Movie Recommendation System

Abstract

GitHub Repository: https://github.com/Levi477/Recommendation-Engine

Website Link: https://recommendation-engine-eight.vercel.app

Demo Video : https://youtu.be/9i2GlXGPWhg

Project Page Link : https://recommendation-engine-eight.vercel.app/project-page

Resources : GeeksforGeeks for algorithms, Wikipedia for algorithms, IMDB for inspiration

Contents

1 Introduction

This project implements a Movie Recommendation System using seven different machine learning techniques and **hosted backend on GCP (Google Cloud Platform) VM**. It explores both content-based and user-based filtering through models like KNN, GMM, Clustering, SVM, and Neural Networks. The goal is to compare the performance and efficacy of these algorithms in recommending movies based on metadata or user interactions.

2 Preprocessing and Feature Engineering

2.1 Data Sources

- TMDB 5000 Movie Dataset (with metadata such as genres, keywords, overview, etc.)
- User interaction data simulated or gathered from user rating/feedback (if applicable)

2.2 Cleaning and Normalization

- Removed duplicates and null entries.
- Converted genres and keywords from stringified JSON to usable lists.
- Normalized text by removing special characters, applying lowercasing, and tokenizing for text-based models.

2.3 Feature Extraction

- **Title, Overview, Genres, Keywords, Cast, Crew**
- Used TF-IDF Vectorization on overview text for content similarity.
- Built a feature vector for each movie combining cast, crew, keywords, and genres.

2.4 User-based Feature Engineering

- Mapping user interactions (likes, watches) into a binary matrix.
- Labeling movie IDs as indices for neural networks.

2.5 Vectorization Techniques

- **TF-IDF**: Used for textual similarity from overviews.
- **CountVectorizer**: Used for bag-of-words models with keywords, cast, and genres.
- **PCA (Optional)**: Dimensionality reduction applied in some cases to reduce noise.

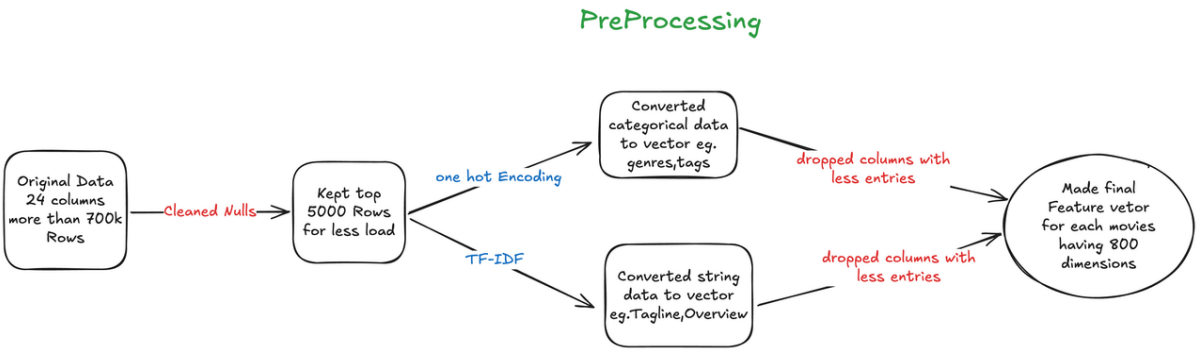

Figure 1: Feature Extraction Pipeline

3 Approaches

3.1 K-Nearest Neighbors (KNN)

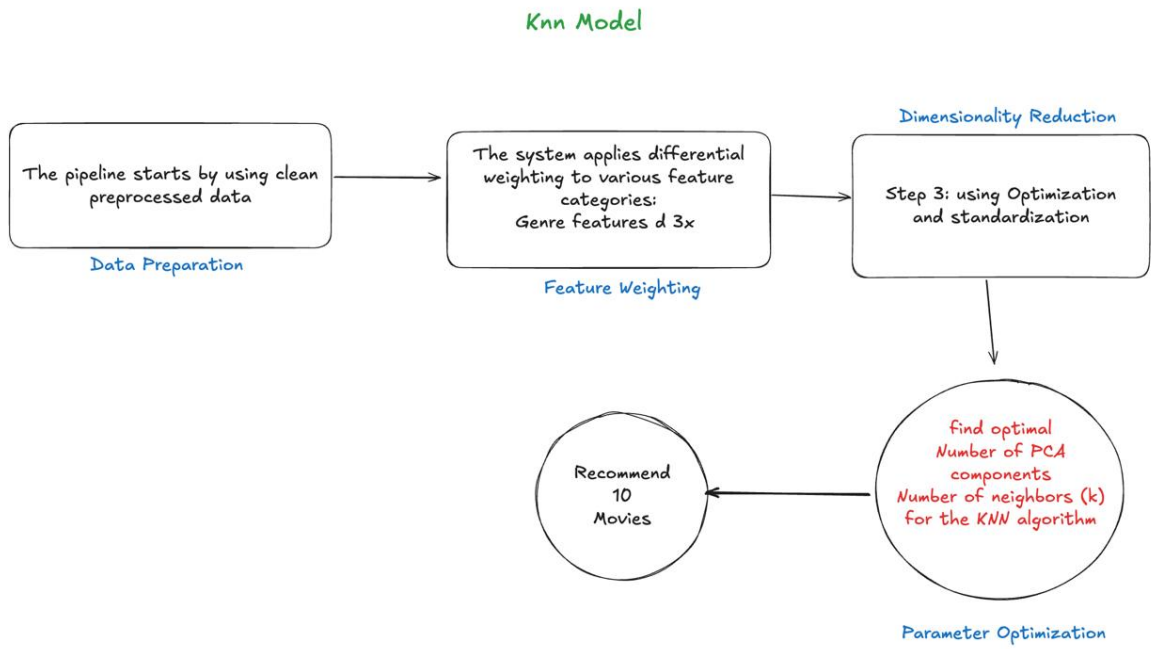

Figure 2: KNN Similarity

1. We use **fuzzy matching** to find the closest matching movie title from the dataset, even if the input has typos or partial names. We select the match with a similarity score above a fixed threshold.
2. Each feature (like genres, keywords) is given a different weight. For example, genres are given 3 times more weight than keywords. This ensures more important features have a stronger influence.

