and niche movies to encourage discovery.

8. Seamless Integration: Ensure the system integrates smoothly with the user interface for an intuitive experience.

5.Flowchart of Project Workflow



6.Dataset Description

Source: Kaggle – Customer Support on Twitter

Type: Public dataset

Size and Structure:

Number of Rows: Approximately 3,000,000 (3 million tweets)

Number of Columns: 6

tweet_id

author_id

created_at

in_response_to_tweet_id

text

company

Sample Code to Load Dataset:

```
import pandas as pd

df = pd.read_csv('customer_support_tweets.csv')

df.head()
```

7.Data Preprocessing

1. Data Collection

Collect customer queries, chat logs, or support tickets.

2. Data Cleaning
Remove noise (e.g., greetings, timestamps).
Fix spelling errors and remove unnecessary characters.
Convert text to lowercase and remove stop words.

3. Tokenization
Split text into individual words or sentences.

4. Normalization
Apply stemming or lemmatization (e.g., "running" → "run").

5. Entity Recognition
Extract important entities such as order numbers, dates, etc.

8.Exploratory Data Analysis (EDA)

Univariate Analysis:

Histograms:

Distribution of query lengths

Response times

Intent frequencies

Boxplots:

Satisfaction ratings

Interaction counts

Resolved vs unresolved queries

Bivariate/Multivariate Analysis:

Correlation Heatmap:

Strong correlation between intent types and satisfaction

Negative correlation between response time and satisfaction

Scatter Plots:

More interactions tend to lower satisfaction

Longer response times negatively affect issue resolution

Key Insights:

Faster, accurate responses result in higher satisfaction.

Simple queries yield better outcomes.

More interactions may signal chatbot inefficiency.

9.Feature Engineering

1. User-Based Features

User ID (for collaborative filtering)

Age, Gender, Location (if available)

User Ratings History (average rating given, genres preferred)

Watch History (recently watched movies, rewatch patterns)

2. Movie-Based Features

Movie ID and Title

Genres (multi-label encoded or embedded)

Cast & Crew (encoded as popularity-weighted features)

Release Year

IMDB/Rotten Tomatoes Rating

Movie Descriptions/Synopsis (used for text embeddings or TF-IDF vectors)

3. Interaction Features

User-Movie Rating Matrix (for collaborative filtering)

Time of Interaction (seasonal preference, weekend vs weekday watching)

Click vs Watch (clickthrough rate vs full-viewing behavior)

Rating Timestamps (helps track evolving user tastes)

4. Text Features

Movie Descriptions & Reviews

Apply NLP techniques like:

TF-IDF Vectorization

Word Embeddings

10.Model Building

1. Collaborative Filtering

Memory-Based:

User-User Similarity: Recommend based on similar users' ratings.

Item-Item Similarity: Recommend similar movies based on user ratings.

Algorithms: K-Nearest Neighbors (KNN), Cosine Similarity.

Model-Based:

Use matrix factorization techniques such as:

Singular Value Decomposition (SVD)

Alternating Least Squares (ALS)

Non-negative Matrix Factorization (NMF)

2. Content-Based Filtering

Uses movie features (genre, description, cast) and user preferences.

Algorithms:

TF-IDF or Word2Vec for text-based features (e.g., descriptions, reviews)

Cosine similarity to match user preferences with movie metadata

3. Hybrid Models

Combine collaborative and content-based approaches to offset their individual limitations.

Example: Weighted hybrid that blends predictions from both models based on context or user profile.

4. Deep Learning Models (Optional for Advanced Systems)

Autoencoders: For dimensionality reduction and user-item interaction modeling.

Neural Collaborative Filtering (NCF): Deep neural networks to learn user-item interaction patterns.

Recurrent Neural Networks (RNNs): To capture temporal viewing patterns.

