


UNIVERSITY OF SOUTH WALES

PRIFYSGOL DE CYMRU

**FACULTY OF COMPUTING, ENGINEERING & SCIENCE
SCHOOL OF COMPUTING & MATHEMATICS**

STATEMENT OF ORIGINALITY

This is to certify that, except where specific reference is made, the work described in this project is the result of the investigation carried out by the student, and that neither this project nor any part of it has been presented, or is currently being submitted in candidature for any award other than in part for the MSc Data Science degree of the University of South Wales.

Signed  (student)

Date SEPTEMBER 30, 2024

(This statement must be bound into each copy of your Project Report)

Utilising machine learning algorithms in movie recommender systems

Joebright Boakye-Dankyi
School of Computing and Mathematics
University of South Wales
Treforest, United Kingdom
30098870@students.southwales.ac.uk

Abstract— In recent times, movie recommender systems have made it easier for one to find a movie to watch, instead of an extensively laborious internet search for movies, systems have been designed to recommend movies without looking into large databases. This work delves into the use of machine learning algorithms to design movie recommender systems. The use of content-based filtering methods, such as cosine similarity and collaborative-based filtering, such as K-Nearest Neighbors (KNN) and the Surprise library package with huge datasets, was used to build algorithms that recommended movies to a user, the recommender system development comprised of data pre-processing, model applications, training, and evaluations. The Cosine Similarity model, with a precision of 1.0, recall of 65.2% and an F1 score of 78.9%, achieved the best results as it focused on the genre-based recommendation about content-based filtering. On the other hand, K-Nearest Neighbors (KNN), with a precision of 0.23%, recall of 0.79% and an F1 score of 0.4%, shows how difficult the model had to recommend relevant movies to a user within a collaborative filtering approach. The surprise library package model within a collaborative filtering approach demonstrated much stability with minimal Root Mean Square (RMSE) and Mean Absolute Error (MAE) of 1.1128 and 0.8746, respectively. These recommendation systems can be used for specific results and will be of great importance to users.

Keywords—Content-based filtering, Collaborative filtering, Cosine Similarity, K-Nearest Neighbors (KNN), Surprise library package, precision, recall, F1 score, Root Mean Square Error (RMSE), Mean Absolute Error (MAE).

I. INTRODUCTION

The prolific introduction of recommendation systems has changed digital content consumption by offering users personalised suggestions custom-made to their preferences. With the increasing volume of content on streaming platforms, these systems play a vital role in enhancing user experience. They enhance user experience across platforms like Netflix, Amazon, eBay and YouTube and have drawn significant attention from researchers aiming to improve accuracy and efficiency. Recommender systems assist users in navigating lots of content available on most streaming platforms, most of these systems are built using advanced machine learning techniques. Most of these techniques highlight collaborative filtering, content-based filtering, and hybrid filtering [1].

Collaborative Filtering is a well-known technique in recommendation systems; it is primarily based on the idea that users are likely to enjoy items that either they or people with similar preferences have liked previously. Memory-based collaborative filtering includes user-item filtering, which recommends items based on similar users' preferences, and item-item filtering, which suggests similar items to those already liked by the user. These methods rely on similarity

measures like cosine similarity [2]. Model-based approaches, such as Singular Value Decomposition (SVD), use machine learning to predict user ratings, offering better speed and scalability. Although effective, memory-based methods can struggle with sparse data and high computational demands. Collaborative Filtering (CF) techniques, specifically Matrix Factorization (MF), can be used for making group recommendations [3]. Content-based filtering (CBF) approach recommends items based on their similarity to what users have previously shown interest in, analyzing item attributes such as genre or textual data. It operates freely of other users' preferences, making it effective in conditions where there are more items than users. Content-based filtering leverages methods like Term Frequency-Inverse Document Frequency (TF-IDF) and Singular Value Decomposition (SVD) to create latent features, allowing the system to recommend items similar in content to a user's past interactions [1]. The system extracts contextual information from items and builds models to provide personalised recommendations without relying on collaborative data from other users. A hybrid movie recommendation combines collaborative filtering and content-based methods to produce a preliminary list of movie suggestions. CF finds user preferences, while CBF approaches capture movie attributes like genre, director, and actors. When sentiment analysis is applied to user reviews, their scores are adjusted, and positive sentiment enhances movie ratings while negative sentiment lowers them. [4] The framework also employs big data technologies, specifically the Spark platform, to increase the efficiency and scalability of the recommendation process, with tests showing reduced runtime as computational nodes in the Spark cluster increase.

