POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Project report for Numerical Analysis for Machine Learning course

MyNextMovie

MSc in Computer Science and Engineering - Artificial Intelligence

Author 1: Fontana Fabrizio - 10828904

Author 2: Terzi Diego - 10816783

Professor: Miglio Edie

TA: Caldana Matteo

Academic year: 2024-2025

---

1. Introduction

The vast amount of movie content available today makes it increasingly difficult for users to discover films that match their preferences. To address this challenge, recommendation systems have become a fundamental tool, offering personalized suggestions based on user behavior and movie characteristics. However, traditional recommender systems often struggle with issues such as data sparsity and the cold-start problem[1], limiting their ability to provide accurate and diverse recommendations. In this project, we set out to build a movie recommender system that leverages generative AI, specifically Variational Autoencoders (VAEs)[2], to enhance the quality of the recommendation. Furthermore, we incorporated collaborative filtering[3], which analyzes past user-item interactions to generate more relevant movie suggestions. By implementing the approach outlined in the reference paper[4], we constructed a system capable of learning complex user preferences and generating tailored movie recommendations. The development process followed a structured approach: as described in the paper, we selected, cleaned, and prepared a suitable dataset for training; then, we implemented the VAE-based recommendation model and trained it; and finally, we designed a graphical user interface (GUI) to allow users to interact with the system seamlessly. This report details the steps taken to create the system.

2. Data

An important step in developing the VAE-based movie recommender system involved a data processing pipeline, as described below.

[1] The challenge in recommendation systems where new users or items lack prior interactions, making it difficult to generate relevant recommendations.

[2] A type of neural network used for generative modeling, capable of learning a structured latent space to generate new data points.

[3] A technique that predicts user preferences based on shared behaviors among users.

[4] Carmen Pei Ling Tung, Su-Cheng Haw, Wan-Er Kong, Palanichamy Naveen (2024). Movie Recommender System based on Generative AI. DOI: 10.1109/PCDS61776.2024.10743897

2.0.1 Dataset Acquisition and Loading

The dataset selected for this project was the widely-used MovieLens 100K dataset, available here. This dataset comprises 100,000 ratings provided by 943 users on 1682 movies. It is structured into three main files: `u.data`, `u.user`, and `u.item`.
Upon downloading and extracting the dataset, these files were loaded into pandas DataFrames for manipulation. The `u.data` file, containing the rating information, was loaded with columns corresponding to `user_id`, `item_id`, `rating`, and `timestamp`. The `u.user` file, providing user demographic data, was loaded with columns `user_id`, `age`, `gender`, `occupation`, and `zip`. Similarly, the `u.item` file, detailing movie attributes including genre information, was loaded with a pipe delimiter and columns encompassing `item_id`, `movie_title`, and a binary representation of 19 movie genres.

2.1. Data Cleaning

The first data cleaning step involved filtering the ratings to include only those from users who had rated a minimum of 20 movies. This threshold was established to ensure that each included user had a substantial interaction history, providing enough data points for the VAE model to learn individual preferences effectively and contribute reliably to the collaborative filtering process. Users with fewer ratings were excluded to mitigate the cold-start problem.
Additionally, the loaded user and item DataFrames were inspected for missing values. Any null entries discovered in these dataframes were imputed by filling them with zeros using the `fillna(0)` method. This step was taken to ensure that the datasets were complete and consistent, preventing errors in downstream processing and model training.

2.1.1 Handling Data Sparsity

A key characteristic of the MovieLens 100K dataset is its inherent sparsity. This means that the vast majority of possible user-item interactions (i.e., ratings) are missing, as users typically only rate a small fraction of the available items. This limited explicit interaction data posed challenges during model training, particularly impacting the loss calculation, which required careful handling to ensure stable and effective learning for the VAE.

2.2. Data Transformation for Model Input

To prepare the filtered and cleaned rating data for the VAE, which expects a matrix input representing user interactions, a transformation was performed. A user-item interaction matrix was constructed using the `pivot_table` function from the ratings DataFrame. User IDs became the rows, item IDs the columns, and the corresponding ratings filled the cells where an interaction existed. The explicit ratings (values 1-5) were normalized to a range between 0 and 1. This process resulted in a dense matrix of shape [number of users] $\times$ [number of items], where each entry represented either a normalized rating (for items the user rated) or 0 (for unrated items), forming the direct input for the VAE model.

