NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SC4020 Data Analytics and Mining

Recommendation Systems: A Comprehensive Study

Group 11

Lim Kang Wei

Ng Yong Jian

Oo Yifei

Yeoh Ming Wei

Table of Contents

Abstract

In the rapidly evolving domain of recommender systems, a variety of algorithms have been proposed and tested for efficiency, accuracy, and scalability. In the field of recommendation systems, the three main types are content-based filtering, collaborative filtering and the hybrid approach. This technical review delves deep into the implementation and comparative analysis of three prominent algorithms, which correspond to the three types of recommendation systems: K-Nearest Neighbors (KNN), Matrix Factorization, and Deep Neural Networks (DNN). Using two distinct datasets—MovieLens, centered around movie recommendations, and KKBox Music, focusing on musical content—this study aims to furnish insights into the strengths, weaknesses, and nuances of each algorithm's performance. This review not only offers a systematic implementation of the algorithms but also provides insights and to guide the optimal selection and fine-tuning of algorithms in real-world recommender system applications.

Introduction

Recommendation systems have transformed the way businesses and platforms engage with users by providing tailored suggestions based on various data inputs. The essence of these systems lies in predicting the preference or rating a user would give to an item. They play an indispensable role in platforms like Netflix, Amazon, and Spotify.

In designing advanced recommendation systems, the three major types are collaborative filtering, content-based filtering and hybrid methods, which is a mix of using content-based filtering and collaborative filtering. Collaborative filtering-based recommender systems solely rely on past interactions between users and items in order to provide personalised recommendations. The features of every individual item are not considered.

Unlike collaborative methods that only rely on the user-item interactions, content based approaches use additional information about users and/or items. For a movies recommender system, this additional information can be, the personal information (age, sex) for users and the characteristics (genre, duration) for the movies (items). Then, the idea of content based methods is to try to build a model, based on the available "features", that explain the observed user-item interactions.

A hybrid recommendation system is a special type of recommendation system which can be implemented in various ways like by using content and collaborative-based methods to generate predictions separately and then combining the prediction or we can just add the capabilities of collaborative-based methods to a content-based approach (and vice versa).

For this review, we have selected one key algorithm for each of the three categories, providing a comparative analysis on their performance and applicability in real-world scenarios. The three algorithms are K-Nearest Neighbors (KNN), Matrix Factorization, and Deep Neural Networks (DNN), representing content-based filtering, collaborative filtering and hybrid recommendation systems respectively.

To fully analyse the performance and difference between these algorithms, we will leverage two diverse datasets—MovieLens and KKBox Music. MovieLens, a widely-acknowledged benchmark in movie recommendation research, provides a rich array of user-movie interactions. In contrast, KKBox Music, as one of Asia's leading music streaming platforms, presents a musical dataset with its unique challenges and dynamics.

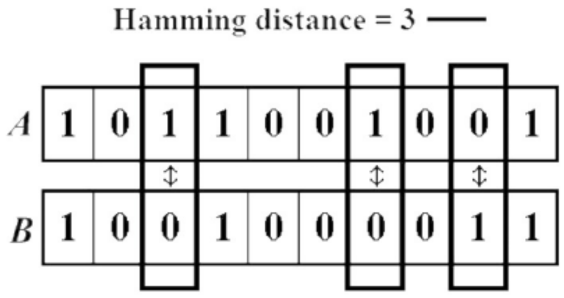
This technical review will chronicle our journey of implementing these three algorithms on the aforementioned datasets, critically analysing their performances, and drawing comparisons to show a clearer picture of their functionalities and applications.

Methods

First, we obtained the movie dataset and cleaned it by filling missing values, inconsistent entries, drop columns and removing duplicates (if necessary). Based on how the dataset is structured, the genre column is separated by ' | ', so we did one hot encoding using 'latin-1'. For content-based recommenders, we removed items that did not belong to any of the genres as it is redundant to have items without its genre labelling; For user behavior-based recommenders, these items are kept because it depends on the user rating on the items to make recommendations. A pivot table will segregate out items that aren't rated by any users already.