In this paper, the aim is to create a personalised movie recommender system that enhances user experience by providing custom-made movie suggestions, of which we explore and implement machine learning techniques, focusing on collaborative filtering and content-based filtering. The goal is to design an accurate movie recommendation system by reviewing current techniques, preprocessing the MovieLens dataset, performing exploratory data analysis, building machine learning models using various recommendation approaches like K-Nearest Neighbors (KNN), Surprise library package, Cosine Similarity and optimizing them through performance evaluation.

This paper is organised as follows. Literature review, which discusses an overview and background of related works and knowledge gaps in the field of movie recommender systems but with more emphasis on Collaborative, Content and Hybrid filtering techniques using Machine learning models and algorithms provided in Section II. Section III describes a detailed methodology of how each machine learning model was built and eventually used in making movie recommendations to a user. Section IV presents the experimental results and evaluations of the models developed

in the methodology. The final Section V details the discussions, conclusions, and limitations of the study provided.

II. LITERATURE REVIEW

Content-based filtering operates by analyzing the characteristics of items and comparing them to the user's preferences. In the context of movie recommendations, this approach leverages attributes such as genre, actors, and plot to suggest relevant content to users. Content-based filtering uses cosine similarity and Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer as the primary techniques. Cosine similarity calculates the similarity between two vectors, such as movie descriptions, by measuring the cosine of the angle between them. TF-IDF vectorizer, on the other hand, enhances the model by quantifying the importance of words within the text data, enabling more precise filtering of content based on genre, cast, or other textual attributes [5]. One of the advantages of content-based filtering is its ability to recommend specifically to a user's historical interests. For example, if a user previously liked horror movies, the system will likely recommend similar titles based on those features. Content-based filtering is extremely efficient in cases where a user's preference is known, and it does not depend on other users' data. However, one limitation noted is the "filter bubble" problem, where the system constantly recommends similar content, limiting the variety of suggestions offered to users [6]. Content-based filtering requires the creation of complex feature vectors from movie metadata, which can be computationally expensive when the dataset grows. Computational cost is increased when the system needs to handle text-based features such as movie descriptions, as encoding these features will require methods like one-hot encoding can result in sparse matrices, thus requiring substantial processing power [7].

Collaborative filtering involves analyzing the behaviour of multiple users to make recommendations. The system assumes that users who have shown interest in similar items will likely have common preferences in the future. It leverages the preferences of similar users to make suggestions [8]. This technique can be further divided into user-based and item-based collaborative filtering. User-based collaborative filtering is recommending items to a user based on the preferences of other users with similar tastes. Thus, if two users share a similar history of movie preferences, the system might recommend movies that a user has watched to the other user. [8] On the other hand, Item-based collaborative filtering focuses on the relationships between items thus, movies that are frequently watched together or rated similarly are grouped, and the system recommends them to users who have shown interest in similar items.

Matrix factorization is one of the most implemented methods for collaborative filtering in recommendation systems, where user-item interaction matrices are decomposed to predict preferences. Work done by the researcher was among the pioneers in applying matrix factorization techniques to uncover latent user-item interactions [9]. More complex iterative approaches were developed by redefining these pioneered methods by integrating regularization techniques and tackling the challenges of data sparsity. Another specific form of matrix factorization is the Non-Negative Matrix Factorization (NMF), which decomposes the user-movie rating matrix into two non-negative matrices, which in turn uncovers latent factors that represent both movie characteristics and user

preferences [10]. NMF's strength lies in its capacity to capture subtle patterns in user behaviour, making it specifically effective for recommendation systems. The iterative improvement of matrix approximations is another area of advancement. Iterative refinement through SVD-I (Singular Value Decomposition-Iterative), which further optimizes predictions by repeatedly decomposing and re-estimating missing values in the rating matrix. This iterative approach results in superior accuracy compared to usual matrix factorization techniques, as improvements are demonstrated in recommender systems across multiple datasets [9]. Collaborative filtering tends to outperform content-based filtering in terms of predictive accuracy, but it suffers from the "cold-start problem," where it cannot make recommendations for new users or items with insufficient data. To overcome the individual limitations of content-based and collaborative filtering methods, they proposed the use of hybrid models [11].