2.2.1 Saving Processed Data

Finally, the processed and cleaned data, including the filtered ratings, the user information with imputed values and the item information with imputed values, were saved into a new directory structure to isolate the cleaned data and make it readily available for the subsequent model building and training phases.

3. Model

The core of our movie recommendation system is built around a Variational Autoencoder, a type of generative model capable of learning latent representations of input data in an unsupervised manner. In this section, we describe the architecture and components of the model, as implemented in TensorFlow/Keras.
The model is composed of three key components: Encoder, Sampling Layer and Decoder.

3.1. Encoder

The encoder takes as input a sparse user rating vector of size n_{items}, where each entry represents whether the user has interacted with a given item. The encoding path consists of a stack of dense layers with progressively decreasing dimensions:

- Dense layer with 1024 units, L2 regularization, and LeakyReLU[5] activation.
- Dense layer with 512 units, L2 regularization, and LeakyReLU activation.
- Dense layer with 256 units, L2 regularization, and LeakyReLU activation.

The output of the encoder consists of two vectors:

- μ: the mean vector of the latent distribution.
- $\log \sigma^2$: the logarithm of the variance vector.

These are used to sample latent variables via the reparameterization trick.

3.2. Sampling Layer

The sampling layer bridges the encoder and decoder by generating latent variables from the distribution defined by the encoder's outputs. This component introduces stochasticity into the model while ensuring that gradients can still be propagated during training.

3.2.1 Reparameterization Trick

To allow gradients to flow through the stochastic latent variables, we use the *reparameterization trick*. Instead of sampling z directly from a distribution parameterized by μ and $\log \sigma^2$, we sample from a standard normal distribution and shift and scale the result deterministically:

$$z = \mu + \exp\left(0.5 \cdot \log \sigma^2\right) \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

This technique converts the sampling process into a differentiable operation, making it compatible with gradient-based optimization. In our implementation, this computation is encapsulated within a Keras `Lambda` layer, ensuring both clarity and serializability of the model.

3.3. Decoder

The decoder reconstructs the input from the latent vector $\mathbf{z}$. It mirrors the structure of the encoder:

- Dense layer with 256 units, L2 regularization, and LeakyReLU activation.
- Dense layer with 512 units, L2 regularization, and LeakyReLU activation.
- Dense layer with 1024 units, L2 regularization, and LeakyReLU activation.

[5]LeakyReLU is a variant of the ReLU activation function that allows a small, non-zero gradient when the unit is inactive, mitigating the *dying ReLU* problem.

The sigmoid activation ensures that outputs are in the range $[0, 1]$, which can be interpreted as a probability score for each item.

3.4. Custom VAE Class

We encapsulate the encoder and decoder into a subclass of `keras.Model`, allowing us to define custom training logic, including the composite loss function (detailed below), and to manage model saving/loading with configuration metadata.

3.5. Loss Function

3.5.1 Reconstruction Loss

This term quantifies how well the VAE can reconstruct the input data after encoding it into the latent space and then decoding it. In the context of the user-item matrix, this loss measures the difference between the original ratings and the ratings predicted by the decoder.
Initially, during preliminary experiments, the reconstruction loss was naively calculated over the entire user-item matrix, including the zero values imputed for unrated movies. Given the inherent sparsity of typical recommendation datasets, where the vast majority of entries are zero (representing unobserved interactions), this approach led to significant training instability and ultimately poor model performance. The model was overwhelmingly incentivized to predict zero for most items to minimize the loss, effectively ignoring the scarce but important signal from the actual user ratings. This resulted in a model that failed to learn meaningful representations or generate accurate reconstructions of observed interactions.
Recognizing this issue, the reconstruction loss calculation was refined. As implemented in the `reconstruction_loss` function within the `CustomVAE` class, the loss is now computed exclusively on the observed ratings (non-zero entries in the input matrix). This is achieved by applying a mask that selects only the positions corresponding to actual ratings. This corrected approach ensures that the model's learning is driven by the discrepancy between the true observed ratings and their reconstructions, focusing its capacity on modeling the actual user preferences and item characteristics. The loss used is based on the mean squared error (MSE) cal-

culated solely on the masked elements:

$$\mathcal{L}_{\text{reconstruction}} = \frac{\sum_{(i,j)\in\text{Observed}}(R_{ij} - \hat{R}_{ij})^2}{|\text{Observed}|}$$

where R_ij is the true rating of user i for item j, $\hat{R}_{ij}$ is the reconstructed rating, and |Observed| is the number of observed ratings. This targeted approach allowed the model to learn effectively from the available interaction data.

