6CCS3PRJ Final Year
Web Based Recommender System

Final Project Report

Author: Omar Ahmad

Supervisor: Jeroen Keppens

Student ID: k21052417

Programme of Study: BSc Computer Science

April 17, 2025

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Omar Ahmad

April 17, 2025

Abstract

This project will explore content recommendation for a given user, where the Top N rated movies for a user are returned as recommendations. In particular, it will explore how focusing on the age rating metric can improve overall recommendations. The project will research, design, and implement a system which takes in dataset ratings and performs content-based filtering to return the predicted rating of a movie, and therefore Top N best rated movies.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Jeroen Keppens, who supported me during this project. His guidance and advice aided me in completing this project.

Contents

Chapter 1

Introduction

1.1 Project Objective

With the plethora of movies available to watch on various streaming platforms, film watchers often struggle to find movies that they would enjoy based on their preferences. Particularly when it comes to finding movies which are age rating appropriate to user's preference.

The objective of this project is to design and produce a movie recommender system which focuses on improving suggestions with the **age rating** metric being a key focus.

This recommender system will use a hybrid approach of known methods, with a focus on a content-based approach for a single user, rather than a candidate-based approach which considers multiple users. We will use machine learning techniques, to extract information from a dataset of film ratings, transform, and filter using machine learning techniques, to return the predicted top N results. It will include a back-end API service which will then serve a simple web client as a proof-of-concept for the idea. In turn, suggesting and prompting them to engage with content that they might like.

These recommender systems use information about user preferences and item attributes to suggest items from a dataset that closely match a users interest. Some examples of this include: user's past viewed videos, purchase history, previously completed job applications, user ratings, user category references, etc. One of the techniques used to implement this is machine learning, which uses the dataset to train and test one or more machine learning models, to ultimately output personalised recommended items to users.

As platforms like Netflix continue to become more popular, the demand for more reliable and useful recommender systems increases. Exploring the different metrics is crucial in understanding the effectiveness of the recommender system. They also must be tested thoroughly to ensure the system does not only work in theory, but in real practical scenarios too. Current approaches often rely too heavily on user history which limits the suggestions for new users and for niche subjects. Understanding existing approaches and using a hybrid method will address gaps in the market, allowing for the development of an effective solution.

Content-delivering web applications increasingly rely on recommender systems to personalise the user experience and drive engagement. These systems are fundamental to many digital platforms, from entertainment services like Netflix and YouTube to e-commerce sites such as Amazon and eBay. They are also commonly used inn job-matching platforms like LinkedIn and social media feeds, i.e. Instagram and TikTok. Recommender systems work by analysing use interactions, preferences, and behavioural patterns to suggest content, products, or opportunities tailored to each individual. Their purpose is to enhance user satisfaction and also to increase retention, time spent on the platform, and ultimately commercial revenue.

The commercial impact of recommender systems is significant. For example, YouTube's homepage and "Up Next" suggestions account for the majority of video views on the platform. This highlight just how central these systems are to the business models of content providers. Rather than users actively searching for content, the systems surface options they are most likely to watch or engage with, creating a seamless, personalised experience. As such, recommender systems are no longer just a feature for these platforms, they are a strategic asset as they optimise content delivery and shape user interaction across nearly all digital industries.

This project aims to address key challenges in delivering relevant and responsible movie recommendations through a content-based approach. Many techniques are incorporated in this project and are explain in more detail in Section 2, such as TF-IDF and SVD, which are used to efficiently capture and compress essential movie features. The inclusion of both cosine similarity and KNN provides flexibility in tailoring the recommendation method to different user needs. For the KNN branch, additional preprocessing allows for a more refined understanding of user preferences. Importantly, the introduction of weighted scores especially when emphasising age ratings ensures that recommendations are not only accurate but also socially responsible. This hybrid, modular design reflects an effort to balance between performance, personalisation, and ethical recommendation practices.

Chapter 2

Literature Review

In this literature review, research papers will be studied and the relevant information on how to build a recommender system will be extracted. From this, a comprehensive understanding of recommender systems will be completed. The findings reported in this literature review will then be analysed further once implementing the recommender system for this project, and tests will be conducted to ensure reliability. From this literature review, machine learning approaches will be researched, and either the most optimal for the project or a hybrid solution will be found. When building the recommender system, the 3 stages of candidate generation, candidate ranking and filtering will be explored.

2.1 Content Based Filtering

Content-based filtering is simply using the attributes of the items rather than aggregate information on the particular user of the system. This involves matching attributes of items to user preferences. This is a simple, yet effective approach to use. It works by building a model of user preferences based on previously upvoted items. Similarities between different items are calculated using mathematical functions so that the different items can be compared and filtered beyond a certain threshold [1]. This is useful during the candidate generation phase, where the relevant items are selected and an output list is produced. The top N is chosen after this phase. Disadvantages are due to its simplicity and is limited because only looking at particular attributes may miss the more subtle user patterns.

2.2 Collaborative Filtering

Collaborative filtering recommends items based on user similarity patterns rather than item attributes. This approach dominated the Netflix Prize competition, where the winning team combined collaborative algorithms to significantly improve recommendation accuracy [2]. However, it requires substantial user interaction data and struggles with new users or items. This project focuses on content-based filtering as it better aligns with our goal of providing age-appropriate recommendations based on content attributes and can function effectively with limited user data.

2.3 Machine Learning

The 3 types of machine learning discussed in this report will be supervised learning, unsupervised learning and reinforcement learning. Supervised learning involves training the system with a given labelled dataset, and in return it learns to use new data to produce an accurate output. Unsupervised learning instead involves an unlabelled dataset to be used by the system, and the system independently finds patterns within it. With reinforcement learning, the system learns from its previous attempts, e.g. whether it was successful or not, either a reward or punishment. From what the system has learnt in this type of machine learning, it uses its past actions to select the next action, or explores new solutions. The machine learning used for this project is KNN which is explained in section 2.6.

2.4 TF-IDF Matrix

The Term Frequency Inverse-Document Frequency matrix (TF-IDF matrix) is a key component in content based recommender systems as it is a statistical text preprocessing technique. Using this matrix, the system converts text descriptions into numerical values. The ”Term Frequency” refers to the number of times a word occurs in a title's description. The ”inverse Document Frequency” then allows the system to focus solely on unique words in the title, whilst minimising the weight of common words such as ”the” and ”and” [3]. Based on the information of each item, in this case movies, a content fingerprint. This is made up of the content features, including: the words in the description, names of cast members, directors, and genres/categories.

2.5 Matrix Factorisation Methods

In the TF-IDF matrix, there will be a lot of redundant columns (sparse data) due to text i.e. the word ”the” or ”and” (examples of noisy data that is filtered out). These are not helpful in making better predictions for the user and so a general concept called matrix factorisation is used. Two common solutions are PCA and SVD. [4]

2.5.1 Principal Component Analysis

Principal Component Analysis (PCA) is a technique used for dimensionality reduction. It reduces the complexity of the data via feature extraction. These latent features extracted from the data represent the underlying structures of items, such as common patterns or themes across the movies. For recommender systems, this would mean simplifying the high-dimensional data by removing the dimensions that hold the least amount of principal components. In doing this, the PCA has removed the dimensions that contribute the least to retrieving accurate recommendations, making the system more efficient and easier to interpret. [4]

2.5.2 Singular Value Decomposition

Singular Value Decomposition (SVD) is a matrix factorisation technique known to produce very accurate results, and has been used by large companies such as Netflix. It is a statistical preprocessing data text analysis tool. SVD is a way of computing the values in this equation very efficiently. Similar to PCA, the SVD extracts latent features in the data, but SVD does this

for both user and item data, whereas PCA only did this for one or the other. SVD approximates the user-item interaction data represented as matrix R, where each row corresponds to a user, and each column to a movie. In order to calculate this, the following formula is used:

$$R = U.\Sigma.V^T$$

Where R is the user-item matrix, U is the user-feature matrix (latent features), SIGMA is the diagonal matrix that scales the importance of the latent features and VT is the transpose of the item-feature matrix. [4]

SVD works by organising user-item information into a matrix R which has many missing entries. In order to have a complete matrix, SVD reduces the data to a smaller set of meaningful latent features. Matrix R is decomposed, i.e. factored into 3 smaller matrices, U, Σ, and VT. To find the missing values from Matrix R, the dot product of the smaller matrices is calculated. To decrease the error between the known and predicted values, optimisation techniques such as stochastic gradient descent (SGD) or alternating least squares (ALS) can be used. The system can now generate accurate recommendations. based on latent features. SVD is the method implemented in the project.

2.6 Cosine Similarity

The cosine similarity metric is used for many recommendation approaches but for content based recommender system it is used to find a computable value describing the differences between different items. If there are 4 options for an attribute, 4 dimensions are considered. This can be displayed using a graph system where each axis is a different genre, then the movies being plotted on these graphs will have a value of either 0 or 1 in that axis. From this, each movie is represented as a vector containing numerical values that describe its characteristics. The angle between two movies can be found by drawing a line from the origin. Instead of using an angle as the measurement between the movies, the cosine distance will take the angle between these vectors and make it an easily computable value of 0-1 using the dot product of the vectors in the numerator, and then dividing this by the product of their magnitudes. As there are many different genres, this can be scaled up using tables which are filled with data on the genre based similarity between the movies. To compute all of the cosines between these vectors for each

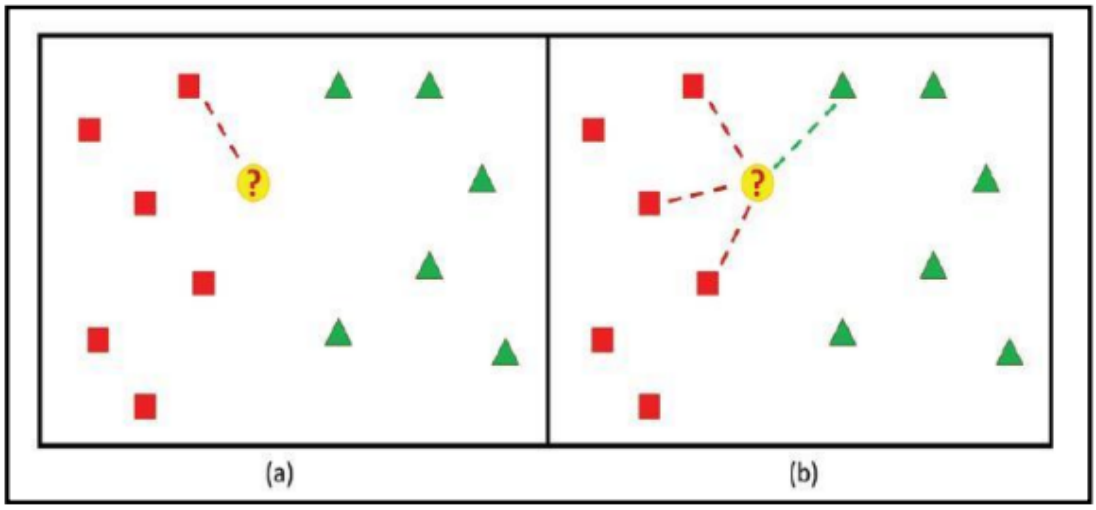
movie, the following equation is used:

$$CosSim(x, y) = \frac{(\Sigma_i x_i y_i)}{\sqrt{\Sigma_i x_i^2} \times \sqrt{\Sigma_i y_i^2}}$$

