MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

A LEARNING-BASED MOVIE RECOMMENDATION SYSTEM

December 11, 2022

Haotian Zhang
Yuanpei College

Runbo Zhang
Yuanpei College

Shangcheng Zhao
School of Psychological and Cognitive Sciences

Yelin Bao
School of Psychological and Cognitive Sciences

Contents

1 Introduction

1.1 Why Are Recommendation Systems So Important?

The revolution in the entertainment industry accelerates the increase of entertainment resources. With the great number of entertainment products flushing in, people can be overwhelmed when choosing from various options. Recommendation systems, which are collections of algorithms used to recommend items to users based on information taken from users, are developed to overcome infobesity. These systems are commonly used by online retailers, streaming services, and social media platforms to improve the user experience and increase engagement while increasing business sales as well.

1.2 Popular Recommendation Algorithms

There are three main classes of algorithms used in recommendation systems: content-based filtering, collaborative filtering, and a hybrid solution of these two[2].

Content-based filtering methods analyze a set of features of items relevant to the user and learn the user's preference[1]. When recommending new items to the user, the system basically matches up the preference learned from previous features to new items. The advantages of content-based filtering are obvious: it can directly represent users' personal preferences, and thus features good interpretability and personalization. Content-based filtering also excels at solving the cold-start problem, a problem that occurs for new items with insufficient behavioral data. However, this approach can also be problematic. First, it doesn't take the global evaluation of items into account, which results in recommending low-quality items to the users. Second, extracting features and learning users' preferences can be time-consuming.

In contrast to content-based filtering, collaborative filtering is user-based. This kind of method first collect the ratings or preference given by different users and then suggest items to users based on their similar tastes and interests. With consideration of all users' ratings or preferences, collaborative filtering can guarantee the quality of recommended items. However, it also has some other disadvantages. First, there might be insufficient users to approximate. More importantly, it has to overcome the cold-start problem.

1.3 Project Goal

In this project, we aimed to build movie recommendation systems with the application of traditional machine learning methods we learned from previous lectures as well as DL

algorithms involving DNN and VAE. These algorithms cover all types of methods mentioned above with distinct levels of complexity and different capabilities to handle different sizes of datasets.

In addition to exploiting recommendation algorithms, we also expect our models to be used directly. Therefore, we write a small demo with a decent UI to recommend movies by calling our DNN model. With that, users can get a sense of our model.

2 Using Traditional ML Methods to Recommend Movies

2.1 Dataset and Python Library

In this session, we used the publicly available data from MovieLens-100K Dataset[1]. In the MovieLens dataset, there are two files containing information about movies and users. The users' file consists of 100,000 ratings (1-5) from 943 users on 1682 movies, while the movies' file consists of the title and genres of different movies. These two files have been preprocessed and manipulated in order to build our systems.

We also do a detailed analysis of MovieLens-100K dataset in *analysis4ml-100k.ipynb* file. Please refer to it.

With respect to implementation, in this session, we mainly use a Python library called Surprise. Surprise (stands for Simple Python Recommendation System Engine) is a Python library for building and analyzing recommendation systems that deal with explicit rating data. It provides various ready-to-use prediction algorithms ranging from baseline algorithms, neighborhood methods, and matrix factorization-based methods.

2.2 Preprocessing

Rating distribution

First, we described the rating distribution to make sure the dataset is suitable for modeling. Results showed that the rating scores ranged from 0.5 to 5, mostly concentrated from 3 to 5. The distribution of rating looks a little negatively skewed, but still suitable for modeling. Next, we examined the rating distribution by users. Results showed that each user evaluated at least 20 movies. Most users evaluated about 20 to 100 movies. The largest number of rating records per user is 2698.

Build dictionaries and extra functions

Before modeling, we did some preparation including creating lists that can connect movie ID and movie title from the movie dataset and getting the rank of each movie from the rating dataset. Correspondingly, we wrote functions to extract users' ratings and movie IDs based on these lists.