3.5.2 KL Divergence Loss

This term acts as a regularizer, encouraging the distribution of the latent space (encoded by the encoder) to be close to a predefined prior distribution, typically a standard normal distribution. Minimizing the KL divergence ensures that the latent space is continuous and well-structured, allowing for smooth interpolation between data points and enabling the generation of diverse outputs. The KL divergence loss for a Gaussian latent distribution with mean μ and log-variance $\log \sigma^2$ is given by:

$$\mathcal{L}_{\text{KL}} = -0.5 \sum_{d=1}^{D} (1 + \log \sigma_d^2 - \mu_d^2 - e^{\log \sigma_d^2})$$

where D is the dimension of the latent space, and μ_d and $\log \sigma_d^2$ are the mean and log-variance for the d-th dimension. This term was implemented in the 'kl_loss' function.

3.5.3 Total Loss

The total loss function that the VAE sought to minimize during training was the sum of the reconstruction loss and the KL divergence loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{reconstruction}} + \mathcal{L}_{\text{KL}}$$

While the custom VAE class includes parameters `beta` and `kl_weight` for potentially weighting these loss components, the provided code for the total loss calculation sums them directly, implying equal weighting ($\beta = 1.0$, $kl\_weight = 1.0$).

4. Training

Following the data preprocessing and the definition of the Variational Autoencoder (VAE) architecture, the phase of model training was undertaken. This stage involved optimizing the VAE's parameters to effectively learn the underlying patterns in the user-item interaction data (ratings), enabling it to generate latent representations and reconstruct the input.

4.1. Data Splitting and Preparation

The cleaned and normalized user-item matrix, representing user ratings for movies, served as the input data for the VAE. The matrix was normalized to scale the rating values by applying the transformation (rating - 1) / 4 and setting the value corresponding to imputed zeros (-0.25 after normalization) back to 0. Subsequently, a common split ratio of 80% for the training set and 20% for the validation set was employed.

4.2. Model Compilation

The VAE model, consisting of the previously defined encoder and decoder networks, was compiled before training. The compilation step involves configuring the model for training by specifying the optimizer and the loss function. The Adam optimizer was chosen for this task. Adam is an adaptive learning rate optimization algorithm that has proven effective in training deep neural networks. It computes adaptive learning rates for each parameter by leveraging estimates of first and second moments of the gradients facilitating efficient convergence during the training process.

4.3. Training Execution

The training was executed using the `fit` method of the compiled VAE model. The model was trained on the training data (`train_data`), with the same data provided as the target output since the VAE is a self-supervised model aiming to reconstruct its input. The training proceeded for a maximum number of epochs, iterating through the training data in batches of a specified size (`batch_size=128`). The choice of a `batch_size` of 128 elements per iteration is not arbitrary; this size, being a power of two, is often optimized for processing on modern hardware (such as GPUs), allowing for more efficient memory utilization and significant acceleration of the training process. Using mini-batches, as opposed to processing the entire dataset at once, also introduces a degree of stochasticity that can help the model avoid local minima and generalize more effectively.

The maximum number of `epochs` was set to 200, providing a wide window for the model to converge. However, the most critical aspect was the implementation of an `EarlyStopping` callback. This intelligent mechanism continuously monitored the validation loss (`val_loss`) throughout the training process. If the validation loss did not show improvement for a predefined number of epochs (`patience=8`), training was automatically halted. Furthermore, the `restore_best_weights=True` argument ensured that the model weights corresponding to the epoch with the lowest validation loss were restored. This is a safeguard against overfitting, ensuring the model halts at its 'sweet spot' of learning, where it generalizes optimally on unseen data, while also conserving computational resources.
These specific values for hyperparameters like `batch_size`, `epochs`, and `patience`, alongside the latent dimension (`latent_dim=64`) were not arbitrarily chosen. Instead, they represent the culmination of iterative experimentation and fine-tuning. Various combinations of these hyperparameters were explored to identify the configuration that yielded the best performance on the validation set, all while remaining consistent with best practices and insights derived from existing literature, including the foundational principles outlined in the provided paper.
The training process generated a history object that recorded the loss and validation loss values at the end of each epoch that was subsequently used to visualize the training progress and assess the convergence and stability of the model.