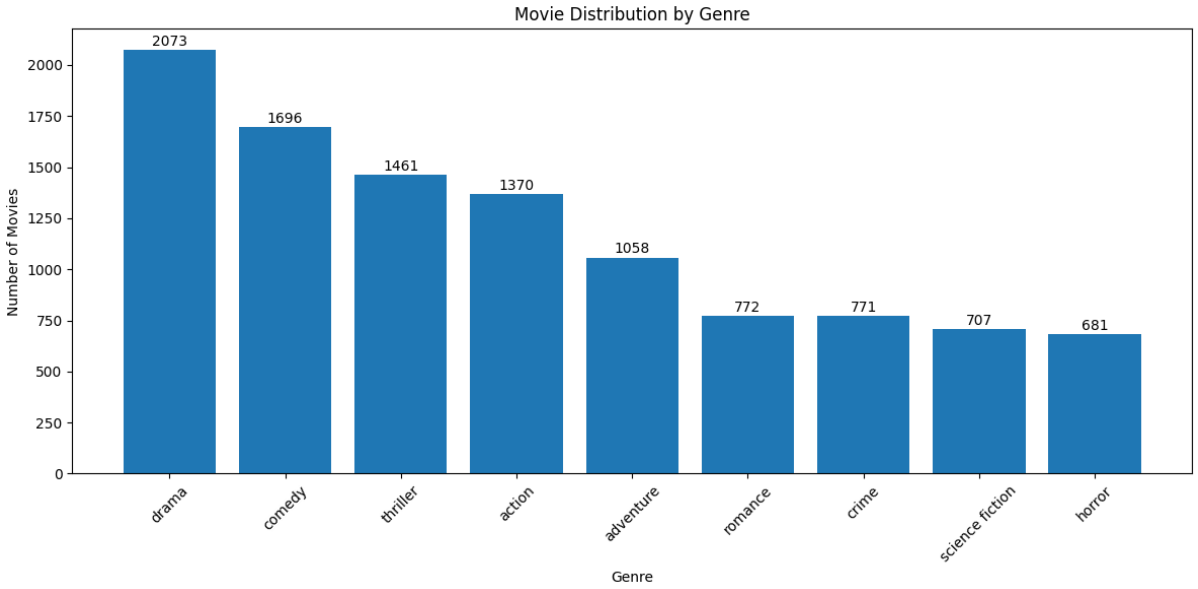

Figure 3: Movie Distribution by Genre

3. PCA reduces the number of features while keeping most information. It helps speed up the system and reduces noise. We use a **cumulative explained variance** plot to decide the number of components needed to retain at least 95% of the variance.

4. KNN is used to find movies similar to the input movie.

   Formula (Euclidean Distance):

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

   Here, p and q are two movies represented as n-dimensional vectors. The smaller the distance $d(p, q)$, the more similar the movies are.

5. To evaluate recommendations, we use the KNN algorithm to find the closest movies to the input movie in the feature space.

3.2 Gaussian Mixture Model (GMM)

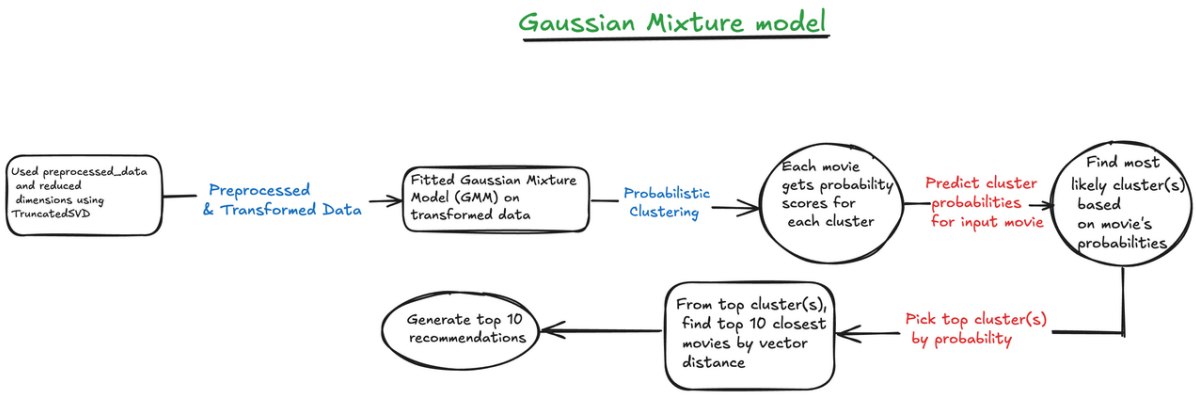

Figure 4: Gaussian Mixture Model Clustering

$$P(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

1. Use EM algorithm to find optimal parameters.
2. Assign users to clusters based on probabilities.
3. Recommend popular movies from that cluster.

3.3 K-Means Clustering

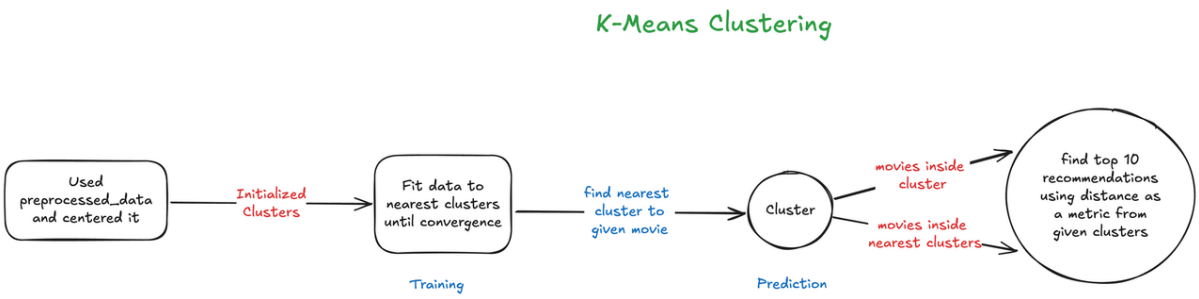

Figure 5: K-Means Clustering

1. We first cleaned the dataset by removing irrelevant columns like `title` and `imdb_id`, then normalized all numerical features using min-max scaling.

2. **K-Means clustering** was applied on the preprocessed feature vectors to group similar movies. We chose $k = 7$ based on experimentation.

3. Initial cluster centers were chosen using a variation of the K-Means++ initialization that considers the squared distance of each point from existing centers.

4. In each iteration (epoch), every movie is assigned to the nearest cluster center. Then, the cluster centers are recalculated as the mean of all points within the cluster.

5. The process repeats for 20 epochs or until convergence (when cluster centers stop changing).

6. Once clustering is complete, recommendations can be made by selecting movies that belong to the same cluster as the input movie. If there are not enough options, the algorithm selects from the nearest cluster center.

