Recommender Systems

PROJECT REPORT

Book Recommendation System: A Neural Networks Approach

COURSE INSTRUCTOR: Mr. Syed Zain-ul-Hassan

GROUP MEMBERS:

Alishba Subhani - **20K-0351**

Farzan Ansari - **20K-1636**

Mannahil Miftah - **20K-0234**

Sondos Suhail - **20K-1057**

OVERVIEW

The main objective of this project is to build a recommendation system based on collaborative filtering techniques using neural networks. It aims to predict user ratings for books and provide personalized book recommendations to users.

DATASET

The code loads the ratings dataset from a CSV file using `pd.read_csv()`. It performs some initial exploratory analysis on the dataset, such as finding the number of unique users (`n_users`), the number of unique books (`ratings.book_id.nunique()`), and the count of ratings for each rating value (`ratings.rating.value_counts()`).
The code loads the ratings dataset from a CSV file using `pd.read_csv()`. It performs some initial exploratory analysis on the dataset, such as finding the number of unique users (`n_users`), the number of unique books (`ratings.book_id.nunique()`), and the count of ratings for each rating value (`ratings.rating.value_counts()`).

DATA PREPROCESSING

The user and book IDs in the ratings dataset are converted to 0-based indices by subtracting 1 from their values. The ratings are then transformed into binary values based on a threshold: ratings less than 3 are set to 0 (negative interaction) and ratings greater than or equal to 3 are set to 1 (positive interaction).

AUTOENCODER FOR DIMENSIONALITY REDUCTION

The code loads pre-trained weights of an autoencoder model (`autoencoder.pth`) and applies dimensionality reduction to the book features. The `books` array is transformed using the encoder part of the autoencoder model.

MODEL - AUTOENCODER

The code defines a PyTorch module named `Autoencoder` that represents the autoencoder model. The autoencoder consists of an encoder and a decoder, both implemented as sequential layers. The encoder reduces the input dimension to the specified `hidden_dimension`, and the decoder reconstructs the original input dimension from the hidden representation.

DATASET PREPARATION

The code prepares the dataset for training the recommendation model. It creates `X` as the input tensor containing user and book indices and `y` as the target tensor containing the corresponding ratings. The dataset is then split into training and testing sets using `random_split()`. Data loaders are created to load the data in batches during training (`train_loader`) and testing (`test_loader`).

MODEL - RECOMMENDER

The code defines a PyTorch module named `Recommender` for the recommendation model. The recommender takes user IDs and book IDs as input and embeds them into lower-dimensional representations using `nn.Embedding`. The embeddings are concatenated and passed through a series of fully connected layers with ReLU activation. The final output is a sigmoid activation for predicting the ratings.

TRAINING THE RECOMMENDER MODEL

The code trains the recommender model using the training data. It specifies an optimizer (`optim.AdamW`), a learning rate scheduler (`LambdaLR`), and a loss criterion (`nn.MSELoss`). It iterates over the specified number of `epochs` and performs forward and backward passes through the model using mini-batches. The training loss and test loss are computed and displayed after each epoch.

EVALUATING THE RECOMMENDER MODEL

The code loads the trained recommender model weights (`recommender.pth`) and evaluates the model's accuracy on the ratings dataset. It iterates over each user in the dataset, feeds their user ID and book ID to the recommender, and compares the predicted ratings with the actual ratings. The overall accuracy score is calculated as the average accuracy across all users.

RANDOM FOREST CLASSIFIER

The code prepares the book embeddings and ratings data for training a Random Forest Classifier (RFC) model. The book embeddings and ratings are merged into a single dataframe (`ratings_merged`). The data is split into training and testing sets (`X_train_2`, `X_test_2`, `y_train_2`, `y_test_2`) using `train_test_split() and the RFC model is initialized (`rfc = RandomForestClassifier()`). The RFC model is trained on the training data using the `fit()` method, and predictions are made on the testing data using the `predict()` method. The accuracy score of the RFC model is computed by comparing the predicted ratings with the actual ratings.

RECOMMENDER FUNCTION FOR TOP-N RECOMMENDATIONS

The code defines a function named `RECOMMEND_N_TOP()` that takes a user ID and returns the top N recommendations for that user. Inside the function, it first identifies the books that the user has not interacted with by creating a binary array `uninteracted_books` with 1s for books the user has not rated. The uninteracted books are then passed through the recommender model to get the predicted ratings. The uninteracted books are sorted based on the predicted ratings, and the top N books are selected. The function returns a dataframe containing the titles of the top N recommended books.

