National University of Computer & Emerging Sciences

(Karachi Campus)

Recommender Systems Project Report

Comparison of Collaborative Filtering and Neural Networks

Members:

21K-4554 Bilal Sohail
21K-3323 Muhammad Shaheer
21K-4556 Muhammad Anas

Abstract

This report provides a comprehensive comparison of collaborative filtering and neural network-based recommender systems using the MovieLens dataset. The collaborative system leverages user-item interactions, while the neural network approach uses deep learning techniques to model user preferences. The comparison is based on accuracy, scalability, and computational efficiency.

Introduction

Recommender systems play a pivotal role in various domains, such as e-commerce and streaming services, by tailoring content to users' preferences. This project aims to compare two fundamental approaches—collaborative filtering and neural network-based recommendations—based on their performance and applicability.

Dataset

The MovieLens dataset contains:
- 100,000 ratings from 943 users on 1,682 movies.
- User demographic data (age, gender, occupation).
- Movie metadata (genres).

Key Preprocessing Steps:

- Merging user ratings with movie genres.
- Splitting data into training and testing sets.

1. Collaborative Filtering System

Implementation:

```
[6]  # convert into interaction matrix
     interaction_matrix = ratings.pivot(index="userid",columns="itemid",values="rating")
     interaction_matrix.head()
```

```
[7]  # compute similarity matrix
     user_similarity_matrix = compute_similarity_matrix(interaction_matrix)
     user_similarity_matrix.head()
```

The function calculates a similarity matrix between users based on their ratings. Missing ratings (NaN) are replaced with 0 to ensure consistency. Then, cosine_similarity computes how similar each pair of users is, resulting in a matrix with similarity scores. These scores are organized in a DataFrame for clarity.

```
[8]  # make predictions and fill the interaction matrix
     predicted_ratings = make_predictions(interaction_matrix,user_similarity_matrix)
     predicted_ratings.head()
```

This function predicts missing ratings in the interaction matrix. It skips items already rated by the user. For unfilled entries, it calculates a weighted average of ratings given by similar users. The weights are the similarity scores. If no similar users have rated the item, the prediction remains NaN.

```
def calculate_mse(predicted_  (parameter) test_interaction_matrix: Any
    test_interaction_matrix = test_interaction_matrix.fillna(0)
    # Calculate MSE
    mse_test = mean_squared_error(test_interaction_matrix.values, predicted_train_matrix.values)
    return mse_test
[10]                                                                                    Python
```

The function calculates the Mean Squared Error between test interaction matrix and predicted interaction matrix, ignoring missing values in the test data. This shows how well the system matches user preferences.

Strengths:

- Exploits user-item interactions.
- Effective for dense datasets.

Weaknesses:

- Struggles with sparse datasets.
- Cold start problem for new users and items.

4. Neural Network-Based Recommender System

Implementation:

The neural network-based recommender system models user-item interactions using a simple feedforward neural network built with PyTorch. It predicts user preferences by learning patterns in the training data.

```
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
Y_train_tensor = torch.tensor(Y_train, dtype=torch.float32).view(-1, 1)

X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
Y_test_tensor = torch.tensor(Y_test, dtype=torch.float32).view(-1, 1)
```

Instead of computing a similarity matrix, the data preparation process for the Neural Network involves converting the feature matrices (X_train and X_test) and target labels (Y_train and Y_test) into PyTorch tensors. This step prepares the data for input into the

neural network, which will automatically learn relationships between features during training.

```
def forward(self, x):
    # Apply LayerNorm first
    x = self.layer_norm_1(x)
    for layer in self.layers:
        x = layer(x)

    x = self.classifier(x)

    return x
```

In the Neural Network, predictions are made during the forward pass through the model. The input features are passed through several layers (including normalization, linear transformations, activations, and dropout), and the final layer (self.classifier) generates a prediction. These predictions are raw logits, which can be converted to probabilities using a sigmoid function.

```
def evaluate(model, test_loader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():

        for inputs, labels in test_loader:

            inputs, labels = inputs.to(device).to(torch.float32), labels.to(device
            # Forward pass

            outputs = model(inputs).to(torch.float32)
            # Track loss and accuracy
            outputs = torch.sigmoid(outputs)
            predicted = (outputs > 0.5).float()
            correct += (predicted == labels.view(-1, 1)).sum().item()
            total += labels.size(0)



    accuracy = correct / total
    return accuracy
```

In the evaluation step, the Neural Network computes predictions for the test set. It uses a sigmoid function to convert raw logits into probabilities and thresholds them at 0.5 for binary classification. The accuracy is calculated as the percentage of correct predictions (predicted matches labels) over the total test samples.

Strengths:

- Can learn complex patterns in data.
- Handles non-linear relationships effectively.

Weaknesses:

- Requires more computational resources.
- Can overfit on small datasets.

5. Comparison Metrics

- RMSE: Measures prediction accuracy.
- Diversity: Evaluates the variety of recommendations.
- Scalability: Assessed based on runtime and memory usage.

6. Results

Metric	Collaborative Filtering	Neural Network Filtering
Accuracy	82.5238	83.72
MSE	11.11	

Computational Efficiency:

Collaborative filtering is more computationally efficient for small to medium datasets but struggles to scale with large, sparse data. In contrast, neural network-based recommenders are more resource-intensive and usually require accelerated computing through GPUs.

Scalability:

In contrast, neural network-based recommenders are more resource-intensive but offer better scalability through GPU acceleration and parallel processing, making them suitable for large-scale, complex systems.

7. Discussion

Both systems have unique strengths:
- Collaborative filtering excels in leveraging user-item interactions effectively.
- Neural networks shine in capturing complex patterns but are computationally intensive.
Hybrid systems could combine the strengths of both for enhanced recommendations.

8. Conclusion

This project demonstrates the trade-offs between collaborative filtering and neural network-based filtering. Each approach suits specific use cases, and their integration holds the potential for superior performance.

9. References

- GroupLens Research: MovieLens Dataset
- Scikit-learn Documentation
- PyTorch Documentation
- Github Repo: https://github.com/bilalsohailmirza/recommender-systems