



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Numerical Analysis for ML

Author: **Federica Maria Laudizi**

Student ID: 10724111

Repository: [Click here](#)

Academic Year: 2024-25

Contents

Contents	i
1 Scope	1
1.1 Introduction	1
1.2 Aim	1
2 Model's architecture	3
3 Methodology	7
3.1 Dataset Cleaning and Pre-processing	7
3.1.1 Data Loading and Cleaning Operations	7
3.1.2 Frameworks and Technologies	7
3.2 Recommendation Workflow	7
3.2.1 The VAE implementation	7
3.2.2 Evaluation	9
3.2.3 Comparison and Visualization	9
3.2.4 User Interface	12
3.2.5 Implementation and Integration	15
4 Results and Conclusions	17

1 | Scope

1.1. Introduction

Movie recommender systems have transformed the way users discover and engage with films by leveraging advanced algorithms to personalize suggestions. Traditional recommendation techniques, including collaborative filtering and content-based approaches, often struggle with issues such as limited diversity, inability to capture complex user preferences, and the cold-start problem for new users and items. Recent advances in artificial intelligence, particularly generative AI models , have introduced new possibilities to improve recommendation quality, being able to generate rich and individualized recommendations, improving diversity and capturing nuanced user interests within a continuous latent space. This report explores the integration of generative AI, in particular the **Variational Autoencoders (VAE)** into movie recommendation systems, highlighting its advantages, implementation strategies, and impact on recommendation accuracy and personalization.

1.2. Aim

As the reproduced paper did not report any results, the identified scope for this project was to **implement the recommender system** in order to verify its functionality and compare its results with traditional techniques.

2 | Model's architecture

The main model implemented and analyzed in the paper is the variational autoencoder, a type of generative neural network architecture. At its heart, a VAE still has the same structural components as a traditional autoencoder: an encoder and a decoder.

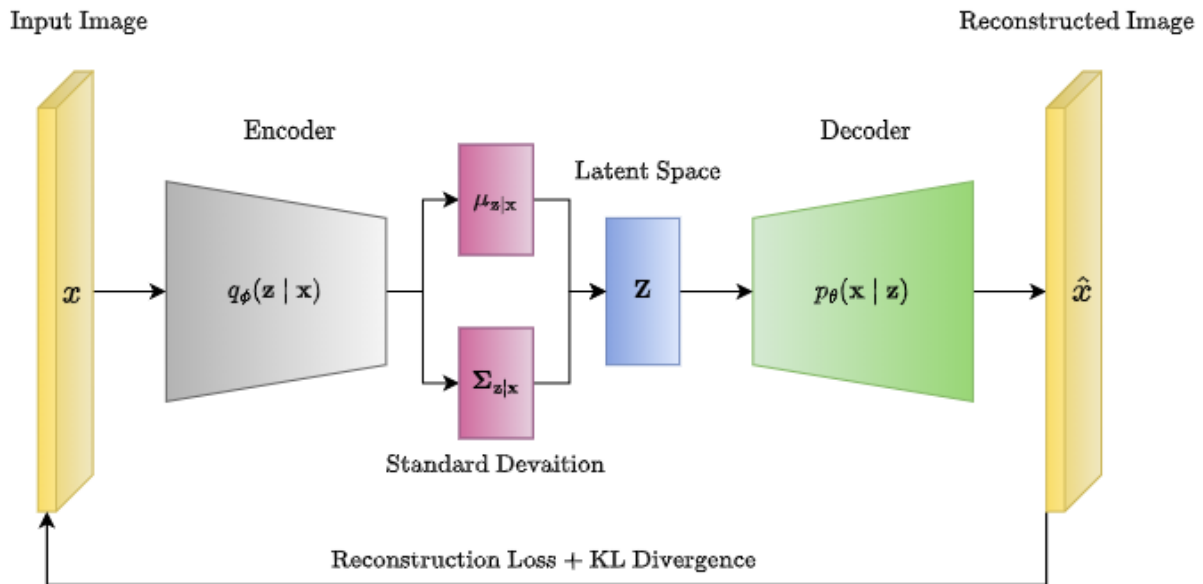


Figure 2.1: VAE architecture

- **Encoder:** The encoder's job is to take an input and compress it into a compact, latent representation. Traditional autoencoders output a single point in the latent space. However, a VAE outputs a distribution (usually a Gaussian) over the latent space.
- **Decoder:** The decoder samples a point from this distribution in the latent space and attempts to reconstruct the original input as accurately as possible.

The key advantage of VAEs lies in their generative capabilities. Because a VAE learns to model the underlying probability distribution of the input data, it can generate new samples that are similar to the original data. Moreover, unlike traditional autoencoders, VAEs promote

a continuous and smooth latent space. This means that we can interpolate between points in this space to generate smoothly transitioning variations of our data.