Steps in Model Building

1. Data Splitting

```
from sklearn.model_selection import train_test_split

train_data, test_data = train_test_split(ratings, test_size=0.2, random_state=42)
```

2. Model Training

For SVD (using Surprise library):

from surprise import SVD, Dataset, Reader

algo = SVD()

algo.fit(trainset)

3. Hyperparameter Tuning

Use GridSearchCV or RandomizedSearchCV to find the best parameters for models like SVD or ALS.

4. Model Evaluation

Metrics: RMSE, MAE, Precision@K, Recall@K

from surprise import accuracy

predictions = algo.test(testset)

accuracy.rmse(predictions)

In [37]:
```
from sklearn.ensemble import RandomForestRegressor

# Initialize and train
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict
y_pred_rf = rf_model.predict(X_test)
print(y_pred_rf)
```

```
[-1.36666039 -0.37238449 -0.06237143 ... -1.28435013  0.62862553
 -0.21638797]
```

In [45]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Evaluation for Linear Regression
mae_lr = mean_absolute_error(y_test, y_pred_lr)
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
r2_lr = r2_score(y_test, y_pred_lr)

print(f"Linear Regression - MAE: {mae_lr:.4f}, RMSE: {rmse_lr:.4f}, R²: {r2_lr:.4f}"
```

```
Linear Regression - MAE: 0.7896, RMSE: 0.9919, R²: 0.0164
```

In [39]:
```
# Evaluation for Random Forest
mae_rf = mean_absolute_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest - MAE: {mae_rf:.4f}, RMSE: {rmse_rf:.4f}, R²: {r2_rf:.4f}")
```