Content-based recommender - KNN Clustering

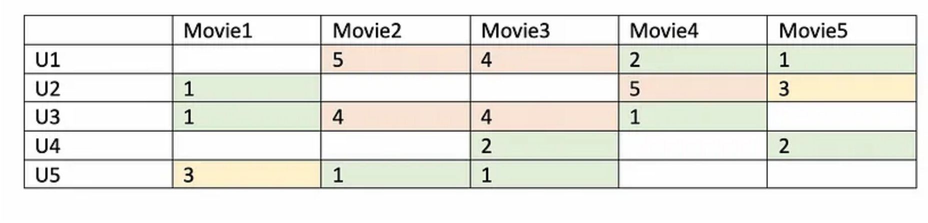
We employed the movie-genre matrix to store information about movies and their genres in matrix. Each row represents a movie, each columns represents a column. The KNN algorithm takes in the movie genres as the parameters, and used Hamming distance as the metric for evaluation. Considering that the movie genres are categorical variables, conventional metrics such as Minkowski distance and Manhattan distance would be less suitable, thus Hamming distance is instead used in our model. The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different. In our case, it represents the number of different genre columns.



Finally, when a user inputs their movie preferences or searches for a specific movie, the KNN algo searches for the K-nearest neighbors and recommends them to the user. Likewise for the KKBox music dataset.

Collaborative filtering-based recommender - Matrix Factorisation

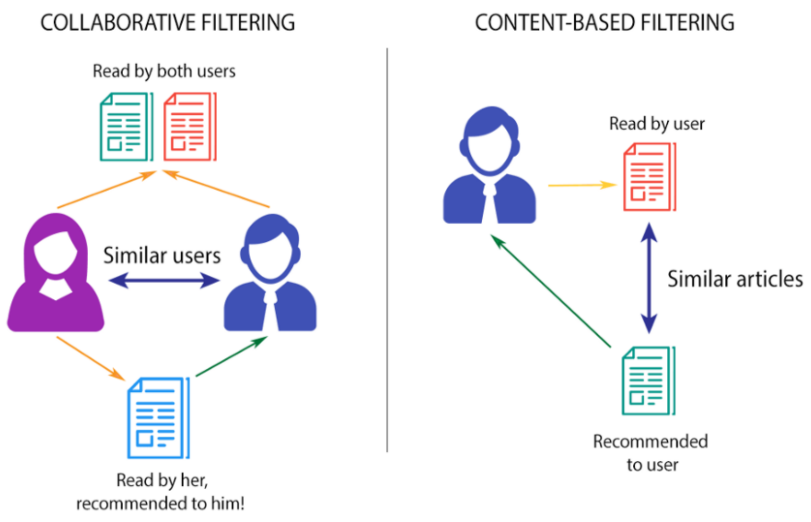
Matrix factorisation is a technique used in collaborative filtering to find the relationship between items and users. It is a way to generate latent features when multiplying two different kinds of entities. Here's a simplified breakdown on how it works:
A user-item rating matrix where each entry represents the rating or feedback that a user gave to an item (movie in this context)



	Movie1	Movie2	Movie3	Movie4	Movie5
U1		5	4	2	1
U2	1			5	3
U3	1	4	4	1	
U4			2		2
U5	3	1	1		

The goal is to find two matrices, one for users and one for items that multiply to approximate the original matrix. The user matrix has one row for each user and one column for each latent factor, while the item matrix has one row for each latent factor and one column for each item. The latent factors are the features or dimensions that represent the characteristics of the items and the preferences of the users. The dot product between a user row and an item column gives the predicted rating for that user and item.

In our case study, we only applied this algorithm to the MovieLens context where we have users rating the movies, but not applied to the KKBox context because we don't have the user-song interaction. It's not possible to identify the underlying patterns and preferences of the users and to estimate how they would rate the songs based on the song's genre alone.



In the real-world, the rating matrix is very sparse since not everyone watches every movies. The missing entries in the rating matrix would be replaced by the dot product of the factor matrices.

Hybrid recommender - Deep Neural Networks

The goal is to choose a suitable item according to the user's profile. To enhance the performance of collaborative filtering (CF) recommender systems, which suffer from low accuracy, linear latent factor, and cold-start problem due to insufficient training data, a deep neural network-based approach is employed. This is a hybrid approach which uses user and item vectors to encapsulate users' and items' data to train on high dimensionality non-linear data to provide more accurate recommendations.

The DNN model is built and trained with two key sources of input:

User Data: It understands users' preferences and behaviour by embedding each user into a continuous vector representation.
Item Data: It similarly transforms items into vector representations, considering characteristics like genre and other features.