2.3 Evaluation Approaches

Proper evaluation of RS is also an important topic. In addition to adopting the traditional model performance indicators such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), we also sought to take other performance indicators into account. We calculated the HitRate and cHitRate for each model. After recommending the top-N items to the users, the number of items in the intersection of recommend set and actually rated set is defined as the number of hits. The Hit Rate is then calculated by dividing the number of hits by the number of users. The cumulative Hit Rate further adds a threshold for the test set, indicating that it just considers the movies a user really likes. We also evaluated models by Average Reciprocal Hit Rank (ARHR). Similar to cHitRate, this evaluation method takes the quality of hits into account, which is represented by the rankings of hits in the rating dataset. The evaluation process followed the steps below: First, MAE and RMSE were calculated by comparing the actual rating and the predicted rating. Next, to calculate the rest performance indicators, we first ran the top-N recommendation on the validation set created by the leave-on-out method. Then, using the approaches depicted above, we calculated HitRate, cHitRate, and ARHR, respectively.

2.4 Content-Based Models

The Rocchio algorithm is a commonly used relevance feedback algorithm in Information Retrieval which helps refine queries[6]. While building recommender systems, the Rocchio algorithm can help with learning and updating user profiles for vectorial representation. Its basic idea is creating a new vector, called the center of mass, by averaging the items of documents (i.e., movie genres) in each category (i.e., defined by ratings). When recommending new items, the algorithm compares how much the new document resembles the center of mass to decide whether it belongs to the category. Here, we used TF-IDF for document transformation. The recommendation result of a representative user was shown.

2.5 Collaborative filtering models

Item-based KNN

The item-based k-nearest neighbors (KNN) algorithm relies on item feature similarity. When a KNN makes a prediction about an item (say, a movie), it will calculate the 'distance' between the target item and every other item in its database. It then ranks the distances and returns the top 'k' nearest neighbor movies as the most similar movie recommendations. In

this paper, we used cosine similarity as a measurement of the distance between items.

User-based KNN

The user-based k-nearest neighbors (KNN) algorithm relies on the similarity between different users. When a KNN makes a prediction about a user's preference, it will calculate the 'distance' between the target user and every other user in its database. It then ranks the distances and returns the top 'k' nearest neighbor users' preference as the recommendations.

SVD

The Singular Value Decomposition (SVD) is generally used as a dimensionality reduction technique in machine learning. SVD is a matrix factorization technique, which reduces the number of features of a dataset by reducing the space dimension from N-dimension to K-dimension (where K<N). In the context of the recommender system, SVD starts with a utility matrix (users × items, A) with rating scores as elements. With the implementation of singular value decomposition, the initial utility matrix is decomposed into a latent factor matrix(U), a matrix describing the strength of each latent factor(S), and a matrix representing the similarity between items and latent factors(V). The SVD decreases the dimension of the utility matrix A by extracting its latent factors. It maps each user and each item into a low-dimensional latent space. This mapping facilitates a clear representation of relationships between users and items. Based on the extracted latent factors, the recommender system will show a user a few other movies that are similar to the one chosen previously. Of note, the similarity is measured by cosine distance.

NMF

The non-negative matrix factorization (NMF) algorithm is used in a vast number of fields including image processing, text mining, clustering, and collaborative filtering. Its most significant assets are speed, ease of interpretation, and versatility. Similar to SVD, NMF relies on matrix factorization. An initial preference matrix (items × users, V) is approximated by the production of two different matrices, which represent the relationship between latent factors and items(M) or users(H), respectively. All the values in the matrices are equal to or greater than zero. The greater the weight, the more "determined" the column (segment) is by the variable in the row. When recommending movies to a user, the algorithm finds attraction weight towards the certain movie in columns of the matrix. After sorting the values in descending order, movies are proposed to the users to match their preferences.

2.6 Conclusion

Algorithm	RMSE	MAE	HitRate	cHitRate	ARHR	Time Efficiency
Rocchio	2.9250	2.7778	0.0000	0.0000	0.0000	/
Item KNN	0.9788	0.7610	0.0000	0.0000	0.0000	11min
User KNN	0.9787	0.7547	0.0000	0.0000	0.0000	3min17s
SVD	0.8779	0.6732	0.0361	0.0424	0.0133	3min45s
NMF	0.9297	0.7128	0.0000	0.0000	0.0000	4min8s
SVD tuned	0.8775	0.6734	0.0311	0.0363	0.0068	4min21s

