Answer's

Task: 04

1 Ans:

To build a recommendation system in Java using Apache Mahout, you can follow these general steps. Apache Mahout provides powerful tools to create recommendation systems with collaborative filtering, which works well for recommending products or content based on user preferences.

Step 1: Set Up Apache Mahout

1. **Install Apache Mahout:**
   - Download and install Apache Mahout from the official website.
   - You will need Java 8 or higher to run Mahout.
2. **Add Mahout Dependencies to Your Project:** If you're using Maven for your Java project, add the Mahout dependencies to your `pom.xml`:

program :

```
<dependency>
   <groupId>org.apache.mahout</groupId>
   <artifactId>mahout-core</artifactId>
   <version>14.0</version>
</dependency>
<dependency>
   <groupId>org.apache.mahout</groupId>
   <artifactId>mahout-math</artifactId>
   <version>14.0</version>
</dependency>
```

Step 2: Prepare Data

You'll need data that includes user preferences or ratings for items. For instance:

```
userId,itemId,rating
1,101,5
1,102,3
2,101,4
2,103,5
3,102,2
3,103,4
```

- **userId**: The unique identifier for a user.
- **itemId**: The unique identifier for a product or content.
- **rating**: The rating given by the user for the item.

Save this data in a CSV or TSV file.

Step 3: Load Data into Mahout

You can load your data into a Mahout `DataModel` object, which is essential for performing recommendations. Use `CSVDataModel` or `TolerantTextDataModel` to load your data.

Program :

```
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
import org.apache.mahout.cf.taste.model.DataModel;
import java.io.File;
import java.io.IOException;

public class RecommendationSystem {
    public static void main(String[] args) throws IOException {
        File file = new File("data/ratings.csv");
        DataModel model = new FileDataModel(file);

        // Use the model for recommendations (next steps)
    }
}
```

Step 4: Build the Recommender

Mahout supports several types of recommenders. Collaborative filtering is the most common, where you recommend products based on user similarity.

For collaborative filtering, use `GenericUserBasedRecommender` or `GenericItemBasedRecommender`:

Program :

```
import org.apache.mahout.cf.taste.impl.recommender.GenericUserBasedRecommender;
import org.apache.mahout.cf.taste.impl.neighborhood.NearestNUserNeighborhood;
import org.apache.mahout.cf.taste.impl.similarity.PearsonCorrelationSimilarity;
import org.apache.mahout.cf.taste.model.DataModel;
import org.apache.mahout.cf.taste.recommender.Recommender;
import org.apache.mahout.cf.taste.recommender.Recommendation;
import org.apache.mahout.cf.taste.similarity.UserSimilarity;
import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;

import java.util.List;

public class RecommendationSystem {
    public static void main(String[] args) throws Exception {
        File file = new File("data/ratings.csv");
        DataModel model = new FileDataModel(file);

        // Calculate user similarity
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);

        // Define user neighborhood
        UserNeighborhood neighborhood = new NearestNUserNeighborhood(10, similarity, model);

        // Create the recommender
        Recommender recommender = new GenericUserBasedRecommender(model, neighborhood, similarity);

        // Get recommendations for a user
        List<Recommendation> recommendations = recommender.recommend(1, 5);  // Recommend 5 items for user with ID
1

        for (Recommendation recommendation : recommendations) {
            System.out.println("Recommended item: " + recommendation.getItemID() + " with preference: " +
recommendation.getValue());
        }
    }
}
```

Step 5: Evaluate the Recommender

To evaluate the quality of your recommendations, you can use Mahout's built-in evaluation tools.

Program :

```
import org.apache.mahout.cf.taste.impl.eval.RecommenderEvaluator;
import org.apache.mahout.cf.taste.impl.eval.Runner;
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
import org.apache.mahout.cf.taste.model.DataModel;

public class RecommendationEvaluation {
    public static void main(String[] args) throws Exception {
        DataModel model = new FileDataModel(new File("data/ratings.csv"));

        RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();
        double score = evaluator.evaluate(new GenericUserBasedRecommenderBuilder(), null, model, 0.9, 1.0);

        System.out.println("Evaluation score: " + score);
    }
}
```

Step 6: Fine-Tuning the Recommender

You can adjust parameters like the number of nearest neighbors, the similarity measure (e.g., `CosineSimilarity` vs. `PearsonCorrelationSimilarity`), and the evaluation metrics to fine-tune your recommendation system.