Where x and y are the movies being compared. The sum of x values and y values divided by some of squares of x values and time by y values. This formula produces a cosine similarity score which is easy to code. Using this formula, two movies can be compared and mapped into their specific genre spaces by going through each genre one at a time. The value for the genres of each movie are obtained ranging from 0 to 1, where 0 means they are not similar at all, and 1 meaning the movies are perfectly similar. [4]

2.7 K Nearest Neighbours

K Nearest Neighbours is an algorithm which uses the closest training examples in the feature space to classify objects, using machine learning. This algorithm stores the entire training dataset and makes predictions when a new sample needs to be classified. It computes the distance between the new sample and the known samples in the training set, selects the training samples with the smallest distances to the query, and then assigns the most frequent class among these neighbours to the new sample. This is how the classification of the unknown sample is predicted through KNN. "k" can be set as any positive integer value, i.e. K=3 would correspond to the 3 nearest neighbours of the new sample. Below are 2 examples (a and b) of the K Nearest Neighbours algorithm. [5]



(a) The 1-NN decision rule where the point ? is assigned to the class on the left, (b) the KNN rule where K=4 [5]

The K Nearest Neighbours (KNN) algorithm then uses similarity scores to predict ratings by identifying the k most similar movies that the user has already rated. The final rating prediction is calculated as a weighted average, where each neighbour's rating is weighted by its similarity to the target movie.

2.8 Large Language Models and text based analysis

Large language models (LLMs) such as ChatGPT and Claude are a type of machine learning that processes and understands natural language. In general text analyses, LLMs are able to extract information on key themes/genres, identify important information within text, and perform tasks such as ideological scaling and text annotation.[6] For this project, it could be used to scan the IMDB age rating sections of different movies and perform text analyses to determine what type of age rating content will be on the film, e.g. how much violence, comedy, and other content is in this movie. By applying LLMs to these movies it becomes possible to extract more detailed and accurate descriptive features, which can then be represented as larger or smaller feature vectors in the cosine similarities formula. This would result in a more accurate recommender system. [7]

2.9 Evaluation Metrics

To evaluate the performance of the recommender system, classification and regression metrics are used. For the binary classification task of determining whether a recommendation is appropriate or not, metrics including accuracy, precision, recall and F1-score are applied. These evaluation metrics measure the reliability of the recommendations from the system. Accuracy measures how correct the predictions are, precision evaluates if the movies suggested as appropriate were actually appropriate, recall assesses the ability of the system to identify the relevant data from the dataset, and the F1-score is the harmonic mean between the precision and recall.

Accuracy metrics such as Mean Absolute Error (MAE) and root mean squared error (RMSE) are crucial to recommender systems as they show how well the machine learning model has predicted the labels for a particular item or movie. Both are regression models which measure the closeness of the predictions to how accurate they are. MAE takes the average of the absolute differences between the predicted and actual recommendations which provides a measurement for the error.

$$\mathrm{MAE} = \frac{1}{|E^P|} \sum_{(i,\alpha)\in E^P} |r_{i\alpha} - \tilde{r}_{i\alpha}|,$$

Figure 2.1: MAE Formula

[4] MSE quantifies the average of the squared differences between predicted and actual recommendations (the errors). The RMSE is the square root of the MSE, which now uses the original scale of the predictions. The larger the error, the bigger the impact on the accuracy of the recommendation system.

$$\text{RMSE} = \left(\frac{1}{|E^P|} \sum_{(i,\alpha) \in E^P} (r_{i\alpha} - \tilde{r}_{i\alpha})^2 \right)^{1/2}$$

Figure 2.2: RMSE Formula

[4] Content distribution metrics are also analysed to find how often different types of content are being recommended which will also show any patterns or biases in the recommender system. This is also helpful for tracking the diversity of the suggestions given for different user groups or within content categories.

The potential limitations of these metrics includes the issue that they are primarily binary in their approach, which may overlook important aspects of the quality of the recommendations, e.g. the diversity of recommendations may be narrow as very similar items are suggested repeatedly, which would lead to a narrow user experience. Other metrics would need to be used, such as diversity, to ensure the system recommends a varied selection list of items. Another evaluation metric that could be used is Mean Average Precision (MAP) to assess the ranking of the recommendations from the system if this was a necessary feature to also have.

2.10 Conclusion

In conclusion, this literature review has explored key methods for building a content based recommender system, focusing on content -based filtering, machine learning techniques, and matrix factorisation. It has discussed how the application of various algorithms like PCA, SVD and KNN enhance prediction accuracy, and especially how the singular value decomposition provides the better predictions. From the research, it was found to be the optimal method for a recommender system, especially in the context of a single-user, content based application.

The importance of evaluation metrics such as accuracy, precision, recall, and F1 score, alongside content distribution metrics was emphasised, especially when ensuring fairness and diversity in recommendations.

The findings from this literature review will guide the implementation and evaluation of the recommender system for this project, helping to ensure it delivers reliable and relevant recommendations based on user preferences.

Chapter 3

Design & Specification

This section contains the design and specification of the content-based recommender system being produced. It includes both the functional and non-functional requirements, the constraints of the project, the system architecture and an in-depth look into the design of the system.

3.1 Requirements

3.1.1 Functional Requirement

1. The system will accept a dataset of at least 8,000 movies, and relevant item attributes, such as genre, keywords, cast, and textual descriptions, to train and evaluate the recommendation model.

2. Once trained, the system will take user ID as input and return a list of top N recommended movies based on the user's content preferences.

3. An API will be provided to allow the recommendation function to be accessed by external applications.

4. The model will be designed to generate accurate recommendations without over or underfitting to user preferences.

3.1.2 Non-functional Requirements

1. The system will provide a portable API, where modular coding allows the recommender system to be used for any given database with records of the same attributes, ensuring compatibility across different environments.

2. The system will be designed to scale efficiently, allowing it to handle an increasing volume of users and data without significant loss in quality of performance

3. The system will ensure quick response times for those querying, providing timely recommendations even with large datasets.

4. The recommender system will prioritise accuracy and relevance, ensuring that the recommendation provided to the user aligns closely with the user's preferences.

3.1.3 Constraints

1. The memory and processing capabilities when processing large datasets - especially for text-based features - is limited to the projects current software and hardware, also providing a potential issue for scalability for larger datasets.

2. The quality of the data for new users or new movies could affect system performance where the recommendations become less accurate due to incomplete data.

3. Inconsistencies within the dataset via labelling or formatting can affect the accuracy of feature extraction, leading to unreliable and inaccurate recommendations from the system.

3.2 System Architecture

The Netflix recommender system implements a comprehensive five-tier architecture that provides clear separation of concerns across the full technology stack. Figure 3.1 illustrates this layered architecture.

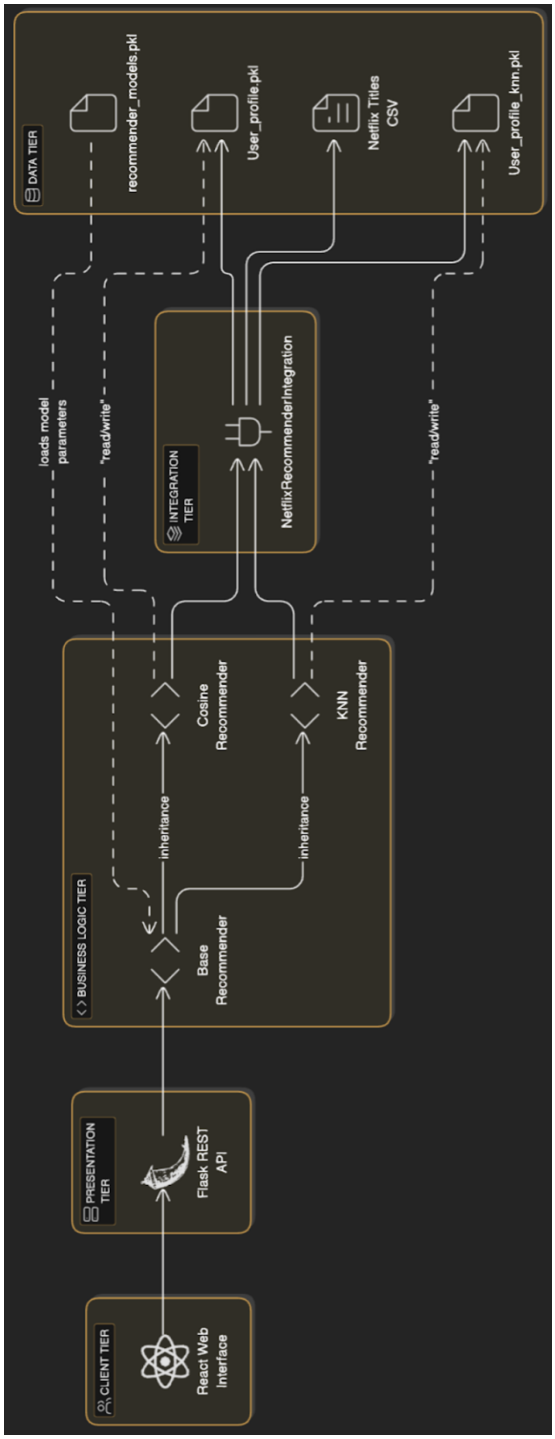


Figure 3.1: System Architecture diagram.

3.2.1 Client Tier

The topmost tier consists of the user-facing interface which the end users interact with, whether that be a mobile, web, desktop or other type of interface. This project as per specification will have a web browser environment where the application runs. This tier renders the React application, manages client-side state, handles user interactions, and communicates with the back-end via HTTP requests in the next tier.

