VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

TOURIST ATTRACTION RECOMMENDATION SYSTEMS BASED ON NCF MODELS

By
Duong Tran Nhat Minh
ITDSIU20032

A thesis submitted to the School of Computer Science and Engineering in partial fulfilment of the requirements for the degree of Bachelor of Data Science

Ho Chi Minh City, Vietnam
2025

TOURIST ATTRACTION RECOMMENDATION SYSTEMS BASED ON NCF MODELS

APPROVED BY:

_____ ,

Dr. Nguyen Thi Thanh Sang

THESIS COMMITTEE

ACKNOWLEDGEMENT

I sincerely appreciate Dr. Nguyen Thi Thanh Sang's competent advice. Her dedicated advice and assistance provided me with the ideal conditions to execute this thesis project. Dr. Nguyen Thi Thanh Sang has imparted significant professional information while inspiring and motivating me throughout the learning and research processes.

I am grateful to all the teachers and lecturers who have taught and guided me over my academic career. Their skills, experience, and dedication have given me confidence and ability to succeed.

Lastly, I appreciate the thesis evaluation committee for their valuable work in examining and grading my thesis.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Tourism is a key driver of economic growth and cultural exchange, and Vietnam’s diverse landscapes and heritage offer great potential for tourism development. Many tourists have trouble finding the right attractions for them, which limits their experience and makes lesser-known places less visible. Using tour guides and travel blogs is inefficient and unreliable; therefore, we need smart, data-driven solutions. This project extends prior work with content-based and collaborative filtering as foundations to construct TravelMate, an advanced machine learning recommendation system with a Neural Collaborative Filtering (NCF) model. TravelMate identifies complicated user-item relationships by merging multiple variables such as user information, item details, and ratings to provide more accurate suggestions. A rigorous assessment demonstrates the system's capacity to understand user preferences and make relevant recommendations. TravelMate's scalable recommendation engine aims to improve user satisfaction, promote underrepresented regions, and drive growth in Vietnam's tourist industry.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Tourism in Vietnam is growing rapidly and attracts large numbers of both domestic and international tourists every year. Even with this fast growth, there is still a big difference in the tools available to tourists to help them explore the country well.

Because so many people use tour guides, the cost of travel might rise, making it expensive for tourists. Some people seek local guidance, which is often useful but limited and difficult to obtain. Travelers frequently spend time researching activities by reading blogs, participating in forums, and watching YouTube videos. However, these sources can be overwhelming and do not always provide specialized help tailored to everyone's needs and goals.

As a result, the tourism industry began to explore for innovative digital solutions. Without these technologies, many travellers are unable to fully enjoy their journeys.

1.2 Problem Statement

Vietnam's tourism industry faces a main challenge because there is no digital tool that can suggest tourist attractions based on what people like. Travelers frequently struggle to locate appropriate recommendations without the assistance of tour guides or time-consuming research, which can stifle one of Vietnam's most rapidly developing sectors. Unlike other global travel locations, where personalized recommendation systems are becoming common, Vietnamese travellers continue to rely primarily on traditional techniques to organize their trips.

The thesis's goal is to create an app that uses machine learning to recommend destinations for travels. Recommendations will be based on the traveller's profile and interactions with the places they've been. It will also concentrate on using complex filtering techniques, such as deep learning-based neural collaborative filtering (NCF) models, to increase recommendation accuracy.

1.3 Scope

This thesis project focuses on the design and development of a personalized tourist attraction recommendation system created specifically for the tourism market. The research concentrates on leveraging machine learning techniques, primarily content-based filtering, collaborative filtering, and advanced neural collaborative filtering (NCF) to generate accurate recommendations based on user profiles, item attributes, and user-item interaction data.

The research scope includes:

- Developing and testing baseline recommendation models using content-based and collaborative filtering methods as preliminary approaches.
- Designing, implementing, and evaluating a neural collaborative filtering model that integrates multiple user and item features to improve recommendation performance.
- Analysing the effectiveness of the proposed system in providing personalized attraction suggestions that enhance user experience.
- Focusing exclusively on attraction recommendations without extending to other travel-related services such as accommodation, transportation, or dining.
- Tailoring the system to the cultural and tourism context of Vietnam, considering local preferences and challenges.

1.4 Objective

The main goal of this research is to create TravelMate, an application that recommends tourist attractions based on what each traveller likes. Using machine learning and travellers' reviews, the application tries to give accurate and useful recommendations that overcome the problems found in traditional travel planning.

The purpose of this research is to advance recommendation technology by applying neural collaborative filtering models. They improve the system's ability to study users' actions and guess their preferences, so the suggestions are more accurate.

Besides, TravelMate is working to make a positive impact on Vietnam's tourism industry. With its easy-to-use and personalized tool, the program motivates travellers to explore more places and help local businesses which leads to a fairer spread of tourism. It can assist local economies, create new opportunities for communities and encourage the growth of tourism over time.

.

CHAPTER 2

RELATED WORK

In tourism, personalized recommendation systems are vital because they assist travellers enjoy their trips by suggesting what they like. For this topic, a good example is the **Tourist Attractions Recommender System for Bangkok, Thailand** [1] which integrates collaborative filtering and content-based filtering. The system described in The Tourist Attractions Recommender System for Bangkok, Thailand on ResearchGate helps tourists find interesting places in Bangkok based on their previous visits and likes.

In the same way, the project aims to build a personalized recommendation system called TravelMate that uses machine learning to suggest travel sites to tourists based on their preferences and ratings. The goal of both systems is to make planning trips easier by suggesting useful attractions.

However, TravelMate goes further than regular hybrid filtering by using deep learning-based neural collaborative filtering (NCF) models. They can identify difficult and non-linear changes in user actions that regular filtering misses. Besides, the NCF approach in TravelMate allows the system to adjust to changing user preferences in real time, even when some user-item interaction data is missing. As a result, the recommendations become more accurate, varied and can be used for larger groups.

In contrast, the Bangkok system's mix may struggle to deal with extensive, scattered data and adjust to the different ways people use it. With limited user feedback, simple algorithms can have difficulty recommending items accurately.

Because of this, TravelMate is especially useful because it uses neural collaborative filtering, making its recommendation system more adaptable and intelligent. On the other hand, these advanced models can be more complex and require more data to use

their benefits. In contrast, the Bangkok system is simpler and requires less computing power which could make it more suitable for places with fewer resources.

In general, both projects offer helpful ideas for personalized tourism recommendations, but TravelMate stands out by using deep learning models to fix the issues of traditional filters and give users a better and more flexible experience.

CHAPTER 3

THEORETICAL BACKGROUND

3.1 Recommendation System

Recommendation system is a software tool that provides suggestions to users by analysing their preferences, behaviours, and interactions. Its primary goal is to help users by filtering and presenting items that are most relevant to their interests. Recommendation systems are widely used across various industries, including e-commerce, streaming services, social media, and tourism.

There are several common approaches to recommendation systems. Content-based filtering recommends items similar to those a user has liked before by examining item features. Collaborative filtering, in contrast, identifies patterns by analysing preferences and behaviours of many users to suggest items favoured by similar users. Hybrid methods combine both approaches to improve accuracy and overcome issues such as the cold-start problem, where new users or items lack sufficient data.

Recent improvements in machine learning and deep learning have made these systems much better. With neural collaborative filtering, neural networks are used to understand the relationships between users and items, leading to better and more flexible recommendations. They make it easier for users by recommending things that are likely to interest them and by helping them find items they might not have noticed. For instance, e-commerce sites help sell more by recommending products that match users' interests, streaming services improve engagement by adjusting content and tourism apps suggest places that appeal to visitors.

There are still some issues, for example, dealing with data that is not very abundant, making the system scalable and handling privacy matters. Even with

these issues, recommendation systems are still vital for making digital platforms more personal and efficient.

3.2 Baseline Model 1: Content – Based Filtering Model

Content-based filtering is a recommendation system technique that suggests items to users based on the attributes of the items they have previously interacted with. The idea is that users will prefer items that have features similar to those they have rated highly or interacted with before. Content-based filtering focuses on the features of items and the individual user's interactions, offering a highly personalized experience. [3]

Key Concepts in Content-Based Filtering

- **Item Features**: Items in a recommendation system are defined by a set of features, which are essential for making recommendations. For example, in a tourism recommendation system, features could include location, type (e.g., museum, park), user ratings, and popularity. The more detailed the features, the better the system can tailor recommendations.
- **User Profiles**: A user profile reflects the preferences of an individual user, typically derived from their past interactions with items. The profile is built by analysing the features of items they have interacted with. It is updated over time to reflect changing preferences.

Process of Building a Content-Based Filtering Model

- **Feature Extraction**: The first step is to extract relevant features for all items, which are then stored in a feature matrix. For tourist attractions, this could include location, category, user ratings, and accessibility.
- **User Profile Construction**: User profiles are built based on past interactions. This is represented as a vector, with each element reflecting the user’s interest in specific features.
- **Similarity Calculation**: The system calculates the similarity between the user profile and item profiles using similarity measures like **cosine**

similarity or **Euclidean distance**. Items with the highest similarity scores are selected for recommendation.