To ensure accurate recommendations, the model is trained using a dataset that contains user interactions with the items, such as user ratings.

For the MovieLens Dataset, the training process aims to minimise the mean square error in predicting user ratings for movies, then making recommendations based on the predicted ratings. On the other hand, for the KKBox Music Dataset, the training process aims to minimise the binary classification error in predicting if there are recurring listening event(s) triggered within a month after the user’s very first observable listening event.

By inputting a user's ID and a list of items, the model predicts how much the user might like each movie or song, then makes recommendations to the user based on these predictions. For example, in the case of recommending movies to users, the model would first predict the ratings on a list of movies that would likely be given by a user. Then, to recommend 10 movies to the users, we would select the top 10 movies with the highest rating predicted.

Experiments

Strengths and weaknesses of the algorithms used for recommendation systems

KNN Content-based Clustering

The strength of KNN Content-based clustering is that it is **simple to implement** and its concept is rather easy to understand compared to the other content-based clustering methods. This can be seen through its algorithm where there is **no explicit training step** and all the work is done during prediction. This helps with scalability as well, since whenever you add in new data, there is no need to train a new model. In addition, the KNN content-based clustering algorithm has only a single hyperparameter, *k*. This makes **hyperparameter tuning easy and straightforward**.

However, since the entire training data is processed for every prediction, it has **high computation cost** and requires **high memory storage** as it stores all the training data. The time complexity for each prediction is **O(MNlog(k))** where M is the dimension of the data, N is the size or the number of instances in the training data. Another weakness for this algorithm would be that it can be **challenging to choose the optimal number of neighbours**, k. Especially in our case where we do not have a ground truth to check upon (unsupervised learning). KNN Content-based clustering is **sensitive to outliers and irrelevant features** as well as it depends on the relative distance between the data points. The issue may be escalated in higher dimension datasets, which people call the "curse of dimensionality" due to the overwhelming number of features. Besides that, KNN content-based clustering **assumes equal importance to all features** as KNN expects points to be close in all dimensions. Fortunately, this issue can be adjusted by choosing an appropriate distance measure and by appropriate preprocessing to scale feature importance.

Matrix Factorization

Matrix Factorization is a popular method in collaborative filtering for recommendation systems. It is very **good at handling sparse and incomplete data**. The algorithm of Matrix Factorization is capable of filling in missing values and predicting ratings for unseen items or users. The missing values are estimated based on patterns and similarities in the data by using non-negative matrix (NMF) on a sparse matrix(from scipy.sparse import csr_matrix) or by doing imputation using KNN to fit transform the sparse matrix (from fancyimpute import KNN). In addition, Matrix factorization is good at **reducing dimensionality and complexity** of the data. It can compress a large matrix into smaller matrices that capture essential information and reduce noise, improving the efficiency and scalability of the recommender system. This can be achieved by using PCA from the sklearn decomposition library: This projects a matrix to a lower-dimensional space that captures the most important information. The most important information in this context refers to the information that explains the most variance in the data and is captured by the principal components (represented in eigenvectors with highest eigenvalues which shows the directions in which the data varies the most). Matrix factorization can **effectively discover latent features and patterns** as well. Matrix factorization can reveal hidden similarities and preferences among users and items, enhancing the quality and diversity of recommendations. Latent features can represent various aspects of the data such as user preferences and item attributes. It can capture complex relationships and patterns that aren't apparent from the raw data.

As much as matrix factorization is useful, it has its own drawbacks. Matrix factorization can suffer from the **cold start** problem, which occurs when there is not enough data or information about new users (User cold start) or items (Item cold start) to make accurate recommendations. To address user cold start, Functional Matrix Factorization where Hybrid Matrix Factorization-Decision Tree Recommender System can be used while Collective Factorization can be used to link user ratings to item text to address item cold start. Matrix Factorization also suffers from limited interpretability. Matrix Factorization can be **difficult to interpret and explain**, especially when dealing with high-dimensional data or complex models. They rely on latent features and patterns that are not directly observable and can be difficult to explain.