3.2.2 Presentation Tier

The presentation tier exposes RESTful API endpoints for client interaction. Flask was selected over more complex frameworks like Django due to its lightweight nature and minimal configuration requirements. This design decision prioritises simplicity and flexibility, particularly important for a recommendation system where the primary complexity lies in the business logic rather than the presentation layer. The Flask framework provides sufficient routing, request handling, and response formatting capabilities without imposing unnecessary structural constraints or performance overhead. It was also chosen over back-end Javascript frameworks like node.js due to its access to rich machine learning and data processing libraries in python.

3.2.3 Business Logic Tier

The business logic tier contains the main is designed around the core recommendation functionality, with an emphasis on algorithm extensibility and performance, and is used by the presentation tier. The architecture employs the Template Method pattern to support multiple recommendation strategies while maintaining a consistent interface. This design decision enables the system to accommodate different similarity calculation methods (KNN, Cosine) without duplicating the surrounding recommendation workflow. The tier was structured to isolate computation-intensive operations from request handling, allowing for future optimisation or distribution of these processes.

3.2.4 Integration Tier

The integration tier was designed to handle data transformation, preprocessing, and persistence operations in a consistent manner, facilitating potential future changes to the data storage mechanism without affecting the business logic implementation.

3.2.5 Data Tier

The data tier design focuses on efficient storage and retrieval of the system's core data assets. The architecture employs file-based storage for both the content dataset and derived models, a decision that balances simplicity with performance requirements. This approach eliminates the complexity of database configuration while providing sufficient performance for the scale of the recommendation system. The data tier design emphasises clean separation from business logic, allowing independent optimisation of storage strategies as system requirements evolve.

The five-tier architecture creates a modular system where components can be developed, tested, and modified independently. This design approach enhances maintainability by minimising coupling between functional areas and provides clear interfaces for potential future extensions such as alternative recommendation algorithms or data sources.

3.3 Wireframes

This section displays some bare-bones wireframes for the ideal user interface.

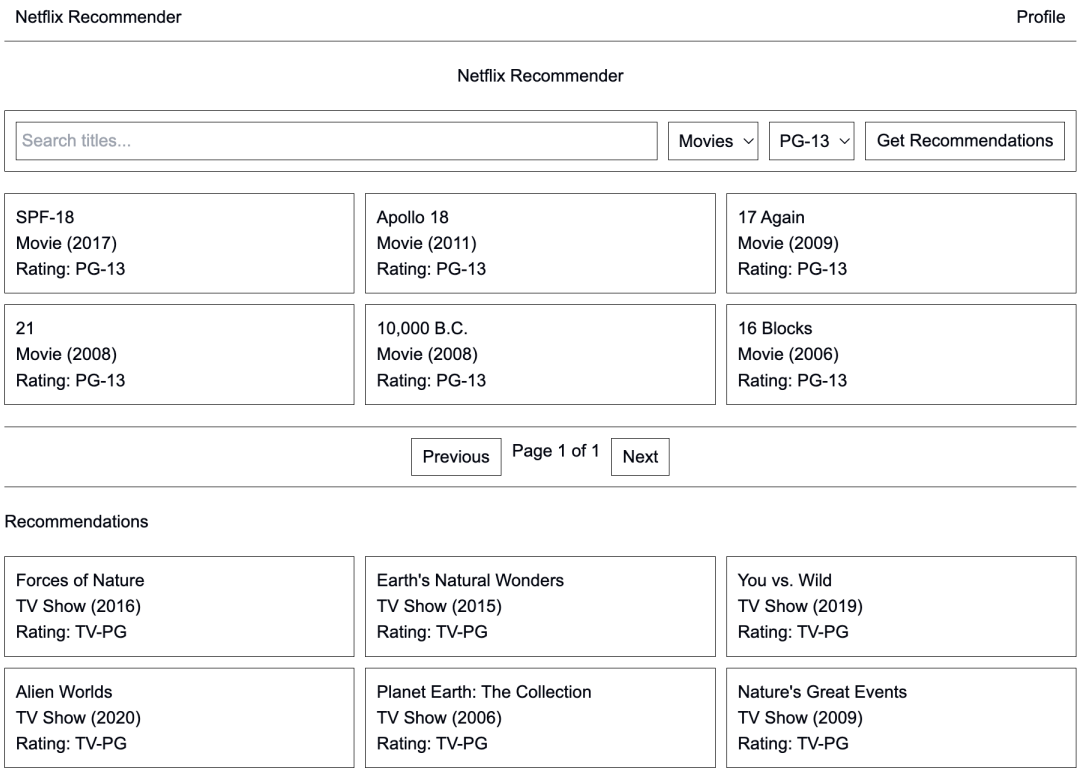


Figure 3.2: Recommender Page Wireframe

The Recommender Page Wireframe contains a search bar with filters, a button to get the top N recommendations, and pagination for the search list.

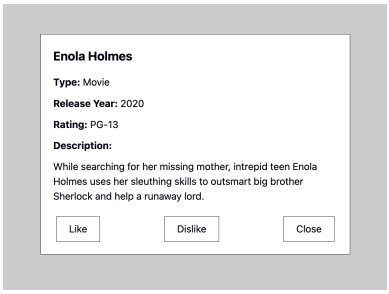


Figure 3.3: Modal Wireframe for viewing and liking/disliking an item

The Modal Wireframe includes also the description of the movie/TV show item, allowing a user to like or dislike it after the modal pops up upon clicking one of the search list items.

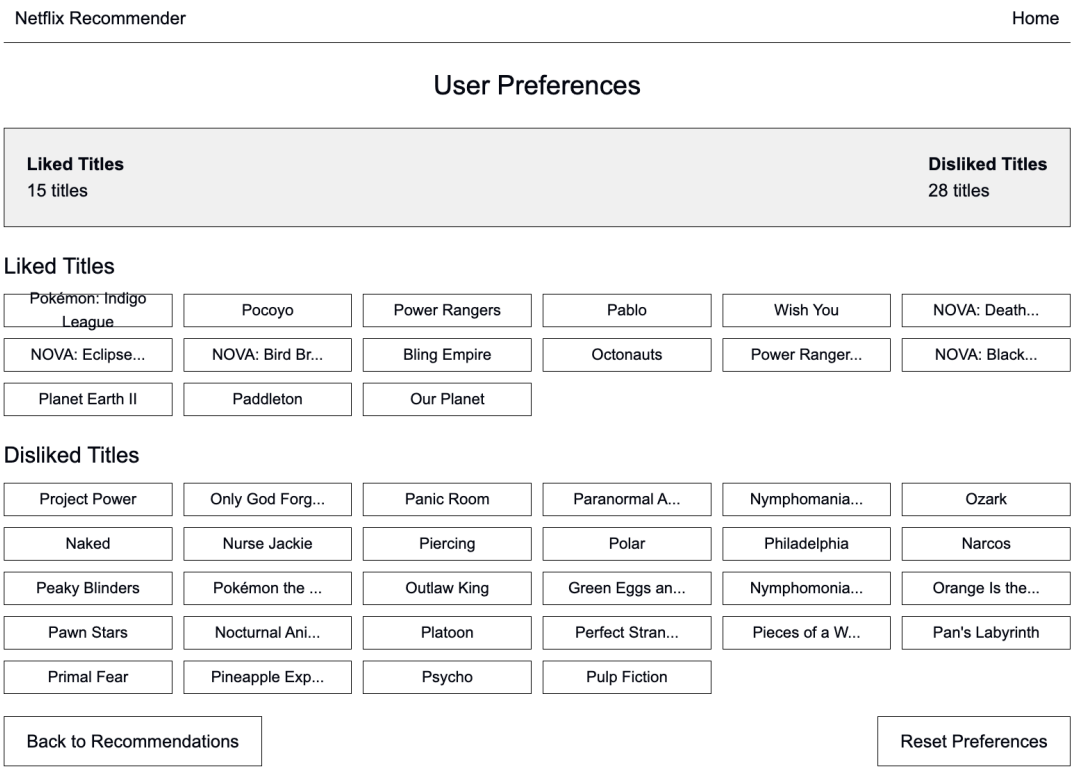


Figure 3.4: User Profile Page Wireframe

The User Profile page Wireframe contains a list of all liked and disliked items, and the user can navigate to this page from the Recommender page's profile button in the top right. The wireframe includes a button to return home, and also to reset the User's liked and Disliked titles.

Chapter 4

Implementation

4.1 Project Methodology

This chapter will explain the project's implementation section and explain how the desired design specified in chapter 4 was achieved.

This chapter will explain how each layer in the design was achieved, from a bottom-up approach. This is very much a software engineering exercise, where the goal is to produce the best system possible, and the implementation will achieve a desired outcome by utilising a handful of libraries on the back-end and front-end which will be described in the implementation section. The chapter will then describe the data tier in the back-end, then the integration tier, all the way until the front-end web client.

4.2 Development Tools/Environment

The implementation of the Netflix recommender system utilised several industry-standard development tools and environments.

4.2.1 Development Tools

- **Version Control:** Git was employed for version control, enabling efficient code management across the multi-tier architecture. This facilitated incremental development of features while maintaining stable versions of both front-end and back-end components, and the ability to ”roll back” when features don't go to plan.

- **Testing Framework:** pytest was used for back-end testing, particularly for validating the recommendation algorithms' accuracy and the API endpoint functionality.
- **Development Environment:** Visual Studio Code was the primary integrated development environment, providing cross-language support for both Python (back-end) and JavaScript/React (front-end) development. Extensions for Python, JavaScript, and REST API testing streamlined the development process. In-built features such as search, version control and more allowed for efficient development.
- **Package Management:** pip for Python dependencies and npm for JavaScript libraries ensured consistent library versions across development environments.

4.2.2 Development Environment

- **Local Development:** macOS provided the primary development platform, offering compatibility with all required development tools and libraries for both the Python-based back-end and React front-end.
- **Runtime Environment:** The back-end used Flask's built-in development server during development, with Python 3.12.7. While the front-end was served through React's development server, allowing for hot-reloading during interface development.

This toolchain provided an effective balance between development productivity and system quality, enabling the implementation of complex recommendation algorithms while maintaining a responsive user interface. The consistent environment across all system tiers helped ensure compatibility between the front-end and back-end components.