- **Recommendation Generation**: Items that show the highest similarity to the user's profile are recommended.

Advantages and Limitations

Content-based filtering uses what people like to offer personalized recommendations, making it a good choice for suggesting new things. Still, it can lead to users seeing only similar items which cuts down on the range of options. The model's effectiveness is influenced by the quality of the item features. The system also faces challenges with new items, known as the **cold-start problem**, where no interaction history is available. [3]

In conclusion, while content-based filtering provides highly personalized recommendations, it is important to address its limitations, such as over-specialization and cold-start issues, to enhance its applicability in real life scenario.

3.3 Baseline Model 2: Collaborative Filtering Model

Collaborative filtering uses the preferences of other users to predict what a user might like. It assumes that if users have enjoyed the same things before, they will probably enjoy similar things in the future. Unlike content-based filtering, collaborative filtering uses how users interact with items and their common behaviour to suggest personalized recommendations. [4]

Key Concepts in Collaborative Filtering

- **User-Item Interaction**: Collaborative filtering relies on the interactions between users and items, such as ratings, clicks, purchases, or views. These interactions are used to determine relationships between users or between items, providing the foundation for recommendations.
- **User Profiles**: Instead of building a profile based on item features, collaborative filtering creates user profiles based on their interactions with items. These profiles are constructed by identifying patterns in user

preferences—such as which items a user has rated or interacted with most frequently.

Process of Building a Collaborative Filtering Model

- **Data Collection**: The first step is gathering data on user-item interactions. This typically takes the form of a user-item matrix, where rows represent users and columns represent items, with entries indicating the interactions (e.g., ratings, views).
- **Similarity Calculation**: The system calculates similarities between users or items. In **user-based collaborative filtering**, the similarity between users is determined by comparing their interaction history. In **item-based collaborative filtering**, the system compares items based on user interactions, recommending items that similar users have liked.
- **Prediction Generation**: Once similarities are calculated, the system predicts a user's preferences for items they haven't yet interacted with, based on the preferences of similar users or the similarity between items.
- **Recommendation Generation**: The system generates recommendations by selecting items with the highest predicted ratings or most likely to be preferred by the user, based on the similarities calculated.

Advantages and Limitations

Collaborative filtering is good at finding recommendations for users by comparing their preferences with those of similar people which helps find new and different items. It does not depend on knowing the features of items which is helpful when those features are not easy to define. Still, the cold-start problem exists, making it difficult to recommend new users or items that have not interacted much. Also, it may not work well when data is sparse, since the user-item interaction matrix is often missing values which affects the accuracy of predictions. [4]

In short, collaborative filtering relies on data from many users to help provide personalized suggestions. Although it is good at finding new things and using fewer item-specific features, it still needs to deal with issues such as cold-start problems and sparse data to be more effective in real-world settings.

3.4 Neural Collaborative Filtering Model

Neural Collaborative Filtering (NCF) is a recommendation system model that leverages deep learning techniques to capture complex, non-linear relationships between users and items. Unlike baseline filtering methods, which rely on linear similarity calculations, NCF uses neural networks to model intricate patterns in user-item interactions. This allows NCF to provide more accurate and personalized recommendations by learning from large-scale, sparse datasets. [5],[6]

Key Concepts in Neural Collaborative Filtering

- **User-Item Interactions**: Like traditional collaborative filtering, NCF is built on the concept of user-item interactions. These interactions, such as ratings, views, or purchases, form the foundation for the recommendation process. The difference lies in how NCF models these interactions: using deep neural networks to learn latent patterns in the data rather than relying on simple similarity metrics.
- **Embedding Layers**: In NCF, both users and items are represented as dense vectors through embedding layers. Each user and item is mapped to a low-dimensional vector space where similar users or items are positioned closer together. These embeddings are learned during the training process, allowing the model to capture latent features that affect user preferences.
- **Non-linear Activation Functions**: NCF introduces non-linear activation functions into the model, enabling it to learn complex relationships between users and items that traditional linear models cannot capture. This makes the model more capable of handling varied and evolving user behaviour.

Architecture of Neural Collaborative Filtering

The architecture of Neural Collaborative Filtering typically involves three primary components:

- **Embedding Layer**: The embedding layer transforms the user and item indices into dense vectors. For each user and item, a corresponding embedding vector is learned, capturing their latent features.
- **Multilayer Perceptron (MLP)**: After the embeddings are obtained, they are fed into a Multi-Layer Perceptron (MLP), which consists of several hidden layers. The MLP processes the combined user and item embeddings, learning complex, non-linear relationships between users and items.
- **Output Layer**: The output layer produces the predicted rating or preference score for a given user-item pair. This score represents the likelihood of the user interacting with the item, and it is typically passed through a sigmoid or SoftMax activation function depending on the task (e.g., rating prediction or binary classification for click prediction).

Basic Model Diagram

The basic architecture of Neural Collaborative Filtering can be visualized as follows:

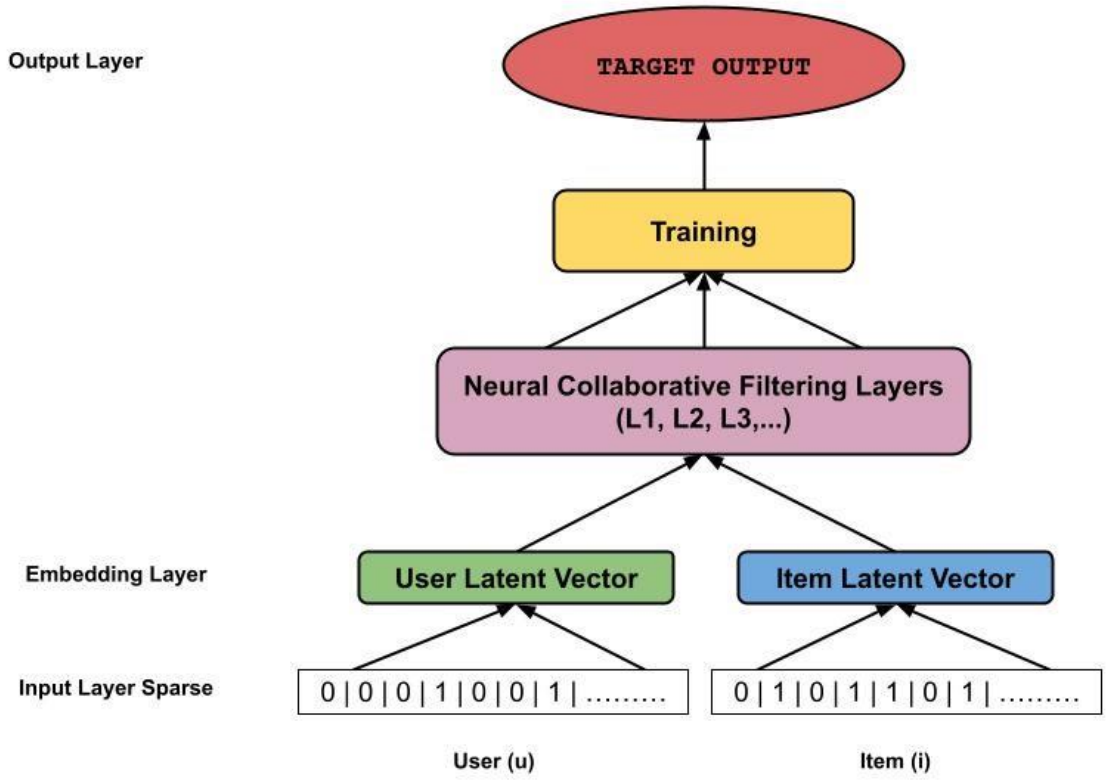


Figure 1: Neural Collaborative Filtering Model Diagram

Advantages and Limitations of Neural Collaborative Filtering

Advantages:

- **Non-linear Relationship Learning**: NCF can model complex, non-linear interactions between users and items that traditional methods (like matrix factorization) cannot capture.
- **Scalability**: NCF can efficiently handle large datasets due to its deep learning framework, making it well-suited for high-dimensional and sparse data.

- **Adaptability**: As user preferences evolve, NCF can adapt more effectively by learning dynamic and context-aware embeddings.

Limitations:

- **Computational Complexity**: NCF requires substantial computational resources for training, especially with large datasets, as it involves deep neural networks.
- **Data Sparsity**: While deep learning models can mitigate sparsity to some extent, NCF still faces challenges when there is insufficient data for new users or items.
- **Cold-Start Problem**: New users or items with limited interaction data may not have well-defined embeddings initially, reducing the model's ability to make accurate recommendations for them.