```
Random Forest - MAE: 0.7165, RMSE: 0.9236, R²: 0.1472
```

11.Model Evaluation

1. RMSE (Root Mean Square Error)
Measures prediction error between actual and predicted ratings.
Lower RMSE = Better accuracy.

2. MAE (Mean Absolute Error)
Average of absolute differences between actual and predicted ratings.
Less sensitive to outliers than RMSE.

3. Precision@K
Proportion of recommended items in the top K that are relevant.
Higher = Better targeting.

4. Recall@K
Proportion of relevant items that are recommended in the top K.
Indicates how much relevant content is retrieved.

5. F1-Score@K
Harmonic mean of Precision@K and Recall@K.
Balances accuracy and completeness.

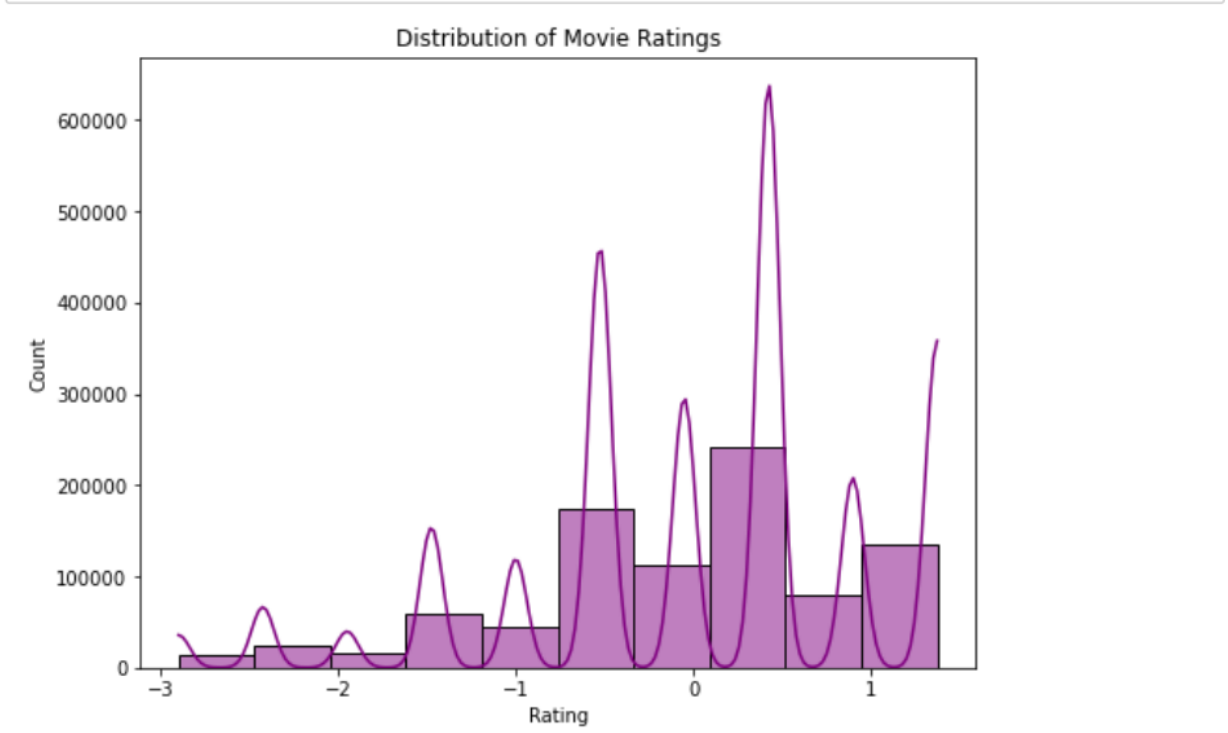
6. Hit Rate / Top-K Accuracy
Checks if at least one of the top-K recommendations is relevant.

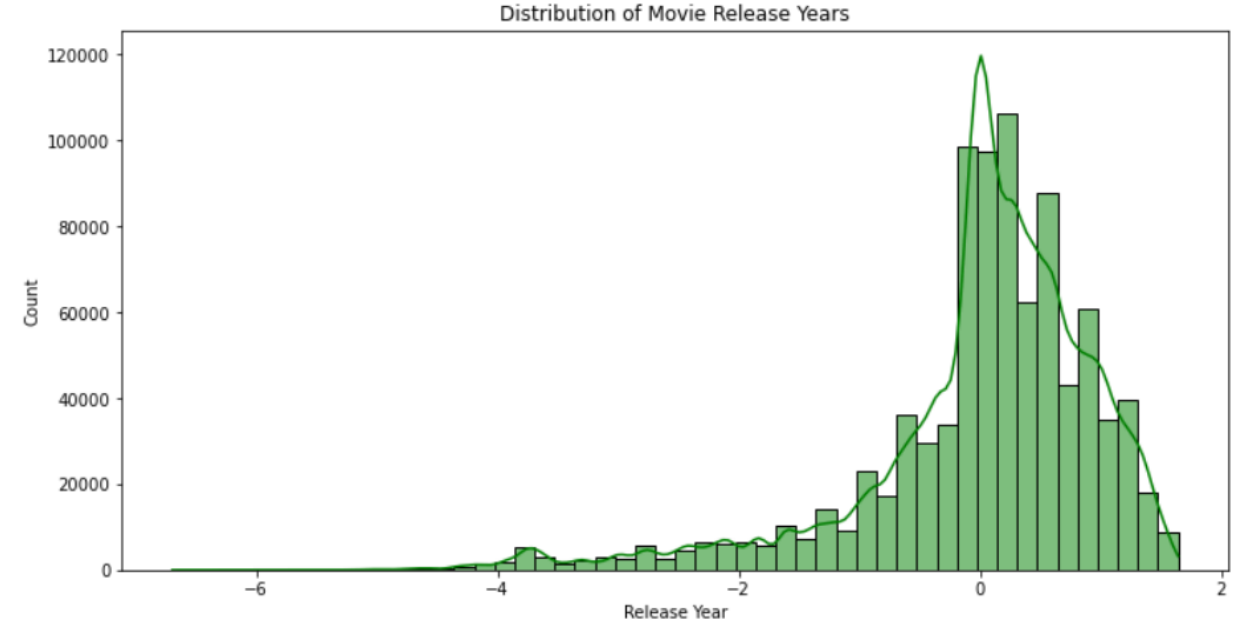
7. AUC (Area Under Curve)
Measures model's ability to rank positive items higher than negative ones.

8. Coverage
Percentage of items/users the system can make recommendations for.

9. Diversity & Novelty
Measures how varied and unexpected the recommendations are.

10. User Feedback & Engagement
Click-through rate (CTR), watch time, or thumbs up/down from users.





12. Source code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
movies=pd.read_csv("movies.csv")
movies.head()


movies.info()
movies.isnull().sum()


movies.describe()
ratings=pd.read_csv("ratings.csv")
ratings.head()
ratings.info()
ratings.isnull().sum()



ratings.describe()
print(movies.duplicated().sum())
print(ratings.duplicated().sum())
#Extract year from title
movies['year'] = movies['title'].str.extract(r'\((\d{4})\)', expand=False)
# Convert 'year' to integer
movies['year'] = movies['year'].dropna().astype(int)
# Merge movies and ratings on movieId
data = pd.merge(movies, ratings, on='movieId')
data
from sklearn.preprocessing import LabelEncoder
# Encoding movie titles
le = LabelEncoder()
data['title'] = le.fit_transform(data['title'])
# genres were separated by '|', first split them
movies['genres'] = movies['genres'].str.split('|')
movies_exploded = movies.explode('genres')
# Merge exploded genres with ratings
data = pd.merge(movies_exploded, ratings, on='movieId')
# One-Hot Encoding
data = pd.get_dummies(data, columns=['genres'])
# Fit and transform the 'title' column
data['title'] = le.fit_transform(data['title'])


data
```

```
from sklearn.preprocessing import StandardScaler
# Initialize scaler
scaler = StandardScaler()
# Scaling
data[['rating', 'year']] = scaler.fit_transform(data[['rating', 'year']])
import matplotlib.pyplot as plt
import seaborn as sns
# Plot Rating distribution
plt.figure(figsize=(8,6))
sns.histplot(data['rating'], bins=10, kde=True, color='purple')