Table 1: Comparison results among traditional ML methods

Algorithm	Recommendations	Algorithm	Recommendations
User KNN	Galaxy of Terror (Quest) Android Alien Contamination I'm the One That I Want Elling Wonder Lesson Faust American: The Bill Hicks Story Dylan Moran: Monster Bill Hicks: Revelations	Item KNN	Galaxy of Terror (Quest) Looker Android Alien Contamination Master of the Flying Guillotine Eye for an Eye Amazing Panda Adventure, The Three Wishes Cure Gordy
SVD	American Beauty High Noon Good, the Bad and the Ugly Eternal Sunshine of the Spotless Mind Godfather Do the Right Thing Chinatown Singin' in the Rain Usual Suspects American History X	NMF	Rebel Without a Cause Before Night Falls Stalker Big Sleep Farewell My Concubine Jason's Lyric Reign Over Me Persona Holy Mountain All Watched Over by Machines of Loving Grace
Rocchio	King Kong Babes in Toyland Passage to India Australia Red River Kung Fu Panda: Secrets of the Masters Beauty and the Beast Labyrinth Aguirre: The Wrath of God National Velvet		

Table 2: Recommendation for userID 85

Results (Table 1) showed that collaborative filtering models outperformed the content-based model. SVD showed well performance while predicting user ratings. More importantly,

it had the highest HitRate, cHitRate, and ARHR, suggesting that it excelled in solving the top-N recommendation problem. The top 10 movies recommended to user 85 were shown in Table 2. Movies recommended by User KNN and Item KNN algorithms overlap slightly, while other recommenders have various outputs.

3 Using DNN to Train a Recommendation System

In this session, we aim to adapt DNN models to our recommendation system. We still use the dataset Movie-Lens 100k here.

3.1 Preprocessing

This dataset can be divided into 3 parts: User, Movie, and Rating.

1. *User:* Identity information about users. It contains 'UserID', 'Age', 'Gender', 'Occupation', and 'Zipcode'.

2. *Movie:* Detailed information about rated movies. It contains 'ID', 'Title', 'Release_date', 'Video_release_date', 'IMDb_URL', and bool values of genres (including 'Unknown', 'Action', 'Adventure', 'Animation', 'Children's', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', and 'Western').

3. *Rating:* Rating information made by users. It contains 'UserID', 'MovieID', 'Rating', and 'Timestamp'.

The detailed preprocessing steps are as follows:

Step 1: Remove features that are irrelevant to training. We delete 'Zipcode' for *Users*, 'Release_date', 'Video_release_date', 'IMDb_URL' for *Movies*, and 'Timestamp' for *Ratings*.

Step 2: Digitize features. For *Users*, the 'Age' attribute, ranging from 0 to 100, is divided into 7 classes each with a corresponding number attached to it. Besides, 'Gender', which takes value from 'Female' and 'Male', is translated to 0 or 1, while 21 unique numbers are assigned to different occupations. For *Movies*, we transferred 'Title' into text vectors with a length of 15. However, in practice, we discover that the last 7 dimensions of text vectors are identical for nearly 95% data. Thus, We compressed the length to 8.

Step 3: Feature enhancement. Based on 100,000 Rating entries, for each user, we calculate the rating numbers with respect to each genre, which approximately values the user's preference. We label the top 3 favorite genres for each user by 1 and assign 0 to the other 16 less preferred genres. In this way, we get genre preference vectors for each user. While for each movie, we calculate the average rating among users, which serves as an important metric of the movie's quality.

After preprocessing, we receive 100,000 entries of data. Each entry includes the user id, user age, user gender, user occupation, user preference, movie id, movie genre movie title text feature vector, and movie average score as long as the rating score.

3.2 Network Architecture and Results

Given the preprocessed data, we implement our model using DNN method.

Preliminary model

The model implements a fully connected neural network (as shown in Figure 1), containing 4 hidden layers each with neurons at the number of 128, 256, 128, and 64. ReLU activation function is used after each hidden layer, while dropout is adopted to reduce overfitting. In the training process we select 85% of the data for training, 15% of the data for testing, define Mean Square Error (MSE) as the error loss function and select Adaptive Moment Estimation (Adam) as the optimizer. We trained 50 epochs under a batch size of 256 to learn the parameters. The validation loss of this model is about 1.1, but the test loss is as high as 1.62.