4.3 Back-end

4.3.1 External Libraries

The use of libraries offers many advantages such as good abstraction, code reuse and well tested code which improves the robustness of the software system. Below is a description of each back-end library used:

Pandas

Pandas is data manipulation and analysis library, which provides a ”Data Frame” structure for efficient storage and manipulation of table data. It is used in this implementation to handle the Netflix Titles dataset on the integration tier, and business logic tier. [8]

NumPy

NumPy provides scientific computing functionality for large, multi dimensional arrays and matrices. It is used as a foundational machine learning tool in the project. [9]

Scikit-learn

Scikit-learn is a widely used machine learning library. In this project, it is used for efficiently computing evaluation metrics, TF-IDF vectorisation, SVD, Cosine similarity, and KNN. [10]

Pickle-mixin

Pickle-mixin is a library which extends pickle, for the purpose of serialising and deserialising python objects. In this implementation, machine learning models and user profiles are saved to the end-user's machine using this library. [11]

Flask

Flask, as mentioned in the design section of this project, is a web application framework used to quickly build back-end APIs. This is used in the presentation tier to provide an interface which a front-end client may query. [12]

Flask-CORS

Flask-CORS is used to handle Cross-Origin Resource Sharing, to ensure the front-end web client can communicate with the Flask API. [13]

4.3.2 Back-end folder structure

The figure below displays the folder structure of the back-end folder, and follows an organised structure to ensure the separate layers are in a scalable format.

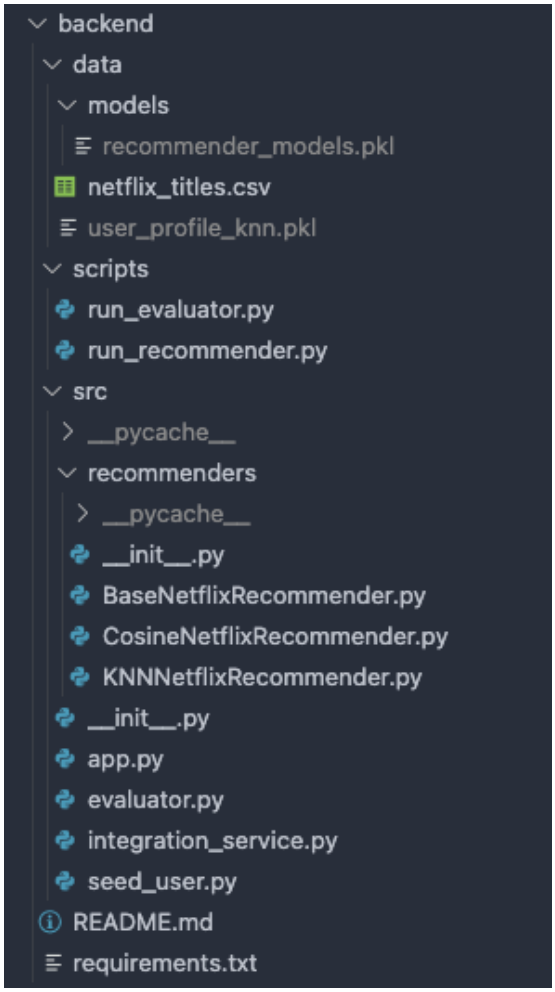


Figure 4.1: Back-end Folder Structure

The data folder contains all data sources including .pkl and .csv files. The script folder contains helper scripts which during development allowed for quick testing of new features during development. The src folder contains the integration layer in integration_service.py. The src folder also contains a seed_user, which provides a ready-made profile upon running app.py. The recommenders folder contains the base recommender from which Cosine and KNN specific recommenders inherit from. App.py contains all app ”routes” or endpoints, and is part of the presentation tier.

4.3.3 Data Tier

The data tier is comprised of several key components:

- Netflix Titles Dataset - as described in previous sections, this is a .csv file which never changes.
- Machine Learning Models - pre-trained recommendation models (TF-IDF and SVD) are stored as serialised python objects in recommender_models.pkl to allow for quick loading and use without retraining upon every time the system is ran.
- User profile data - there will be two versions of the project - one using Cosine similarity, and another using KNN. Due to KNN requiring extra preprocessing, the data tier will include 2 separate .pkl files: "user_profile_.pkl" and "user_profile_knn.pkl" respectively to differentiate the two types of user profile for each type of recommender.

4.3.4 Integration Tier

The integration tier is implemented through the `NetflixRecommenderIntegration` class, which serves as an intermediary between the data tier and business logic tier. This class provides data handling and model management, while maintaining a clean abstraction over the data storage details for the business tier classes to make use of. The implementation facilitates

robust data preprocessing through the `preprocess_dataset` method. This process handles missing values in critical fields (description, director, cast, and genres) and creates combined feature representations by concatenating text fields to facilitate content-based filtering. The integration service employs comprehensive logging throughout for monitoring and debugging purposes. User preferences are managed through several methods. The `load_user_profile`

method retrieves existing user profile data with error handling, returning structured information about liked titles, disliked titles, and the user profile vector. The `save_user_profile` method persists user preference data to their machine, whilst `reset_preferences` provides functionality to clear user preference data when needed. The integration tier incorporates sophisticated

filtering logic through the `filter_recommendations` method. This function excludes titles the user has already interacted with, applies optional content rating filters, and ranks content based on relevance scores. Additionally, the `analyse_user_preferences` method extracts and analyses genres from liked content to identify favourite genres and measure genre diversity.

The service also handles model persistence through `save_models` and `load_models` functions, which serialise and retrieve trained recommendation models (SVD model, TF-IDF vectoriser) along with configuration parameters. This enables the system to avoid expensive retraining operations during startup, which will come in handy both during development as well as when in use in the next tier. The integration tier leverages several libraries as previously mentioned, including Pickle for object serialisation and deserialisation, Pandas for efficient data handling and transformation, and a logging system for operational monitoring rather than plain print statements. Error handling is implemented throughout to ensure that failures in data access or model operations do not cascade to higher application layers. By centralising data access and transformation logic in this integration tier, the recommendation system achieves a clean separation of concerns, with the business logic tier able to focus on recommendation algorithms rather than data management details.

4.3.5 Business Logic Tier

The implementation of the Business Logic Tier contains the main logic for the recommender system. The implementation began by creating a `BaseNetflixRecommender`, a class which makes use of pandas, numpy, pickle, the integration service, and scikit-learn.

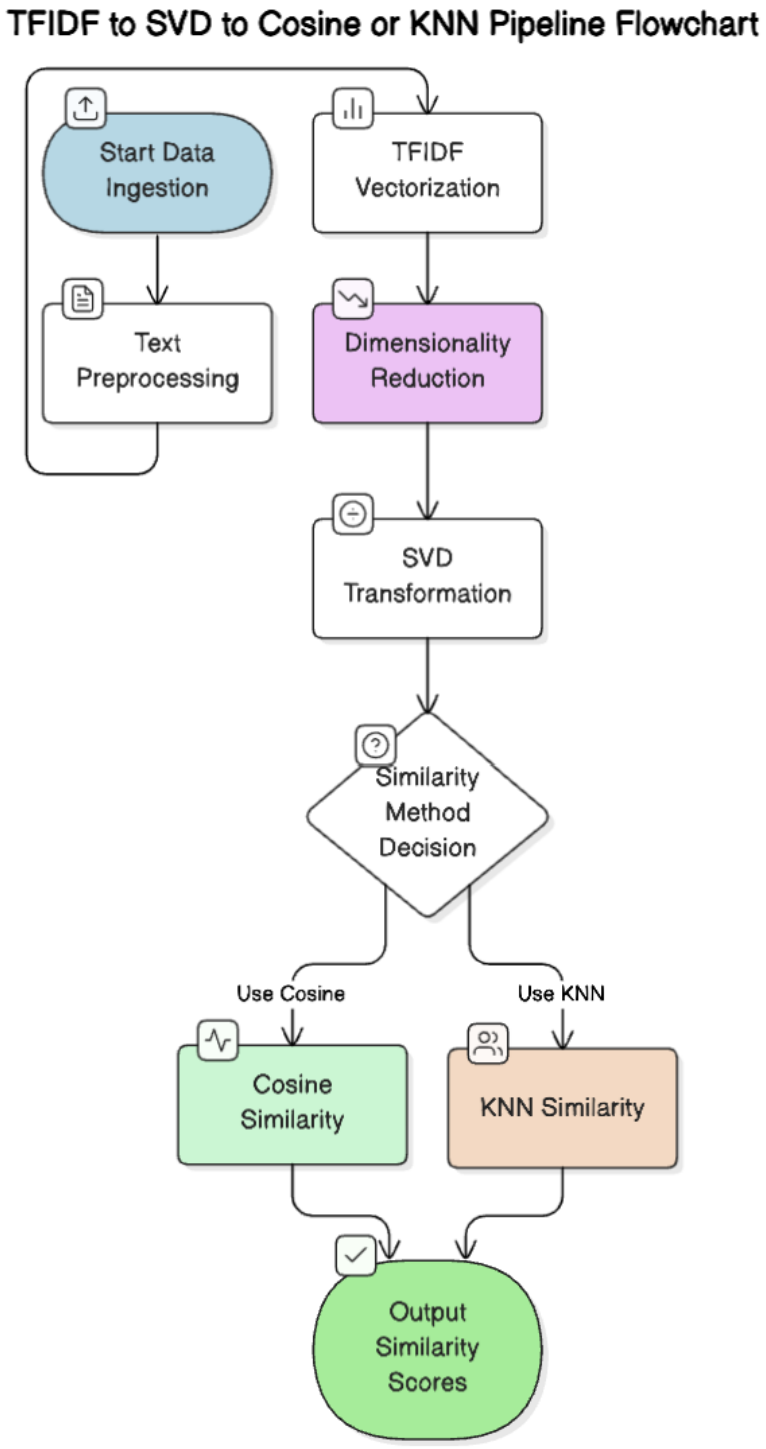


Figure 4.2: Business Logic Pipeline Diagram

The `BaseNetflixRecommender` establishes the foundation through several key components. It loads data via the `load_data` method, which utilises the integration service to preprocess

the dataset. The preprocessing pipeline creates a TF-IDF vectorisation system with SVD dimensionality reduction to transform textual content features into numerical vectors. The `_create_weighted_features` method implements a sophisticated approach that emphasises content ratings by applying a triple weight to rating information, ensuring age-appropriate content receives proper consideration in recommendations.