3.5 Models Comparison Table

Table 1 Baseline Models and NCF model Comparison

Aspect	***Content-Based Filtering***	***Collaborative Filtering***	***Neural Collaborative Filtering***
Methodology	Matches item features with user profiles	Finds patterns in user-item interactions	Learns non-linear relationships with deep learning
Data Dependency	Requires detailed item attributes	Requires user-item interaction data	Requires user-item interaction data
Cold Start Problem	Struggles with new users/items	Struggles with new users/items	Same as collaborative filtering
Flexibility	Limited to feature similarity	Limited by linear patterns	Can model complex, non-linear patterns
Scalability	Suitable for small datasets	Scalable to large datasets	Highly scalable with large datasets
Personalization	High (based on user preferences)	Moderate (relies on group behaviour)	High (learns user-item interactions deeply)
Accuracy	Moderate	Moderate to high	High

CHAPTER 4

METHODOLOGY

4.1 Theoretical Models

4.1.1 Recommendation System

The purpose of this research is building and optimizing a Neural Collaborative Filtering (NCF) model to deliver personalized recommendation system for tourism. The NCF model represents users and items as low-dimensional embeddings that capture latent features, enabling the neural network to learn complex, non-linear interactions between them. Initially, the model is trained using only user-item rating data to predict preferences and recommend item to users.

The model is improved by including more user and item details to make personalization and predictions more accurate. Using this approach, the model can better represent what individuals like and the situation they are in.

To optimize, we can try different ways to prepare and transform your data and you can also adjust the model's hyperparameters such as the size of the embedding, the network's design, the learning rate and regularization. They make the model better at using what it has learned and prevent it from overfitting.[6], [7]

4.1.2 TravelMate Prototype

After developing the recommendation model, a prototype will be created as a web application to demonstrate the system's functions. The prototype will feature basic user authentication, including sign-up and login capabilities, enabling personalized user sessions. User data will be collected through the website and

securely stored in a PostgreSQL database. The recommendation model will be integrated with the website backend, allowing real-time generation of personalized attraction suggestions based on each user's data. The website will be made using Streamlit which ensures a straightforward and interactive experience for users. The prototype allows us to check how well the model works and how easy it is to use in practice.

4.2 Applying Baseline Models

4.2.1 Content-Based Filtering Model

In this project, the content-based filtering model uses information about tourist attractions such as their categories and quality scores, together with user ratings to suggest personalized recommendations. Every attraction is described by a feature vector that includes its category and quality metrics, with the quality metrics weighted to show their importance

User profiles are constructed by aggregating the feature vectors of attractions each user has rated, weighted by their ratings. This profile captures the user's preferences in terms of attraction categories and quality.

The recommendation process calculates cosine similarity between user profiles and item feature vectors, identifying attractions that align closely with a user's past preferences. Items with the highest similarity scores are recommended to the user.

Evaluation of the model compares predicted preferences based on similarity scores with actual user ratings, using metrics such as root mean squared error (RMSE) and accuracy.

4.2.2 Collaborative Filtering Model

The collaborative filtering model in this project is designed to recommend tourist attractions based on patterns in user-item interactions. It uses the user-item ratings dataset to identify similarities between items by analysing how users have rated different attractions.

The core of this model involves creating a user-item matrix, where each row represents a user and each column represents an item, the entries are normalized user ratings. Using this matrix, the model calculates item-item similarities through cosine similarity, enabling it to identify attractions that are rated similarly by users.

Predictions for unseen items are made by estimating a user's rating as a weighted average of their ratings for similar items, with the weights derived from item similarity scores. This item-item approach leverages the collective behaviour of users to suggest attractions highly rated by users with similar tastes.

The model's effectiveness is assessed by comparing predicted ratings against actual ratings using metrics such as root mean squared error (RMSE), mean absolute error (MAE), and accuracy.

4.3 Neural Collaborative Filtering Model

4.3.1 Model Architecture

The model is based on Neural Collaborative Filtering (NCF) and uses user attribute information to predict how users will rate items. It uses embedding layers for categorical data and dense layers for continuous data which allows the model to learn complicated relationships between features.

1. Input Layers

- **User ID Input:** Receives a single integer representing the user, which is mapped into a dense vector through an embedding layer.
- **Item ID Input:** Receives a single integer representing the item, also mapped into a dense vector via an embedding layer.

- **User Numeric Features Input:** A vector containing continuous user attributes such as Age, Gender (binary numeric), and Budget.
- **User Categorical Feature Input (GroupComp):** An integer representing the user's categorical group, fed into a dedicated embedding layer.

2. Embedding Layers

- **User Embedding:** Transforms the discrete user ID into a dense latent vector of fixed size (embedding dimension), capturing latent user preferences.
- **Item Embedding:** Similarly transforms the item ID into a latent vector representing item characteristics.
- **GroupComp Embedding:** Transforms the categorical GroupComp attribute into a dense vector, enabling the model to learn meaningful representations of user groups beyond simple one-hot encodings.

3. Flatten Layers

Each embedding output is flattened from a 2D tensor to a 1D vector to enable concatenation with other features.

4. Feature Concatenation

The flattened user embedding, item embedding, weighted user numeric features, and GroupComp embedding are concatenated into a single feature vector. This combined vector represents all relevant information for predicting the user's rating for the item.

5. Fully Connected Dense Layers

The concatenated features are passed through two fully connected (dense) layers:

- The first dense layer with 128 neurons and ReLU activation captures complex, non-linear feature interactions.

- A dropout layer follows to reduce overfitting by randomly disabling neurons during training.
- The second dense layer with 64 neurons and ReLU activation further refines feature interactions, followed by another dropout layer.

6. Output Layer

A final dense layer with a single neuron and linear activation outputs the predicted rating as a continuous value, matching the normalized rating scale.

7. Training Configuration

- The model is trained using the Mean Squared Error (MSE) loss function, suitable for regression tasks predicting continuous ratings.
- The Adam optimizer is used for efficient gradient-based optimization.

4.3.2 Model Diagram

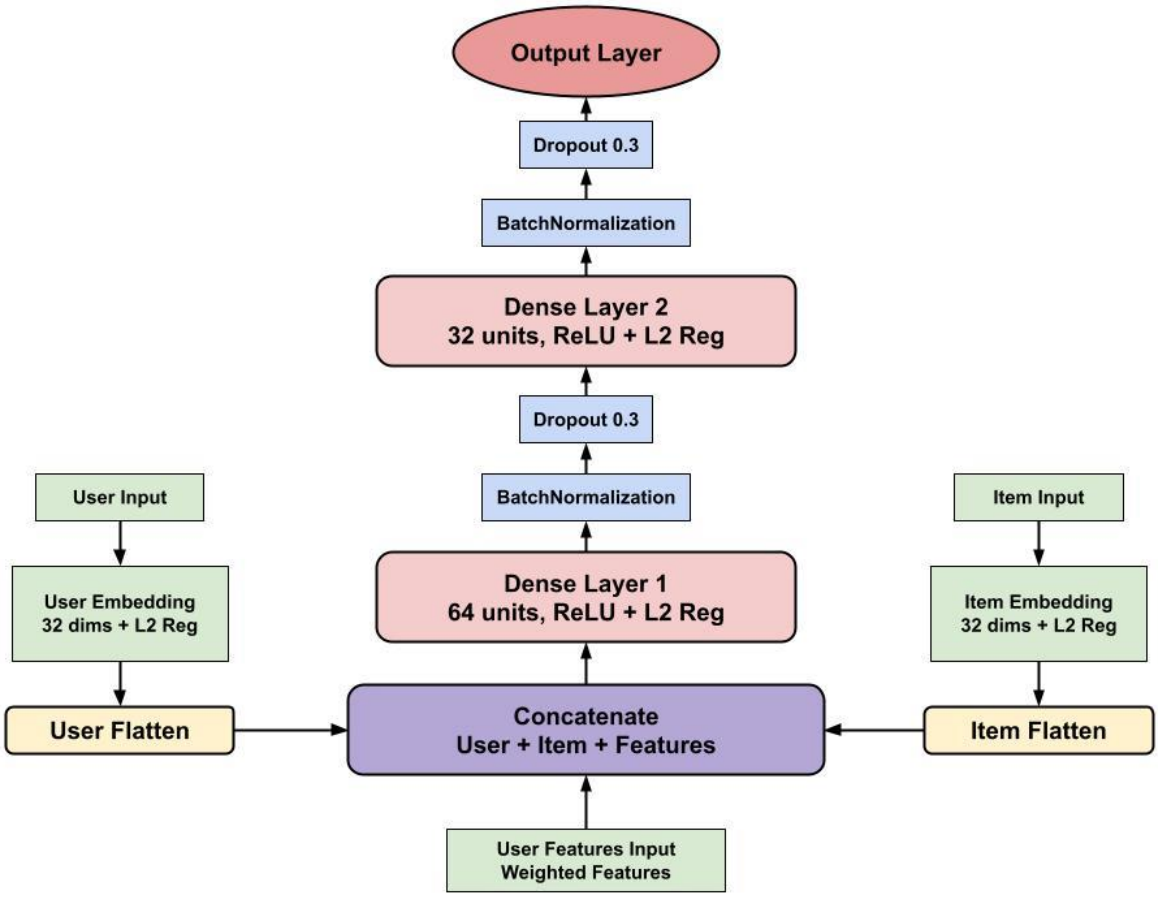


Figure 2: TravelMate NCF Model Diagram

The diagram shows the architecture of the Neural Collaborative Filtering (NCF) built for this thesis project application - TravelMate.

