Code_Data_Processing

June 26, 2025

0.0.1 Data Processing : Reproducing Popularity and Gender Bias in Music Recommenders with Cross-Domain Extension to Books

Team 2 Elif Deger Nataliya Kharitonova

In this notebook we have the code which we implemented for the data processeing steps before starting working on the recommender algorithms

0.1 LFM DATASET

- The LFM-2b dataset used in our study is considered derivative work according to paragraph 4.1 of Last.fm's API Terms of Service (https://www.last.fm/api/tos). The Last.fm Terms of Service further grant us a license to use this data (according to paragraph 4).
- The exact dataset we are using is LFM-2b Dataset, which is a subset of LAST FM dataset and an extension of the LFM-1b dataset and was created by the respective authors of the paper we are replicating "Analyzing Item Popularity Bias of Music Recommender Systems: Are Different Genders Equally Affected?". Unfortunately due to licensing issues (see: https://www.cp.jku.at/datasets/LFM-2b/) the dataset is not avaliable to public. We had to contact the authors ourselves, and they were very kind to provide us with the dataset.

We start with pre-processing data for the music recommendation systems. In the paper the authors apply the following data pre-processing steps:

1. Keep only user-track interactions with playcount (PC) greater than 1 to reduce noise.
2. Consider only tracks listened to by at least 5 different users.
3. Consider only users who listened to at least 5 different tracks.
4. Restrict listening events to the last 5 years to focus on recent popularity trends.
5. Convert interactions to binary form: 1 if user listened to the track at least once, 0 otherwise.
6. Sample 100,000 tracks uniformly at random to include a balanced variety of track popularities.
7. To evaluate split users into train, validation, and test sets (60-20-20) with 5-fold cross-validation, training on full interactions of train users and using 80/20 item splits for validation and testing.

Step 1:

- Load the TSV file containing user listening counts.
- Filter out rows where the `count` is **1 or less**.
- Save the filtered data to a new TSV file called `filtered-listening-counts.tsv`.

[ ]:
```python
import pandas as pd

df = pd.read_csv('listening-counts.tsv', sep='\t')

# Filter rows with count > 1
filtered_df = df[df['count'] > 1]
num_unique_users = filtered_df['user_id'].nunique()
num_unique_tracks = filtered_df['track_id'].nunique()

print(f"Unique user_id count after filtering: {num_unique_users}")
print(f"Unique track_id count after filtering: {num_unique_tracks}")

filtered_df.to_csv('filtered-listening-counts.tsv', sep='\t', index=False)
```

Step 2:

- **Load two TSV files**:
  - `filtered-listening-counts.tsv`: contains listening data filtered by count > 1.
  - `users.tsv`: contains user metadata.
- **Merge** the two datasets **on `user_id`** using an **inner join**
- **Save** the merged dataset as `merged-users-listening.tsv`.

[ ]:
```python
import pandas as pd

filtered_df = pd.read_csv('filtered-listening-counts.tsv', sep='\t')
users_df = pd.read_csv('users.tsv', sep='\t')

merged_df = pd.merge(users_df, filtered_df, on='user_id', how='inner')

merged_df.to_csv('merged-users-listening.tsv', sep='\t', index=False)

print(f"Merged data saved with {merged_df['user_id'].nunique()} unique users.")
```

Step 2 - continue and Step 3

- **Load** the merged listening and user data from `merged-users-listening.tsv`.
- **Filter tracks**:
  - Keeps only tracks listened to by **at least 5 unique users**.
- **Filter users**:
  - From the remaining data, keeps only users who have listened to **at least 5 unique tracks**.
- **Save** the final filtered dataset to `filtered-merged-listening.tsv`.