Deep Neural Network (DNN)

DNN are very **efficient in handling large amounts and complex data**. In the context of recommendation systems, this capacity ensures that DNNs can sift through extensive user-item interactions, making them a viable choice for large-scale platforms. One of the significant strengths of DNNs lies in their ability to **model non-linear relationships in data**. By utilising non-linear activation functions such as ReLU, Sigmoid, and Tanh, DNNs can capture interactions that traditional linear models might miss. This characteristic is especially vital in recommendation systems, where user-item interactions often have intricate, non-linear patterns that can influence recommendations. Deep learning architectures like DNN, with their multiple layers, are able to effectively **capture complex patterns and dependencies**. For example, in hybrid recommendation systems, a DNN can discern intricate dependencies between various features, ensuring more personalised and accurate recommendations. Another notable advantage of DNNs is their inherent **robustness to missing and noisy data**. Due to their architecture and training methods, DNNs can still make accurate predictions even when confronted with missing or noisy data points. This robustness is crucial for recommendation systems, where user behaviour can be erratic and data may often be incomplete or ambiguous.

However, DNN also has its own flaws. Training Deep Neural Networks, particularly for large-scale datasets, necessitates **significant computational resources**. It's a well-documented fact that DNNs, especially deep architectures, often require high-performance GPUs for efficient and timely training. Beyond training, deploying and maintaining such models in real-time recommendation systems can be exceedingly resource-intensive, potentially leading to increased operational costs and complexities. One of the most frequently cited drawbacks of DNNs is their **"black-box" nature**, wherein the intricate interplay of weights and activations makes it difficult to interpret the model's decision-making process. In the realm of recommendation systems, this opaqueness can be problematic. Understanding the rationale behind why a certain item was recommended is pivotal not only for fostering user trust but also for gleaning valuable business insights. Without this understanding, making iterative improvements or justifying recommendations becomes a challenge. Another known limitation of DNNs in the context of recommendation is their performance in **"cold-start" scenarios**. When introduced to new users or items with limited data, DNNs can struggle to generate accurate recommendations. This is because these networks heavily rely on existing data to form patterns, and in the absence of sufficient data for new users or items, their performance can be suboptimal. Like many machine learning models, DNNs are **susceptible to issues like overfitting and underfitting**. Overfitting is where the model performs exceptionally well on the training data but poorly on unseen data. On the other hand, underfitting occurs when the model fails to capture the underlying patterns in both training and test datasets. Regularization techniques and hyperparameter tuning are often needed to mitigate these issues, adding another layer of complexity to model training and validation.

Parameter setting

KNN Content-based Clustering

The parameters for a KNN model are simple and straightforward. For the first parameter n_neighbors, which is the number of neighbors to consider in the KNN algorithm. It determines how many similar items to retrieve in the clustering process. This parameter is chosen based on the goal of the recommender system, smaller ‘n_neighbors’ values (e.g., 5 or 10) results in highly personalised recommendations. When ‘n_neighbors’ is small, the system focuses on the very nearest neighbors, leading to recommendations that closely match a user's specific tastes. In addition, smaller ‘n_neighbors’ values are more appropriate in relatively small datasets. On the other hand, larger ‘n_neighbors’ values (e.g., 50 or 100) considers a broader range of items, which can help in suggesting a more varied set of items to users. Such recommendations are more stable and less sensitive to outliers or rare items. It's suitable for scenarios where the user base is diverse and preferences vary widely. In our specific case, tuning recommendations based on changing 'K' leads us to same recommendations. Our reasoning is because the input of the model is the one hot encoded features, which are an array of binaries. There're a lot of similar movies/songs that share the same 3 sets of 'ones' in other words same genre. So, even if we change the number of 'K', they will still refer to the same items. Because the items can only be so different in the sense of switching little binary values. For the metric to be used, we have chosen Hamming distance, instead of the conventional choice of Minkowski distance or Manhattan distance, since in our dataset the variables to be considered are the genres, artists, composer, etc. (based on the dataset, MovieLens or KKBox), and they are categorical variables. Thus, Hamming distance would be the most appropriate metric in our case. For the algorithm, we have chosen ‘auto’, which lets the KNN model itself to determine the best approach between ‘ball_tree’, ‘kd_tree’ and ‘brute’.

Matrix Factorization

Number of Latent Factors: The number of latent factors is a hyperparameter that determines the dimensionality of the latent space and the expressiveness of the model. A higher number of latent factors can capture more complex relationships and patterns in the data but can also lead to overfitting and increased computation time. It's a balance: too few factors might lead to underfitting, while too many can lead to overfitting and longer computation times. Selecting the right number usually requires cross-validation. In our technical review, this is represented by various aspects of the movies and users such as genre, rating, popularity, or actors are not apparent from the raw data. Which are obtained from 'W' and 'H' matrices in NMF.