It uses three inputs: a User ID and Place ID (both turned into 32-dimensional embeddings to show hidden behavioural patterns) and weighted user demographic features such as age, gender, budget and group composition preferences.

The user and place embeddings are merged and then combined with the pre-processed demographic features to create a single representation that uses both collaborative and content-based filtering.

The combined feature vector is sent to a two-layer neural network with ReLU activations, batch normalization and dropout regularization (64→32 units) which then produces a linear output that predicts the rating a user would give to the place.

4.3.3 Model Optimization

There are 5 main steps in the optimization process for this NCF model that help improve both performance and training stability. [5], [6], [7]

First, try to make the embedding size smaller. When embeddings are smaller, the model is simpler which can make training faster and less likely to overfit. We will also place batch normalization layers after the dense layers. Batch normalization stabilizes training and speeds it up by normalizing the inputs in each mini-batch which improves the flow of gradients and reduces internal covariate shift.

After that, we need to enhance the model's L2 regularization specifically for the embedding layers. Which will penalize large weights, encouraging the model to learn simpler, more generalizable features and thus reducing overfitting.

Thirdly is adjusting the number and size of units in the dense layer architecture. This change aims to streamline the model, balancing capacity and efficiency to improve learning without unnecessary complexity.

And then, by increasing the optimizer's learning rate, the model can speed up and learn faster, though it must be carefully tuned to avoid instability.

Finally, early stopping is added into the training configuration. Early stopping monitors validation performance and halts training when improvements plateau, preventing overfitting and saving computational resources.

These optimizations result in a more efficient, faster-training, and more robust model.

CHAPTER 5

IMPLEMENTATION

5.1 Recommendation System Implementation

5.1.1 Data Preparation

Loading Datasets

The model utilizes two primary datasets from Kaggle: Synthetic Tourism Dataset for Recommender Systems. [2]

- **User-Item Interaction Data (rats.csv)**: Contains user ratings of items, serving as the core interaction matrix.
- **User Features Data (ufeat.csv)**: Contains demographic and profile attributes of users, providing additional information to enrich user representations.

Selecting Relevant User Attributes

From the user features dataset, only four attributes were selected based on their predictive relevance:

- **Age** (numeric)
- **Gender** (categorical binary)
- **Budget** (numeric)
- **GroupComp** (categorical nominal)

This selection ensures focus on key variables that influence user preferences without introducing noise from irrelevant features.

Handling Missing Values

Missing values in the selected user features are filled with zeros to maintain dataset integrity and avoid errors during model training.

Encoding Categorical Features

- **Gender** is converted into a binary numeric variable, mapping "Male" to 1 and "Female" to 0.
- **GroupComp**, the original nominal categorical variable with multiple categories is changed into a dense embedding using a specific method. Before that, the data is converted into integer indices for each category, so the model can learn a detailed vector for every category.

Encoding User and Item Identifiers

User and item IDs in the interaction dataset are label encoded into continuous integer indices. This encoding facilitates efficient embedding lookup and numerical processing within the neural network.

Normalizing Rating Values

The rating scale is normalized to the range [0, 1] by dividing all ratings by the maximum rating value. This normalization stabilizes training by bounding the output values and improving numerical conditioning.

Merging User Features with Interaction Data

The user features are merged with the user-item interaction data based on user identifiers. This integration ensures that each interaction record is accompanied by the corresponding user attribute values required for model input.

Separating Numerical and Categorical Features for Model Input

User features are split into two groups for input to the model:

- **Numerical features:** Age, Gender, and Budget are passed as a numeric feature vector.

- **Categorical feature:** GroupComp is passed separately as an integer index to be input into an embedding layer.

Train-Test Splitting

The processed dataset is randomly split into training and testing subsets (commonly 80% training, 20% testing) to enable unbiased evaluation of model performance.

Conversion to Numeric Arrays with Explicit Data Types

All inputs (user IDs, item IDs, numeric features, categorical indices, and ratings) are converted into NumPy arrays with explicit data types (int32 for categorical and ID inputs, float32 for continuous features and labels). This step ensures compatibility with TensorFlow's model input requirements and computational efficiency.

5.1.2 Model Implementation

After data preparation steps mentioned above, this is the main process of implementing the Neural Collaborative Filtering model with multiple attributes from two datasets provided.

Distribute weights for each attribute in the model:

```
# Apply weights
# First find indices of each attribute in X_user_features
age_idx = feature_cols.index('Age')
gender_idx = feature_cols.index('Gender')
budget_idx = feature_cols.index('Budget')

groupcomp_indices = [i for i, c in enumerate(feature_cols) if c.startswith('GroupComp_')]

n_groupcomp = len(groupcomp_indices)
if n_groupcomp == 0:
    raise ValueError("No GroupComp columns found after one-hot encoding.")

# Define weights per feature
weights = np.ones(X_user_features.shape[1], dtype=np.float32) * 0.0
weights[age_idx] = 0.3
weights[gender_idx] = 0.3
weights[budget_idx] = 0.2
# Distribute GroupComp total weight 0.2 evenly
for idx in groupcomp_indices:
    weights[idx] = 0.2 / n_groupcomp

# Apply weights
X_user_features_weighted = X_user_features * weights
```

Building the NCF model:

The model takes three inputs: user IDs, item (place) IDs, and a vector of user features. Each user and item ID is converted into a dense embedding vector with regularization

The embeddings are flattened and concatenated together with the user features into a combined input. This combined vector is then passed through two fully connected layers with ReLU activations and dropout for regularization

The model will then output a single continuous value using a linear activation

```
def build_ncf_model(num_users, num_items, user_feat_dim, embedding_size=50):
    user_input = Input(shape=(1,), name='user_input')
    place_input = Input(shape=(1,), name='place_input')
    user_features_input = Input(shape=(user_feat_dim,), name='user_features_input')

    user_embedding = Embedding(num_users, embedding_size,
                               embeddings_regularizer=tf.keras.regularizers.l2(1e-6),
                               name='user_embedding')(user_input)
    place_embedding = Embedding(num_items, embedding_size,
                                embeddings_regularizer=tf.keras.regularizers.l2(1e-6),
                                name='place_embedding')(place_input)

    user_flat = Flatten()(user_embedding)
    place_flat = Flatten()(place_embedding)

    # Concatenate embeddings and weighted user features
    combined = Concatenate()([user_flat, place_flat, user_features_input])

    x = Dense(128, activation='relu')(combined)
    x = Dropout(0.5)(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(1, activation='linear')(x)

    model = Model(inputs=[user_input, place_input, user_features_input], outputs=output)
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model
```

Finally, the model is trained with inputs including user IDs, item IDs, and user features, and evaluated using MSE loss and RMSE metrics.

```
def evaluate_model(model, X_test_list, y_test):
    y_pred = model.predict(X_test_list)
    rmse = np.sqrt(np.mean((y_test - y_pred.flatten())**2))

    y_pred_bin = (y_pred.flatten() >= 0.5).astype(int)
    y_test_bin = (y_test >= 0.5).astype(int)

    auc_roc = roc_auc_score(y_test_bin, y_pred_bin)
    accuracy = np.mean(y_pred_bin == y_test_bin)

    print(f"RMSE: {rmse:.4f}")
    print(f"AUC-ROC: {auc_roc:.4f}")
    print(f"Accuracy: {accuracy:.4f}")

    return accuracy, rmse, auc_roc
```

5.1.3 Model Optimization

Detailed steps in the optimization process:

In the model architecture, reduce the embedding size from 50 to 32 – changed the default parameter in build_ncf_model() function. Also, Add BatchNormalization() after each Dense layer to improve training stability

```
# Concatenate embeddings and weighted user features
combined = Concatenate()([user_flat, place_flat, user_features_input])

x = Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(1e-6))(combined)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)
x = Dense(32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(1e-6))(x)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)
output = Dense(1, activation='linear')(x)
```

Enhance L2 regularization for Embeddings

Original:

```
user_embedding = Embedding(num_users, embedding_size,
                           embeddings_regularizer=tf.keras.regularizers.l2(1e-6),
                           name='user_embedding')(user_input)
place_embedding = Embedding(num_items, embedding_size,
                            embeddings_regularizer=tf.keras.regularizers.l2(1e-6),
                            name='place_embedding')(place_input)
```

