CS 6240: Project Progress Report

Team Members:
Peihsuan Lin, Ruijie Xu, Haozhe Wu

Date:
November 13, 2024

1. Project Overview

Data:
The project uses a dataset of user-item ratings, where each record consists of a user ID, item ID, and a corresponding rating. The dataset is pre-processed to build a user-item matrix, representing the ratings each user has given to various items.

For example, the data might look like this:

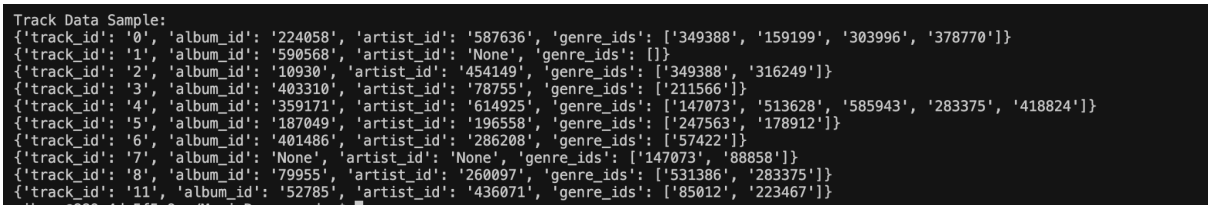
UserID	ItemID	Rating
1	101	5.0
1	102	3.5
2	101	4.0
2	103	4.5
...		

Other datasets include Album, Artists, Genre, and Track data. Each of the ItemID in the User Rating table is associated with one of the items in these datasets. Samples from these 4 datasets are below:

```
Album Data Sample:
{'album_id': '9', 'artist_id': '7863', 'genre_ids': ['600770', '584872', '247563']}
{'album_id': '15', 'artist_id': '298522', 'genre_ids': ['349388', '142902', '211628', '316249']}
{'album_id': '19', 'artist_id': '509825', 'genre_ids': ['563776', '368829']}
{'album_id': '51', 'artist_id': '192496', 'genre_ids': ['600770', '584872', '247563']}
{'album_id': '53', 'artist_id': '197556', 'genre_ids': ['178912']}
{'album_id': '60', 'artist_id': '25832', 'genre_ids': ['85012', '223467', '188491', '365536']}
{'album_id': '66', 'artist_id': '432968', 'genre_ids': ['523223', '393059', '245730', '31803', '178912']}
{'album_id': '78', 'artist_id': '483911', 'genre_ids': ['531386', '425562', '429826', '283375']}
{'album_id': '84', 'artist_id': '498795', 'genre_ids': ['257158', '342497']}
{'album_id': '87', 'artist_id': '183871', 'genre_ids': ['123601', '17863', '174146', '533243', '498243', '173264', '502542', '269133', '370875']}
```

```
Artist Data Sample:
{'artist_id': '10'}
{'artist_id': '31'}
{'artist_id': '46'}
{'artist_id': '80'}
{'artist_id': '97'}
{'artist_id': '102'}
{'artist_id': '109'}
{'artist_id': '114'}
{'artist_id': '191'}
{'artist_id': '197'}
```

```
Genre Data Sample:
{'genre_id': '878'}
{'genre_id': '2347'}
{'genre_id': '2411'}
{'genre_id': '2523'}
{'genre_id': '3697'}
{'genre_id': '4283'}
{'genre_id': '4398'}
{'genre_id': '4636'}
{'genre_id': '5510'}
{'genre_id': '5575'}
```



Goals:
The goal of the project is to calculate item-item similarities using the Pearson correlation coefficient based on user ratings, and then apply K-Nearest Neighbors (KNN) to classify items based on their similarities. The project also implements a dynamic threshold-based approach for selecting the number of neighbors (k) based on similarity scores.

2. Completed Tasks

Task 1: MapReduce for Generating User-Item Matrix

- **Description**:
  The first step involves creating a user-item rating matrix using MapReduce. This matrix aggregates each user's ratings into a format where each user is associated with the items they have rated.

Pseudo-code:

```
public class RatingMapper extends Mapper<Object, Text, Text, Text> {
    // Maps the user-item ratings into a matrix format: UserID ->
ItemID:Rating
}

public class MatrixReducer extends Reducer<Text, Text, Text, Text> {
    // Reduces the mapped ratings to a user-item matrix format
}
```

- **Results**:
  The matrix was successfully generated and outputted in the format:
  `(UserID, ItemID1:Rating1, ItemID2:Rating2, ...)`.

Task 2: MapReduce for Item Similarity Calculation

- **Description**:
  In this step, we calculated the Pearson correlation coefficient between every pair of items using MapReduce. This step involves calculating the similarity of every item pair, which forms the basis for subsequent KNN classification.