7. This method enables content-based recommendations using movie embeddings derived from metadata.

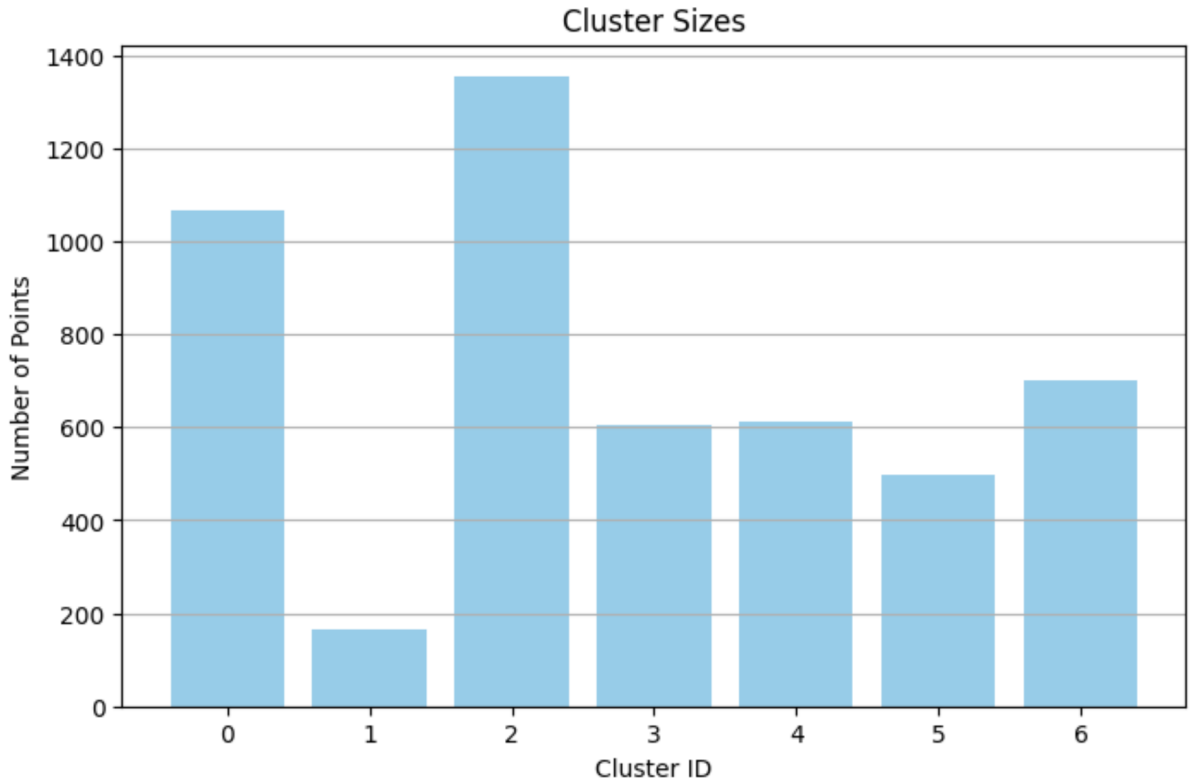

Figure 6: Distribution of Movies Across K-Means Clusters

8. **Euclidean Distance Formula:**

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

where p and q are n-dimensional feature vectors representing two movies.

3.4 Support Vector Machine (SVM)

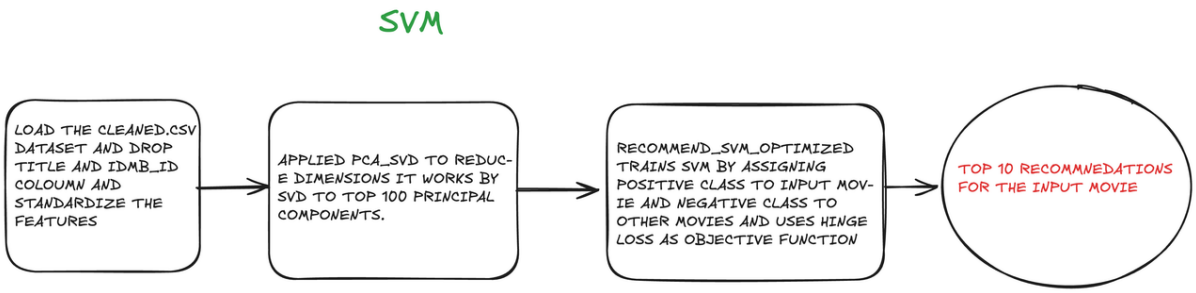

Figure 7: Support Vector Machine

$$f(x) = \mathrm{w}^T x + b$$

1. Load and Prepare Data(standardization).
2. Do Dimensionality reduction using PCA using SVD(top 100 components).
3. Learn a linear classifier that tries to separate the input movie from all others using a hinge-loss-based optimization (inspired by SVM).
4. compute scores based on distance from hyperplane and returned top movies.

3.5 Artificial Neural Network (ANN)

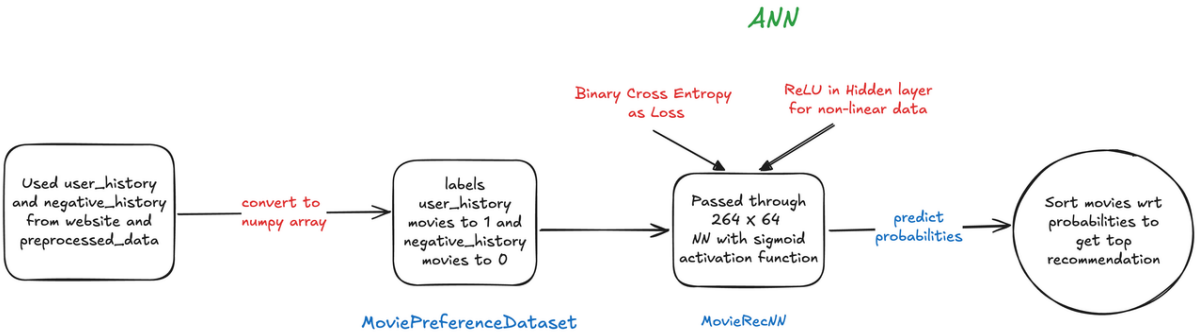

Figure 8: Neural Network Architecture

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

1. I uses User interactions (eg. CLicking the Movie) to filter out movies that user liked and disliked
2. Make an array of positive-movies and negative-movies form above information to assign them label of 1 and 0 respectively
3. Train model on given dataset of positive and negative movies.
4. Use the weights gotten in previous step to predict movies with highest probability.
5. Applied sigmoid activation function to get final probability
6. Recommend movies with highest predicted scores.