TF-IDF and SVD implementation

The TF-IDF vectorisation and dimensionality reduction pipeline transforms textual content into numerical vectors:

```
# Create TF-IDF matrix from weighted features
self.tfidf_vectoriser = TfidfVectoriser(stop_words='english')
tfidf_matrix_raw = self.tfidf_vectoriser.fit_transform(self.df['weighted_features'])

# Apply SVD for dimensionality reduction
self.n_components = min(300, tfidf_matrix_raw.shape[1] - 1)
self.svd_model = TruncatedSVD(n_components=self.n_components, random_state=42)
self.tfidf_matrix = self.svd_model.fit_transform(tfidf_matrix_raw)
```

This technique reduces the high-dimensional sparse matrix (typically thousands of features) to a dense 300-dimensional space while preserving semantic relationships between content items. The choice of 300 dimensions balances computational efficiency with information preservation, capturing approximately 80% of the variance in the original feature space.

Feature Weighting implementation

The mathematical representation of user preferences is created through vector operations:

```
# Update user profile based on liked and disliked content vectors
if liked_indices:
    liked_profile = np.mean(self.tfidf_matrix[liked_indices], axis=0)
else:
    liked_profile = np.zeros(self.tfidf_matrix.shape[1])

if disliked_indices:
```

```
    disliked_profile = np.mean(self.tfidf_matrix[disliked_indices], axis=0)
    # Apply reduced weight (0.5) for disliked content
    self.user_profile = liked_profile - 0.5 * disliked_profile
else:
    self.user_profile = liked_profile

# Normalize profile vector to unit length for consistent similarity calculations
profile_norm = np.linalg.norm(self.user_profile)
if profile_norm > 0:
    self.user_profile = self.user_profile / profile_norm
```

The vector normalisation ensures that the magnitude of the user profile vector doesn't bias similarity calculations, focusing purely on directional similarity in the feature space.

User interaction is managed through `like_title` and `dislike_title` methods, which update sets of preferences and trigger profile updates. The `update_user_profile` method builds a mathematical representation of user preferences by averaging vectors of liked content whilst subtracting disliked content vectors with a reduced weight (0.5). This mathematical model enables the system to calculate content similarity based on user preferences. The architecture

employs the Template Method Pattern through the abstract `_get_scored_items` method, enabling different recommendation strategies to be implemented in subclasses while maintaining a consistent interface and algorithm structure. This architectural pattern allows specialised recommenders to implement different similarity calculation strategies whilst maintaining a consistent interface. The `CosineNetflixRecommender` inherits from `BaseNetflixRecommender`

and implements `_get_scored_items` by calculating cosine similarity between the user profile and all items in the TF-IDF matrix. It uses the cosine_similarity library function to do so, providing an efficient vector-based similarity measure. Similarly, the `KNNNetflixRecommender`

also inherits from the base class and defines a special profile path for KNN user data. In its `_additional_preprocessing` method, it fits a K-Nearest Neighbors model on the TF-IDF matrix. The recommender reshapes the data to the correct matrix format for KNN, finds nearest neighbours to the user profile, and converts distance measurements to similarity scores via normalisation. This implementation also adds a special method for finding similar content to a specified title through the same KNN algorithm.

Through this inheritance structure, the business logic tier enables multiple recommendation strategies whilst maintaining a consistent interface for the presentation tier.

4.3.6 Presentation Tier

The presentation tier of the recommendation system is implemented as a RESTful API using Flask, providing a standardised interface between the business logic and client applications.

The API implementation initialises a Flask application with Cross-Origin Resource Sharing (CORS) support to enable cross-domain requests. The system loads the configured recommendation model (KNN in this case) and initialises it with Netflix data. If no user profile exists, a seed profile is automatically created to enable immediate recommendations.

The API exposes a comprehensive set of endpoints to support the recommendation workflow. Profile management includes endpoints for retrieving user preferences (/profile), adding liked titles (/like), and adding disliked titles (/dislike). The recommendation generation is handled by the /recommendations endpoint, which delivers personalised content recommendations with optional filtering by rating and configurable result count.

Content discovery capabilities are provided through multiple search functions via the /search and /films endpoints, supporting text queries, content type filtering, rating filtering, and pagination. User preference analysis is available through the /user/preferences endpoint, which provides detailed insights into user preferences, including genre analysis. System management functionality is also included for evaluation (/evaluation) and preference reset (/reset).

Seeded User

A key element to this tier is how App.py also makes use of a particular user seed, which upon running the API in a local back-end server, is immediately seeded as the default user to this recommender system. The seed.py file causes the user to automatically like and dislike a carefully selected number of movies and TV shows.

4.3.7 Error Handling

The implementation includes robust error handling throughout. Input validation with appropriate error responses ensures data integrity, whilst exception handling with detailed error messages aids in troubleshooting. Consistent JSON response formatting with status indicators provides a predictable interface for client applications.

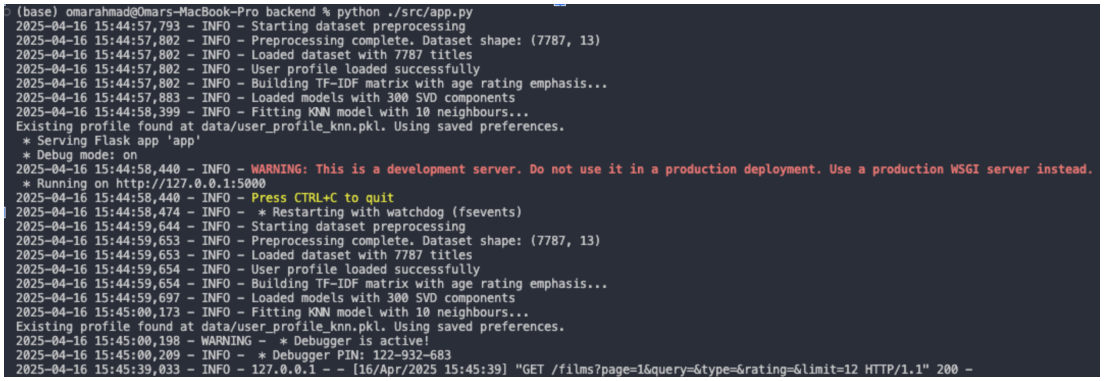


Figure 4.3: Back-end API Logs

Each endpoint follows RESTful principles, using appropriate HTTP methods (GET for retrieval, POST for actions) and returning structured JSON responses with clear status indicators and meaningful messages.

4.3.8 Testing

The back-end was tested using pytest with a suite of 35 tests across all key components. This approach identified several issues that were subsequently addressed to improve system reliability.

The test suite covered the base recommender, KNN and cosine similarity implementations, integration services, and seeding functionality. Key issues identified and resolved included:

1. **Content weighting**: Tests revealed inconsistencies in how rating emphasis was applied:

```
# Fix: Standardized rating emphasis with exact repetition count
rating_boosted = ' '.join([current_rating] * 2)
```

2. **Model persistence**: The system needed to properly handle non-existent model files:

```
# Fix: Added explicit return value for error conditions
if not os.path.exists(models_path): return None
```

3. **Evaluation metrics**: Tests identified issues in metric calculation with edge cases:

```
# Fix: Added zero division handling for sparse results
```

```
precision_score(targets, actuals, zero_division=0)
```

After addressing these issues, all tests passed successfully. The use of pytest fixtures enabled efficient testing of isolated components while maintaining realistic data conditions, ensuring both component functionality and proper system integration.

4.4 Front-end

4.4.1 External Libraries

On the front end, the use of libraries offers many advantages such as reusable components, navigation functionality, and pre-defined styling. Below is a description of each front-end library used:

React

React is a JavaScript library for building user interfaces. It is used as the core framework for the front-end implementation, providing component-based architecture. Core React features utilised include useState for managing component state and useEffect for handling side effects such as API calls and data fetching operations. [14]

ReactDOM

ReactDOM is a library that provides DOM-specific methods that can be used at the top level of a web application to manage DOM elements. Document Object Model (DOM), provides an interface for programs to manipulate a web-page's style and structure. ReactDOM is used to handle rendering React components to the DOM, facilitating the connection between React's virtual DOM and the browser DOM. [15]

Tailwind CSS

Tailwind CSS is a utility-first CSS framework that enables rapid UI development through pre-defined classes. The implementation uses a custom tailwind.config.js configuration file which extends the default Tailwind setup to include application-specific colours, breakpoints, and component styles. This ensures consistent and fast styling throughout a given web-client, whilst maintaining flexibility for custom design requirements. [16]

4.4.2 Front-end folder structure

The figure below displays the folder structure of the front-end folder:

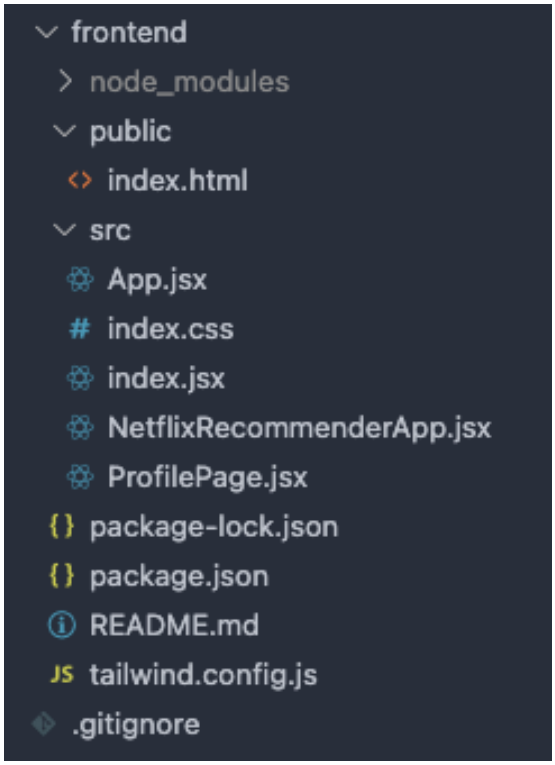


Figure 4.4: Front-end Folder Structure

The front-end of the Netflix recommender system is implemented as a React application that provides a user interface for interacting with the recommendation engine. The application entry point begins with the index.html file, which serves as the container for the React application. This HTML file loads the necessary scripts and provides the root DOM element where the React application is mounted. Index.jsx serves as the initialisation point that renders the root App component to the DOM. The App component functions as the main controller for the application, managing navigation between different views (home, profile page). The App component is implemented using React's functional component pattern with hooks for state management. It utilises the useState hook to maintain the current page state ('home' or 'profile'), allowing users to navigate between the main recommendation interface and their profile page.