4.3.1 Model Saving

Upon completion of the training, the trained VAE model was saved to disk in the Keras format. This saved model encapsulates the learned weights and the model architecture, allowing it to be loaded later for making predictions without needing to retrain. The model was saved to the path `saved_models/vae_model.keras`.

4.4. Evaluation Metrics Calculation

Following the training and saving of the model, its performance was evaluated on the validation data.
Initially, the trained VAE was utilized to predict ratings for all users within this validation set. The raw outputs from the decoder, which are scaled between 0 and 1 due to the sigmoid activation and prior normalization, were then post-processed. This involved inverting the normalization to map the predicted values back to the original 1-5 rating scale and clipping any outliers to ensure they conformed to the valid rating range.
For quantitatively assessing the recommendation system's performance, especially in a ranking context where presenting a limited list of relevant items is paramount, we specifically chose widely accepted top-N evaluation metrics: Precision@5, Recall@5, and F1-score@5. This choice is deliberate because, in real-world scenarios, such as the one we are addressing, users typically focus on the top few recommendations presented.

- **Precision@5** measures the proportion of the top 5 recommended items that are truly relevant to the user. It improves the quality and minimizes irrelevant suggestions in the displayed list.
- **Recall@5** indicates the proportion of all relevant items for a user that were successfully captured within the top 5 recommendations. This metric ensures that the system does not miss valuable suggestions that an user would appreciate.
- The **F1-score@5**, as the harmonic mean of Precision and Recall, provides a balanced assessment of the model's performance. It helps in understanding the overall effectiveness by considering both the accuracy of the recommendations and their coverage of the user's preferences.

These metrics were computed for each user by comparing their top 5 predicted movies against their actual rated items. The average values across all users in the validation set then provided a robust indicator of the model's overall recommendation quality,.

5. User interaction

This section explores how users interact with our movie recommendation system. We've implemented two distinct modes to accommodate different user needs: a guest mode for casual users and a personalized mode for registered users.

5.1. Guest Mode

For users who prefer not to create an account, we've designed a simple guest mode. Guests can select their preferred movie genre from a dropdown menu (Action, Comedy, Drama, Horror, Sci-Fi), or opt to see recommendations across all genres. The system then displays the four highest-rated movies in the chosen category, allowing guests to explore content without any prior interaction history.

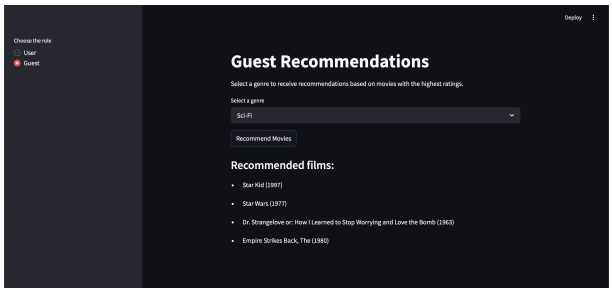

Figure 1: Guest page

5.2. User Authentication

To offer personalized recommendations, we've implemented a login system. Users must enter their user ID and password to access their personal profile, which enables the system to track their preferences and past interactions. This functionality has the purpose of generating personalized suggestions over time and improving the accuracy of recommendations.

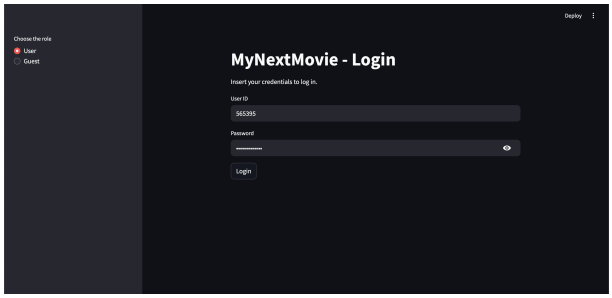

Figure 2: Login page

5.3. Personalized Recommendations

Once logged in, users enter the core functionality of the system. They are presented with four movie recommendations, generated by our VAE model. Each movie includes a rating slider (0-5) that allows users to express their opinion.
Instead of offering generic movie suggestions, the system adapts with every interaction, fine-tuning its recommendations to reflect user preferences in real time.