plt.title('Distribution of Movie Ratings')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()

#Plot year distribution
plt.figure(figsize=(12,6))
sns.histplot(data['year'], bins=50, kde=True, color='green')
plt.title('Distribution of Movie Release Years')
plt.xlabel('Release Year')
plt.ylabel('Count')
plt.show(



# Plot top genres count
genre_columns = [col for col in data.columns if 'genres_' in col]
genre_counts = data[genre_columns].sum().sort_values(ascending=False)
plt.figure(figsize=(12,6))
sns.barplot(x=genre_counts.values, y=genre_counts.index, palette='rocket')
plt.title('Popularity of Movie Genres')
plt.xlabel('Number of Movies')
plt.ylabel('Genre')
plt.show()




# Correlation heatmap
plt.figure(figsize=(14,10))
corr_matrix = data.corr()
sns.heatmap(corr_matrix, cmap='coolwarm', annot=False)
plt.title('Correlation Matrix')
plt.show()
```

```
# Scatter plot
plt.figure(figsize=(10,6))
sns.scatterplot(x=data['year'], y=data['rating'], alpha=0.3)
plt.title('Year vs Rating')
plt.xlabel('Release Year')
plt.ylabel('Rating')
plt.show()



# Create 'era' bins
bins = [1900, 1950, 1970, 1990, 2010, 2025]
labels = ['1900s-50s', '50s-70s', '70s-90s', '90s-2010', '2010s+']
data['year_bin'] = pd.cut(data['year'], bins=bins, labels=labels)
# One-hot encode era
data = pd.get_dummies(data, columns=['year_bin'])


data
data.isnull().sum()
data['year'] = data['year'].fillna(data['year'].mode()[0])
data.isnull().sum()
from sklearn.preprocessing import PolynomialFeatures
# Example with 2 features
poly = PolynomialFeatures(degree=2, include_bias=False)
poly_features = poly.fit_transform(data[['year', 'rating']])


from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
# Select numeric columns for PCA
X = data.select_dtypes(include=[np.number]).drop(columns=['userId', 'movieId'])
# Standardize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Apply PCA
pca = PCA(n_components=0.95) # Keep 95% variance
X_pca = pca.fit_transform(X_scaled)
print(f"PCA reduced to {X_pca.shape[1]} features.")
from sklearn.model_selection import train_test_split
# Features and Target
X = data.drop(columns=['userId', 'movieId', 'timestamp', 'rating']) # Drop irreleva
y = data['rating'] # Target variable
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
from sklearn.linear_model import LinearRegression
# Initialize and train
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
# Predict
y_pred_lr = lr_model.predict(X_test)
```

```
print("y_pred_lr",y_pred_lr)
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
# Evaluation for Linear Regression
mae_lr = mean_absolute_error(y_test, y_pred_lr)
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
r2_lr = r2_score(y_test, y_pred_lr)