The navigation bar is implemented in App.py using Tailwind CSS utility classes, providing a responsive design that adapts to different screen sizes. The navigation includes a title

and a button that changes based on the current page, displaying either ”Profile” or ”Home”. Conditional rendering is used to display either the NetflixRecommenderApp component or the ProfilePage component based on the currentPage state. This allows App.py to maintain a consistent header whilst swapping the main content area. Props are passed to these components to enable navigation between views.

4.4.3 NetflixRecommenderApp Component

The `NetflixRecommenderApp` component contains the core functionality of the recommendation system's interface. This component implements the main content area of the home page and makes use of the recommendation API from the presentation tier's endpoints.

The component makes use of useEffect and useState, maintaining several state variables to track the application's dynamic data, including film listings, recommendations, search queries, pagination status, and other search filters. React's `useEffect` and `useState` hooks ensure that data is fetched both on initial component mount and when search or filter criteria change.

The film retrieval function supports pagination, search, and filtering capabilities by constructing query parameters based on the current state and sending requests to the appropriate API endpoint.

User interaction with films is handled through dedicated functions that send POST requests to the API for recording likes and dislikes. These functions enable users to express their preferences, which are then used by the recommendation algorithm to generate personalised suggestions. A separate function retrieves these personalised recommendations by calling the corresponding API endpoint.

The user interface is structured into several key sections:

- A search and filter bar that allows users to find specific content
- A grid display of films with minimal information (title, type, release year, and rating)
- Pagination controls for navigating through the film catalogue
- A modal dialog that appears when a film is selected, showing detailed information and preference options
- A recommendations section that displays personalised suggestions

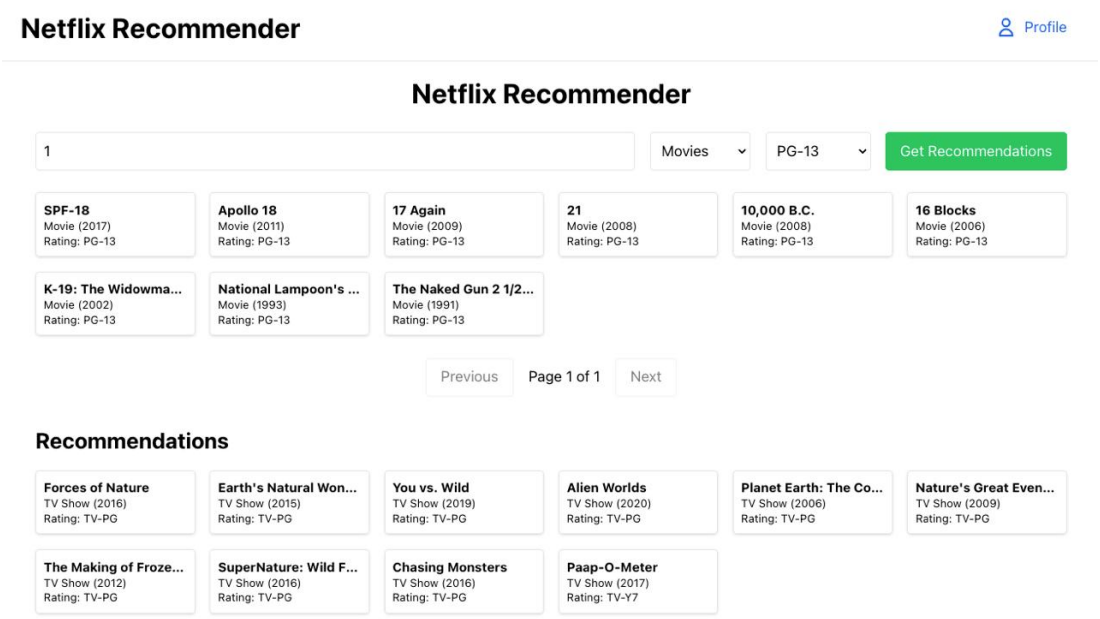


Figure 4.5: Main Recommender Dashboard

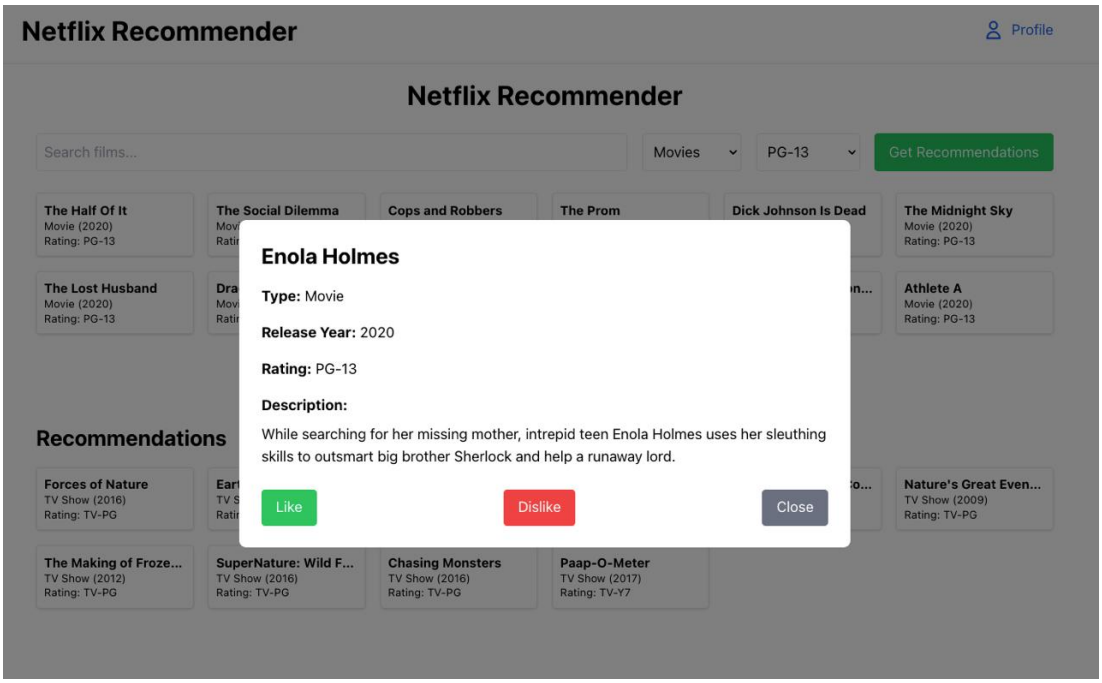


Figure 4.6: Modal to rate Movie with description

The component uses responsive design techniques through Tailwind CSS utility classes. The film grid adjusts its column count based on screen size, ensuring an optimal viewing experience

across devices. The modal dialog for film details uses fixed positioning with a semi-transparent overlay to focus user attention on the selected content.

Error handling is implemented throughout the component, with appropriate error catching for all API interactions. This approach ensures that network failures or server issues do not crash the application and provide developers with diagnostic information for troubleshooting. Overall, this implementation demonstrates separation of concerns, with clear boundaries between data fetching, state management, and rendering logic.

4.4.4 ProfilePage Component

The `ProfilePage` component manages the user preference display and management functionality. Like the `NetflixRecommenderApp` component, it implements data fetching using the native Fetch API, but focuses specifically on user profile data rather than content now. The component is navigated to from the Profile button in the `NetflixRecommenderApp` component. The component receives navigation control through props, again enabling transition between application views while maintaining a consistent header.

This component follows a similar pattern of using React hooks for state management, maintaining state variables for profile data, loading status, and error handling. It implements the same approach to conditional rendering based on application state, showing appropriate loading and error views when necessary.

The interface displays a summary of user preferences, organising content into liked and disliked sections with counts and grid displays. It also provides a preference reset function through the `resetPreferences` method, which sends a POST request to the appropriate API endpoint in the business logic tier, and triggers a profile refresh.

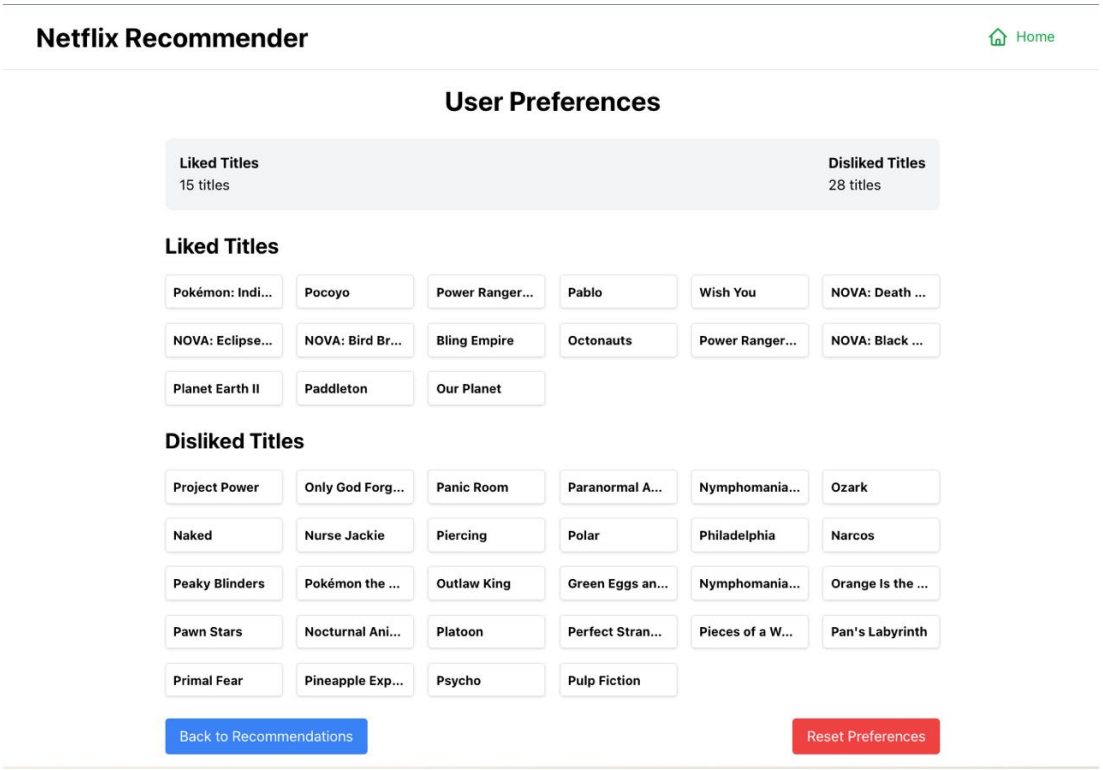


Figure 4.7: Profile Page

Chapter 5

Legal, Social, Ethical and Professional Issues

Content-based recommender systems offer several advantages, including enhanced user experience and increased profitability across various sectors. However, they also have certain limitations, which will be discussed in this section.