[ ]:
```python
import pandas as pd

# Load merged data
merged_df = pd.read_csv('merged-users-listening.tsv', sep='\t')

# First, filter tracks with at least 5 unique users
track_user_counts = merged_df.groupby('track_id')['user_id'].nunique()
tracks_to_keep = track_user_counts[track_user_counts >= 5].index
filtered_df = merged_df[merged_df['track_id'].isin(tracks_to_keep)]

# Then, filter users with at least 5 unique tracks
user_track_counts = filtered_df.groupby('user_id')['track_id'].nunique()
users_to_keep = user_track_counts[user_track_counts >= 5].index
filtered_df = filtered_df[filtered_df['user_id'].isin(users_to_keep)]

print(f"Filtered data has {filtered_df['user_id'].nunique()} users and␣
 ↪{filtered_df['track_id'].nunique()} tracks.")

filtered_df.to_csv('filtered-merged-listening.tsv', sep='\t', index=False)
```

Step 4

- Load dataset and convert `creation_time` to datetime.
- Keep events from 2008 onward and save.
- Filter rows with valid demographics (`gender` m/f, valid `country`, or valid `age`).
- Save demographic-filtered data.
- Filter again for events since 2008 and save.
- Exclude rows with `gender` = 'n' and `age` = -1, then save.
- Drop rows with any missing values and save cleaned data.
- Print counts of rows and unique users/tracks at key steps.

[ ]:
```python
import pandas as pd

df = pd.read_csv('filtered-merged-listening.tsv', sep='\t')

# Convert creation_time to datetime format
df['creation_time'] = pd.to_datetime(df['creation_time'])

# Define cutoff date (January 1, 2008)
cutoff_date = pd.Timestamp('2008-01-01')
recent_df = df[df['creation_time'] >= cutoff_date]

print(f"Remaining events after 2009: {len(recent_df)}")
print(f"Unique users: {recent_df['user_id'].nunique()}")
print(f"Unique tracks: {recent_df['track_id'].nunique()}")

recent_df.to_csv('filtered-since-2008-listening.tsv', sep='\t', index=False)
```

```
df = pd.read_csv('filtered-since-2008-listening.tsv', sep='\t')

# Apply demographic filtering and keep rows where at least one demographic␣
 ↪attribute is valid
filtered_df = df[
    (df['gender'].isin(['m', 'f'])) |
    (
        df['country'].notna() &
        (df['country'].str.strip() != '') &
        (df['country'].str.lower() != 'nan')
    ) |
    (df['age'] != -1)
]


print("Number of rows with at least one valid demographic attribute:",␣
 ↪len(filtered_df))

num_users = filtered_df['user_id'].nunique()
num_tracks = filtered_df['track_id'].nunique()
print(f"Unique user_id count: {num_users}")
print(f"Unique track_id count: {num_tracks}")

filtered_df.to_csv('2.filtered-demographics-listening.tsv', sep='\t',␣
 ↪index=False)

df = pd.read_csv('2.filtered-demographics-listening.tsv', sep='\t')

df['creation_time'] = pd.to_datetime(df['creation_time'])

df_filtered = df[df['creation_time'] >= pd.Timestamp('2008-01-01')]

num_users = df_filtered['user_id'].nunique()
num_tracks = df_filtered['track_id'].nunique()

print(f"Remaining rows after 2008: {len(df_filtered)}")
print(f"Unique user_id count: {num_users}")
print(f"Unique track_id count: {num_tracks}")

df_filtered.to_csv('filtered-demographics-2008onward.tsv', sep='\t',␣
 ↪index=False)

df = pd.read_csv('filtered-demographics-2008onward.tsv', sep='\t')

filtered_df = df[(df['gender'] != 'n') & (df['age'] != -1)]
```

```
print(f"Rows after excluding gender 'n' and age -1: {len(filtered_df)}")
print(f"Unique user_id count: {filtered_df['user_id'].nunique()}")
print(f"Unique track_id count: {filtered_df['track_id'].nunique()}")

filtered_df.to_csv('fixed-last.tsv', sep='\t', index=False)

df = pd.read_csv('fixed-last.tsv', sep='\t')

# Drop rows with any missing values
clean_df = df.dropna()

clean_df.to_csv('fixed-last-clean.tsv', sep='\t', index=False)

print(f"Original rows: {len(df)}")
print(f"Cleaned rows (no missing values): {len(clean_df)}")
print(f"Dropped rows: {len(df) - len(clean_df)}")
```