Hybrid recommender systems combine collaborative filtering and content-based filtering to mitigate the weaknesses of both approaches, these approaches have been especially successful in e-commerce and entertainment platforms. Despite the potential of hybrid systems, computational challenges remain. This is because each item must be analyzed and compared with the user's preferences, resulting in significant computational overhead. A recommender system named RECOM effectively suggests movies depending on the user's interests, demonstrating the effectiveness of content-based filtering, and collaborative filtering in conjunction with machine learning algorithms [12]. Several machine learning algorithms are employed to enhance the performance of recommender systems. Among the most used are K-means clustering, principal component analysis (PCA), and self-organizing maps (SOM). K-means clustering is a basic algorithm used in collaborative filtering. It groups users based on common features such as age, gender, and past viewing history. The system then suggests movies based on the cluster to which a user belongs [13]. K-means is relatively simple and computationally efficient but suffers from scalability issues when applied to large datasets. The cold-start problem also persists in this algorithm, as it relies on user interaction data for clustering. PCA is a dimensionality reduction technique used to enhance K-means clustering by reducing the number of features through the identification of the most significant features in a dataset, PCA improves the efficiency of clustering algorithms and speeds up the recommendation process [14]. [13] Self-Organizing Maps (SOM) is an unsupervised learning algorithm that maps high-dimensional data onto lower-dimensional spaces, it is commonly used in combination with PCA to improve the accuracy of movie recommender systems. SOM is capable of clustering users based on non-linear relationships between features, making it a powerful tool for identifying hidden patterns in user preferences but it is more computationally intensive compared to K-means clustering.

Experimental outcomes have shown that hybrid systems, combining autoencoders and SVD, are more likely to reduce the mean squared error (MSE) in predicted ratings and user satisfaction. Deep learning has been shown to positively impact movie recommendations by reducing dimensions and handling non-linear relationships in data [15]. These systems combine Collaborative Filtering (CF) and Singular Value Decomposition (SVD) to make recommendations. Machine learning models utilize a multi-cloud setting and offer powerful solutions for successful movie recommendation

systems by leveraging advanced clustering techniques and multi-cloud infrastructure [16]. Its ability to scale efficiently and deliver very accurate, customized recommendations makes it a valuable advancement over some other recommender systems, making it an effective tool for handling large datasets in real-world applications.

Despite the advancements in movie recommendation systems, several knowledge gaps persist. Most existing studies focus mainly on either content-based or collaborative filtering methods. There is a need for more research on models that effectively integrate both approaches to enhance recommendation accuracy. While many studies utilize supervised machine learning algorithms, there is limited exploration of unsupervised learning techniques in this domain suggesting that these approaches could uncover patterns in user behaviour that supervised methods might overlook [17]. User experience remains a less explored area in the context of movie recommendation systems. Research should probe into how user interface design, real-time adaptation and personalized feedback can improve user satisfaction and engagement with recommender systems [18].

III. METHODOLOGY

Python programming language was used for this work, and three machine-learning methods were applied to produce the movie recommendation system. The K-Nearest Neighbors(KNN) based recommendation with cosine as the metric was used for collaborative filtering. Cosine Similarity recommendation, where the cosine function was applied on feature vectors produced with the Term Frequency Inverse Document Frequency (TF-IDF) (*TfidfVectorizer()*) function, was considered for content-based filtering and the Surprise library with KNN Basic based recommendation model.

The data was obtained from the GroupLens Research Group, with specific extraction of the MovieLens 25M dataset which contains 25 million movie ratings, and 1 million tag applications applied to 62,000 movies by 162,000 users. Including tag genome data with 15 million relevance scores across 1,129 tags released in 2019.

Fig.1. shows the steps used in creating the selected recommendation models and how the outputs were obtained.

In exploring the data, “Movies” and “Ratings” were used, and every movie was made to have a unique “*movieId*” by dropping duplicated *movieId* from the column. 10,000 movies were selected randomly with unique *movieId* from the movies DataFrame, a *random_state* of 42 was set as a seed. The resulting movies DataFrame then had these 10,000 randomly selected unique movies after inspections. The sorted movie DataFrame was then merged with the ratings DataFrame using *movieId* as a common key, and unwanted columns, such as genres, were removed after the merge. Every row in the ratings DataFrame now had access to additional data in the sorted movies DataFrame for the corresponding *movieId*. The idea here was to work with numeric data from this point. With a *random_state* of 49 as seed, 100,000 rows from the ratings DataFrame were selected randomly.