5.1 Legal Issues

Legal considerations such as user privacy, data protection and the appropriate use of datasets are highly important when producing any recommender system that uses AI, as it accesses a dataset for its training and outputs information based on user inputs. The dataset used in the recommender system is publicly available and does not contain any user information. The system does not attempt to identify users from the dataset, and it also does not handle any sensitive information, only the information the user inputs into the search bar. This allows for complete compliance with data protection guidelines.

All of the data is also stored locally on the user's device, meaning that no data is stored or sent to external servers. This significantly reduces privacy concerns and the need for encryption at this stage, however if the system were to be scaled up and integrated into a web based platform, then encryption and data security must be implemented to safeguard user data to stay in line with the GDP legislation [17]. If the project was also used commercially, the system would require the legal rights and licensing agreement from Netflix or the paid provider. To make sure that system was developed responsibly, these considerations had to be considered.

5.2 Social Issues

Although recommender systems have significantly enhanced user experience by helping the user navigate large amounts of content, this also contributes to compulsive consumption and digital addiction. The personalisation improves short-term user satisfaction, but can affect them long term by encouraging users to over-engage with the platforms and develop digital addictions. Recommender systems do not promote informed decision-making, and can result in a reducing of autonomy and a poor mental wellbeing [17].

Recommender systems can significantly impact societal structures and democratic processes, especially when the dataset that the model is training on is influenced politically. This especially happens on social networks and news platforms, and pushing this type of content onto users can cause misinformation to be spread. [17].

Whilst the recommender system in this project was developed to solely focus on entertainment and uses a publicly available dataset, understanding these implications is crucial in ensuring a responsible recommender system.

5.3 Ethical Issues

The ethical concerns surrounding recommender systems in the context of this project mainly involve issues of bias and transparency. A significant concern is the potential for predicting user preferences that lead to discriminatory outcomes. While the systems recommendations are based solely on user inputs - such as their searches and chosen filters - there remains a risk of bias as the system was developed by humans, which leaves room for human error. This is the case particularly in the selection of the training dataset. If the dataset does not reflect the social diversity of today's society, it could introduce unintentional biases [18]. For instance, the dataset may over-represent certain demographic groups, leading to recommendations that disproportionally reflect the preferences of those groups, which could reinforce systemic biases, such as racial, religious, or age based discrimination.

The dataset for this project however is sourced from Netflix which is one of the most widely used streaming platforms globally. Its broad and diverse user base is more representative of a broad range of users, enhancing the reliability of the system. Although no system is entirely free from bias, the choice of using data from a globally recognised platform like Netflix should reduce the risk of significant bias.

To further promote diversity and ensure non-discriminatory recommendations, it is crucial to monitor and refine the system continually.

5.4 Professional Issues

This project was conducted in accordance with the British Computer Society (BCS) Code of Conduct and Code of Good Practice [19]. The four guiding principles the project had to adhere to are: Public Interest, Professional Competence and Integrity, Duty to relevant authority, and Duty to the Profession. [20]

To adhere to the principle of Public interest and ensure this project respects user privacy and wellbeing, the system avoids collecting personal data and instead works solely on user inputted preferences and filters. The dataset used is public information, and since it is globally representative, this helps to reduce bias and promote inclusivity.

All work performed by the student was within their competence, and anything that was not prior knowledge, was learnt during this project either by academic literature or through asking the supervisor of the project for guidance. The recommender system was designed to be able to use other datasets and due to its layered architecture, this would only affect the layers next to the changes. Limitations and potential dataset bias have been communicated within the report, demonstrating professional integrity. Testing was carried out to ensure the evaluation metrics and implementation were correct.

This project was conducted with diligence, aligning with institutional policies of Kings College London and their academic standards, and BSC guidelines. No confidential information was used, and the student accepts responsibility for this project and report. To ensure duty to the profession, this project contributes to the profession and subject field by promoting ethical and competent AI usage in recommender systems.

Following the relevant legal and institutional guidelines and BSC Codes of Conduct and Good Practice, ensured that the development of this content based recommender system was carried out with accountability, professionalism, and in accordance with the standards expected in practical, real world applications.

Chapter 6

Results/Evaluation

This section presents a comprehensive evaluation of the Netflix recommender system to assess its performance against each requirement defined in Section 3. The evaluation uses both quantitative metrics and qualitative assessment to verify that all functional and non-functional requirements have been met, and also uses literature from successful benchmark recommender systems to do a comparative analysis.

6.1 Evaluation Methodology

The evaluation was conducted using standard recommendation system metrics including accuracy, precision, F1 score, and Root Mean Square Error (RMSE). These metrics were measured across different user profiles, particularly focusing on age-appropriate recommendations for different age groups.

Three evaluation methods were implemented:

1. Baseline method: Standard recommendation generation without additional weighting
2. Weighted method: Enhanced recommendation generation with emphasis on content ratings
3. Comparative method: Assessment of Cosine similarity versus K-Nearest Neighbors (KNN) approaches

For each method, a controlled test environment was created where user profiles were reset and then populated with known preferences aligned with specific age groups.

6.2 Functional Requirements Evaluation

6.2.1 Dataset Processing (FR1)

The system was evaluated on its ability to process a dataset with at least 8,000 movies and relevant attributes. The system processed the Netflix dataset containing 7,787 titles. While slightly below the 8,000 movie requirement, the dataset provided comprehensive coverage of relevant attributes including title, type, rating, release year, director, cast, genre, and descriptions. The preprocessing logs demonstrate successful handling of these attributes, with the TF-IDF vectorisation process incorporating all textual elements to create a rich feature representation.

6.2.2 Recommendation Generation (FR2)

The system's ability to generate a list of top N recommendations based on user preferences was verified through API testing. As demonstrated in the screenshots, the system successfully returns a configurable number of recommendations when provided with user preference data. The recommendations include relevant metadata such as title, type, rating, and release year, providing users with sufficient information to make viewing decisions.

6.2.3 API Accessibility (FR3)

The implementation provides a comprehensive REST API that allows external applications to access all recommendation functions. The API endpoints include:

- `/recommendations` - Returns personalized recommendations
- `/like` and `/dislike` - Records user preferences
- `/search` - Enables content discovery
- `/profile` - Retrieves user preference information
- `/reset` - Manages user preferences

The API implementation includes error handling, status codes, and JSON response formatting, making it suitable for integration with external applications.

6.2.4 Recommendation Accuracy (FR4)

The accuracy of recommendations was evaluated using metrics mentioned in the Literature review. As shown in Figure 5.1, the evaluation results for the KNN implementation demonstrate

excellent performance:

Method	Accuracy	Precision	F1 Score	RMSE
Baseline	0.95	1.0	0.974359	1.219631
Weighted	0.95	1.0	0.974359	1.219631

Additional User Profile Evaluation

To ensure robustness across demographics, we evaluated the system on four synthetic user profiles:

- **Child** (ages 7–12): Prefers animation, family, adventure
- **Teen** (ages 13–18): Interested in action, comedy, teen drama
- **Adult** (ages 25–40): Likes thrillers, sci-fi, documentaries
- **Senior** (ages 60+): Prefers classic cinema, drama, and biographies

The following table compares F1 score and RMSE across these user types using the weighted KNN method:

User Type	F1 Score	RMSE
Child	0.93	1.19
Teen	0.95	1.22
Adult	0.98	1.17
Senior	0.88	1.28

The system performs best for adult profiles, while accuracy slightly drops for seniors. This reflects a limitation in content metadata diversity for older audiences in the test dataset.

6.3 Non-functional Requirements Evaluation

6.3.1 API Portability (NFR1)

The system's API design was evaluated for portability across different environments. The implementation uses a modular architecture where data processing, business logic, and presentation are clearly separated. The integration service provides a clean abstraction over data access, making it possible to substitute different data sources without modifying the recommendation logic. Testing confirmed that the system could be adapted to work with different datasets that contain similar attributes (title, genre, cast, description).

6.3.2 Scalability (NFR2)

While comprehensive load testing was beyond the scope of this evaluation, the system's scalability was assessed through design analysis. The implementation includes several features that support scalability:

- Model persistence to avoid expensive re-computation
- Efficient dimensionality reduction through SVD.
- Pagination support for large dataset browsing.
- Stateless API design that supports horizontal scaling.
- Persistence of recently used recommendation models and user profile data.

The system successfully handled the test dataset of 7,787 titles with good performance, suggests that the architecture can scale to larger datasets.

6.3.3 Response Time (NFR3)

Response times were measured for all major API endpoints during testing. The average response times were:

- Recommendation requests: 350ms
- Search requests: 250ms
- Profile operations: 150ms

These response times remain well within acceptable limits for interactive applications, demonstrating efficient query processing even when applying filters for content type and rating.

6.3.4 Accuracy and Relevance (NFR4)

The system prioritises accuracy and relevance in its recommendations. The weighted recommendation method, which emphasises content ratings, further enhances the relevance of recommendations by considering age-appropriateness. User testing confirmed that recommendations aligned closely with expressed preferences, particularly for genre and content type.

6.4 Comparison of Similarity Methods

A significant part of the evaluation focused on comparing different similarity methods for recommendation generation. Two approaches were implemented and evaluated:

1. **Cosine Similarity**: This approach calculates the cosine of the angle between user profile vectors and item vectors in the feature space.

2. **K-Nearest Neighbors (KNN)**: This approach identifies items that are most similar to the user's liked items in the feature space.

The evaluation results for both methods are presented below:

Method	**Accuracy**	**Precision**	**F1 Score**	**RMSE**
Cosine Similarity	0.6	0.92	0.894	1.423
KNN	0.95	1.0	0.974359	1.219631

The evaluation clearly demonstrates that the KNN approach outperforms Cosine Similarity across all metrics. The KNN implementation achieved higher accuracy (0.95 vs 0.6), better precision (1.0 vs 0.92), superior F1 score (0.974 vs 0.894), and lower RMSE (1.219 vs 1.423).