3.6 Content-Based Filtering

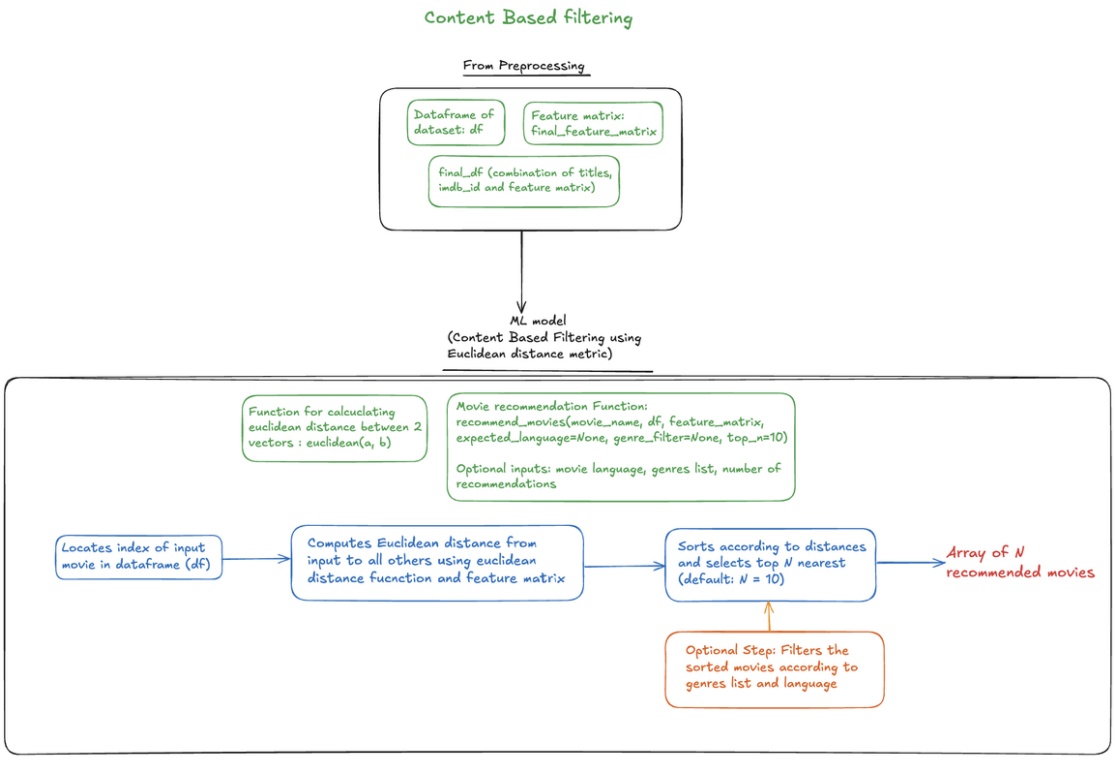

Figure 9: Content-Based Filtering

Euclidean distance:

$$\text{euclidean}(a, b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

1. Take the feature matrix (created using TF-IDF and encoding techniques) and calculate the similarities between selected and other movies.
2. Sort the movies according to similarities (euclidean distances).
3. Recommend the top N similar movies to user's selected movie.

Similarity Distribution

Steps:

1. First obtain the index of selected movie from the data frame.
2. Calculate the Euclidean distances between the vector of selected movie and all movies using feature matrix and index.
3. Sort these distances and plot graph of top 50 distances.
4. Movies with lower distances are recommended.