Objective Function: The objective function is a measure of how well the model fits the data and can be chosen based on the specific problem and data. The choice of objective function determines how the discrepancy between the predicted and actual ratings is measured. Common objective functions include squared error and cross-entropy.

Initialization: Initialization can significantly influence the performance and convergence of Matrix Factorization and performance of the algorithm. Random initialization can lead to different local minima and affect the stability of the algorithm, while deterministic initialization can improve the convergence and reproducibility of the results. The init parameter can be set to 'random' for random initialization or 'nndsvd' for deterministic initialization using the non-negative double singular value decomposition method.

Optimization Algorithm: The optimization algorithm iteratively updates the factor matrices to minimize the objective function. Among the ones you mentioned, Alternating Least Squares (ALS) is particularly popular for Matrix Factorization due to its efficiency, especially when parallelized. However, Stochastic Gradient Descent (SGD) offers more flexibility and can sometimes achieve better results, though at the cost of more fine-tuning.

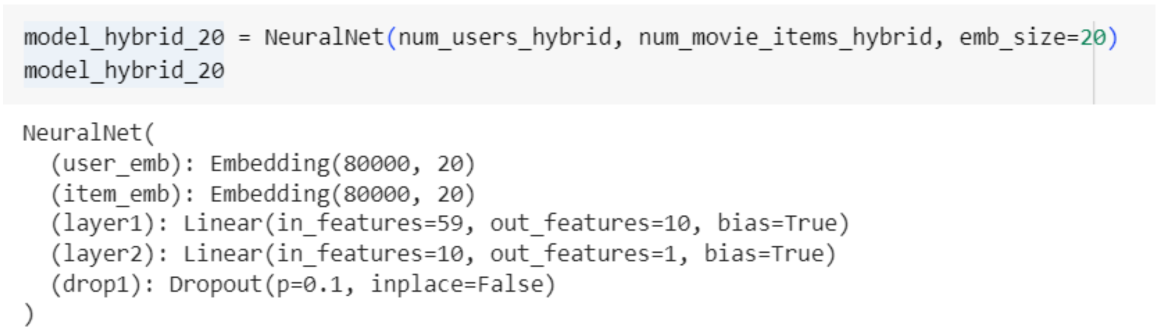
Deep Neural Network (DNN)

The parameters for the DNN algorithm mainly consist of the hyperparameters of the neural network. Firstly, the number of epochs. This is defined as the number of times the entire training dataset is passed through during the training of a neural network. Generally, training for more epochs can help the model learn and fit the training set better. However, too many epochs can lead to overfitting. In this study, we set the number of epochs to 20. The second hyperparameter is the learning rate. This is a hyperparameter that controls the size of the steps the optimization algorithm takes during training. A higher learning rate allows the model to converge faster but risks overshooting the optimal weights. A lower learning rate leads to slower convergence but may help find a more accurate and stable model. In some applications, decaying learning rate is used as it offers the advantages of both high and low learning rates. Initially, a high learning rate is utilised for fast convergence and is then reduced to ensure stability and to prevent overshooting. The learning rate is set to 0.001 in this study. The third hyperparameter is the embedding size. It refers to the dimension of the embedding vectors. Embeddings are vector representations of discrete variables that can reduce the dimensionality of categorical variables, which enables the model to capture meaningful relationships and patterns within the categorical data. This is particularly useful in recommendation systems as it allows the system to find nearest neighbours in the embedding space, making it efficient to generate recommendations based on user preferences or cluster categories. In this study, we experimented with embedding sizes of 20, 30, 40 and 50 to determine which size obtains the optimal results. The model with the optimal embedding size is then used to make the recommendations for users. For both of the datasets, the optimal embedding size is 50 as it yields the least test error.

The architecture of the model are as follows:

The model starts with two embedding layers. The first embedding layer embeds user IDs into continuous vectors of a specific size when the model is initialised. It captures user preferences and characteristics. The next embedding layer embeds item IDs into vectors of the same dimension as the first one. It represents the characteristics of items, which may include genre, language, and more based on the dataset. Next, are two fully connected (linear) hidden layers. The first hidden layer processes the concatenated input, which includes user embeddings, item embeddings, and the characteristics of the items. The second hidden layer further processes the output from the first hidden layer. The number of neurons in the hidden layers are set to 10. To prevent overfitting, we added a dropout to the first hidden layer with a dropout rate of 0.1. For the MovieLens Dataset, the model outputs the predicted ratings. For the KKBox Music Dataset, the model outputs the likelihood the user will listen to the song again within a month after the user's very first observable listening event.

