

CS5481: Data Engineering - Assignment2 Solution

Question 1 - LLM for data engineering

(20 marks) LLMs' fast, articulate answers to expert questions can help data engineers discover datasets, write and debug code, document procedures, and learn new techniques as they build data pipelines. In this question, you are required to write suitable prompts for ChatGPT to achieve the following targets.

1. (5 marks) Assume you need datasets to train a recommender system that predicts user preferences for products. Use ChatGPT (GPT-4) to find relevant datasets. List the prompts you used and the corresponding outputs from ChatGPT.
2. (5 marks) Use ChatGPT to preprocess a sample from the **Movielens-1M** dataset for training a recommender system. List the prompts, inputs, and outputs from ChatGPT.
3. (10 marks) Generate a preprocessing script using ChatGPT to format the dataset for a Collaborative Filtering model using Surprise or TensorFlow. Correct any errors in the generated code, use the revised version to preprocess the dataset, and display the first 5 processed entries.

Solution

1. Prompt to ChatGPT: I need datasets to train a recommender system that predicts user preferences for products. Please suggest relevant datasets and provide links to access them.

Output from ChatGPT:

1. **Movielens-1M**: A dataset containing 1 million ratings for movies.
 - [Link to Movielens-1M](#)
2. **Amazon Product Reviews**: A dataset with reviews and ratings across various product categories.
 - [Link to Amazon Reviews](#)
3. **Goodbooks-10k**: A dataset of 6 million ratings for 10,000 books.
 - [Link to Goodbooks-10k](#)
4. **Jester Joke Dataset**: A dataset of continuous ratings (-10 to 10) for jokes.
 - [Link to Jester Dataset](#)
5. **Netflix Prize Dataset**: A dataset of 100 million movie ratings.
 - [Link to Netflix Dataset](#)

2. Prompt to ChatGPT: Preprocess a sample from the Movielens-1M dataset for training a recommender system.

Output:

plaintext

Copy code

1. Load the dataset.
2. Filter columns to focus on user ID, movie ID, and rating.
3. Normalize ratings (optional).
4. Split the dataset into training and testing sets.

Preprocessing Example:

python

Copy code

```
import pandas as pd

# Load the Movielens dataset
df = pd.read_csv('ratings.dat', sep='::', engine='python', names=['UserID', 'MovieID', 'Ra

# Select necessary columns
df = df[['UserID', 'MovieID', 'Rating']]

# Normalize ratings (optional)
df['Rating'] = df['Rating'] / 5.0

# Split into training and test sets
train_df = df.sample(frac=0.8, random_state=42)
test_df = df.drop(train_df.index)

# Display sample of preprocessed data
print(train_df.head())
```

3. Prompt to ChatGPT: Generate a preprocessing script using Surprise to format the Movielens-1M dataset for a Collaborative Filtering model.

Prompt to ChatGPT:

plaintext

Copy code

Generate a preprocessing script using Surprise to format the Movielens-1M dataset for a Co

Initial Output:

python

Copy code

```
from surprise import Dataset, Reader
import pandas as pd

# Load the dataset
df = pd.read_csv('ratings.dat', sep='::', engine='python', names=['UserID', 'MovieID', 'Ra

# Create a Reader object for Surprise
reader = Reader(rating_scale=(1, 5))

# Load the dataset into Surprise format
data = Dataset.load_from_df(df[['UserID', 'MovieID', 'Rating']], reader)

# Train-test split
trainset = data.build_full_trainset()
testset = trainset.build_anti_testset()

# Display first 5 processed entries
print(trainset.all_ratings()[:5])
```