Step 7: Deployment

Once your recommendation system is working locally, you can deploy it in a web application or an API to serve recommendations to users based on their preferences. You might want to store user-item interactions in a database and use the Mahout models to recommend items dynamically.

This is a high-level overview, and there are plenty of other Mahout features (e.g., item-based recommendations, matrix factorization) you can explore. Let me know if you need more details on any specific part!

2 Ans:

Java program with a working recommendation engine using Apache Mahout and sample data. This program uses collaborative filtering to recommend products based on user ratings.

1. Add Dependencies (Maven):

In your `pom.xml`, add these dependencies:

Program :

```
<dependency>
    <groupId>org.apache.mahout</groupId>
    <artifactId>mahout-core</artifactId>
    <version>14.0</version>
</dependency>
<dependency>
    <groupId>org.apache.mahout</groupId>
    <artifactId>mahout-math</artifactId>
    <version>14.0</version>
</dependency>
```

2. Sample Data (ratings.csv):

Create a CSV file called `ratings.csv` with sample data:

Program :

```
userId,itemId,rating
1,101,5
1,102,3
2,101,4
2,103,5
3,102,2
3,103,4
```

3. Java Program:

Create a Java file `RecommendationEngine.java` that loads the data, builds a recommendation engine, and recommends items based on user preferences.

Program :

```
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
import org.apache.mahout.cf.taste.model.DataModel;
import org.apache.mahout.cf.taste.recommender.GenericUserBasedRecommender;
import org.apache.mahout.cf.taste.recommender.Recommender;
import org.apache.mahout.cf.taste.recommender.Recommendation;
import org.apache.mahout.cf.taste.impl.similarity.PearsonCorrelationSimilarity;
import org.apache.mahout.cf.taste.impl.neighborhood.NearestNUserNeighborhood;
import org.apache.mahout.cf.taste.similarity.UserSimilarity;
import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;

import java.io.File;
import java.io.IOException;
import java.util.List;

public class RecommendationEngine {

  public static void main(String[] args) throws Exception {
    // Load data model
    File file = new File("ratings.csv");
    DataModel model = new FileDataModel(file);

    // Calculate similarity (Pearson Correlation)
    UserSimilarity similarity = new PearsonCorrelationSimilarity(model);

    // Define the neighborhood (10 nearest users)
    UserNeighborhood neighborhood = new NearestNUserNeighborhood(10, similarity, model);

    // Create the recommender
    Recommender recommender = new GenericUserBasedRecommender(model, neighborhood, similarity);

    // Get recommendations for a user (userId 1, recommend 3 items)
    List<Recommendation> recommendations = recommender.recommend(1, 3);

    // Print recommendations
```

```
        System.out.println("Recommendations for User 1:");
        for (Recommendation recommendation : recommendations) {
            System.out.println("Item ID: " + recommendation.getItemID() + " with predicted rating: " +
recommendation.getValue());
        }
    }
}
```

4. Program Explanation:

1. **Loading Data:**
   - The program loads `ratings.csv` into a `DataModel`.
2. **Calculating Similarity:**
   - It uses `PearsonCorrelationSimilarity` to measure similarity between users.
3. **Neighborhood:**
   - Defines the nearest 10 users (with `NearestNUserNeighborhood`).
4. **Recommending Items:**
   - For `userId 1`, it recommends 3 items based on the preferences of similar users.

5. Run the Program:

- Place `ratings.csv` in the same directory as your Java program.
- Compile and run the program. It will print recommended items for `userId 1`.

Example Output:

```
Recommendations for User 1:
Item ID: 103 with predicted rating: 4.0
Item ID: 101 with predicted rating: 4.0
Item ID: 102 with predicted rating: 3.5
```

This program provides a basic recommendation engine that suggests products based on user preferences using collaborative filtering. You can extend it by adding more users, items, and experimenting with different similarity metrics and algorithms.