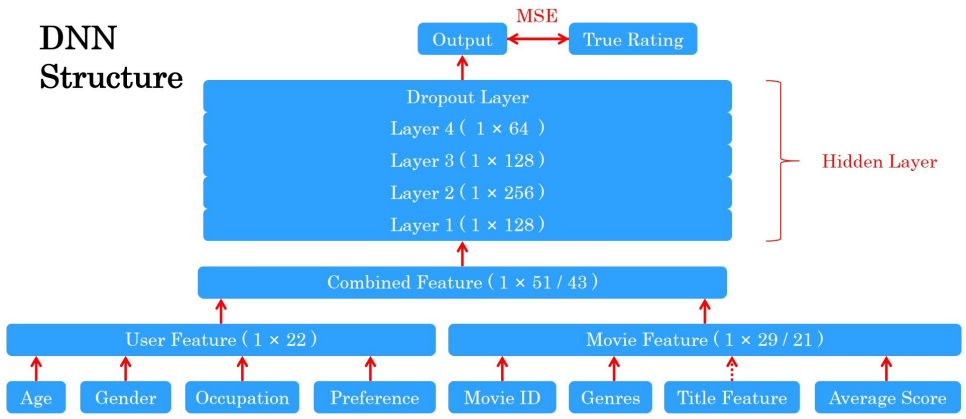


Figure 1: Architecture of DNN recommendation model

Principle of recommending

Given the user's age, gender, occupation, and preferences, we construct a user feature vector. After that, it is combined with different movie feature vectors, while different prediction scores are obtained through the network model. Movies corresponding to combinations with the highest K (can be defined by the user) predicted score is selected as recommended ones.

Drawbacks

We found that in the actual recommendation process, for different user characteristics, the content recommended each time is almost the same. This does not meet the original intention of our recommendation that recommendations should vary from person to person. After locating the source of the problem, we found that the text feature vector of the movie titles is a very confusing thing, not only has relatively large data (nearly a thousand times the one-hot vector), but also has no strong correlation with the movie rating. At the same time, the data of the movie number is relatively large. So we make following improvements.

Final model

We keep the original framework unchanged and modify it on the input data. The text feature vector of movie titles is removed while each movie number is scaled down to 1/100 of its original size.

We got the learning curve shown in Figure 2. It can be seen that the validation loss fluctuates around 1.0, while the eventual test loss was around 1.60, Slight though the improvement is compared to the original one, this model solves the problem of similar recommended content in the previous model and improved diversity.

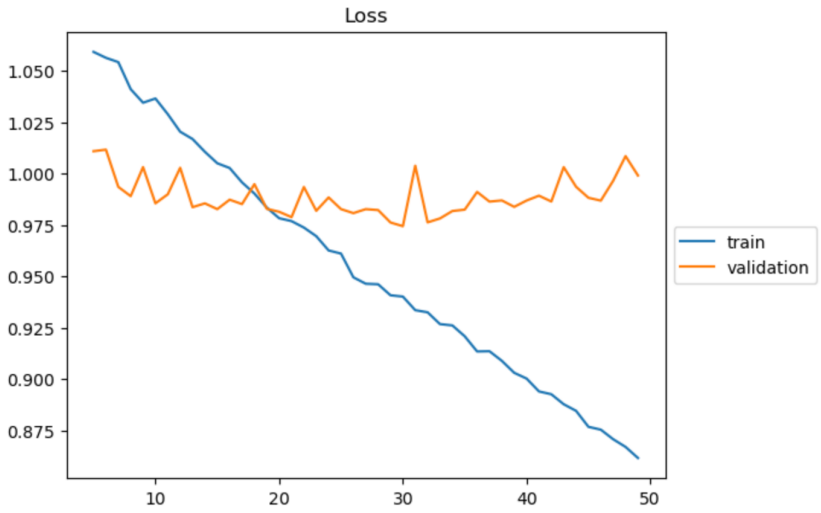


Figure 2: Learning curve of final DNN model

However, the model based on score estimation is not suitable for the recommendation of massive big data, let alone the comparatively large errors. To this end, we further explore new recommendation methods.

4 Using VAE to Train a Recommendation System

4.1 Why VAE?

VAE (Variational Autoencoder) is a type of generative model that uses a combination of deep learning and probabilistic graphical modeling techniques to learn the underlying structure of data. At a high level, a VAE works by encoding data into a lower-dimensional latent space and then decoding that latent representation back into the original data space. This allows the VAE to capture the underlying patterns and structures present in the data, and to generate new data samples that are similar to the training data.

In the context of a recommendation system, a VAE can be trained on user-item interactions (such as ratings or clicks) to learn the preferences and interests of individual users. Once trained, the VAE can be used to generate recommendations for a user by encoding their past interactions into a latent space and then decoding that latent representation to generate a list of items that are likely to be of interest to the user.