The figure below shows an example of the model architecture with embedding size set to 20.

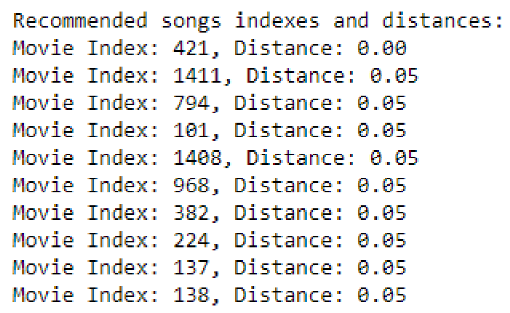
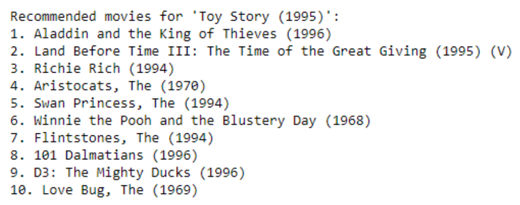
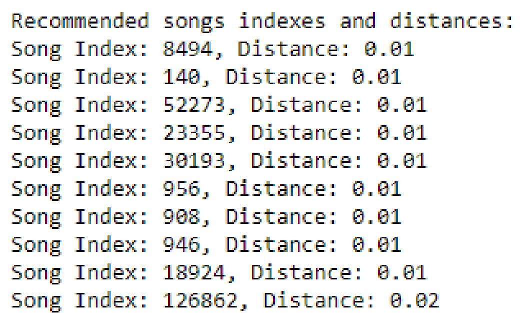
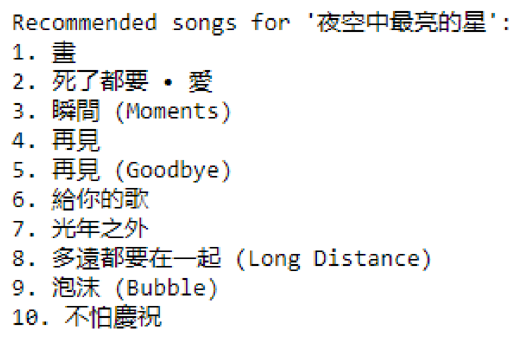


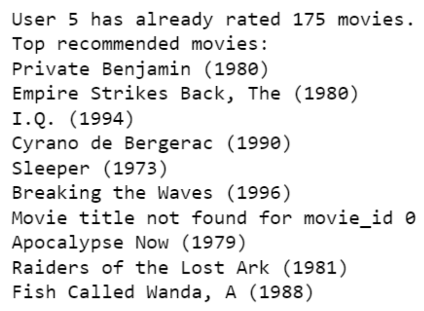
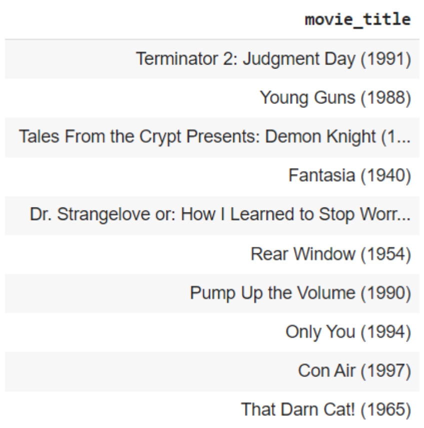
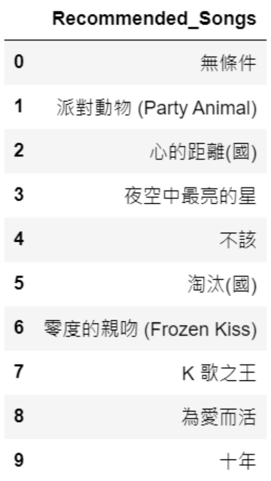
```
model_hybrid_20 = NeuralNet(num_users_hybrid, num_movie_items_hybrid, emb_size=20)
model_hybrid_20
```

```
NeuralNet(
  (user_emb): Embedding(80000, 20)
  (item_emb): Embedding(80000, 20)
  (layer1): Linear(in_features=59, out_features=10, bias=True)
  (layer2): Linear(in_features=10, out_features=1, bias=True)
  (drop1): Dropout(p=0.1, inplace=False)
)
```

Compiled findings

Methodology	MovieLens	KKBox

KNN
Content-based
Clustering
	For our case, we used 'Toy Story (1995)' as our input, using an item-based clustering approach we got the following result.
	