The ratings DataFrame were then pivoted (*pd.pivot()*). This forms the *user-movie matrix* such that the columns represent unique *userIds*, the index represents unique *movieIds* and the values observed within the cells are the *actual ratings* each user gave to a particular movie. For cells where no ratings are given it was seen as *NaN*. The number

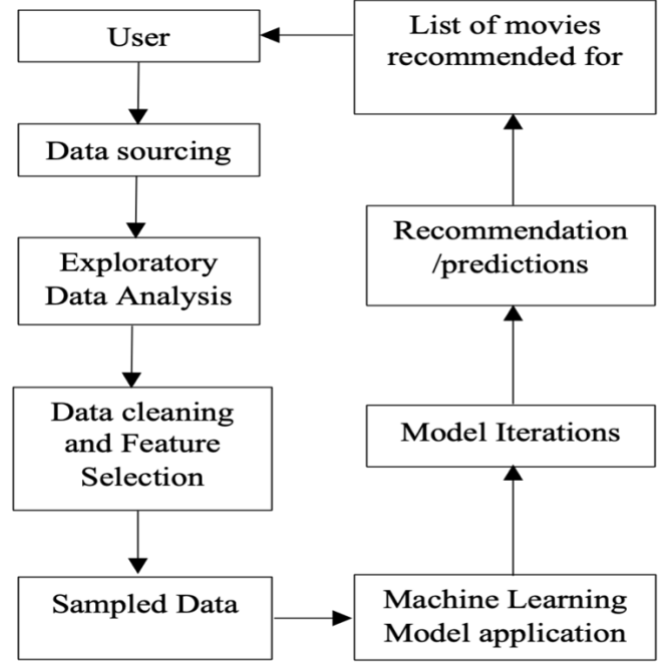


Fig.1: Steps taken to build the movie recommender system for users.

of users that rated each movie was computed and stored in a new DataFrame using *group by movieId*. Where *movieId* was the unique ID of each movie, and rating (*count*) was the number of users who rated each movie. The index was reset by converting the *movieId* from the index back into a regular column using *reset_index (level =0)* in our case, this is *movieId* making the dataframe easily accessible. Each user's number of movies that they have rated was also calculated by using *rating.groupby('userId')* and *['rating'].agg('count')* to give the exact number of movies each user has rated, which was stored in a new dataframe too.

Each user's number of movies that they have rated was also calculated by using *rating.groupby(userId)* and *['rating'].agg('count')* to give the exact number of movies each user has rated, which was stored in a new dataframe too.

All missing values thus *NaN*, were replaced with 0 in the DataFrame, indicating a user never gave a rating. Preliminary summary statistics like mean, count, standard deviations, minimum and maximum values were computed on both the DataFrames containing the number of users that rated each movie and that of the movies that have been rated.

A scatter plot of *movieId* and the number of users that rated each movie was observed, also that of *userId* and the number of movies they rated was seen.

A. K-Nearest Neighbors

This model is classified as a collaborative filtering recommendation, and it was used by calculating the similarities between a particular movie and its other related types. A new DataFrame, *data_final*, is obtained by merging the ratings and movies dataset, which creates new columns named *movieids*, *ratings* and *movie titles*. Train and test sets are obtained by using the *sample function()* to randomly split *data_final* into 80% *train_data* and 20% *test_data* respectively. From the *train_data* a user-item matrix is created with rows indicating users (*userId*), columns representing movies (*movieId*) and values matching users'

ratings. All missing values are replaced with 0 using *fillna(0)*, and the matrix is then converted to a sparse format with *csr_matrix()* to help with computational efficiency.

The K-Nearest Neighbors (KNN) model is developed using the cosine similarity metric to evaluate user similarity. The model is initialised with the *NearestNeighbors()* function and trained on the sparse training matrix with the *fit()* method. The parameter *n_neighbors=20* then specifies that the model will identify the 20 closest neighbors for each user. A user-item matrix is developed for the test_data and the process is repeated for the spare testing matrix too with the *fit()* method. A function *get_user_recommendation()* is defined to predict movie recommendations for a given user (*user_id*). If the user is not found in the training data, an error message is returned. The index of the target user in the training matrix is found using *get_loc()*. Then, the *knn.kneighbors()* function finds the nearest neighbors for that user. The *n_neighbors=n+1* function finds one extra neighbor because the closest neighbor in the matrix is the user themselves. Similar users' indices are retrieved, and the target user is removed from the list using (*similar_users[1:]*), leaving only the valid neighbors.