The Distribution Trick

Instead of a point in the latent space, the encoder of a VAE outputs the parameters that define a probability distribution (usually mean and variance). During training, we sample a point from this distribution to feed into the decoder.

The Reparameterization Trick

Imagine the encoder outputs the mean μ and standard deviation σ of a Gaussian distribution. We then sample a random variable from the distribution as:

$$z = \text{random_sample}(\mu, \sigma)$$

This seems straightforward, but there's a catch. Backpropagation requires calculating gradients with respect to the model's parameters (μ and σ in this case). However, calculating the gradient with respect to z (a randomly sampled variable) is mathematically undefined. This throws a wrench in the training process.

The reparameterization trick offers an elegant solution. Instead of directly sampling z from its distribution $\mathcal{N}(\mu, \sigma^2)$, we decompose z into a deterministic component and a stochastic component that is independent of the parameters we want to optimize.

1. **Introduce a new random variable:** We introduce another independent random variable, typically a standard normal variable ϵ . This new variable ϵ serves as a source of randomness entirely separate from the sampling process within the VAE.
2. **Reparameterize the latent variable:** Instead of directly adding noise, we create a scaled version of the new random variable ϵ using the standard deviation σ predicted by the encoder. We achieve this with element-wise multiplication:

$$z = \mu + \sigma \cdot \epsilon$$

Now, both μ and σ are deterministic outputs from the encoder, and ϵ is a separate source of randomness that we can control.

Why Does This Work? Because the new equation only involves deterministic operations, we can now calculate gradients with respect to μ and σ during backpropagation. This allows the network to learn and adjust its parameters effectively. Therefore, although the reparameterization trick uses a deterministic transformation, the final sampled latent variable (z) retains the desired statistical properties of the original Gaussian distribution. This ensures the VAE maintains its ability to explore the latent space effectively.

Loss Function

The VAE's loss function is composed of two parts:

- **Reconstruction Loss:** Just like in a traditional autoencoder, this part measures the difference between the original input and the output generated by the decoder.
- **KL Divergence:** This term measures the difference between the learned probability distribution over the latent space and a predefined prior distribution (usually a standard normal distribution). The KL divergence term ensures that the learned distribution over the latent space is close to the prior distribution, which helps regularize the model and ensures that the latent space has a meaningful structure.

$$\log p_{\theta} (x^{(i)}) \geq \mathcal{L} (x^{(i)}, \theta, \phi) = \mathbb{E}_z [\log p_{\theta} (x^{(i)} | z)] - D_{KL} (q_{\phi} (z | x^{(i)}) || p_{\theta}(z))$$

$$\underbrace{\mathbb{E}_z [\log p_{\theta} (x^{(i)} | z)]}_{\text{Reconstruct the Input Data}} - \underbrace{D_{KL} (q_{\phi} (z | x^{(i)}) || p_{\theta}(z))}_{\text{KL Divergence}}$$

$$\theta^*, \phi^* = \arg \max_{\theta, \phi} \sum_{i=1}^N \mathcal{L} (x^{(i)}, \theta, \phi)$$

3 | Methodology

3.1. Dataset Cleaning and Pre-processing

3.1.1. Data Loading and Cleaning Operations

The model was trained using the "100K Dataset" from [MovieLens](#). This dataset provides a rich collection of user ratings along with movie metadata such as titles, genres, and release dates, which is essential for both rating prediction and content-based analysis.

Data cleaning was performed by reading the ratings (`u.data`) and metadata (`u.item`) files. In the pre-processing step:

- Users who rated less than 20 movies were filtered out.
- The raw ratings were pivoted into a user-movie matrix with missing ratings replaced by zeros.
- For the implementation of variational autoencoder (VAE), the ratings were normalized to the $[0,1]$ range.
- Movie metadata was used to extract genre features;

3.1.2. Frameworks and Technologies

The dataset was managed using **Pandas** dataframes. These scripts leverage Python libraries such as NumPy, scikit-learn, TensorFlow Keras, matplotlib, seaborn and Streamlit to efficiently handle data operations and model training and implement the user interface.

3.2. Recommendation Workflow

3.2.1. The VAE implementation

The VAE, implemented in `movie_recommender.py`, is designed to learn latent representations of user preferences from the user-movie rating matrix. The key details of its implementation:

- **Input and Data Pre-processing:**
Each user is represented by a rating vector of length n_{movies} . The ratings are normalized

to the $[0,1]$ range to match the output range of the decoder.