Optimized:

```
user_embedding = Embedding(num_users, embedding_size,
                           embeddings_regularizer=tf.keras.regularizers.l2(1e-5),
                           name='user_embedding')(user_input)
place_embedding = Embedding(num_items, embedding_size,
                            embeddings_regularizer=tf.keras.regularizers.l2(1e-5),
                            name='place_embedding')(place_input)
```

Complete Dense Layer Architecture Overhaul

Key changes are:

- **Architecture**: 128→64→1 ➜ 64→32→1 (more efficient)

- **L2 Regularization**: Added kernel_regularizer=tf.keras.regularizers.l2(1e-6) to Dense layers
- **Batch Normalization**: Added BatchNormalization() after each Dense layer
- **Dropout**: Reduced from 0.5 to 0.3

Original:

```
x = Dense(128, activation='relu')(combined)
x = Dropout(0.5)(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(1, activation='linear')(x)
```

Optimized:

```
x = Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(1e-6))(combined)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)
x = Dense(32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(1e-6))(x)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)
output = Dense(1, activation='linear')(x)
```

Increase optimizer learning rate

```
model = Model(inputs=[user_input, place_input, user_features_input], outputs=output)
model.compile(optimizer=Adam(learning_rate=0.002), loss='mse')
return model
```

5.2 TravelMate Prototype Implementation

5.2.1 Web Application Development

The web application is built using Streamlit in **app_clean.py**. It handles UI rendering, user interaction, and integrates the recommendation engine.

- Page configuration and session state initialization:

```
# Set page configuration
st.set_page_config(
    page_title="Tourism Recommender System",
    page_icon="🌍",
    layout="wide"
)

# Initialize session state
if 'authenticated' not in st.session_state:
    st.session_state['authenticated'] = False
if 'user_id' not in st.session_state:
    st.session_state['user_id'] = None
if 'username' not in st.session_state:
    st.session_state['username'] = None
```

- User authentication UI with login and registration forms using Streamlit forms:

```
# Authentication UI
if not st.session_state['authenticated']:
    tab1, tab2 = st.tabs(["Login", "Register"])

    with tab1:
        with st.form("login_form"):
            st.subheader("Login")
            username = st.text_input("Username")
            password = st.text_input("Password", type="password")
            submit_login = st.form_submit_button("Login")

            if submit_login:
                if not username or not password:
                    st.error("Please enter both username and password")
                else:
                    success, result = login_user(username, password)
                    if success:
                        st.session_state['authenticated'] = True
                        st.session_state['user_id'] = result
                        st.session_state['username'] = username
                        st.success("Login successful!")
                        st.rerun()
                    else:
```

```
                    st.error(result)

with tab2:
    with st.form("register_form"):
        st.subheader("Register")
        new_username = st.text_input("Username")
        email = st.text_input("Email")
        new_password = st.text_input("Password", type="password")
        confirm_password = st.text_input("Confirm Password", type="password")

        # Optional profile information
        st.subheader("Profile Information (Optional)")
        age = st.slider("Age", 10, 100, 30)
        gender = st.selectbox("Gender", ["Male", "Female", "Other"])
        budget = st.number_input("Budget", 0, 10000, 500)
        group_comp = st.selectbox("Group Composition",
                                  ["Solo", "Family", "Couple", "Friends", "Other"])

        submit_register = st.form_submit_button("Register")


         if submit_register:
            if not new_username or not email or not new_password:
                st.error("Please enter username, email, and password")
            elif new_password != confirm_password:
                st.error("Passwords don't match")
            else:
                success, result = register_user(
                    new_username,
                    email,
                    new_password,
                    age=age,
                    gender=gender,
                    budget=budget,
                    group_comp=group_comp
                )
                if success:
                    st.success("Registration successful! Please log in.")
                else:
                    st.error(result)
```

- After authentication, user profile management, rating interface, and personalized recommendations are presented dynamically.

- The app loads datasets, pre-processes them, and integrates user inputs with the recommendation engine (more details below).

5.2.2 Database Architecture and Authentication

The database layer is implemented using **SQLAlchemy ORM** with PostgreSQL, defined in **database.py** and accessed in **auth.py**.

- Database models in database.py:

```
# Define models (duplicate from streamlit_app.py)
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    username = Column(String, unique=True, index=True)
    email = Column(String, unique=True, index=True)
    hashed_password = Column(String)
    age = Column(Integer, nullable=True)
    gender = Column(String, nullable=True)
    budget = Column(Float, nullable=True)
    group_comp = Column(String, nullable=True)
    created_at = Column(String, default=lambda: datetime.datetime.now().isoformat())

    # Relationships
    ratings = relationship("Rating", back_populates="user")


class Place(Base):
    __tablename__ = "places"

    id = Column(Integer, primary_key=True, index=True)
    place_id = Column(String, unique=True, index=True)
    place_name = Column(String)
    place_id_encoded = Column(Integer, nullable=True)

    # Relationships
    ratings = relationship("Rating", back_populates="place")
```

```
class Rating(Base):
    __tablename__ = "ratings"

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"))
    place_id = Column(Integer, ForeignKey("places.id"))
    rating = Column(Float)
    created_at = Column(String, default=lambda: datetime.datetime.now().isoformat())

    # Relationships
    user = relationship("User", back_populates="ratings")
    place = relationship("Place", back_populates="ratings")
```

- Database setup and initialization is handled by setup_database() (in database.py) which creates the database if missing and creates tables.
- Authentication and user-related database operations in **auth.py**:

```
# Authentication functions
def register_user(username, email, password, age=None, gender=None, budget=None, group_comp=None):
    with get_db() as db:
        # Check if user already exists
        existing_user = db.query(User).filter(User.username == username).first()
        if existing_user:
            return False, "Username already exists"

        existing_email = db.query(User).filter(User.email == email).first()
        if existing_email:
            return False, "Email already exists"


def login_user(username, password):
    with get_db() as db:
        user = db.query(User).filter(User.username == username).first()

        if not user:
            return False, "User not found"

        if not verify_password(password, user.hashed_password):
            return False, "Incorrect password"

        return True, user.id
```

5.2.3 Recommendation Engine Implementation

The recommendation engine uses a Neural Collaborative Filtering (NCF) model built with TensorFlow/Keras inside **app_clean.py**.

- The NCF model architecture:

```
# Build the NCF model
def build_ncf_model(num_users, num_items, user_feat_dim, embedding_size=32):
    """
    Build an optimized Neural Collaborative Filtering model
    with reduced complexity and improved regularization.
    """
    # Use more efficient embedding size (32 instead of 50)
    user_input = Input(shape=(1,), name='user_input')
    place_input = Input(shape=(1,), name='place_input')
    user_features_input = Input(shape=(user_feat_dim,), name='user_features_input')

    # Add stronger regularization for embeddings
    user_embedding = Embedding(num_users, embedding_size,
                               embeddings_regularizer=tf.keras.regularizers.l2(1e-5),
                               name='user_embedding')(user_input)
    place_embedding = Embedding(num_items, embedding_size,
                                embeddings_regularizer=tf.keras.regularizers.l2(1e-5),
                                name='place_embedding')(place_input)
```

```
user_flat = Flatten()(user_embedding)
place_flat = Flatten()(place_embedding)

# Add batch normalization for feature inputs to improve training
user_features_normalized = tf.keras.layers.BatchNormalization()(user_features_input)

# Concatenate embeddings and weighted user features
combined = Concatenate()([user_flat, place_flat, user_features_normalized])

# More efficient network architecture with batch normalization
x = Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(1e-6))(combined)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)  # Lower dropout for better efficiency

x = Dense(32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(1e-6))(x)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)

output = Dense(1, activation='linear')(x)


model = Model(inputs=[user_input, place_input, user_features_input], outputs=output)

# Use Adam optimizer with a slightly higher learning rate and beta parameters
optimizer = Adam(learning_rate=0.002, beta_1=0.9, beta_2=0.999)
model.compile(optimizer=optimizer, loss='mse')

return model
```

- The model is trained on user-place rating data combined with user demographic features (age, gender, budget, group composition).
- Predictions are generated for unrated places, and top personalized recommendations are displayed.

5.2.4 System Integration and Optimization

- The system integrates UI, database, and the recommendation engine seamlessly in **app_clean.py**.

- Database calls and authentication are modularized via **auth.py**.
- The model is cached and loaded efficiently to prevent retraining on every run:

```python
# Cache model to avoid reloading
@st.cache_resource
def load_cached_model(model_path, num_users, num_items, user_feat_dim):
    """Load and cache the model to improve performance"""
    model = build_ncf_model(
        num_users=num_users,
        num_items=num_items,
        user_feat_dim=user_feat_dim
    )
    try:
        model.load_weights(model_path)
        print("Model loaded from cache")
    except:
        print("No cached model found")
    return model
```

- User ratings are saved and fetched dynamically, allowing incremental updates.
- User profile updates dynamically affect recommendation results.
- The system applies data preprocessing, caching, and early stopping during model training for performance and to avoid overfitting.
- User interface elements update reactively based on session state and database content.