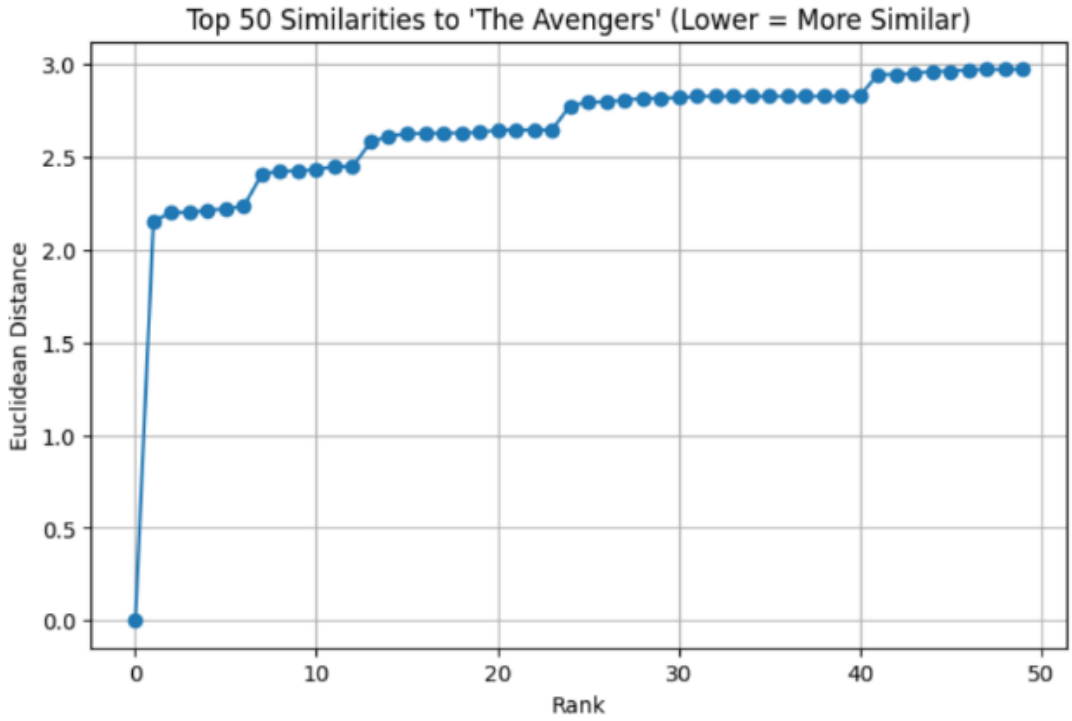

Figure 10: Content-Based Filtering - Similarity Scores

3.7 Bayesian Recommendation

3.7.1 Traditional approach

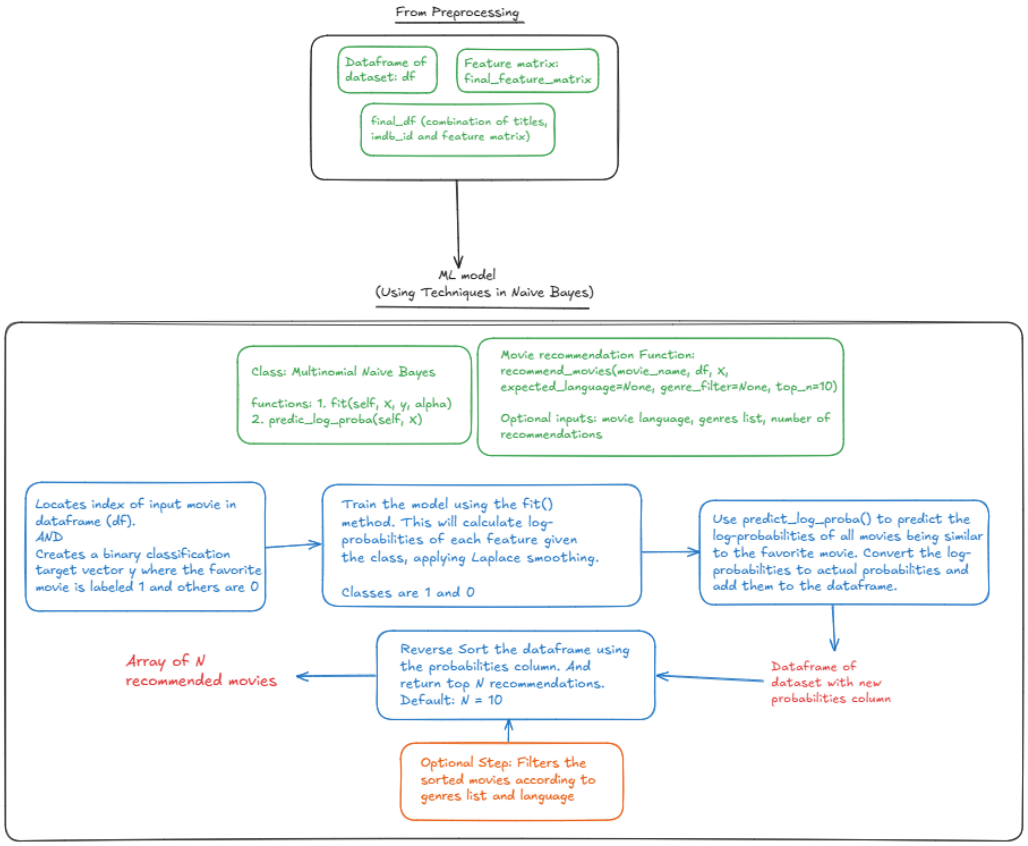

Figure 11: Bayesian Model: Similar to Traditional Method

Formula:

$$logP(c \mid \mathbf{x}) = logP(c) + \sum_{i=1}^{n} x_i \cdot logP(w_i \mid c)$$

1. In this method, we create two classes, 1 for selected movie and 0 for all other remaining movies
2. Then we calculate the prior log probability for each class (log P(c)) and log probabilities of the features/words for that class (log P(wi | c)) along with Laplace smoothing.
3. After this, we iterate through all movies and calculate the log probabilities of these movies for the given classes according to formula.
4. Then top N movies with high probabilities for class 1 are recommended

Top Influential Words

It visualizes the most influential words for a given movie's class.

Steps:

1. Select the index of the selected movie from the dataframe.
2. Fit the Naive Bayes model with the binary target vector indicating that the selected movie is the positive class.
3. Extract the top N influential words based on their log-probabilities from the feature_log_prob attribute of the model.
4. Plot these words along with their log-probabilities in a bar chart.
5. Also, We can manually check if these words are connected to recommended movies