- **Encoder Architecture:**

The encoder takes the normalized rating vector as input and passes it through three fully-connected (dense) layers with ReLU activation:

- First dense layer: 1024 neurons.
- Second dense layer: 512 neurons.
- Third dense layer: 256 neurons.

The output of these layers is then split into two separate dense layers to compute:

- The latent mean vector z_{mean} (with a default dimension of 50).
- The latent log-variance vector $z_{\text{log_var}}$ (also of dimension 50).

- **Reparameterization Trick:**

The latent vector z is computed as:

$$z = z_{\text{mean}} + \exp(0.5 \cdot z_{\text{log_var}}) \cdot \epsilon,$$

where ϵ is sampled from a standard normal distribution.

- **Decoder Architecture:**

The decoder reconstructs the user rating vector from the latent space. It mirrors the encoder but in reverse:

- First dense layer: 256 neurons with ReLU activation.
- Second dense layer: 512 neurons with ReLU activation.
- Third dense layer: 1024 neurons with ReLU activation.
- Output layer: A dense layer with n_{movies} neurons and a sigmoid activation function, ensuring that the reconstructed ratings fall within the $[0,1]$ range.

- **Loss Function and Training:**

The VAE is trained using a composite loss function that consists of:

1. **Reconstruction Loss:** Calculated using the binary cross-entropy between the input and the reconstructed output, scaled by n_{movies} .
2. **KL Divergence Loss:** Regularizes the latent space by measuring the divergence between the learned latent distribution and a standard normal distribution.

The overall loss is the sum of these two components. The model is compiled using the Adam optimizer, with early stopping (patience of 8 epochs) applied during training. Default training parameters include a batch size of 64 and 10 epochs.

- **Outputs:**

The final VAE model outputs a reconstructed user rating vector, aiming to be as close

as possible to the original normalized input while also maintaining a meaningful latent representation.

3.2.2. Evaluation

Model performance was evaluated using metrics such as RMSE, MAE, precision, recall, and F1 score. The evaluation functions, embedded in the code, also ensure that:

- Only the top-10 recommendations are considered for precision and recall calculations.
- The deterministic results are verified by testing the consistency of the system with varying filtering conditions.

3.2.3. Comparison and Visualization

In order to compare the results of the VAE model and verify its improvements w.r.t. traditional techniques, 3 other methods were implemented:

1. Collaborative Filtering (CF)

CF is a renowned and widely used technique for information filtering that identifies users within a group who share similarities with a particular user. In `model_comparisons.py`, a user-based collaborative filtering method was developed using cosine similarity. This method predicts movie ratings by averaging the ratings of the top- k most similar users.

2. Knowledge-Based Recommender

Knowledge-based systems use explicit knowledge about users and items to make recommendations. This approach is commonly used in domains where structured information is available. Also contained in `model_comparisons.py`, this approach leverages movie metadata to construct user profiles based on a weighted average of movie features. The similarity between user profiles and movie features is then used to generate rating predictions.

3. Hybrid Recommender

To combine the strengths of both CF and knowledge-based methods, a hybrid recommender was implemented. Predictions from both approaches are merged using a weighted sum, controlled by the parameter α (with a default value of 0.6), thus balancing their contributions.

Additionally, visualization functions were developed to generate:

- Bar charts comparing model performance.