5.3 Tools Used

5.3.1 Python

In implementing the NCF model, the python libraries below are used [10], [11]

NumPy

A fundamental library for numerical computing in Python, used here for array manipulation and efficient numerical operations.

Pandas

A powerful data analysis and manipulation library, used to load, clean, and preprocess tabular datasets.

TensorFlow (KerasAPI) [10]

TensorFlow is a comprehensive open-source library developed by Google for machine learning and deep learning applications. In this model, TensorFlow (specifically its high-level Keras API) is used to:

Build the Neural Network Architecture: It provides layers like Embedding, Dense, Flatten, Concatenate, and Dropout to construct the Neural Collaborative Filtering model, enabling the learning of complex user-item interactions and feature representations.

Model Training and Optimization: TensorFlow handles the backpropagation and optimization process using optimizers like Adam, facilitating efficient gradient-based training of the model weights.

Prediction and Evaluation: After training, TensorFlow enables fast inference to predict user ratings on test data.

TensorFlow's flexibility and scalability make it well-suited for implementing and experimenting with sophisticated neural models in recommendation systems.

Scikit-learn

scikit-learn is a versatile Python library for traditional machine learning tasks and data preprocessing. In this project, it supports:

Data Preparation: Utilities such as LabelEncoder convert categorical user and item IDs into numeric indices suitable for embedding layers in TensorFlow.

Data Splitting: The train_test_split function divides the dataset into training and testing subsets, ensuring unbiased evaluation.

Model Evaluation Metrics: Functions like accuracy_score and roc_auc_score assess the quality of predictions, providing interpretable performance metrics alongside regression loss.

scikit-learn complements TensorFlow by offering robust tools for preprocessing and evaluation, essential for building reliable and effective machine learning pipelines.

5.3.2 Streamlit Framework

Streamlit is an open-source Python framework designed to quickly build and deploy interactive web applications, especially for data science and machine learning projects. It allows developers to create user-friendly interfaces with minimal code, enabling easy visualization and exploration of data and models. [8]

Key functions of Streamlit include:

- Simplified UI creation using intuitive Python scripts without needing front-end expertise.
- Support for displaying various media types like charts, maps, images, and videos.
- Real-time interactivity with widgets such as sliders, buttons, and input fields to capture user inputs.
- Automatic updates of the app interface upon code or data changes.
- Easy integration with popular Python libraries like Pandas, Matplotlib, Plotly, and TensorFlow.
- Overall, **Streamlit** streamlines the process of turning data science workflows and machine learning models into web applications efficiently.

5.3.3 PostgreSQL

PostgreSQL is a well-known RDBMS that is open-source, strong and follows industry standards. It can handle different data types, complex queries and transactions which makes it useful for everything from basic websites to large data warehouses. [9]

Key functions of PostgreSQL include:

- Storing, organizing, and managing structured data efficiently using SQL.
- Supporting complex queries, joins, and indexing for fast data retrieval.
- Ensuring data integrity and consistency through ACID-compliant transactions.
- Extensibility with custom functions, data types, and support for procedural languages.
- Advanced features like full-text search, JSON support, and geospatial data handling.

Compared to other databases, PostgreSQL offers advantages such as superior compliance with SQL standards, a rich feature set, and strong community support. Its flexibility allows developers to customize and extend the database to fit specific needs, while maintaining reliability and scalability.

CHAPTER 6

RESULT

6.1 Baseline Models

Content – Based Filtering Model

Statistics:

- Accuracy: 54.69%
- RMSE: 2.1577
- Precision: 1.0
- Recall: 0.6452
- F1 Score: 0.7884

Collaborative Filtering Model

Statistics:

- Accuracy: 91.09%
- RMSE: 0.2967
- Precision: 0.4223
- Recall: 0.3309
- F1 Score: 0.3711

Baseline Models Evaluation

The results of the evaluation indicate that the models perform differently, as shown by their accuracy and RMSE (Root Mean Squared Error). The findings are the following:

Content-Based Filtering Model: The accuracy rate was 54.69% and the RMSE was 2.16. Regardless of efforts to optimize the model by adjusting feature weights, it was

not able to provide accurate recommendations. The relatively low accuracy indicates that the model does not effectively capture user preferences or the relationships between users and places.

Collaborative Filtering Model: The model achieved an accuracy of 91.79% and an RMSE of 0.29. By analysing user-item data, the model revealed patterns and similarities, allowing it to make much better predictions.

Temporary Conclusion

Testing has shown that the collaborative filtering model is more effective than the content-based model. Although the content-based approach includes item features such as categories and quality, its low accuracy and high error mean it is not ready for immediate use. Even after making changes to the model and using advanced techniques, the performance is still not high enough for a real recommendation system.

So, we are going to concentrate on the collaborative filtering model to guide our future work. Because it is highly accurate and has low error, it forms a solid foundation for developing a more advanced system.

6.2 Neural Collaborative Filtering Model

The Neural Collaborative Filtering Model (NCF) was developed using two datasets: rats.csv – users and items interactions data, ufeat.csv – user's features including age, gender, budget, group combinations. [2]

After implementing and adjusting parameters for an optimized model, these are the model's evaluation statistics:

- Accuracy: 79,07%
- RMSE: 0.1585
- Precision: 0.7714
- Recall: 0.8005
- F1 Score: 0.7857

Models' evaluation metrics comparison:

Table 2: Models' Results Comparison

Metric	Content-Based Filtering	Collaborative Filtering	NCF – Before Adjustments	NCF – After Adjustments
Accuracy	54.69%	91.09%	79.51%	79.07%
Precision	1.0	0.4223	0.7615	0.7714
Recall	0.6452	0.3309	0.8318	0.8005
RMSE	2.1577	0.2967	0.1582	0.1585
F1 Score	0.7884	0.3711	0.7951	0.7857

Evaluate NCF model after adjusting parameters for optimization:

The study of the NCF model before and after adjusting some parameters highlights some interesting trends in using various data features and improving the models.

After all the adjustments, although there are no improvements in terms of accuracy and precision, the model has shown improvements in training aspects. For instance, this means that the model can better predict, is less likely to overfit and is trained more reliably which indicates that optimization is more about making the model useful and dependable than just making it accurate.

The optimized model cuts down on the architecture's complexity but still represents how users and items interact effectively. Stronger regularization, batch normalization and early stopping make the training process more stable and help avoid overfitting. Because the model trains and converges faster, it is more efficient and even though some standard metrics drop slightly, the optimized version offers clearer explanations and performs consistently in different situations, making it more suitable and dependable for real-world recommendation systems.

.

Here is the summary of aspects improved after adjusting model's parameters for better optimization:

Table 3: Summary of Impacts After Adjusting Model's Parameters

Parameter	Original Value	Optimized Value	Impact
Embedding Size	50	32	36% parameter reduction
Dense Architecture	128→64→1	64→32→1	~50% parameter reduction
Embedding L2 Reg	1e-6	1e-5	10x stronger regularization
Dense L2 Reg	None	1e-6	Added regularization
Dropout Rate	0.5	0.3	40% reduction, less aggressive
Batch Size	64	128	2x faster training
Learning Rate	0.001	0.002	2x faster convergence
Batch Normalization	None	Added	Better training stability

Comparing NCF model to Baseline model – Collaborative filtering model:

The comparative analysis indicates an evident trade-off between raw classification accuracy and predictive precision. The item-item collaborative filtering model has a 91 % accuracy which is good because it can capture the most frequent patterns of like or dislike, but its RMSE of 0.2967 shows that its numeric rating predictions are off by an average of almost 0.3 stars. By contrast, the NCF model achieves a lower classification

accuracy of 79 %, which is mainly because the NCF model addresses the rating prediction problem as a continuous regression problem, instead of using rating prediction as a proxy to a binary classification problem. More importantly, the RMSE of the NCF model is 0.1585, compared to the collaborative baseline of 0.2967, proving that the ratings that the model predicts are much closer to the user ratings. This significant decrease in error has the effect of making recommendations more finely tuned: users receive recommendations that are more personalized and reflect their subtle tastes, as opposed to the blunt “like/dislike” boundaries.