	`Recommended songs indexes and distances:`
	`Movie Index: 421, Distance: 0.00`
	`Movie Index: 1411, Distance: 0.05`
	`Movie Index: 794, Distance: 0.05`
	`Movie Index: 101, Distance: 0.05`
	`Movie Index: 1408, Distance: 0.05`
	`Movie Index: 968, Distance: 0.05`
	`Movie Index: 382, Distance: 0.05`
	`Movie Index: 224, Distance: 0.05`
	`Movie Index: 137, Distance: 0.05`
	`Movie Index: 138, Distance: 0.05`
	
	`Recommended movies for 'Toy Story (1995)':`
	`1. Aladdin and the King of Thieves (1996)`
	`2. Land Before Time III: The Time of the Great Giving (1995) (V)`
	`3. Richie Rich (1994)`
	`4. Aristocats, The (1970)`
	`5. Swan Princess, The (1994)`
	`6. Winnie the Pooh and the Blustery Day (1968)`
	`7. Flintstones, The (1994)`
	`8. 101 Dalmatians (1996)`
	`9. D3: The Mighty Ducks (1996)`
	`10. Love Bug, The (1969)`
	These outputs are based on items that share similar genre. If a user likes Toy Story, then he/she is most probably going to like the above recommended movies because of shared attributes. In this sense, they all fall under 'Animation', 'Children's', and 'Comedy'.
		For our case, we used '夜空中最亮的星 ' as our input. The parameters considered are the genres, artist, composer, lyricist, and language. The result is as follows:
		
		`Recommended songs indexes and distances:`
		`Song Index: 8494, Distance: 0.01`
		`Song Index: 140, Distance: 0.01`
		`Song Index: 52273, Distance: 0.01`
		`Song Index: 23355, Distance: 0.01`
		`Song Index: 30193, Distance: 0.01`
		`Song Index: 956, Distance: 0.01`
		`Song Index: 908, Distance: 0.01`
		`Song Index: 946, Distance: 0.01`
		`Song Index: 18924, Distance: 0.01`
		`Song Index: 126862, Distance: 0.02`
		
		`Recommended songs for '夜空中最亮的星':`
		`1. 畫`
		`2. 死了都要 • 愛`
		`3. 瞬間 (Moments)`
		`4. 再見`
		`5. 再見 (Goodbye)`
		`6. 給你的歌`
		`7. 光年之外`
		`8. 多遠都要在一起 (Long Distance)`
		`9. 泡沫 (Bubble)`
		`10. 不怕慶祝`
		These outputs are based on items that share similar attributes as listed above. If a user likes song 夜空中最亮的星, he/she potentially will like the above recommended songs too. In our case here, most of them are of language 3, artist 40, and genre 458.

Matrix Factorisation (Collaborative Filtering)	After hyperparameter tuning (for n_components, beta loss, solver and initialisation), the following are the top recommended movies based on other similar user's preferences.  User 5 has already rated 175 movies. Top recommended movies: Private Benjamin (1980) Empire Strikes Back, The (1980) I.Q. (1994) Cyrano de Bergerac (1990) Sleeper (1973) Breaking the Waves (1996) Movie title not found for movie_id 0 Apocalypse Now (1979) Raiders of the Lost Ark (1981) Fish Called Wanda, A (1988) However, we are not able to generate the IDs of the users that are behaving similarly as User 5.	Unable to proceed due to absence of user's preference. Without user's rating on the songs, technically we are not able to recommend to a prompt user. The way Matrix Factorisation works is it depends on users that behave similarly to the prompt user (in this case, it's based on how similar other users rate songs as compared to the prompt user). Without this data, we can't compare and recommend.
Deep Neural Network	The model was trained with inputs such as user ID, movie ID and genre (one-hot encoded). Next, we deploy it to recommend the following movies to User 5.  movie_title Terminator 2: Judgment Day (1991) Young Guns (1988) Tales From the Crypt Presents: Demon Knight (1... Fantasia (1940) Dr. Strangelove or: How I Learned to Stop Worr... Rear Window (1954) Pump Up the Volume (1990) Only You (1994) Con Air (1997) That Darn Cat! (1965)	The model was trained with inputs such as user ID (msno), song ID, genre (one-hot encoded) and song features like artist and language. Next, we deploy it to recommend the following songs to User 5.  Recommended_Songs 0 無條件 1 派對動物 (Party Animal) 2 心的距離(國) 3 夜空中最亮的星 4 不該 5 淘汰(國) 6 零度的親吻 (Frozen Kiss) 7 K 歌之王 8 為愛而活 9 十年