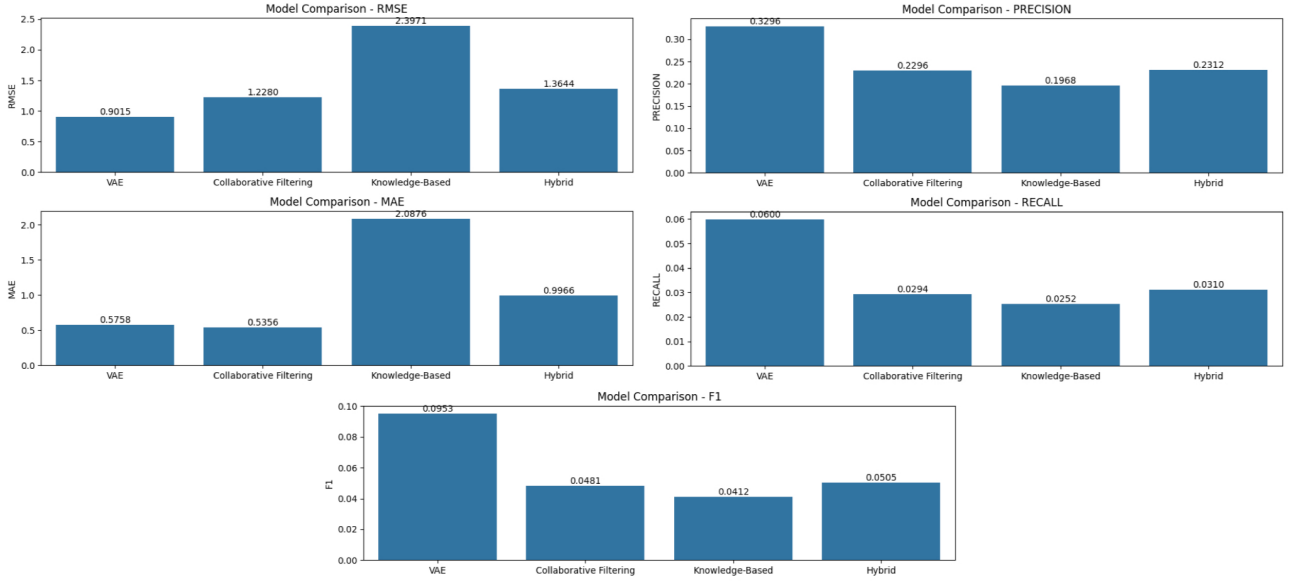


Figure 3.1: Model comparison bar charts

As we can see, VAE consistently outperforms other models in precision, recall, and F1-score while maintaining the lowest RMSE. On the other side, CF performs well in error metrics (MAE and RMSE) but has moderate precision and recall, Knowledge-Based model shows poor performance across all metrics, likely due to its rigid rule-based approach and the Hybrid model performs better than Knowledge-Based but does not surpass VAE, indicating that combining different approaches does not necessarily improve performance in this case.

- Heat-maps illustrating the user-movie rating matrix and similarity matrices.

The heatmap shown below, suggests that user similarity is not strong, which impacts Collaborative Filtering's effectiveness. On the other side, VAE's strong performance is likely due to its ability to learn latent patterns rather than relying on user similarity alone.

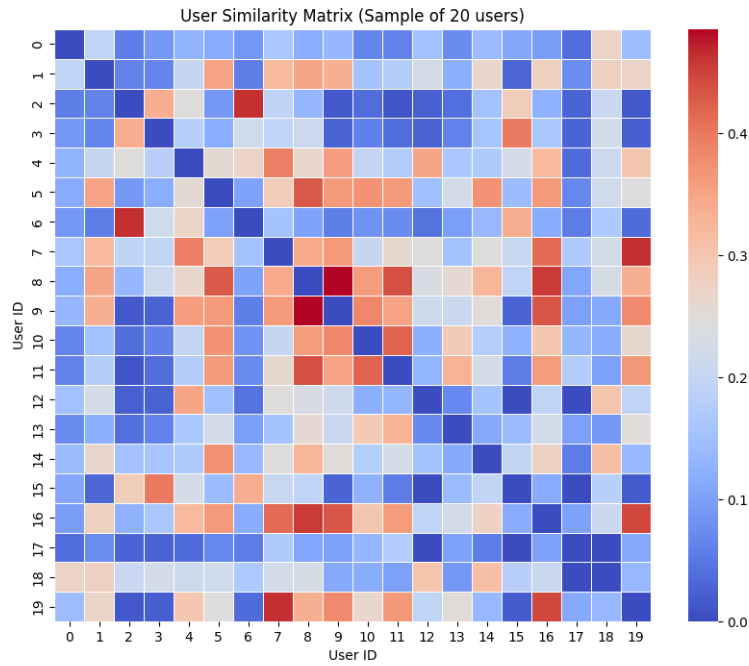


Figure 3.2: Similarity matric

- Sensitivity analysis plots for the hybrid model's α parameter.

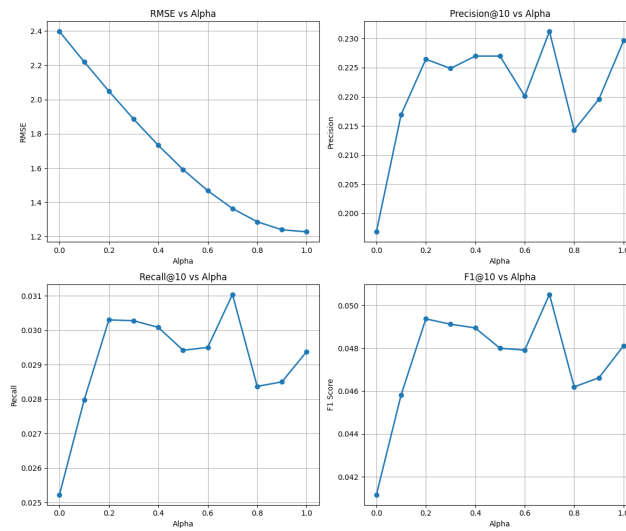


Figure 3.3: Alpha sensitivity

Higher Alpha values reduce RMSE, leading to improved accuracy. Precision and recall improve

significantly at low Alpha but stabilize around 0.2 - 0.3. The optimal Alpha range is around 0.6 - 0.7, where F1-score and recall peak. Beyond Alpha = 0.8, improvements diminish, suggesting limited benefits from further increasing Alpha.