VAE outperforms other recommendation models for the following three advantages. First, VAE can learn the latent features or characteristics of the items in the training set, which allows the model to provide more personalized and accurate recommendations for users. Another advantage is that they can generate new items that are similar to the items in the training set. This can be useful in situations where there are not enough items in the training set to make accurate recommendations. Furthermore, VAE models are able to handle high-dimensional data and can be trained on large datasets. This makes them well-suited for recommendation systems that need to handle a large number of items and users.

4.2 Mathematical Insights

VAE assumes that the input $\mathbf{x}$ is generated by two generative process:

$$\mathbf{z} \sim p_{\theta^*}(\mathbf{z})$$

and

$$\mathbf{x}|\mathbf{z} \sim p_{\theta^*}(\mathbf{x}|\mathbf{z})$$

where $\mathbf{z}$ is an unobserved latent vector whose dimensionality is much lower than $\mathbf{x}$. VAE aims at estimating the parameters θ^* by the MLE method, i.e., $\hat{\theta} = \arg\max_{\theta\in\Theta} p_\theta(\mathbf{x})$. $p_\theta(\mathbf{x})$ is computed by

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})\, p_\theta(\mathbf{z})\, d\mathbf{z}$$

In practice, however, for most $\mathbf{z}$, $p_\theta(\mathbf{x}|\mathbf{z}) \approx 0$. The key idea is to estimate $p_\theta(\mathbf{x}|\mathbf{z})$ with a distribution $q_\phi(\mathbf{x}|\mathbf{z})$ that follows a specific parametric distribution, e.g., normal distribution. In deep learning practice, both p_θ and q_ϕ are parameterized by deep neural networks, also named decoder and encoder.

Naturally, we desire p_θ and q_ϕ to be as close as possible. The closeness between them is quantified by Kullback-Liebler divergence (KL)[3] with the following format:

$$KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[log\, q_\phi(\mathbf{z}|\mathbf{x}) - log\, p_\theta(\mathbf{z},\mathbf{x})] + log\, p_\theta(\mathbf{x})$$

Thus,

$$log\, p_\theta(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[log\, p_\theta(\mathbf{x}|\mathbf{z}))] - KL(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) = \mathscr{L}(\mathbf{x};\theta,\phi)$$

where $\mathscr{L}(\mathbf{x};\theta,\phi)$ can act as the objective function.

With respect to the recommendation system, Liang et al.[4] proposed a VAE for collaborative filtering called Mult-VAE. Mult-VAE takes the user-item binary interaction matrix $\mathbf{x}_u$ as input and learns a latent representation $\mathbf{z}_u$ with lower dimensionality (the encoder). This latent representation is then used to reconstruct the input (the decoder) and to predict the missing interactions. In movie recommendations, interaction is referred to as rating. The top-N recommendation is predicted by taking N items with the highest reconstructed ratings for each user u.

Under an assumption that latent vector $\mathbf{z}_u$ is drawn from a standard Gaussian distribution, interaction matrix $\mathbf{x}_u$ is subject to a multinomial distribution and a neural network f_θ is used to produce a probability distribution $\pi(\mathbf{z}_u)$ over I items, we reconstruct the VAE model in the recommendation system as follows:

$$\mathbf{z}_u \sim N(0, \mathbf{I}_k)$$

$$\pi(\mathbf{z}_u) \propto exp\{f_\theta(\mathbf{z}_u)\}$$

$$\mathbf{x}_u \sim Mult(N_u, \pi(\mathbf{z}_u))$$

4.3 Dataset and Preprocessing

Same as [4] and [5], we choose MovieLens-20M dataset to perform the recommendation system. The dataset contains 20 million rating information regarding 27000 movies and 138,000 users.

In order to imitate the prevalent case for practical recommendation tasks, the explicit

interactions, i.e. ratings, are converted to implicit ones by binarization. That is, only ratings greater than or equal to 4 are considered valid interactions. 10000 users are chosen as the test set as well as their ratings, and models are trained on the rest.

4.4 Testing Metrics

Evaluating the performance of recommendations is much more sophisticated than simple regressions. The key step is to sample 80% of the ratings for each user and compare the rest with model predictions. In this way, necessary information about each user is learned while some interactions remained unknown for us to select the agreeable model.