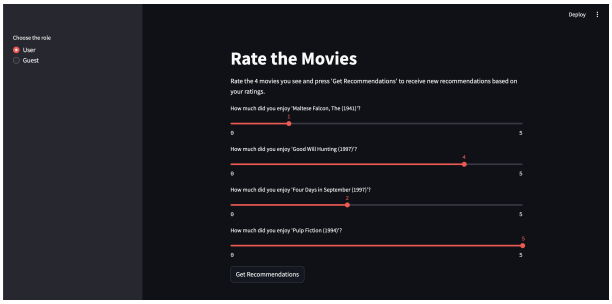

Figure 3: Rating page

6. Conclusions

This project successfully developed and implemented MyNextMovie, a movie recommender system leveraging VAEs and collaborative filtering to address the inherent limitations of traditional recommendation approaches, such as data sparsity and the cold-start problem. Our objective was to create a system capable of generating diverse, personalized, and contextually relevant movie recommendations, thereby enhancing the user's movie discovery experience.
The integration of VAEs, as outlined in the reference paper, proved instrumental in achieving this goal. By learning a structured latent space representation of user preferences and movie characteristics, the VAE model enabled the generation of novel and relevant recommendations, moving beyond simple content-based or collaborative filtering techniques. The system's architecture, as detailed in Section 5 of our report, allowed for both guest access with genre-based recommendations and personalized recommendations for logged-in users, adapting to their evolving tastes through continuous interaction.
A critical aspect of evaluating our model's performance was the analysis of its loss function during training. As depicted in Figure 4, the training process exhibited a consistent reduction in log loss over epochs.

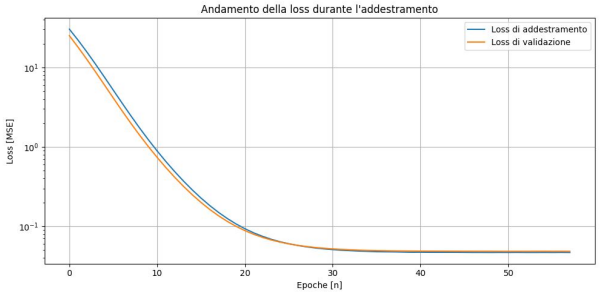

Figure 4: Log Loss during Model Training

The log loss curve demonstrates a stable convergence, indicating that the model effectively learned from the training data without significant overfitting. The gradual decrease signifies that the VAE was progressively minimizing the discrepancy between the predicted and actual user preferences, leading to more accurate recommendations. The absence of sharp fluctuations suggests a consistent training regimen and well-chosen hyperparameters. While the curve shows a clear downward trend, a minimal residual loss is expected due to the inherent complexity and stochastic nature of user preferences in recommender systems. This behavior underscores the model's ability to generalize from observed data to unseen movie-user interactions.

For logged-in users, the recommendations genuinely reflect their evolving tastes, as the VAE dynamically adjusts its latent representations based on new ratings. Moreover, the system has been structured to allow future retraining of the model once a sufficient number of new reviews has been collected, enabling even more accurate and personalized predictions over time.

Despite the promising results obtained, we acknowledge that there is always room for improvement and future development. For future work, we think that a straightforward next step would be to test our model on larger datasets to see how it scales and performs with more extensive user and movie data. This would help us understand its real-world applicability. We could also experiment with making the VAE model more complex, perhaps by adding more hidden layers or increasing the number of neurons in existing layers, to see if a deeper network can capture even more intricate patterns in user preferences and lead to more accurate recommendations.

In conclusion, MyNextMovie has demonstrated strong potential in developing movie recommendation systems using generative AI. Through this project, we've shown that Variational Autoencoders can effectively address some of the key limitations of traditional recommenders, opening up new possibilities for more personalized and engaging movie discovery.