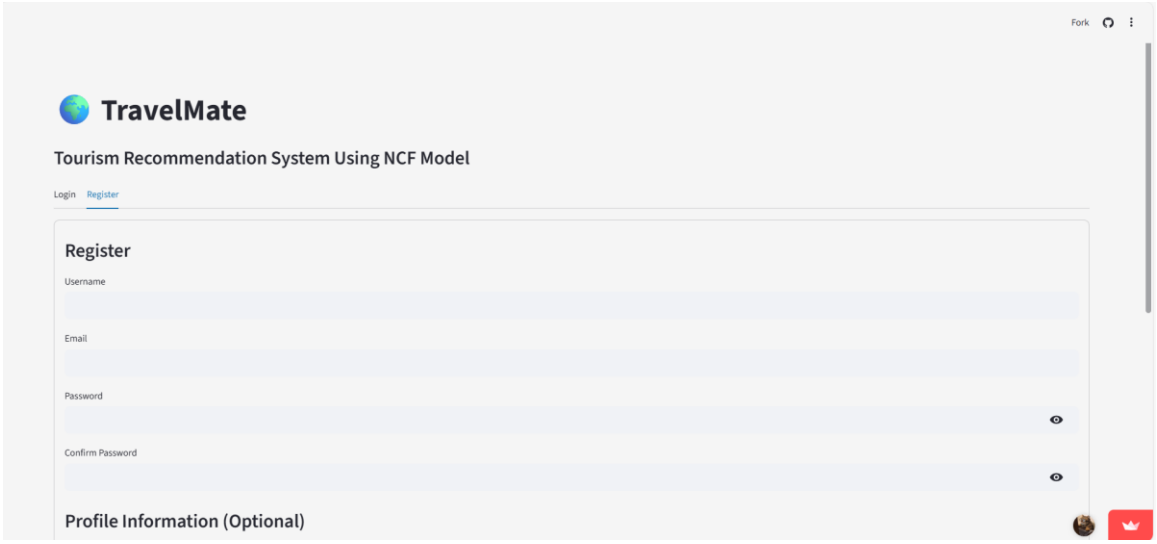
Another advantage of the NCF model is its ability to mitigate the cold-start problem. By integrating both user features and interaction data, NCF can make accurate predictions even for new users or items, unlike collaborative filtering, which struggles with this issue. The NCF model also significantly outperforms in precision, recall, and F1 score (0.7714, 0.8005, and 0.7857, respectively), showing its superior predictive performance across all metrics compared to collaborative filtering's lower scores (0.4223 precision, 0.3309 recall, and 0.3711 F1).

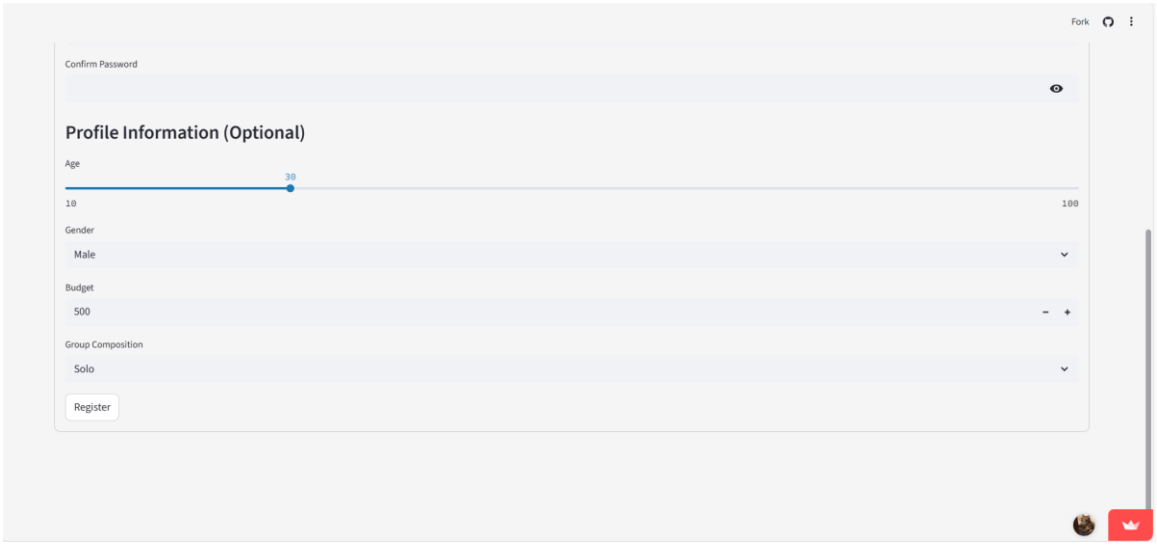
Additionally, the NCF structure is extensible by design, which enables the side-information, e.g., user demographics, temporal context, or item metadata, to be integrated and ranking-oriented loss functions to be used. Such abilities not only make NCF a more precise predictor of individual ratings but also a more generalizable engine of high-quality, long-tail recommendations in practice.

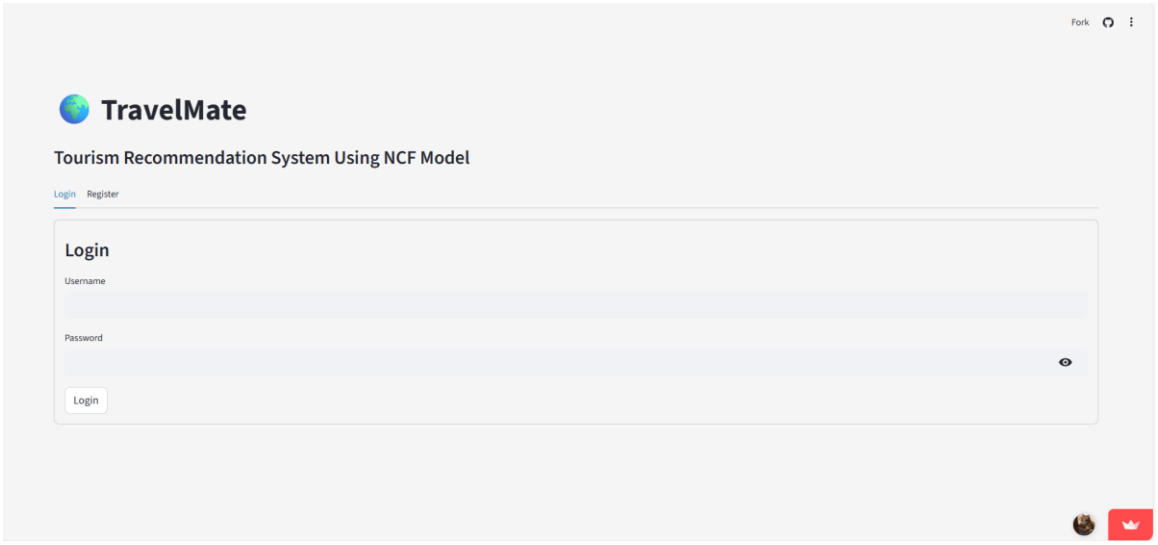
6.3 TravelMate Web Application

The web application is created based on the Streamlit Framework to demonstrate a platform where individuals can create their own accounts and enter their data. In the app, one will be able to visit the available places and rate them. After gathering enough data, such as the basic information of the users and their ratings, the application will give them personalized recommendations using the NCF model that is developed and optimized in this thesis.

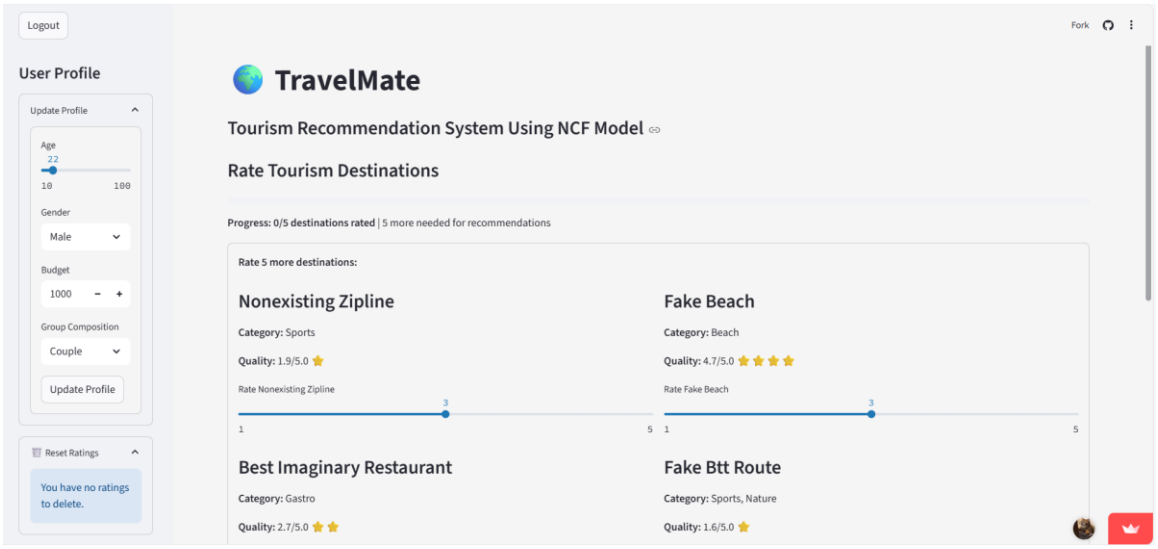
In order to start using the TravelMate application, users are required to create an account by entering necessary information including a username, password, email address, gender, age, and budget. This information is then safely stored in a PostgreSQL database, and users can log in to the application with their credentials.