3.2.4. User Interface

The recommender system is integrated into a user-friendly interface, implemented in **Streamlit**. This allows users to easily input their movie preferences and receive real-time personalized recommendations.

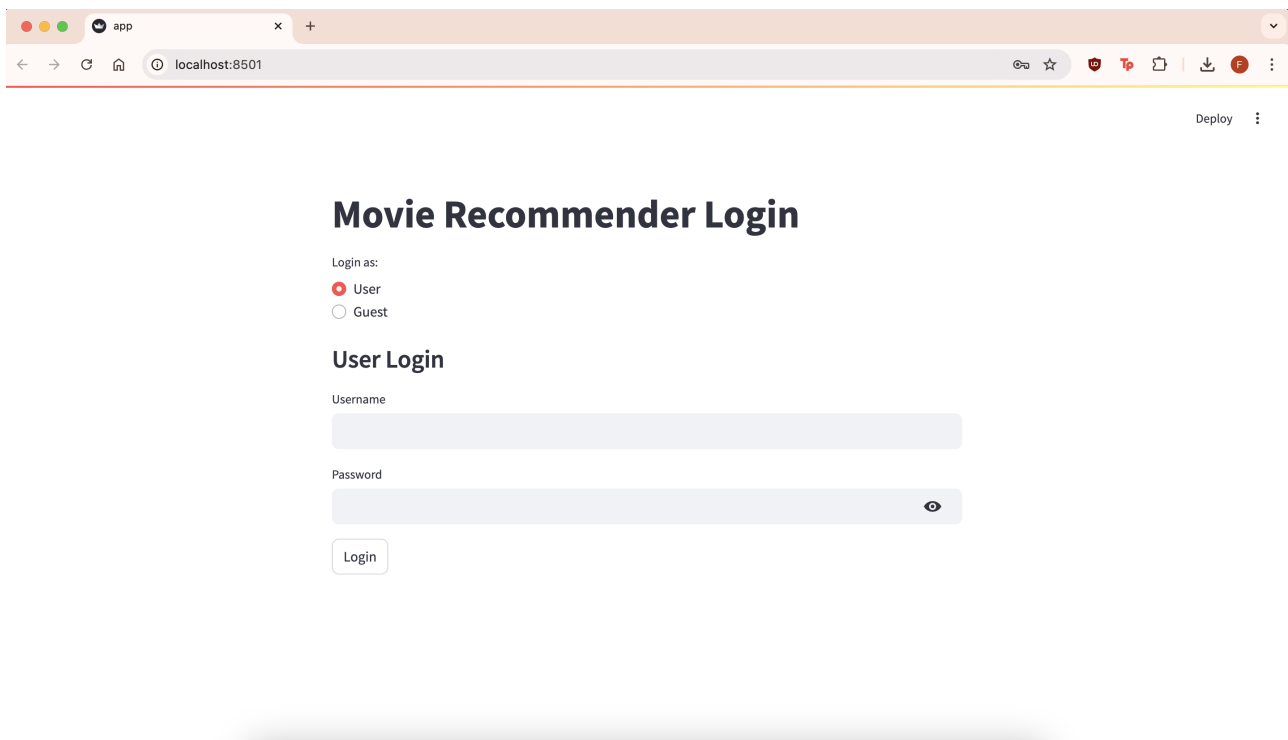
The image shows a web browser window with a single tab titled 'app'. The address bar shows 'localhost:8501'. The page content is a login form for a 'Movie Recommender'. At the top, it says 'Login as:' followed by two radio buttons: 'User' (which is selected) and 'Guest'. Below this is a section titled 'User Login'. It contains two input fields: 'Username' and 'Password'. The 'Password' field has a toggle icon (an eye) to the right of it. At the bottom of the form is a 'Login' button. The browser's top bar includes navigation icons, a star for bookmarks, and a 'Deploy' button with a dropdown arrow on the right.

Figure 3.4: Movie recommender system main page

Guests must select their movie genre preferences (Fig. 3.7) to receive movie recommendations since they do not have a watched movie history that can be used to personalize recommendations. By selecting their preferred genres (Fig. 3.7), guests can provide input that helps the system generate relevant movie suggestions tailored to their interests. If they choose genres, the system recommends random movies from those genres. Users will receive their personalized recommendations by rating (Fig. 3.5) previously seen movies.