Conclusion

Since movie and song recommendations are subjective and we do not know who the users are, there is no ground truth for us to evaluate the performance for our recommendation systems. Nevertheless, it is possible to compare the differences in recommendation provided by the algorithm. However, the use case for KNN Content-Based Clustering is different from that of Matrix Factorization (MF) and DNN. The application for KNN Content-Based Clustering is when the **user searches for a movie name**, then the KNN algorithm will provide the user with a few recommendations, while that for Matrix Factorization Collaborative Filtering and DNN Hybrid Recommendation System is the algorithm will **provide recommendations for the user without any prompt**, one instance would be some recommended movies in the recommendation page. Hence, we can only compare the differences in MF and DNN. In addition, since MF is unable to be used on the KKBox Music dataset, we will only compare the differences in the **movie recommendations** given by the Matrix Factorization and DNN algorithm.

From the compiled findings above, we can see that the movie recommended by the MF and DNN algorithm differs quite a lot, in fact, there is no similar movie recommended by both of the algorithm. MF is based on user ratings, while DNN relies on a different set of inputs. With different inputs, we get different outcomes.

References

A step-by-step explanation of principal component analysis (PCA) (no date) *Built In*. Available at: https://builtin.com/data-science/step-step-explanation-principal-component-analysis (Accessed: 15 October 2023).

Advantages and disadvantages of Deep Learning (2023) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-deep-learning/ (Accessed: 15 October 2023).

Garza, J. (2020) *Why use matrix factorization?*, *Jair G Website*. Available at: https://jairgs.github.io/matrix-factorization/ (Accessed: 15 October 2023).

Joby, A. (2023) *K nearest neighbor or KNN algorithm and it's essence in ML*, *Learn Hub*. Available at: https://learn.g2.com/k-nearest-neighbor (Accessed: 15 October 2023).

Koehrsen, W. (2018) *Neural network embeddings explained*, *Medium*. Available at: https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526 (Accessed: 15 October 2023).

Le, J. (2020) *Recommendation system series part 2: The 10 categories of deep recommendation systems that...*, *Medium*. Available at: https://towardsdatascience.com/recommendation-system-series-part-2-the-10-categories-of-deep-recommendation-systems-that-189d60287b58 (Accessed: 15 October 2023).

MLNerds (2020) *How does KNN algorithm work ? what are the advantages and disadvantages of Knn ?*, *Machine Learning Interviews*. Available at: https://machinelearninginterview.com/topics/machine-learning/how-does-knn-algorithm-work-what-are-the-advantages-and-disadvantages-of-knn/ (Accessed: 15 October 2023).

Rocca, B. (2019, June 12). Introduction to recommender systems. https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada

Science, D. (no date) *What are some advantages and disadvantages of using matrix factorization for recommender systems?*, *Matrix Factorization for Recommender Systems: Pros and Cons*. Available at: https://www.linkedin.com/advice/0/what-some-advantages-disadvantages-using-matrix-factorization (Accessed: 15 October 2023).

Sciforce (2023) *Deep Learning based Recommender Systems*, *Medium*. Available at: https://medium.com/sciforce/deep-learning-based-recommender-systems-b61a5ddd5456 (Accessed: 15 October 2023).

Turing. (2022, August 5). *A guide to content-based filtering in Recommender Systems*. A Guide to Content-based Filtering in Recommender Systems. https://www.turing.com/kb/content-based-filtering-in-recommender-systems

(No date) *Matrix factorization recommender systems and cold start*. Available at: https://sw.cs.wwu.edu/~tuora/aarontuor/materials/presentations/poster5.pdf (Accessed: 15 October 2023).