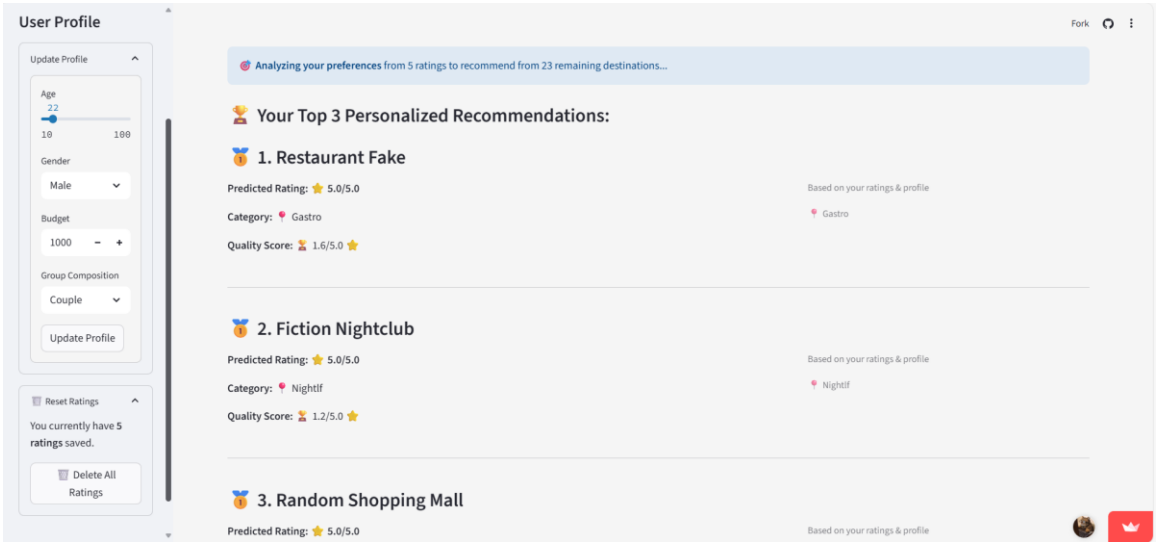




After logging in, users are asked to rate at least five destinations selected among a curated list of twenty.



After these initial ratings are provided, the system will offer three personalized suggestions on their trip



Since this version is only a proof of concept, all user information and ratings are required upfront to show the mechanics of a personalized tourist recommendation system. In later versions, the application will keep track of the visits and ratings of each user over time, thus accumulating a more realistic and richer dataset and providing more accurate recommendations.

CHAPTER 7

DISCUSSION

7.1 Strengths

The strength of this project lies in its implementation of a Neural Collaborative Filtering (NCF) model that integrates multiple data attributes, including user-item ratings and detailed user features. NCF is better than traditional methods such as content-based and classic collaborative filtering because it makes the system more accurate and flexible. NCF uses deep learning to identify the complex and non-linear ways users and items are connected which simpler models cannot do well. Thanks to this, the system can suggest items that are more likely to appeal to each user.

In addition, using several data attributes allows the model to process more detailed information. Adding user features and preferences along with ratings helps the system overcome the cold-start problem which affects users or items with little history. Because of this integration, the system can still make useful recommendations even if there is not much interaction data.

Besides that, NCF models are adaptable and can handle the large and complex data found in tourism platforms. They can keep learning and adjust to users' actions, improving the quality of their recommendations over time.

In summary, using a multi-attribute NCF model in this project improves the accuracy of recommendations and allows the system to handle more data. As a result, users are more satisfied and engaged and the recommendation applications can expand their user base.

7.2 Limitations

Despite many advantages, the project also has several potential limitations, particularly related to data quality and inherent challenges of Neural Collaborative Filtering (NCF) models.

One significant limitation arises when data are not properly handled or pre-processed. Poor data quality such as missing values, inconsistent feature encoding, or imbalanced datasets can badly affected model performance. Since NCF models rely heavily on learning meaningful patterns from the input data, insufficient or noisy data can lead to inaccurate embeddings and therefore, unreliable recommendations. Proper feature engineering, normalization, and handling of sparse data are very critical for these NCF models.

Additionally, NCF models require a large amount of data to train effectively. In scenarios with limited user-item interactions or sparse ratings, the model may struggle to learn, which impacts its accuracy. Even though adding more data helps a lot, there are still problems when dealing with users or items that have never been seen before.

Another limitation is the computational complexity of NCF models. Deep neural networks need to be trained on huge datasets which takes a lot of time and computing power, making it hard to use them in real time or requiring special hardware. NCF models are good at detecting complex patterns, but they may fit the data they are trained on too closely which can make it hard for them to deal with new data. It is important to adjust hyperparameters and apply regularization to prevent this problem.

In general, NCF is powerful, but it needs quality data, enough computing resources and still encounters the same challenges as other deep learning methods in recommendation systems.

7.3 Future Work

The potential of NCF models in personalized recommendation systems is very high and looks very promising. After finishing this project, more improvements can be made

to the model to increase its precision and accuracy. This involves trying out advanced neural network designs, using more contextual and time-related data and using advanced methods to improve how the model generalizes.

A focus for future research should be gathering and combining real-world data from the Vietnam tourism sector. Using real data about how tourists behave and what they like will help the model understand the local area and provide highly relevant suggestions.

Eventually, the project may result in a virtual travel assistant that is tailored to tourists visiting Vietnam. Such an assistant would give dynamic attraction suggestions and also guide travellers in real time, plan their itinerary and offer interactive help, making the trip much better. When the system learns from user actions and uses different types of data, it can guide tourists to the best places in Vietnam in a way that suits them..

CHAPTER 8

CONCLUSION

This thesis research showed that a personalized recommendation system could be successful for Vietnam's tourism industry. With the help of advanced deep learning and the Neural Collaborative Filtering model, we can design a system that collects and analyses what tourists like and suggest suitable places for them. Using information from different sources and good optimization methods makes sure the suggestions are both personal and useful for many people. Moreover, the prototype built with Streamlit demonstrates how this system can simplify travel, proving that the project's future is bright and full of possibilities. In conclusion, the recommendation application can increase user joy and make tourists more interested in Vietnam which will benefits the country's tourism industry grow in the future.

REFERENCES

[1] C. Pasapitch, S. Jatsada, Y. Surakirat, S. Netirak, and B. Khatthaliya, "The Tourist Attractions Recommender System for Bangkok Thailand," *ResearchGate*, Jan. 2020. https://www.researchgate.net/ (accessed May 10, 2025).

[2] V. T. Camacho, "RecommenderSystemTourism_PMPproject," *Kaggle.com*, 2020. https://www.kaggle.com/datasets/vitortcamacho/recommendersystemtourism-pmpproject (accessed May. 10, 2025).

[3] M. J. Pazzani and D. Billsus, "Content-Based Recommendation Systems," *The Adaptive Web*, vol. 4321, pp. 325–341, 2007, doi: https://doi.org/10.1007/978-3-540-72079-9_10.

[4] X. Su and T. M. Khoshgoftaar, "A Survey of Collaborative Filtering Techniques," *Advances in Artificial Intelligence*, Oct. 27, 2009. https://www.hindawi.com/journals/aai/2009/421425/

[5] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural Collaborative Filtering," *Proceedings of the 26th International Conference on World Wide Web - WWW '17*, 2017, doi: https://doi.org/10.1145/3038912.3052569.

[6] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural Graph Collaborative Filtering," *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, Jul. 2019, doi: https://doi.org/10.1145/3331184.3331267.

[7] F. Petroni *et al.*, "An Extensible Event Extraction System With Cross-Media Event Resolution," Jul. 2018, doi: https://doi.org/10.1145/3219819.3219827.

[8] Streamlit, "Streamlit Docs," *docs.streamlit.io*, 2024. https://docs.streamlit.io/

[9] "PostgreSQL: Documentation," *www.postgresql.org*. https://www.postgresql.org/docs/

[10] *TensorFlow*, 2025. https://www.tensorflow.org/recommenders?hl=vi (accessed May. 1, 2025).

[11] Python, “The Python Standard Library — Python 3.8.1 documentation,” *Python.org*, 2020. https://docs.python.org/3/library/

Tourist attraction recommendation systems based on NCF models

ORIGINALITY REPORT

4%	3%	6%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	fastercapital.com Internet Source	1%
2	Poonam Nandal, Mamta Dahiya, Meeta Singh, Arvind Dagur, Brijesh Kumar. "Progressive Computational Intelligence, Information Technology and Networking", CRC Press, 2025 Publication	1%
3	arxiv.org Internet Source	1%
4	Submitted to International University - VNUHCM Student Paper	1%
5	"Recommender Systems Handbook", Springer Science and Business Media LLC, 2022 Publication	1%
6	Submitted to RMIT University Student Paper	1%

Exclude quotes	On	Exclude matches	< 1%
Exclude bibliography	On		