Because implicit interactions are used to train the model, previous metrics such as score and MSE are not suitable for this case. The key metric we used is Recall in the TOP-k recommendation, denoted as $Recall@k$. It is computed by the formula

$$Recall@k = \frac{|\hat{R}_k \cup R_k|}{R_k}$$

where $\hat{R}_k$ is the top-k movies recommended by the model and R_k is the true movies users have selected.

Another metric used in evaluating a recommendation system is the truncated normalized discounted cumulative gain ($NDCG@k$). While $Recall@k$ considers all items ranked within the first k movies to be equally important, $NDCG@k$ uses a monotonically increasing discount to emphasize the importance of higher ranks versus lower ones [4]. It is computed by the formula

$$DCG@k = \sum_{i=1}^{k} \frac{2^{\mathbb{I}[\omega(i) \in R_k]} - 1}{log_2(i+1)}$$

where $\omega(i)$ refers to the movie at rank i predicted. $NDCG@k$ is the linearly normalized version of $DCG@k$.

4.5 Model Architecture

We referred to codes provided by [5] to implement the model. The *utils.py* file in their GitHub repo has provided a set of APIs to help train recommendation models with binary inputs.

The whole architecture includes 2 parts, an encoder, and a decoder. The encoder receives a user-movie interaction matrix $\mathbf{x}$ and computes through several fully-connected layers with a swish activation and a normalization step per layer. At the end of the encoding stage,

the mean and variance of the latent Gaussian distribution are learned. In the decoding stage, latent vector instances $\mathbf{z}$ are drawn from this distribution, which then goes through the decoder network. The decoder network has layers identical to the encoder (including activation functions, layer types, etc.), through which $\mathbf{z}$ can be reconstructed to $\hat{\mathbf{x}}$ which should be identical to $\mathbf{x}$ in the ideal situation.

4.6 Tuning and Results

Parameters to be tuned involve dimensions of latent vector, numbers of hidden units for each layer, number of layers in encoder and decoder, and learning rates. The activation function is Swish in default, which is a nonlinear smoothing function interpolating between a linear function and the ReLU function. To grab the best parameters and activation function, we set up sorts of experiments. Some Representative results are shown in Table 3.

	Model Parameters			Test Result		
	Architecture	Learning rate	Activation	Recall@20	Recall@50	NDCG@100
1	(1024, 2048, 7, 5)	(0.00005, 0.00001)	Swish	0.40338	0.54048	0.43871
2	(1024, 2048, 7, 5)	(0.0005, 0.0001)	Swish	0.3841	0.52038	0.41509
3	(1024, 2048, 7, 5)	(0.005, 0.001)	Swish	0.24266	0.35841	0.27073
4	(1024, 2048, 7, 5)	(0.00005, 0.00001)	ReLU	0.4029	0.54028	0.43826
5	(512, 1024, 5, 3)	(0.00005, 0.00001)	Swish	0.39564	0.53058	0.43052
6	(512, 1024, 5, 3)	(0.0005, 0.0001)	Swish	0.38785	0.52619	0.42154
7	(512, 1024, 5, 3)	(0.005, 0.001)	Swish	0.37588	0.51987	0.40352
8	(512, 1024, 5, 3)	(0.00005, 0.00001)	ReLU	0.39307	0.52875	0.4277

Table 3: Tuning results of the VAE recommendation model. The four-element bundle "Architecture" here represents the number of #(latent variables, hidden units, encoder layers, decoder layers). Learning rate bundles here correspond to 2 learning stages in our implementation.

From Table 3 we can conclude that: (1) Architectures with more parameters tend to grasp a better pattern from the data, until bundle (1024, 2048, 7, 5) where few advancements could be made by another addition (We tried (2048, 4096, 7, 5) and get the nearly exactly same results). (2) To train a huge model like this (e.g. Architecture (1024, 2048, 7, 5) has a total of 154,443,234 parameters), lower learning rates perform better. (3) Swish functions do capture better patterns than ReLUs.