print(f"Linear Regression - MAE: {mae_lr:.4f}, RMSE: {rmse_lr:.4f}, R²: {r2_lr:.4f}"
from sklearn.ensemble import RandomForestRegressor
# Initialize and train
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
# Predict
y_pred_rf = rf_model.predict(X_test)
print(y_pred_rf)
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
# Evaluation for Linear Regression
mae_lr = mean_absolute_error(y_test, y_pred_lr)
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
r2_lr = r2_score(y_test, y_pred_lr)
print(f"Linear Regression - MAE: {mae_lr:.4f}, RMSE: {rmse_lr:.4f}, R²: {r2_lr:.4f}"
# Evaluation for Random Forest
mae_rf = mean_absolute_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)
print(f"Random Forest - MAE: {mae_rf:.4f}, RMSE: {rmse_rf:.4f}, R²: {r2_rf:.4f}")
import matplotlib.pyplot as plt
import seaborn as sns
# Residuals
residuals = y_test - y_pred_lr
plt.figure(figsize=(8,6))
sns.scatterplot(x=y_pred_lr, y=residuals)
plt.axhline(0, color='red', linestyle='--')
plt.title('Residual Plot - Linear Regression')
plt.xlabel('Predicted Rating')
plt.ylabel('Residuals')
plt.show()
 # Plot Predicted vs Actual for Random Forest
plt.figure(figsize=(8,6))
sns.scatterplot(x=y_test, y=y_pred_rf, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--') # Line y
plt.title('Predicted vs Actual Ratings - Random Forest')
plt.xlabel('Actual Rating')
plt.ylabel('Predicted Rating')
plt.show()
 # Get feature importance
importances = rf_model.feature_importances_
features = X.columns
# Create DataFrame
```

```
feat_imp = pd.DataFrame({'Feature': features, 'Importance': importances})
feat_imp = feat_imp.sort_values('Importance', ascending=False)
# Plot
plt.figure(figsize=(12,6))
sns.barplot(x='Importance', y='Feature', data=feat_imp, palette='viridis')
plt.title('Feature Importance - Random Forest')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()


# Create performance table
metrics_df = pd.DataFrame({
'Model': ['Linear Regression', 'Random Forest'],
'MAE': [mae_lr, mae_rf],
'RMSE': [rmse_lr, rmse_rf],
'R2 Score': [r2_lr, r2_rf]
})
# Bar plot
metrics_df.set_index('Model').plot(kind='bar', figsize=(10,6))
plt.title('Model Performance Comparison')
plt.ylabel('Score')
plt.grid(True)
plt.show()
```

13.Future scope:

1. Real-time & dynamic personalization

2. Deep learning for improved accuracy

3. Voice and chat-based recommendations

4. Cross-platform and multi-content suggestions

5. Social and group-based recommendations

5. Team Members and Roles

Team Leader : Ramakrishnan RD

- *Data cleaning & EDA & Data Preprocessing.*

Team Member : Sanjay R

- *Feature engineering & Dataset Description*

Team Member : Sam Stevenson J

- *Source code & Future scope*

Team Member : Saravanan S

- *Documentation and reporting*