For each similar user, their movie ratings are retrieved from the *test_data*, unrated movies for the target user are identified as *unrated_movies*, and if a similar user has rated a movie the target user has not seen yet, it is added to the *recommend_movies* dictionary. If a movie receives ratings from multiple similar users, the ratings are merged. The movie recommendations in the dictionary are sorted in descending order by their ratings. The *top n movies* are selected, and their respective titles are retrieved from the movies dataset. The function returns the list of *recommended movie titles* or a message of *no new recommendations are available*. The *get_user_recommendation()* is fed with *userId 72315* as *get_user_recommendation(user_id=72315, n=20)* and an output of 20 movies is predicted for the user in descending order and model was evaluated for precision, recall and F1-scores.

B. Surprise

The surprise library used in this work falls under a type of collaborative filtering, the data from Ratings were used, with 100,000 rows and 4 columns which had *UserId*, *movieId*, *rating* and *timestamp*. Ratings data was stored in a dataframe called *df_ratings*, and all other useful tools in the library were imported, with class *KNNBasic()* stored as *knn*.

The *KNNBasic* algorithm is then initiated, and cross-validation of a 5-fold approach is done, allowing the model to be trained and tested on various aspects of the data. The Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) were used as the evaluation metrics to check the accuracy of the model's predictions. The observed values of the mean and standard deviation of RMSE and MAE gave more understanding of the model's performance across different folds, which also captured the time it took for the fitting and testing of the model.

The model was then trained on the entire dataset after the cross-validation process to improve its ability to predict. The similarity matrix was calculated as the foundation for making predictions about user ratings for movies. For a precise user with *Id 3* and a chosen movie with *movieId* as *1252*, the model predicted a rating of 5, showing how functional it is. This was

demonstrated by the predicted rating printed alongside the actual ratings from the DataFrame to verify the accuracy of this prediction based on the information given by *userId 3* in the dataset. The ratings DataFrame was merged with the movies dataset and unwanted columns like *timestamps* were dropped, restructuring the data for more analysis.

The function *top_n_movies*, was defined to recommend the top n movies for a given user, where n will define the number of movies to return. It was written such that, it predicts ratings for all movies that a user has not rated, it then sorts these predictions in a descending order and returns the top n movies. This defined function was applied to take in a user with *Id 3*, and it effectively returned the top 5 recommended movies along with their ratings.

C. Cosine similarity

The cosine similarity was applied in content-based filtering, this defines how similar two items are by calculating the cosine of the angle between the two items, which in the case of this work, items are movies. For two movies, say A and B, the cosine angle can be computed to determine how similar they are, the closer the value of the angle approaches zero (0), the more similar both movies are. This can denoted in a written vectors as *Sim(a,b)* (see Equation 1).

$$\text{Sim}(a,b) = \cos(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{|\vec{A}| |\vec{B}|}, \quad (1)$$

Sim(a,b) is the similarity between movies A and B.

For this model, the movielens 25m dataset containing movie ratings, titles, and genres was used. Each movie was uniquely identified by duplicating the dataset and dropping unwanted movie entries based on the movieId. The titles and genres of movies are combined into a single text feature using python's apply function, which employs a lambda function to merge both fields for each row. Genres are then separated by spaces for easy tokenization.

Text vectorization was then performed on the data to convert the merged text into lower cases to prevent the model from treating words with different cases as different tokens. The TF-IDF (Term Frequency-Inverse Document Frequency) is then introduced, thus a unique procedure to transform text data into numerical vectors. This is achieved by using *TfidfVectorizer* from Scikit-learn to convert the combined text into a matrix where each row denotes a movie and each column denotes a unique word from titles and genres. The value seen in the matrix represents how describing words in one movie is relative to the other. This resultant high-dimensional TF-IDF matrix is then converted to a sparse matrix format *csr_matrix* for computational effectiveness and memory. This matrix type is the best for handling large datasets as it stores only non-zero values, and thereby reducing memory to its optimal minimum.