Revised Output (after fixing):

python

Copy code

```
from surprise import Dataset, Reader
import pandas as pd

# Load dataset
df = pd.read_csv('ratings.dat', sep='::', engine='python', names=['UserID', 'MovieID', 'Ra

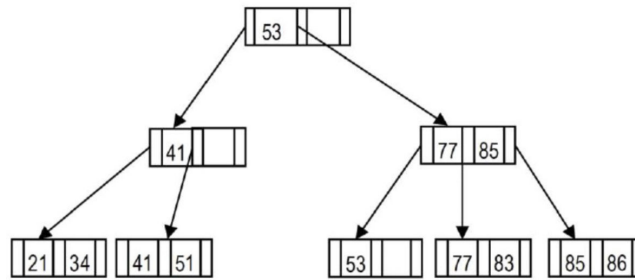
# Use Surprise's Reader and Dataset classes
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(df[['UserID', 'MovieID', 'Rating']], reader)

# Split data into train and test sets
trainset = data.build_full_trainset()
testset = trainset.build_testset()

# Print the first 5 processed entries
for uid, iid, true_r in testset[:5]:
    print(f'UserID: {uid}, MovieID: {iid}, Rating: {true_r}')
```

Question 2 - Data Indexing

(25 marks) Given the following B⁺-tree, please answer following questions.

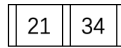


1. What is the value of p for this B⁺-tree? (Note that p is the order of a B⁺-tree)
2. Can you re-build a taller B⁺-tree with the same value of p using the same set of search-key values in the leaf nodes of the given tree? If yes, show the steps by drawing a new diagram whenever the height of the tree increases.
3. Insert the search-key values 84, 19 and 32 in sequence to the given B⁺-tree, and draw a new diagram for each insertion.
4. Suggest a sequence of search-key values to be deleted from the resultant B⁺-tree in Q4.2 to shrink the tree to 2 levels with the **least** number of deletions. Show the steps by drawing a new diagram whenever a node is deleted.

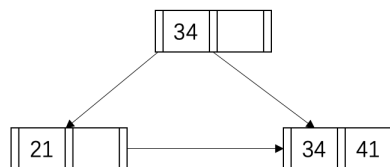
Solution

1. (5 marks) $p = 3$. Because each node has at most 3 pointers.
2. (6 marks) Yes, we can rebuild the B⁺ tree as follows:

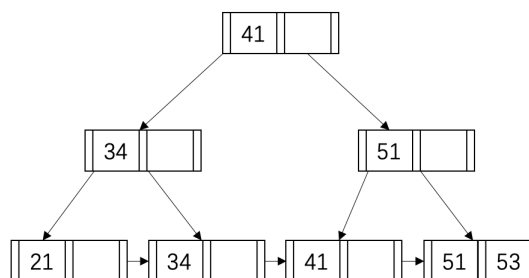
Insert 21, 34



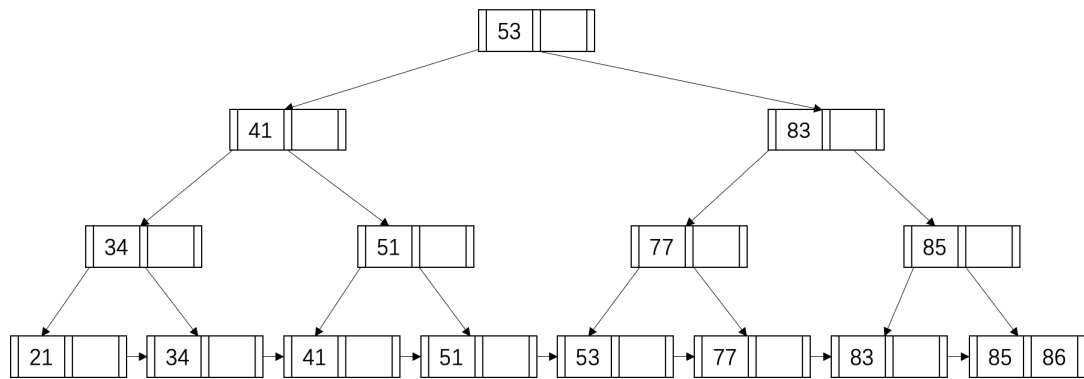
Insert 41



Insert 51, 53

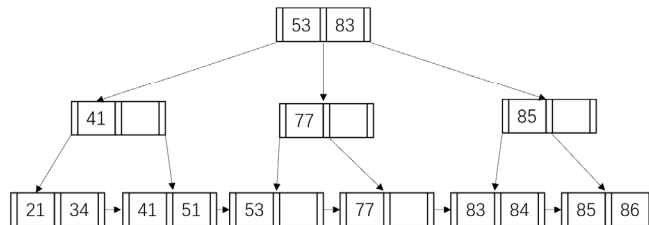


Insert 77, 83, 85, 86

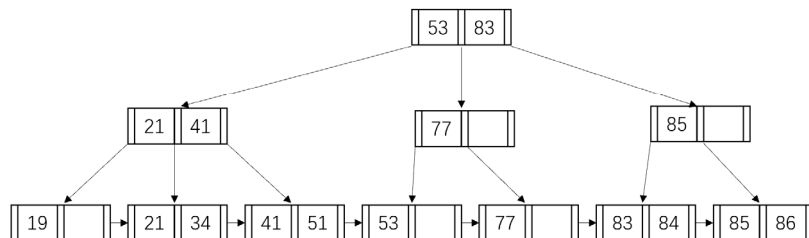


3. (6 marks) The insertions are showed below:

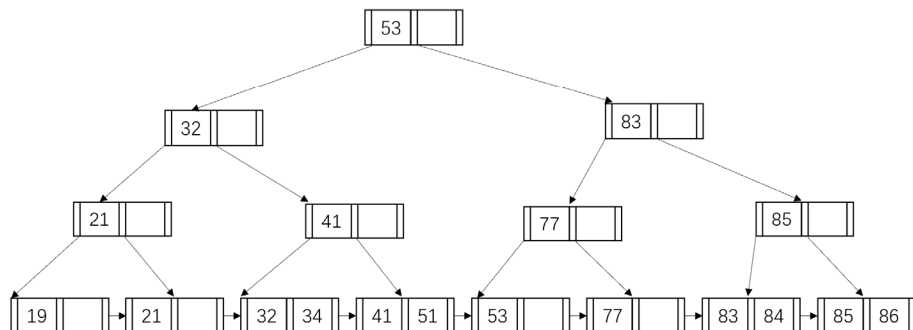
Insert 84



Insert 19

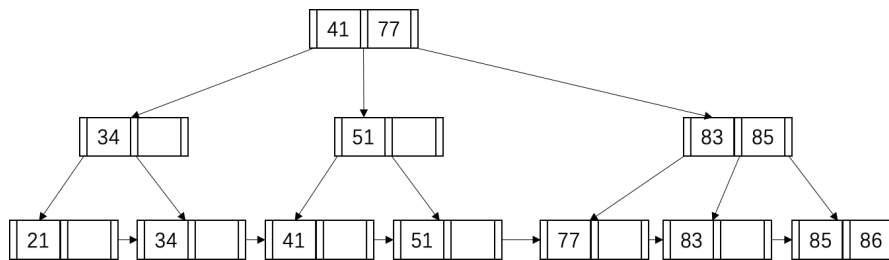


Insert 32

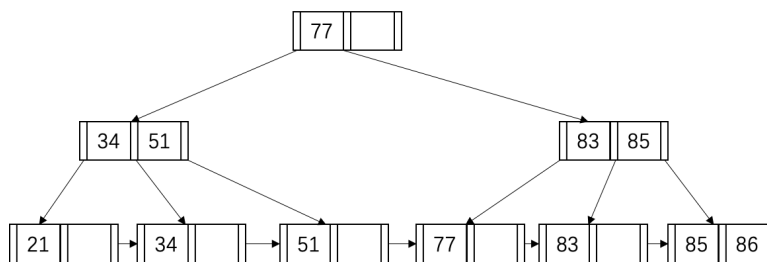


4. **(8 marks)** The least number of deletions is 5, there are many different sequences we can choose, one possible solution is:

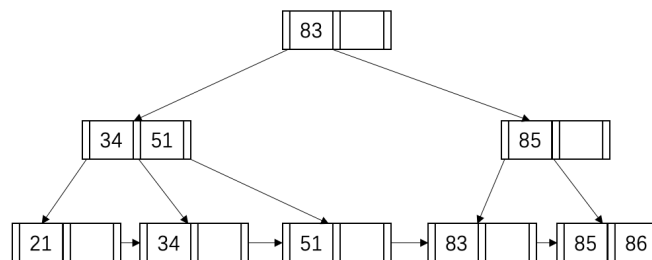
Delete 53



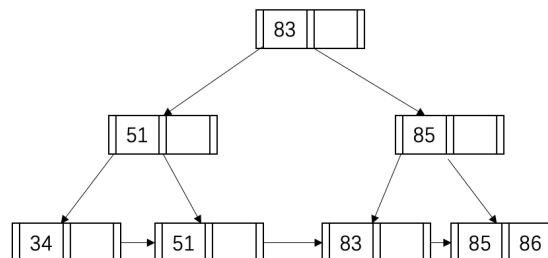
Delete 41



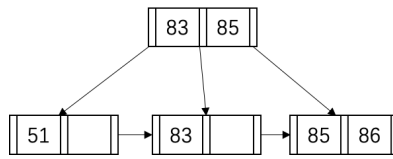
Delete 77



Delete 21



Delete 34



Question 3 - Data Querying

(25 marks) The university held a coding contest where hackers submit solutions to various tasks. Each task has a bonus for the top 3 performers. You are given the following SQL tables:

- Hackers (hacker_id: INT, name: VARCHAR, bank_account: INT)
- Tasks (task_id: INT, description: VARCHAR, bonus: INT)
- Submissions (submission_id: INT, hacker_id: INT, task_id: INT, score: INT, submission_date: DATE)

Assume:

- Each task has a bonus for the top 3 submissions with the highest scores.
 - If there are multiple submissions with the same score, the earliest submission (lower submission_id) is preferred.
 - Hackers can submit multiple times, but only their best submission (highest score) counts for each task.
1. **(5 marks)** Write a query to print the hacker_id, name, and the number of distinct tasks each hacker participated in. Sort the result by the number of tasks in descending order, and then by hacker_id in ascending order if there's a tie.
 2. **(5 marks)** Write a query to find the task_id, description, and the total bonus awarded for each task. Sort the result by task_id in ascending order.
 3. **(5 marks)** Write a query to list the submission_id, hacker_id, name, and score of the highest-scoring submission for each task submitted on 2023-01-01. If multiple submissions have the highest score for the same task, return the submission with the smallest submission_id. Sort the result by task_id in ascending order.
 4. **(5 marks)** Write a query to print the hacker_id, name, and the total score each hacker achieved across all tasks. For each task, only the hacker's best score counts. Sort the result by total_score in descending order and by hacker_id in ascending order if there's a tie.
 5. **(5 marks)** Write a query to find the hacker_id, name, and bank_account of hackers who did not participate in any tasks.

1.

```
SELECT h.hacker_id, h.name, COUNT(DISTINCT s.task_id) AS task_count
FROM Hackers h
JOIN Submissions s ON h.hacker_id = s.hacker_id
GROUP BY h.hacker_id, h.name
ORDER BY task_count DESC, h.hacker_id ASC;
```

2.

```
SELECT t.task_id, t.description, SUM(t.bonus) AS total_bonus
FROM Tasks t
JOIN Submissions s ON t.task_id = s.task_id
GROUP BY t.task_id, t.description
ORDER BY t.task_id ASC;
```

3.

```
WITH MaxScores AS (
    SELECT task_id, MAX(score) AS max_score
    FROM Submissions
    WHERE submission_date = '2023-01-01'
```



```

GROUP BY task_id
)
SELECT s.submission_id, s.hacker_id, h.name, s.score
FROM Submissions s
JOIN MaxScores m ON s.task_id = m.task_id AND s.score = m.max_score
JOIN Hackers h ON s.hacker_id = h.hacker_id
WHERE s.submission_date = '2023-01-01'
ORDER BY s.task_id ASC, s.submission_id ASC;