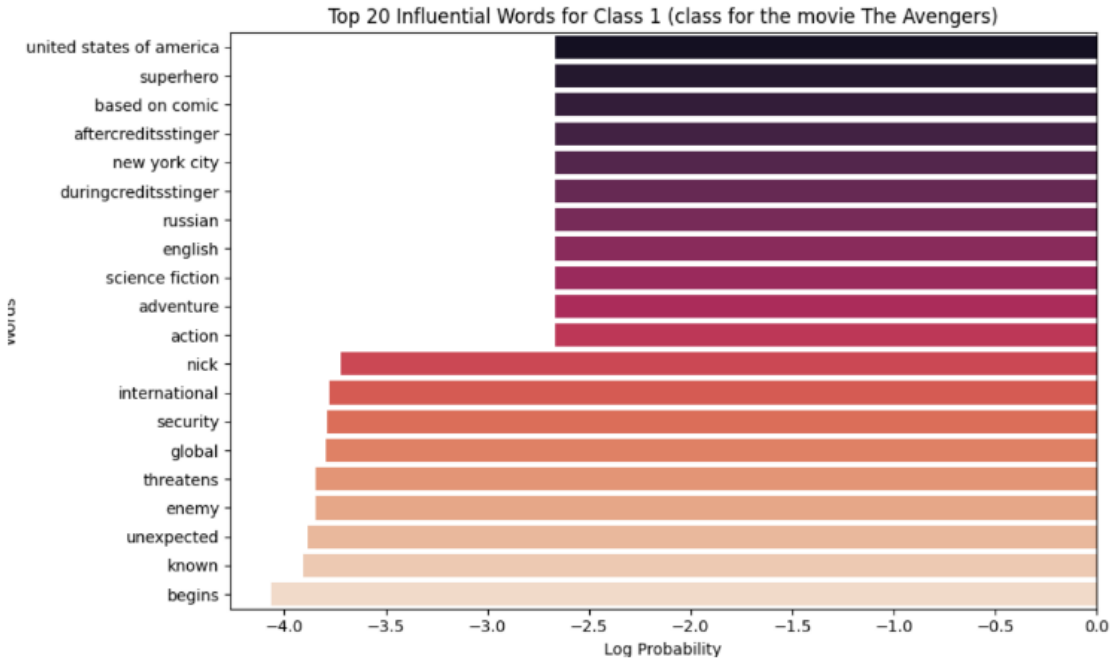

Figure 12: Top Influential words

3.7.2 Modified approach

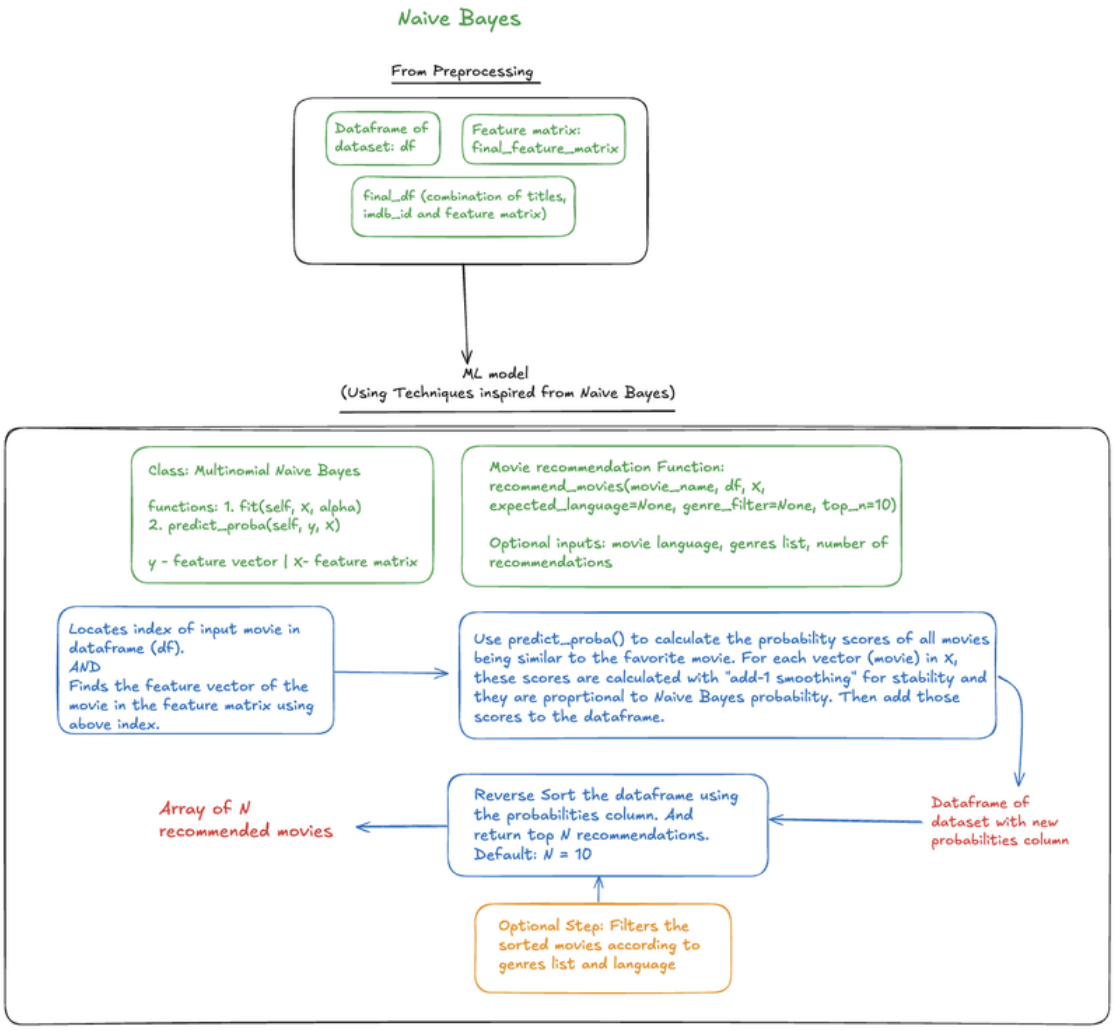

Figure 13: Bayesian Recommendation Model

Formula:

$$Score(d_i, q) \propto \prod_{j=1}^{|V|} (q_j + 1)^{d_{ij}}$$

1. Instead of traditional classification method, we compute the **likelihood of other movies** being generated by the input movie's word distribution.
2. First, we take the selected movie's feature vector. Then we iterate through all of the movies vectors and compute the similarity scores (probability scores) for each movie using the above formula.
3. In the formula, q is selected movie's feature vector and di is any movie's feature vector from the list of all movies.
4. Recommend the top N movies with highest probability scores.

4 Analysis and Results

After implementing and evaluating the seven different recommendation techniques, we compared their output,likliehood or their clusters.

4.1 K-Nearest Neighbors (KNN)

PCA reduces the number of features while keeping most information. It helps speed up the system and reduces noise. We use a **cumulative explained variance** plot to decide the number of components needed to retain at least 95% of the variance. give title for this graph