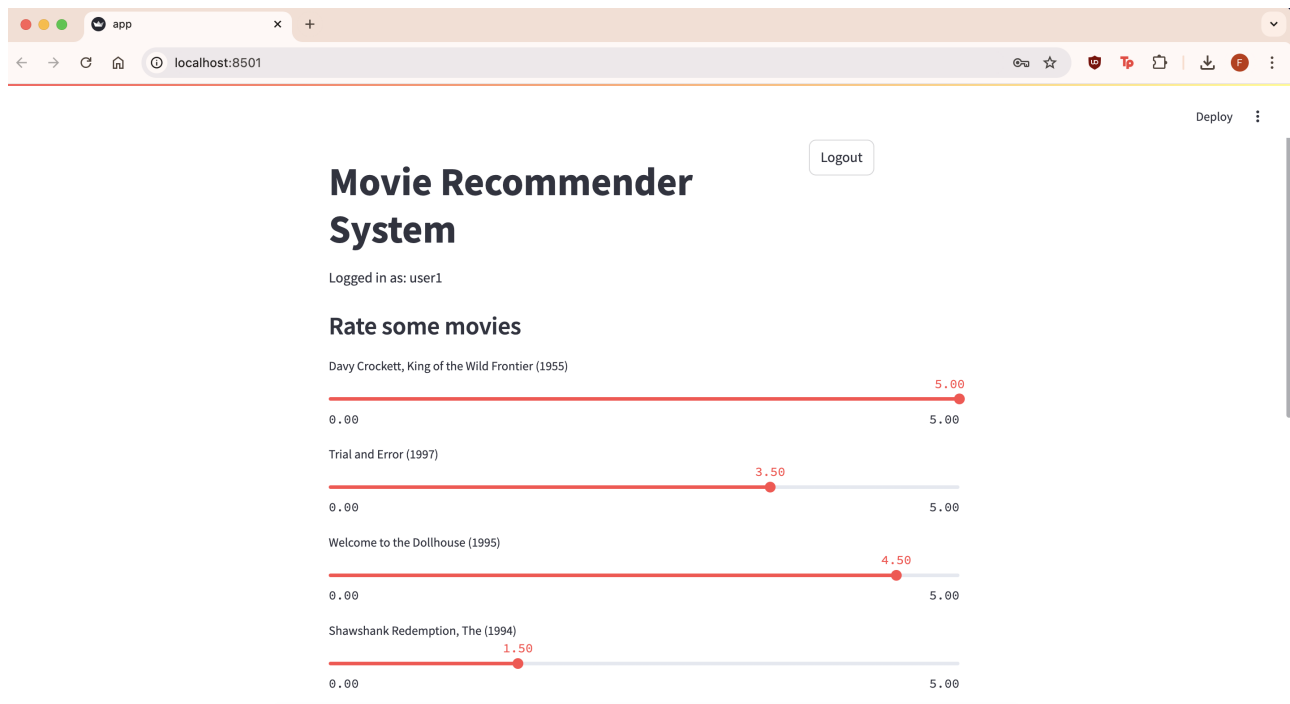


Figure 3.5: Rate watched movies

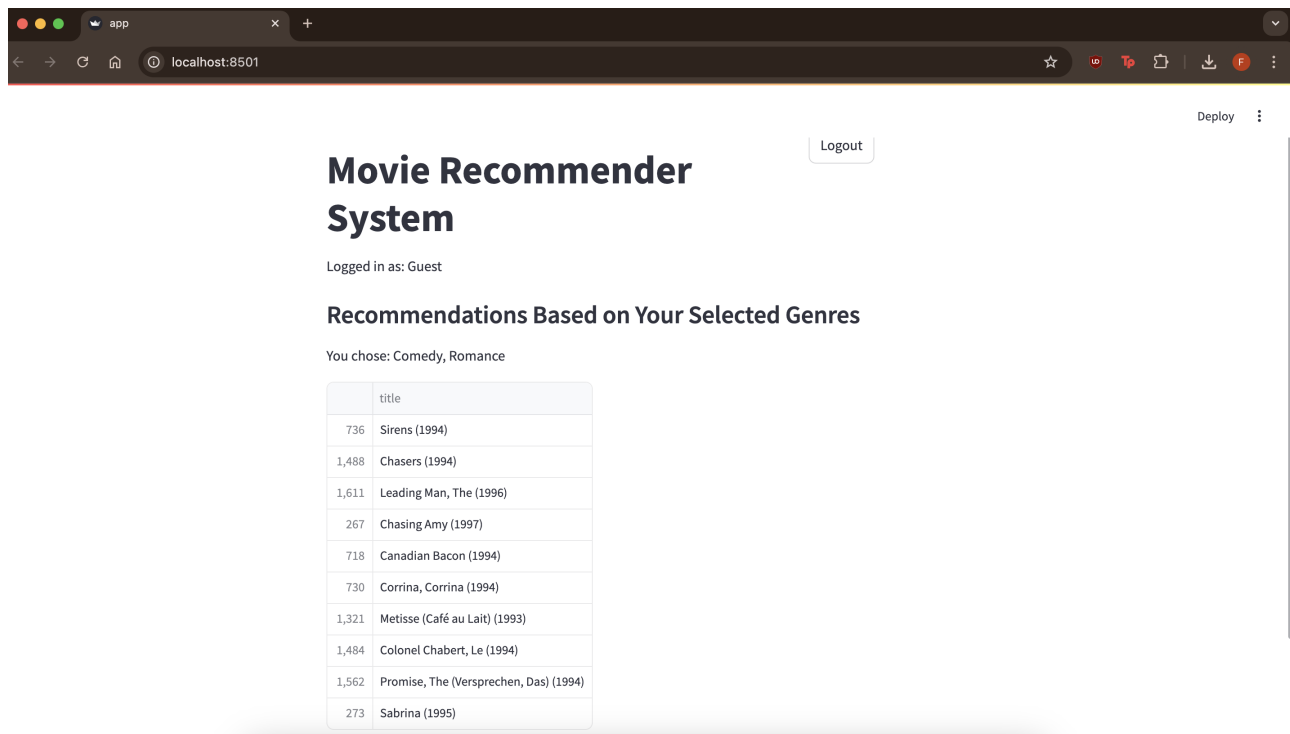


Figure 3.6: Recommendations based on genre

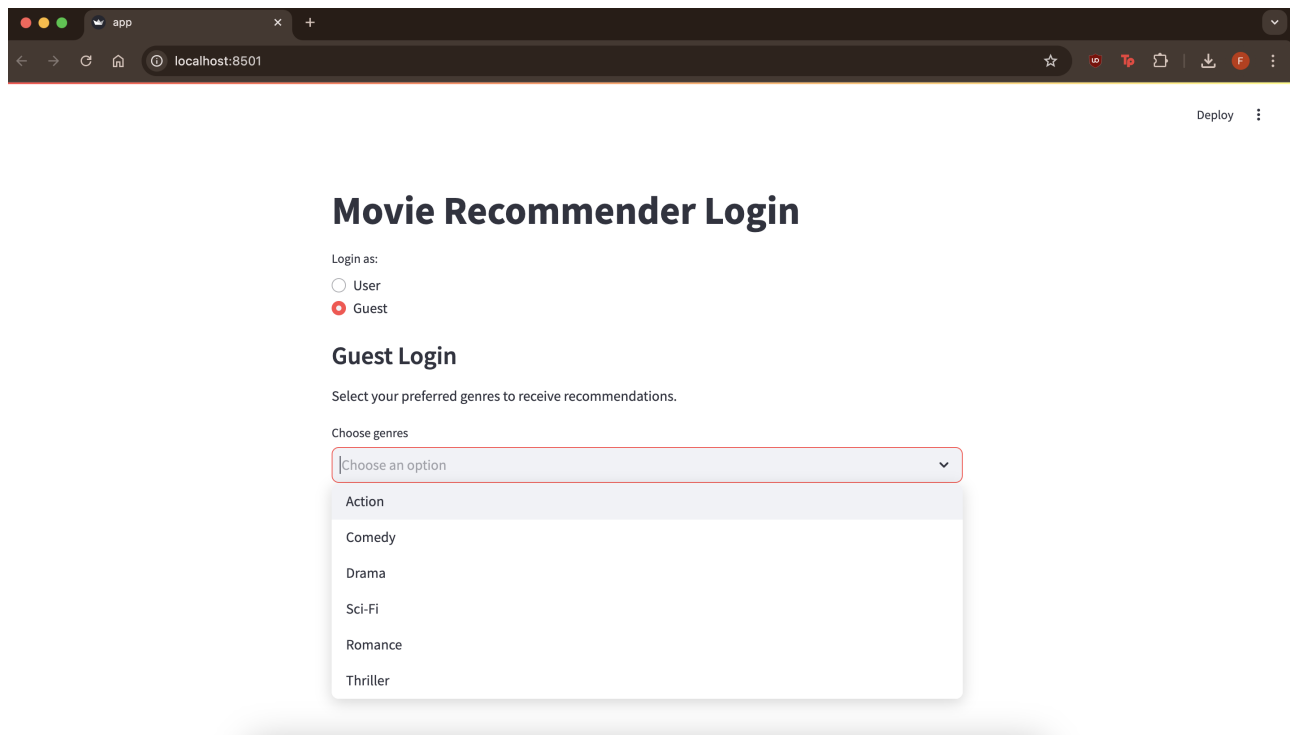


Figure 3.7: Guest genre selection

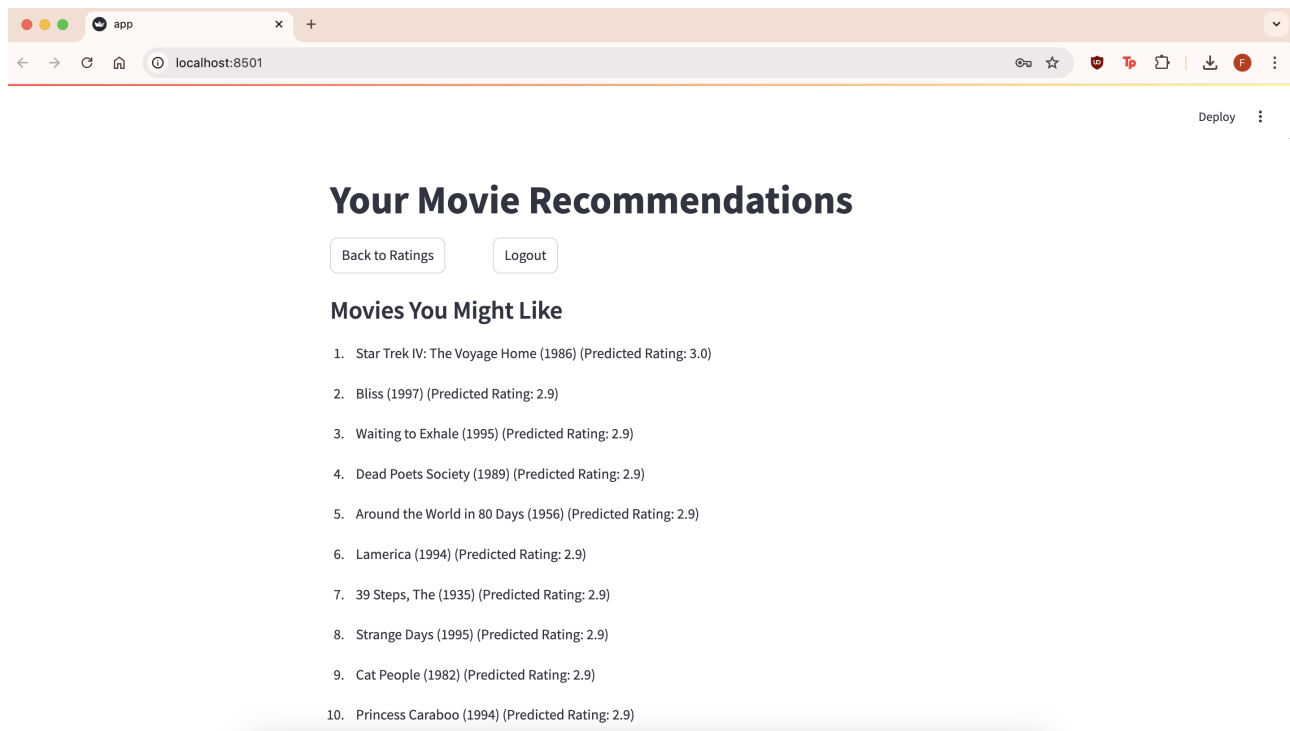


Figure 3.8: User personalized recommendations

3.2.5. Implementation and Integration

The entire workflow was implemented in Python with the code organized into three main scripts:

- `movie_recommender.py` focuses on the VAE-based recommendation approach.
- `model_comparisons.py` implements collaborative filtering, the knowledge-based recommender, their hybrid integration and the visualisation of the metrics.
- `app.py` implements the user interface using the Python library Streamlit.

4 | Results and Conclusions

This report explored the integration of Variational Autoencoders (VAE) in movie recommender systems to address key challenges such as limited diversity, complex user preferences, and the cold-start problem. Through the implementation, it was demonstrated that VAEs can effectively generate meaningful latent representations of users and movies, leading to more personalized and diverse recommendations compared to traditional techniques such as Collaborative Filtering (CF) and Knowledge-Based methods. The experimental results highlight that VAE outperforms other approaches in key performance metrics, including precision, recall, and F1-score, while maintaining competitive error rates in RMSE and MAE. Our visual analyses further confirm that VAEs can capture intricate patterns in user preferences without being overly dependent on explicit similarity metrics. Unlike CF, which struggles with sparsity in user-item interactions, and knowledge-based models, which rely on predefined rules, VAEs offer a flexible, data-driven approach to recommendation generation. In conclusion, leveraging generative AI through VAEs presents a promising direction for enhancing recommender systems by improving recommendation diversity, personalization, and robustness against cold-start issues.