```

4.

```

WITH BestScores AS (
    SELECT hacker_id, task_id, MAX(score) AS best_score
    FROM Submissions
    GROUP BY hacker_id, task_id
)
SELECT h.hacker_id, h.name, SUM(b.best_score) AS total_score
FROM Hackers h
JOIN BestScores b ON h.hacker_id = b.hacker_id
GROUP BY h.hacker_id, h.name
ORDER BY total_score DESC, h.hacker_id ASC;

```

5.

```

SELECT h.hacker_id, h.name, h.bank_account
FROM Hackers h
LEFT JOIN Submissions s ON h.hacker_id = s.hacker_id
WHERE s.hacker_id IS NULL;

```

Question 4 - Recommender System

(30 marks)

1. **(8 marks)** Please write two basic approaches for recommender system and briefly explain them.
2. **(8 marks)** One common challenge in recommender systems is the filter bubble problem. Explain what the filter bubble problem is and how it affects user experience. Suggest at least two strategies to mitigate the filter bubble issue.
3. **(14 marks)** Rating prediction is an important task for a recommender system. Try to implement a recommendation model using [the Goodbooks-10k dataset](#) to predict user ratings. You have two options based on your available compute power:
 - (a) Option 1 (Full Dataset): For students with sufficient compute power, use the entire dataset (6 million ratings for 10,000 books by 53,000 users).
 - (b) Option 2 (Limited Dataset): For students with limited compute power, use a subset of the data. Limit the dataset to the top 5,000 users and top 3,000 books based on the number of interactions (ratings). In your submission, clearly state that you are using the subset due to computational constraints.
 - (c) You can implement the model using user-based collaborative filtering, item-based collaborative filtering, or matrix factorization.
 - (d) Split your chosen dataset (full or limited) into a training set and a test set.
 - (e) Display the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) on the test set.

1.

- 1) User-based Collaborative Filtering: Recommends items based on the preferences of similar users.
- 2) Matrix Factorization: Factorizes the user-item interaction matrix to discover latent factors.

2.

The filter bubble problem occurs when a recommender system repeatedly exposes users to the same type of content, limiting exposure to diverse content. Two strategies to mitigate it:

- 1) Introduce Serendipity: Add randomness to recommendations.
- 2) Diversify Recommendations: Ensure diversity in content by adjusting the algorithm to prioritize content from different categories.

3.

Steps:

- 1) Preprocessing: Limit to 5,000 users and 3,000 books.
- 2) Train-Test Split: Split data into training and test sets.
- 3) Model: Use matrix factorization (NMF).

Sample Code:

```
import pandas as pd

from surprise import Dataset, Reader, NMF

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_absolute_error, mean_squared_error

import numpy as np
```

```

# Load dataset
df = pd.read_csv('goodbooks-10k/ratings.csv')

# Limit data for students with limited compute power
top_users = df['user_id'].value_counts().nlargest(5000).index
top_books = df['book_id'].value_counts().nlargest(3000).index
df = df[df['user_id'].isin(top_users) & df['book_id'].isin(top_books)]

# Split into train and test sets
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

# Prepare dataset for Surprise library
reader = Reader(rating_scale=(1, 5))
train_data = Dataset.load_from_df(train_df[['user_id', 'book_id', 'rating']], reader)
trainset = train_data.build_full_trainset()

# Build and train the model
nmf_model = NMF(n_factors=15, random_state=42)
nmf_model.fit(trainset)

# Predict on test set and calculate errors
predictions = []
for i in range(len(test_df)):
    uid = test_df.iloc[i]['user_id']
    iid = test_df.iloc[i]['book_id']
    actual_rating = test_df.iloc[i]['rating']
    pred

```