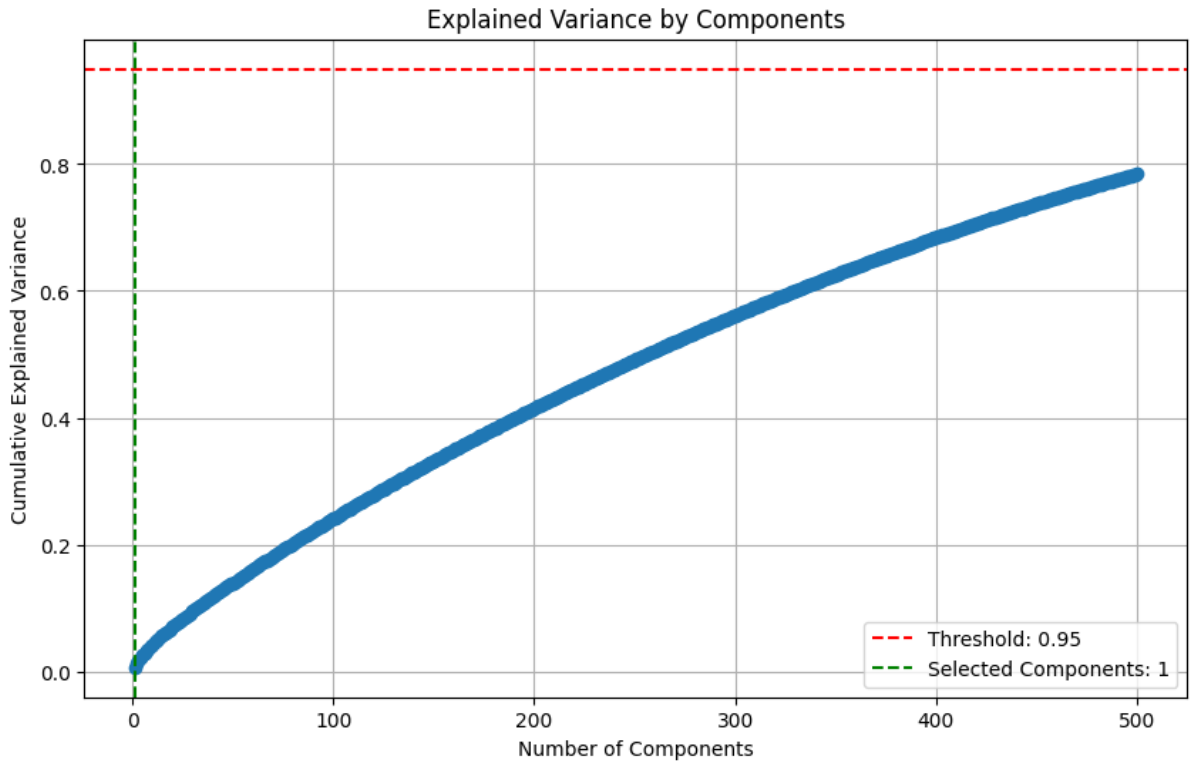

Figure 14: Variance

4.2 K-Means Clustering

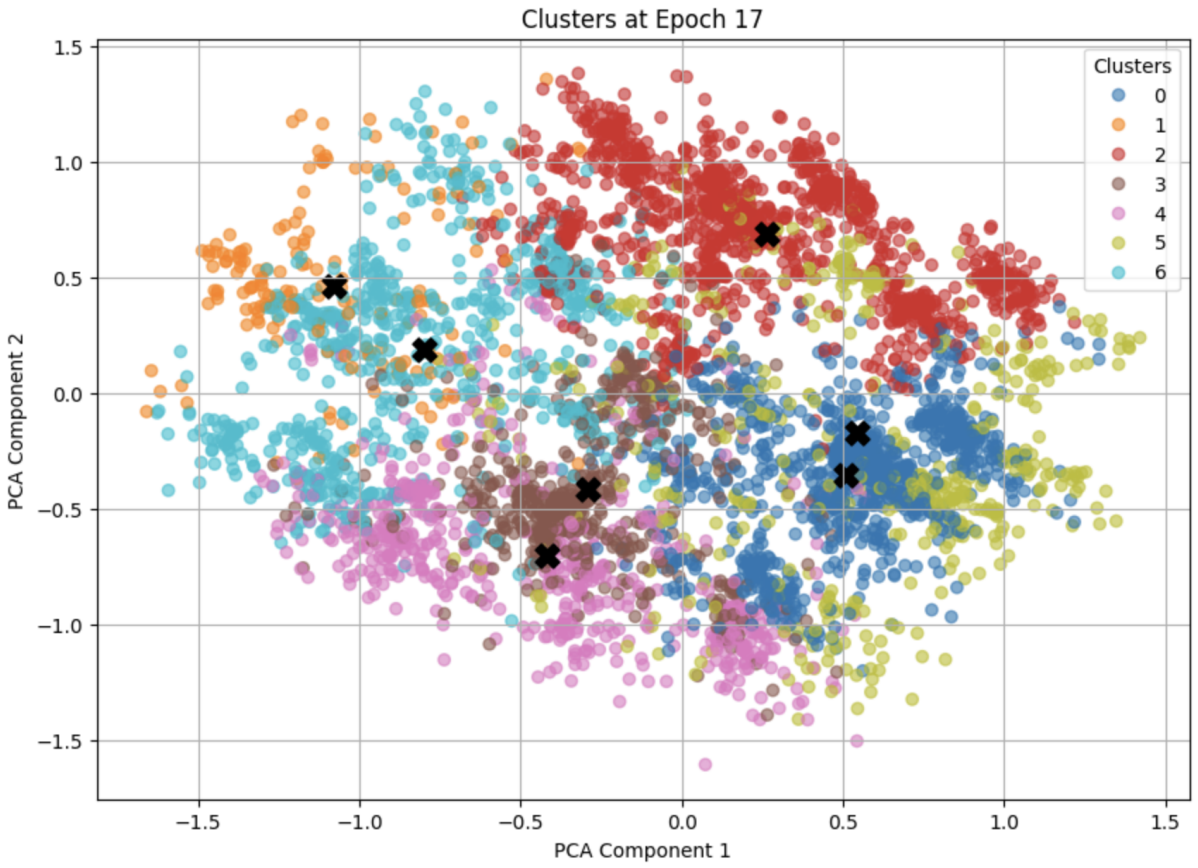

Figure 15: Distribution of Movies Across K-Means Clusters in reduced dimension

4.3 Content-Based Filtering

4.3.1 Genre Importance Matrix

It visualizes the similarity in the genres of selected movie and recommended movies.

Genre Importance Matrix

Steps:

1. Find the genres list of selected movie.
2. Create a loop, which finds the genres list of each recommended movie and creates a matrix of 0s and 1s where 0 indicates specific genre of selected movie is not present in recommended movie and 1 indicates specific genre of selected movie is present in recommended movie.
3. Plot the heat map using that matrix.
4. More number of 1s for recommended shows the similarity between selected movie and recommended movie.