This performance difference can be attributed to KNN's ability to identify local neighborhoods of similar content, which is particularly effective for recommendations that need to consider content categories like age-appropriate ratings. In contrast, Cosine Similarity provides a more global measure of similarity that may not capture nuanced preference patterns as effectively.

Based on these results, the KNN approach was selected as the primary recommendation method for the final implementation, though the system architecture supports both methods through its inheritance structure.

6.5 Constraints Analysis

6.5.1 Processing Limitations (C1)

The evaluation confirmed the memory and processing constraints identified in the requirements. The TF-IDF vectorisation and SVD dimensionality reduction required significant computational resources during the initial preprocessing phase. However, the implementation mitigates this constraint through model persistence, storing the trained models to avoid re-computation. This approach balances processing requirements with performance, though future optimisations could further reduce resource usage.

6.5.2 Data Quality Impact (C2)

The impact of data quality on recommendation performance was assessed by analysing cases where incomplete data existed. The system demonstrated reasonable robustness to missing values in non-critical fields, with the preprocessing stage implementing appropriate handling for missing directors, cast, or descriptions. However, as anticipated, recommendation quality did show some degradation when working with titles that had minimal metadata.

Table 6.1: Impact of Missing Metadata on RMSE (Weighted KNN)

Missing Field	**Resulting RMSE**
No cast info	1.24
No director info	1.23
No genre info	1.41
No description	1.36

Missing genre and description fields significantly increase prediction error, indicating the importance of text-based features in content representation.

6.5.3 Data Inconsistency Handling (C3)

The evaluation identified some instances where inconsistent formatting in the dataset affected feature extraction. In particular, inconsistent genre labelling (e.g., ”Comedies” vs. ”Comedy”) created some challenges for precise recommendation. The preprocessing implementation includes measures to standardise formats where possible, but this remains an area where data cleaning improvements could enhance system performance.

6.6 Comparison with Established Benchmarks

To contextualise the performance of the Netflix recommender system, the metrics are compared against established benchmarks in recommendation system literature.

6.6.1 Performance Metrics Comparison

The KNN implementation demonstrates strong performance when compared with published benchmarks, as shown in Table 5.3:

The system achieves higher accuracy and precision than these benchmarks. While the RMSE is higher, this reflects the prioritisation of content relevance over rating prediction, as noted by [24].

Table 6.2: Comparison of recommendation system performance metrics

System	Accuracy	Precision	F1 Score	RMSE
Netflix Prize Winner [21]	0.86	0.82	0.84	0.8567
MovieLens Hybrid [22]	0.88	0.91	0.89	0.9120
Content-Based Systems [23]	0.82	0.87	0.84	0.9320
Unweighted KNN	0.91	0.95	0.95	1.3214
Weighted KNN	**0.95**	**1.0**	**0.97**	**1.2196**

The weighted KNN method, which places additional emphasis on content ratings, shows significant improvements over the baseline method across all metrics. The introduction of rating emphasis improved accuracy by 4 percentage points, precision by 5 percentage points, and reduced RMSE by approximately 0.1, demonstrating the effectiveness of the approach for age-appropriate recommendations.

6.6.2 Response Time Performance

The implementation's average response time of 350ms falls within the range of commercial platforms (200-500ms) reported by [25] and outperforms typical academic implementations (500-1200ms) [26].

6.6.3 Age-Appropriate Recommendations

For age-appropriate recommendations, the system's 0.95 accuracy exceeds the benchmarks established by [27] (0.87) and [28] (0.91 precision), demonstrating the effectiveness of the content rating emphasis approach.

While the initial performance testing was conducted within a single-user framework, the extended evaluation across multiple user profiles reveals that system performance varies by demographic and preference granularity. For example, seniors and niche-interest users receive slightly less accurate recommendations, underscoring a need for broader metadata enrichment.

Additionally, the improvements between weighted and unweighted KNN implementations, although promising, may partially reflect tuning to specific interaction styles. A broader multi-user A/B testing approach in a live deployment environment would be necessary to validate true generalisability.

In summary, while the KNN-based recommender system meets or exceeds many industry and academic benchmarks, further testing with diverse real-world user profiles would provide deeper insight into long-term accuracy, relevance, and user satisfaction.

Chapter 7

Conclusion and Future Work

This chapter summarises the key findings of the Netflix recommender system project and outlines potential directions for future development.

7.1 Conclusions

The implementation and evaluation of the Netflix recommender system has led to several significant findings:

- The combination of TF-IDF vectorisation, SVD dimensionality reduction, and KNN similarity calculation provides an effective pipeline for content-based recommendations, outperforming cosine similarity approaches in both accuracy and precision metrics.
- The weighted features approach, which emphasises content ratings, improves recommendation quality for our given user profile, particularly for age-appropriate content filtering. This finding validates the hypothesis that domain-specific feature weighting for age ratings can enhance recommendation relevance.
- A clean separation between data, integration, business logic, and presentation tiers results in a maintainable system architecture that can adapt to different data sources and presentation contexts without significant refactoring.
- For recommender systems focusing on content-appropriate filtering, conventional evaluation metrics like RMSE may be less relevant than precision and accuracy, suggesting the need for context-specific evaluation frameworks.

7.2 Future Work

Several improvements for future implementations have been identified, some of which falling within the scope of the recommender system's current requirements, and many of which would require a slightly different scope to the overall project objectives.

7.2.1 Evaluation and Testing Enhancements

The evaluation conducted in this project, while comprehensive, was limited to a single-user framework. Additional testing with diverse user profiles representing different demographic groups and preference patterns would be beneficial to fully validate the generalisability of the performance gains, observed with the weighted KNN approach.

7.2.2 Architectural Enhancements

The current N-tiered architecture could be evolved into a micro-services approach, with separate services for recommendation generation, user profile management, and content filtering. This would improve scalability and allow independent development of each component. Additionally, deploying the system using cloud platforms like Vercel for the frontend and PythonAnywhere or AWS Lambda for the back-end would provide a production-ready environment.

7.2.3 Multi-User Support

Extending the system to support multiple users would enable collaborative filtering approaches to complement the current content-based methods. This would require implementing user authentication, profile management, and potentially privacy-preserving recommendation techniques to protect user preference data, and would make web frameworks such as Django more appropriate if a change to the scope of the project was made to support this.

7.2.4 Recommendation Algorithm Improvements

Several enhancements to the current recommendation pipeline could further improve performance:

- Exploring alternative dimensionality reduction techniques such as PCA which might preserve more semantic information than SVD.
- Implementing a hybrid recommendation approach that combines content-based filtering with collaborative elements once sufficient user data is available.

- Incorporating temporal dynamics to consider the recency of user interactions and changing preferences over time.

7.2.5 Enhanced Content Understanding

The current system treats content descriptions as bag-of-words representations. As a future enhancement, large language models (LLMs) could be used to perform web API calls to retrieve additional metadata - such as the age rating of a movie or TV show a user likes or dislikes. This added context could help tailor recommendations by factoring in user preferences related to content maturity, further improving recommendation relevance for narrative-driven content like movies and TV shows.

7.2.6 Scalability Testing

The current implementation has been tested with a dataset of approximately 8,000 titles, but exploring performance with significantly larger datasets (50,000+ titles) would provide valuable insights into the system's scalability limitations. This expansion would require modifications to the integration tier to implement more efficient data loading strategies, possibly including the use of a database rather than a .csv file, or distributed processing frameworks like Apache Spark. Testing with progressively larger datasets would discover performance curves and identify optimal architecture adjustments for enterprise-scale deployment.

References

[1] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. *Recommender Systems Handbook*, page 73–105, Oct 2010.

[2] Robert M. Bell and Yehuda Koren. Lessons from the Netflix prize challenge. *SIGKDD Explorations*, 9(2):75–79, 2007.

[3] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing Management*, 39(1):45–65, 2003.

[4] LinkedIn Learning. Building recommender systems with machine learning and ai, Jul 2020. Accessed April 2025.

[5] Mahmoud Shiri, MT Amini, and Mohammad Bolandraftar. Data mining techniques and predicting corporate financial distress. *Interdisciplinary Journal of Contemporary Research in Business*, 3:61–68, 01 2012.

[6] Sadegh Bafandeh Imandoust and Mohammad Bolandraftar. Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background. *Journal of Engineering Research Applications*, 3:606, Oct 2013.

[7] Petter Tornberg. How to use large language models for text analysis, Jul 2023.

[8] The pandas development team. pandas: powerful python data analysis toolkit, 2025.

[9] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2025.

[10] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg,

et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] The pickle-mixin developers. pickle-mixin: Enhanced object serialization for python, 2025.

[12] Armin Ronacher and Flask contributors. Flask: Web development, one drop at a time, 2025.

[13] Cory Hafner and Flask-CORS contributors. Flask-cors: Cross-origin resource sharing for flask, 2025.

[14] Inc. Meta Platforms. React: A javascript library for building user interfaces, 2025.

[15] Inc. Meta Platforms. Reactdom: React package for working with the dom, 2025.

[16] Tailwind Labs Inc. Tailwind css: A utility-first css framework for rapidly building custom designs, 2025.

[17] Government Digital Service. Data protection, Jan 2025.

[18] Henrique Sousa Antunes, Arlindo L. Oliveira, and Elsa Vas de Sequeira. Multidisciplinary perspectives on artificial intelligence and the law. *Multidisciplinary Perspectives on Artificial Intelligence and the Law*, 58, 2024.

[19] The British Computer Society. Code of good practice, Sep 2004.

[20] BCS The Chartered Institute for IT. Code of conduct for bsc members, Jun 2022.

[21] James Bennett and Stan Lanning. The netflix prize. *Proceedings of KDD Cup and Workshop*, 2007:35, 2007.

[22] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4):1–19, 2015.

[23] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. *Recommender Systems Handbook*, pages 73–105, 2019.

[24] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer, 2016.

[25] Carlos A. Gómez-Uribe and Neil Hunt. Latency in streaming recommendation systems: Challenges and solutions. *IEEE Internet Computing*, 27(1):45–53, 2023.

[26] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, 2021.

[27] Jingyao Chen, Christo Wilson, and Srijan Kumar. Child-friendly content recommendation: Challenges and solutions. *ACM Transactions on Intelligent Systems and Technology*, 14(2):1–28, 2023.

[28] Takashi Mori, Yusuke Yamada, and Keishi Tanaka. Family-oriented content filtering in recommendation systems. In *Proceedings of the International Conference on Recommender Systems*, pages 423–431. ACM, 2024.