When the text vectorizing was completed, the cosine similarity was then computed using the sparse matrix. The resulting matrix is a *sparse cosine similarity matrix* in which each value represents the similarity between two movies based on their TF-IDF vectors. Each movie is then compared against all the others, which gives a 3209 X 3209 matrix, meaning 3209 is the number of unique movies in the dataset, leaving out all the zero (0) values.

The *recommend_movies_sparse* function was then designed; this function takes a *movie title* as input and returns the top 5 most similar movies for a user. It does so easily by identifying the index of the inputted movie in the dataset. It then obtains the *cosine similarity scores* for that movie relative to all others using an equivalent row in the similarity matrix. The function excludes the input movie itself while sorting the similarity scores in descending order, and top similar movies are selected. The function returns the top 5 most similar movies as the prediction (recommendation). A movie titled “Kill Bill: vol 2 (2004)” was input in the function, and it successfully returned 5 top movies like what was inputted as a recommendation to a user. The model was then evaluated on precision, recall and F1-scores.

IV. EXPERIMENTAL RESULTS AND EVALUATIONS

This section discusses in detail the outcome of the three models when they were applied, their respective evaluations were also interpreted accordingly.

A. K - Nearest Neighbors

The function *get_user_recommendation (user_id=72315, n=20)* provides personalised movie recommendations for user 72315. User 72315 existed in the training data after checks. The function then identified other users with similar movie preferences using the K-Nearest Neighbors (KNN) algorithm. It then gathered movies that these similar users have rated, which the target *user_id* 72315 has not yet seen. These movies were ranked by aggregated ratings and returned a list of the top 20 recommended movie titles in descending order. The topmost 5 of the recommended movies were in the descending order as follows; Keeping the Faith (2000), Abominable Dr. Phibes, The (1971), Superman (1978), Killer Joe (2011), Mindhunters (2004). The model was evaluated by Importing necessary libraries, including precision score, recall score from *sklearn.metrics*. The function *evaluate_recommender()* is then defined to evaluate how accurately the system recommends movies to its users.

The function iterates through each user in the test dataset, compares their rated movies to the model's recommendations, and calculates the overlap between the recommended movies and the ones they rated. If no relevant items or recommendations exist for a user, the function skips to the next user. The function then calculates the precision and recall scores for each user. Precision measures the proportion of recommended movies that were relevant, while recall assesses the proportion of relevant movies that were successfully recommended. The precision and recall figures are stored for each user to calculate the overall performance later. The function then calculates the average precision, recall and F1 scores across all the users. The F1 score is a balanced metric combining precision and recall, giving a single score that reflects the model's overall effectiveness. The function is then run on the test dataset, and the results are computed and printed.

The F1 score of 0.4% reveals the overall poor performance in evaluating the precision and recall of the model. A precision of 0.23% and a recall of 0.79% indicate small portions of the recommended movies were relevant and most

of the relevant movies were not captured by the model's output.

B. Surprise

The function *top_n_movies()* provides a personalized list of movie recommendations for a given user based on predicted ratings. An empty dictionary *predict_ratings* is created, to store the predicted ratings of movies that the *user_id* = 3 hasn't rated yet. The function then loops through all the movies in the *df_ratings_1* DataFrame.

For each movie, it checks if the user has rated it using *trainset.knows_item(movieid)*. If the movie hasn't been rated by the user, the function uses the recommendation model to predict the user's rating for that movie, storing the predicted rating in *predict_ratings* with the *movieid* as the key. After predicted ratings are generated for all unrated movies, the function sorts the movies based on their predicted ratings in descending order, selecting the top 5 movies. The final step prints out the titles and predicted ratings for the top 5 recommendations. The top 5 movies for this *user_id*=3 with predicted ratings of 5 for all the movies, include Paperman (2012), Predicted rating: 5, Triplets of Belleville, The (Les triplettes de Belleville) (2003), Predicted rating: 5, Maltese Falcon, The (a.k.a. Dangerous Female) (1931), Predicted rating: 5, Her (2013), Predicted rating: 5, Louis C.K.: Chewed Up (2008), Predicted rating: 5. These are highly recommended movies for the user based on the model's predictions.