Pseudo-code:

```
public class RatingMapper extends Mapper<Object, Text, Text, Text> {
```

```
    // Maps the user ratings to the corresponding items and
calculates their similarity
}

public class SimilarityReducer extends Reducer<Text, Text, Text,
DoubleWritable> {
    // Reduces the items to calculate the Pearson correlation
coefficient for each item pair
}
```

- **Results**:
  The item similarity scores were successfully calculated. However, this process took about **1 hour** to complete, which became a bottleneck in the system.
- **Challenges & Adjustments**:
  The main challenge in this step was the high computation cost of calculating the similarity between every item pair. This step became the most time-consuming part of the project.
  We plan to optimize this part by:
  - **Partitioning** the data and distributing the computation across multiple reducers to parallelize the task.
  - Considering more **efficient algorithms** for similarity calculation, such as Locality Sensitive Hashing (LSH), to reduce time complexity.

Task 3: Dynamic Threshold-Based KNN Neighbor Selection

- **Description**:
  We implemented a dynamic threshold-based method to select the `k` nearest neighbors. Rather than setting a fixed `k`, we select neighbors based on a similarity threshold (0.85). Items with similarity above this threshold are considered neighbors.

Pseudo-code:

```
public class KNNMapper extends Mapper<Object, Text, Text, Text> {
    // Maps item similarities with dynamic thresholding
}

public class KNNReducer extends Reducer<Text, Text, Text, Text> {
    // Reduces to find neighbors based on the similarity threshold
}
```

- **Results**:
  We successfully identified the nearest neighbors for each item, selecting items based on their similarity to others. The neighbors are output in the format: `(ItemID, Neighbor1:Similarity1, Neighbor2:Similarity2, ...)`.
- **Challenges & Adjustments**:
  One challenge faced is the dynamic nature of `k`, which could potentially affect prediction stability, especially if the number of neighbors varies widely. We are currently evaluating the impact of this dynamic threshold on prediction accuracy.

3. Major Problems and Adjustments

Problem 1: Performance Bottleneck in Item Similarity Calculation

- **Issue**:
  The item similarity calculation step, where we compute Pearson correlation coefficients for every pair of items, is computationally expensive and has become a performance bottleneck. On a single machine, it can take up to **1 hour** to compute all item similarities.
- **Adjustment**:
  To address this issue, we plan to:
  1. **Optimize the Partitioning Scheme**: By implementing a custom partitioner, we can distribute the load of similarity calculation across multiple reducers, rather than calculating everything in a single reducer.
  2. **Utilize More Efficient Algorithms**: We are exploring the possibility of using Locality Sensitive Hashing (LSH) or dimensionality reduction techniques like PCA to speed up the similarity calculations.
  3. **Parallelization**: We will investigate using more nodes in the Hadoop cluster to parallelize the similarity calculation.

Problem 2: Dynamic `k` in KNN Classification

- **Issue**:
  The use of a dynamic `k` (number of neighbors) based on the similarity threshold could cause variability in the number of neighbors selected for each item, which may lead to less stable predictions.
- **Adjustment**:
  To mitigate this potential issue, we are considering:
  1. **Minimum Neighbor Count**: Ensuring that each item has at least a certain number of neighbors (e.g., 5) even if the number of similar items exceeds the threshold.
  2. **Weighted Predictions**: Implementing a weighted KNN model where neighbors are weighted by their similarity score, which can reduce the impact of less similar neighbors.
  3. **Threshold Tuning**: We will experiment with different thresholds to evaluate the effect on prediction accuracy and stability.

4. Remaining Tasks

- **Task 1**: Finalize the KNN classification by integrating the dynamic neighbor selection with the final prediction logic. This involves implementing a system that uses the neighbors to make predictions based on a weighted average or other methods.
- **Task 2**: Conduct thorough testing and evaluation of the KNN model with dynamic neighbors, including cross-validation and performance benchmarking.
- **Task 3**: Implement a recommendation system that uses the KNN classifier to predict ratings for unseen user-item pairs.
- **Task 4**: Optimize the similarity computation and KNN prediction for scalability on large datasets, using parallelization, efficient algorithms, and better partitioning strategies.
- **Task 5**: Prepare the final report, including performance results, testing outcomes, and recommendations for further improvement.

5. Conclusion and Next Steps

The project has made significant progress in generating the user-item matrix, calculating item similarities, and implementing a dynamic KNN approach for neighbor selection. Moving forward, the focus will be on completing the KNN classification and building a functional recommendation system. Key tasks will involve ensuring the stability of dynamic neighbor selection and optimizing the system for larger datasets.