CODE SCREENSHOTS

AUTO-ENCODER

```python
class Autoencoder(nn.Module):
    def __init__(self, input_dimension, hidden_dimension):
        super(Autoencoder, self).__init__()

        self.encoder = nn.Sequential(
            # nn.Linear(input_dimension, input_dimension),
            # nn.ReLU(),
            nn.Linear(input_dimension, hidden_dimension),
            nn.ReLU()
        )

        self.decoder = nn.Sequential(
            nn.Linear(hidden_dimension, input_dimension),
            nn.ReLU()
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
```

RECOMMENDER MODEL

```python
class Recommender(nn.Module):
    def __init__(self, n_users, books_n_features, hidden_size):
        super(Recommender, self).__init__()
        self.user_embed = nn.Embedding(n_users, hidden_size)
        self.book_embed = nn.Embedding(10000, hidden_size)

        self.dense = nn.Sequential(
            nn.Linear(hidden_size * 2, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, 8),
            nn.ReLU(),
            nn.Linear(8, 1),
            nn.Sigmoid()
        )

    def forward(self, user_ids, book_ids):
        users = self.user_embed(user_ids)
        books = self.book_embed(book_ids)

        target = torch.cat([users, books], dim=1)
        target = self.dense(target)
```

RECOMMENDER MODEL TRAINING

```
Epoch 1/100: 100%|██████████| 9339/9339 [01:30<00:00, 102.75it/s, Train Loss=5
Epoch 1/100, Train Loss: 5.9937, Test Loss: 5.8296
Epoch 2/100: 100%|██████████| 9339/9339 [01:24<00:00, 110.68it/s, Train Loss=5
Epoch 2/100, Train Loss: 5.7565, Test Loss: 5.8071
Epoch 3/100: 100%|██████████| 9339/9339 [01:23<00:00, 111.66it/s, Train Loss=5
Epoch 3/100, Train Loss: 5.7224, Test Loss: 5.7999
Epoch 4/100: 100%|██████████| 9339/9339 [01:31<00:00, 102.17it/s, Train Loss=5
Epoch 4/100, Train Loss: 5.6994, Test Loss: 5.8004
Epoch 5/100: 100%|██████████| 9339/9339 [01:28<00:00, 105.83it/s, Train Loss=5
Epoch 5/100, Train Loss: 5.6774, Test Loss: 5.8046
Epoch 6/100: 100%|██████████| 9339/9339 [01:24<00:00, 110.56it/s, Train Loss=5
Epoch 6/100, Train Loss: 5.6526, Test Loss: 5.8039
Epoch 7/100: 100%|██████████| 9339/9339 [01:23<00:00, 111.79it/s, Train Loss=5
Epoch 7/100, Train Loss: 5.6231, Test Loss: 5.8118
```

RFC

```
rfc = RandomForestClassifier(verbose=11, n_jobs=-1)
rfc.fit(X_train_2, y_train_2)
pred = rfc.predict(X_test_2)
accuracyScore = accuracy_score(pred, y_test_2)
accuracyScore
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 4 concurrent workers.

building tree 1 of 100building tree 2 of 100
building tree 3 of 100

building tree 4 of 100
building tree 5 of 100
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  1.2min

building tree 6 of 100
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  1.2min

building tree 7 of 100
building tree 8 of 100
[Parallel(n_jobs=-1)]: Done   3 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  1.2min

building tree 9 of 100
```

RECOMMENDATION

```
In [128]: RECOMMEND_N_TOP(0, 10)
```

Out[128]:

	TOP_N_RECOMMENDATIONS
0	The Complete Maus (Maus, #1-2)
1	The Hate U Give
2	There's Treasure Everywhere: A Calvin and Hobb...
3	The Indispensable Calvin and Hobbes
4	All Things Wise and Wonderful
5	Words of Radiance (The Stormlight Archive, #2)
6	Just Mercy: A Story of Justice and Redemption
7	Unnatural Death (Lord Peter Wimsey, #3)
8	Fair Game (Alpha & Omega, #3)
9	The Obelisk Gate (The Broken Earth, #2)