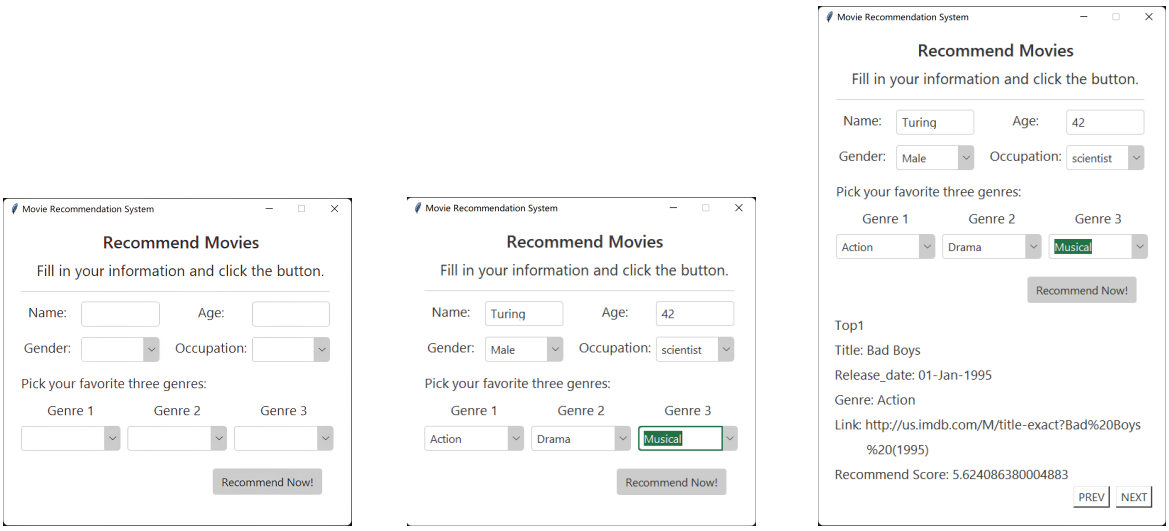
Therefore, we choose combination 1 as our best parameter.

5 Combining with UI: A Movie Recommendation App

With all the above well-performed models available, we are able to set sail to our final destination, a movie recommendation app.

5.1 Features of the App

The movie recommendation app we designed aims to recommend desired movies for users. We write the UI based on Tkinter, a convenient standard Python interface to the Tk GUI toolkit. To achieve our goal, necessary information is collected, for instance, name, age, gender, occupation, and most importantly, 3 different genres that users are interested in (see in figure 3a). Having filled in all the above information and with the "Recommend Now" button clicked, the program automatically loads the model we trained which predict top-3 recommendation result. The recommended result is shown on the screen, accompanied by basic information about the movie including release date, genres to which it belongs, and the web link to the corresponding movie website. Users can conveniently refer to the website by simply clicking the link.



(a) Homepage **(b)** Fill in the blanks **(c)** Recommend results

Figure 3: This is the UI demo for the recommendation system (3a). Let's say, Turing, aged 42, male, working as a scientist, wants to watch some movies (3b). His favorite genres are action, drama, and musical. After clicking the button, the system automatically recommends him top-3 movies, that is, Bad Boys (3c), Mask, and Lone Star. Now, Turing can click the link below and start enjoying!

5.2 Future Prospects

Up to now, we only use our model proposed in Session 3 to make the prediction. This is because only this model fits the scenario of an application only to recommend movies with identities and finite interests selected by the users. However, scenarios are distinct in real life. If we were to build a large movie platform where people can watch, comment, rate, etc., our last model could be used to capture users' implicit interactions, while the app having been implemented could be used to give initial recommendations to new users. They compensate each other.

In other words, all these models proposed and implemented above can be easily integrated together to meet different needs in comprehensive recommend scenarios. In this way, we regard our work to be of great value.

Bibliography

[1] Niels Bogaards and Frederique Schut. Content-based book recommendations: Personalised and explainable recommendations without the cold-start problem. In *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, page 545–547, New York, NY, USA, 2021. Association for Computing Machinery.

[2] Maria Brbić, Ivana Podnar arko, George A. Tsihrintzis, and Maria Virvou. Tuning machine learning algorithms for content-based movie recommendation. 9(3):233–242, sep 2015.

[3] Tommaso Carraro, Mirko Polato, and Fabio Aiolli. Conditioned variational autoencoder for top-n item recommendation, 2020.

[4] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering, 2018.

[5] Vojtech Vancura and Pavel Kordík. Deep variational autoencoder with shallow parallel path for top-n recommendation (VASP). *CoRR*, abs/2102.05774, 2021.

[6] Chong Wang, Yao Shen, Huan Yang, and Minyi Guo. Improving rocchio algorithm for updating user profile in recommender systems. In Xuemin Lin, Yannis Manolopoulos, Divesh Srivastava, and Guangyan Huang, editors, *Web Information Systems Engineering – WISE 2013*, pages 162–174, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.