In evaluating the surprise library package, two common metrics for measuring accuracy were used, the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). RMSE gives more weight to larger errors by squaring the differences between predicted and actual ratings, whereas MAE averages the absolute differences without emphasising large errors. A lower RMSE or MAE indicates better prediction performance.

After performing a five-fold cross-validation was performed. The model is trained and evaluated on these folds as each fold is used as a test set once, while the rest serves as train data. The model's generalisation ability is assessed by this technique as it is tested across different data splits. Each fold's results are recorded for both RMSE and MAE.

the mean RMSE is approximately 1.1128, which can be interpreted as, on average, the model's predictions are about 1.11 points off the actual ratings. The mean MAE is around 0.8746, indicating the average absolute error between predicted and actual ratings.

The results from this cross-validation display consistent performance across the folds, with small standard deviations for RMSE as 0.0051 and MAE as 0.0044. This consistency in the values for both RMSE and MAE suggests that the model is stable and does not significantly overfit or underfit on different data splits. Also, the train time and test time indicate that training is consistent across folds but can slightly vary depending on data size and computational resources.

C. Cosine similarity

The function *recommend_movies_sparse()* is defined and given a movie title “Kill Bill: Vol.2 (2004)”. The function of the model then created the expected relevant movies for *Kill Bill: Vol.2 (2004)* which were selected based on the genre of the given movie, these selected movies were used as the basis

for assessing the recommendations. These expected relevant movies represent the correct or ideal set of recommendations that the model should suggest. A full list of movies is generated and is arranged in descending order. The top 5 movies among the recommended list are Welcome (2007), Arranged (2007), Frenchmen 2 (2007), Guru (2007), and After Sex (2007). The model's evaluation involved comparing the recommended movies which is y_{pred} to the actual relevant movies y_{true} . In y_{true} , movies are marked as 1 if they are relevant and 0 if they are not, while y_{pred} marks movies as 1 if the model correctly predicted them as relevant and 0 if they are irrelevant. This encoding helps in the calculation of precision, recall, and F1 score to assess the model's accuracy in recommending the correct movies.

The precision, recall and F1 scores are calculated using their appropriate functions, with a precision of 1.0, it means that all the recommended moves by the model are relevant to the user based on the chosen movie. The recall score of 0.652, thus 62.5%, can be interpreted as the proportion of relevant movies retrieved by the model. With an F1 score of 0.789, the precision and recall are well-balanced, indicating a strong overall performance of the model.

V. CONCLUSION

Each model built for the recommender system displayed a unique strength and limitation. The KNN model with more personalised characteristics, by identifying similar users, performed poorly with an F1 score of 0.4%, which suggests a smaller balance between precision and recall. Low figures were observed for precision and recall. Thus, 0.23% and 0.79% indicate a small portion of recommended movies were relevant, and the model failed to capture the relevant movies. The surprise model also iterates on a collaborative filtering approach using RMSE and MAE for evaluation, displaying more consistent performance. Cross-validations came out with a mean RMSE of 1.1128 and a mean MAE of 0.8746, with low standard deviations for both RMSE and MAE as 0.0051 and 0.0044. It is safe to say the model's predictions were relatively close to the actual ratings, and there was stability across different splits. The surprise model from its evaluation neither overfit nor underfit the training data.

Finally, Cosine Similarity gives the best balance between performance and relevance, making it the most suitable model for this recommender system when focusing on content-based recommendations. This model showed the overall best performance in terms of precision, recall and F1 scores. A 1.0 precision score indicates that all the movies retrieved were relevant, while a 0.652 recall suggests that 65.2% of relevant movies retrieved were recommended to the user. F1 score of 78.9% demonstrates a properly balanced model robust for a personalised movie recommendation system.

The KNN model struggled with generalisation, and that makes it less suitable for our personalised recommender system with collaborative filtering. Although the surprise model shows stable and accurate predictions, its primary limitation is the complexity of incorporating user-item interactions for personalised recommendations as compared to models like KNN. The cosine similarity model's dependence on genre-based comparisons may limit its capabilities to predict movies beyond strict genre similarities,

thus it can miss out on cross-genre recommendations of movies.

ACKNOWLEDGMENT

I would like to acknowledge Joel and Ieuan for all their assistance and contributions towards my studies.