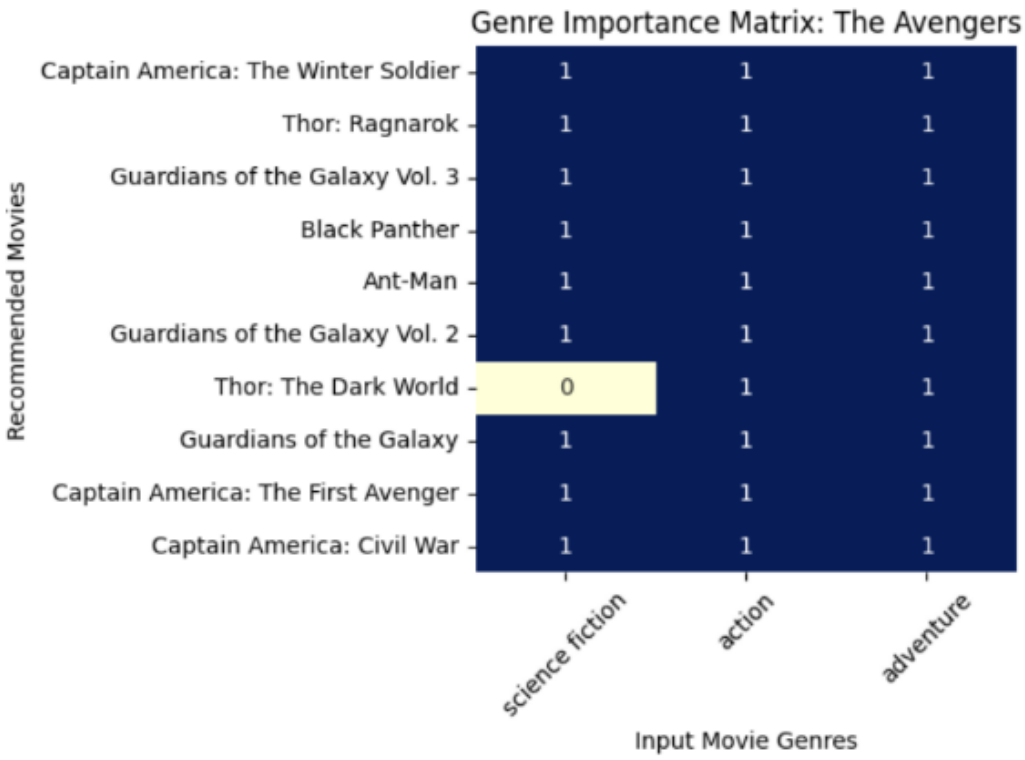

Figure 16: Content-Based Filtering - Genre Importance Matrix

4.4 Bayesian Techniques

4.4.1 Traditional approach:

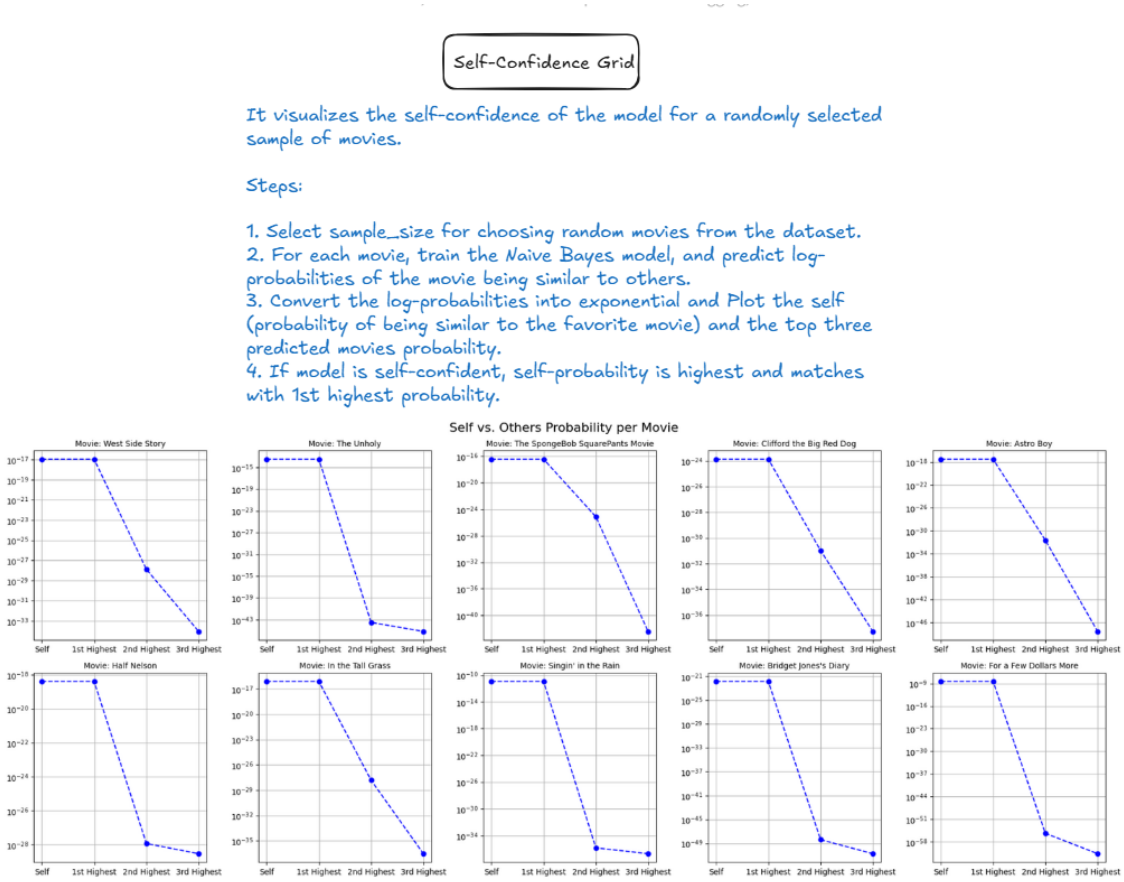

Figure 17: Bayesian Techniques- traditional approach : Self-Confidence Grid

4.4.2 Modified Approach:

It is observed that Self-importance grid is similar to traditional approach but improvement is made in better recommendations.

Genre Importance Matrix

It visualizes the similarity in the genres of selected movie and recommended movies.

Steps:

1. Find the genres list of selected movie.
2. Create a loop, which finds the genres list of each recommended movie and creates a matrix of 0s and 1s where 0 indicates specific genre of selected movie is not present in recommended movie and 1 indicates specific genre of selected movie is present in recommended movie.
3. Plot the heat map using that matrix.
4. More number of 1s for recommended shows the similarity between selected movie and recommended movie.

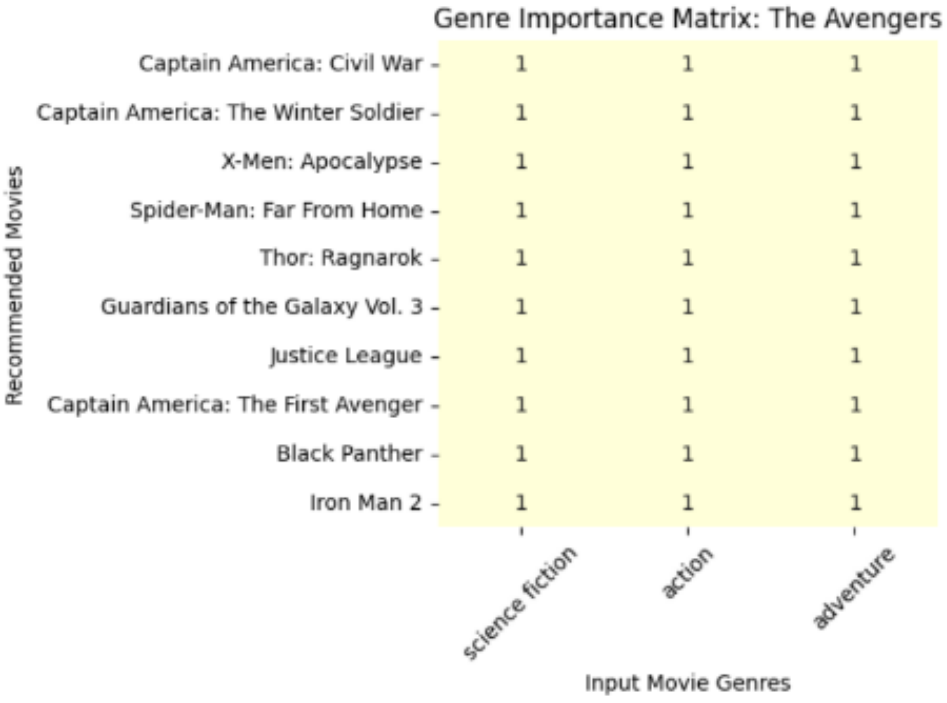

Figure 18: Bayesian techniques - modified approach: Genre Importance Matrix

5 Summary of Model Performance

Model	Accuracy	Personalization	Complexity	Remarks
KNN	High	Medium	Low	Fast, interpretable
GMM	Medium	Low	Medium	Good for group-based suggestions
Content-Based	Medium	High	Medium	Cold-start friendly
K-Means	Medium	Medium	Low	Useful for segmentation
SVM	High	Medium	High	Effective with labeled data
ANN	Very High	Very High	Very High	Excellent accuracy, needs large dataset
Bayesian	Medium	High	Medium	Probabilistic, robust

Table 1: Comparative Analysis of Recommendation Models

5.1 Observations

- **ANN** gave the best overall performance in terms of personalization and accuracy, but requires heavy computational resources.
- **KNN** was fast, interpretable, and reliable for collaborative filtering scenarios.
- **Content-Based Filtering** excelled in personalizing suggestions based on movie features.
- **GMM** is accurate and has great performance.
- **Clustering methods** like GMM and K-Means worked well for grouping similar user preferences but sometimes lacked precision
- **SVM** was effective in classification-based recommendation settings but scaled poorly for large datasets.
- **Bayesian Methods** provided explainable and consistent recommendations.

5.2 Conclusion

There is no universal best algorithm—each has its advantages depending on the specific use-case, data availability, and user behavior. In practice, hybrid models that combine collaborative, content-based, and neural network approaches often yield the most robust performance. There is no universal best algorithm—each has its advantages depending on the specific use-case, data availability, and user behavior. In practice, hybrid models that combine collaborative, content-based, and neural network approaches often yield the most robust performance.

6 Contributions

Deep Gajjar (B23EE1014) : Data Cleaning/Preprocessing and Vectorization, Implemented ANN (Artificial Neural Network) algorithm

Abhyudaya Tiwari (B23CS1085) : Implemented KNN algorithm,Created backend for all the models and Deployed on GCP(Google Cloud Platform)

Neeraj Kumar (B23CS1044) : Implemented K-Means Clustring Algorithm, Created Frontend Website for the project

Pawar Yuvraj Pramod (B23CS1051): Implemented Bayesian (Both Traditional and Modified) and Content Based Filtering models

Bhavya Uchhat (B23CS1074) : Implemented GMM(Gausian Mixtureing Model) algorithm for Reocmmendation

Patil Sanskar (B23CS1050) : Implemented SVM(Scalable Vector Machine) algorithm for Recommendation