REFERENCES

- [1] S. Jayalakshmi, N. Ganesh, R. Čep, and J. S. Murugan, "Movie Recommender Systems: Concepts, Methods, Challenges, and Future Directions," *Sensors*, vol. 22, no. 13, Jul. 2022, doi: 10.3390/s22134904.
- [2] A. Yassine, L. Mohamed, and M. al Achhab, "Intelligent recommender system based on unsupervised machine learning and demographic attributes," *Simulation Modelling Practice and Theory*, vol. 107, Feb. 2021, doi: 10.1016/j.simpat.2020.102198
- [3] F. Ortega, A. Hernando, J. Bobadilla, and J. H. Kang, "Recommending items to group of users using Matrix Factorization based Collaborative Filtering," *Information Sciences*, vol. 345, pp. 313–324, Jun. 2016, doi: 10.1016/j.ins.2016.01.083.
- [4] Y. Wang, M. Wang, and W. Xu, "A Sentiment-Enhanced Hybrid Recommender System for Movie Recommendation: A Big Data Analytics Framework," *Wireless Communications and Mobile Computing*, vol. 2018, 2018, doi: 10.1155/2018/8263704.
- [5] M. J. Pazzani, D. Billsus, "Content-Based Recommendation," *The Adaptive Web Springer Link*, vol. 4321, pp.325–341
- [6] S. Malik, "Movie Recommender System using Machine Learning," *EAI Endorsed Transactions on Creative Technologies*, vol. 9, no. 3, p. e3, Oct. 2022, doi: 10.4108/eetct.v9i3.2712.
- [7] P. Kumar, S. G. Kibriya, Y. Ajay, and Ilampiray, "Retraction: Movie Recommender System Using Machine Learning Algorithms," *Journal of Physics: Conference Series*, vol. 1916, no. 1. IOP Publishing Ltd, May 27, 2021. doi: 10.1088/1742-6596/1916/1/012052.
- [8] V. L. Chetana and H. Seetha, "Enhancing Movie Recommendations: An Ensemble-Based Deep Collaborative Filtering Approach Utilizing AdaMVRGO Optimization," *Traitement du Signal*, vol. 40, no. 6, pp. 2337–2351, Dec. 2023, doi: 10.18280/ts.400602.
- [9] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," In *Proceedings of the 10th international conference on World Wide Web*, pp. 285 – 295, 2001.
- [10] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42(8), pp. 30–37, 2009.
- [11] Wu, C.-S. M., Garg, D., & Bhandary, U. (2018). Movie Recommendation System Using Collaborative Filtering, 23-25 Nov, 2018, IEEE 9th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2018.
- [12] P. Gupta, V. Batra, A. Sharma, D. Narula, and R. Tiwari, "MOVIE RECOMMENDER SYSTEM USING MACHINE LEARNING," 2023. [Online]. Available: <http://www.ijeast.com>
- [13] Z. Wang, X. Yu, N. Feng, Z. Wang, "An improved collaborative movie recommendation system using computational intelligence," *Journal of Visual Languages & Computing*, vol. 25, pp. 667–675, December 2014.
- [14] Y. Zhao, C. Zhou, J. Cao, Y. Zhao, S. Liu, C. Cheng, and X. Li, "Multiscenario combination based on multi-agent reinforcement learning to optimize the advertising recommendation system," 2407.02759, 2024.
- [15] R. Lavanya, E. Gogia, and N. Rai, "Comparison Study on Improved Movie Recommender Systems," *Webology*, vol. 18, no. Special Issue 04, pp. 1470–1478, Dec. 2021, doi: 10.14704/web/v18si04/web18285.
- [16] K. Indira and M. K. Kavithadevi, "Efficient Machine Learning Model for Movie Recommender Systems Using Multi-Cloud Environment," *Mobile Networks and Applications*, vol. 24, no. 6, pp. 1872–1882, Dec. 2019, doi: 10.1007/s11036-019-01387-4.
- [17] V. Paranjape, N. Nihalani, and N. Mishra, "Design and Development of an Efficient Demographic-based Movie Recommender System using Hybrid Machine Learning Techniques," *International Journal of Computers, Communications and Control*, vol. 19, no. 4, 2024, doi: 10.15837/IJCCC.2024.4.5840.

- [18] M. J. Awan *et al.*, “A recommendation engine for predicting movie ratings using a big data approach,” *Electronics (Switzerland)*, vol. 10, no. 10, May 2021, doi: 10.3390/electronics10101215.