Step 5 & Step 6

- Loads a cleaned TSV dataset into `df`.
- Create a binary column `binary_listen` set to 1 for all rows
- Find unique tracks and samples up to 100,000 tracks randomly without replacement.
- Filter the DataFrame to keep only the sampled tracks.
- Drop the original `count` column to keep only binary data.
- Save the sampled data to a new TSV file.

[ ]:
```
import pandas as pd
import numpy as np

df = pd.read_csv('fixed-last-clean.tsv', sep='\t')

# Convert listening counts to binary (1 if listened at least once, 0 otherwise)
# Since the dataset only has listening events, we just set count=1
df['binary_listen'] = 1

# Now, sample 100,000 tracks uniformly at random
unique_tracks = df['track_id'].unique()

# If less than 100k tracks, take all
num_tracks_to_sample = min(100_000, len(unique_tracks))

sampled_tracks = np.random.choice(unique_tracks, size=num_tracks_to_sample,␣
 ↪replace=False)

# Filter dataframe to keep only the sampled tracks
final_df = df[df['track_id'].isin(sampled_tracks)].copy()
```

```
final_df = final_df.drop(columns=['count'])

final_df.to_csv('sampled_100k.tsv', sep='\t', index=False)

print(f"Sampled {num_tracks_to_sample} tracks.")
print(f"Final dataset has {final_df['user_id'].nunique()} unique users and␣
  ↪{final_df['track_id'].nunique()} unique tracks.")
print(f"Total listening events (binary interactions): {len(final_df)}")
```

Step 7

- Loads the dataset `'sampled_100k.tsv'` into `df`.
- Set up 5-fold cross-validation with shuffled user splits.
- Creates a directory `cv_splits` to store the folds.
- For each fold:
  - Split users into train+val and test sets based on KFold indices.
  - Further split train+val users into 60% train and 20% validation users.
  - Filter the main DataFrame into train, val, and test subsets by user IDs.
  - Define `split_input_target` to split each user's interactions into 80% input and 20% target by shuffling their items.
  - Apply this split to validation and test sets (creating input and target parts).
  - Save train, val input, val target, test input, and test target files for the current fold.

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
import os


df = pd.read_csv('sampled_100k.tsv', sep='\t')

users = df['user_id'].unique()
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Prepare output dir
os.makedirs("cv_splits", exist_ok=True)

# Iterate over folds
for fold, (train_val_idx, test_idx) in enumerate(kf.split(users), start=1):
    users_train_val = users[train_val_idx]
    users_test = users[test_idx]

    # Further split train_val into train and val (60% / 20% of total users)
    n_total = len(users)
    n_train = int(0.6 * n_total)
```

```
    n_val = int(0.2 * n_total)

    train_users = users_train_val[:n_train]
    val_users = users_train_val[n_train:]

    # Get user interaction subsets
    train_df = df[df['user_id'].isin(train_users)].copy()
    val_df = df[df['user_id'].isin(val_users)].copy()
    test_df = df[df['user_id'].isin(users_test)].copy()

    def split_input_target(user_df):
        """Split each user's items into 80% input, 20% target."""
        input_rows = []
        target_rows = []
        for user_id, group in user_df.groupby('user_id'):
            items = group.sample(frac=1, random_state=42)  # Shuffle
            split_point = int(len(items) * 0.8)
            input_rows.append(items.iloc[:split_point])
            target_rows.append(items.iloc[split_point:])
        return pd.concat(input_rows), pd.concat(target_rows)

    val_input, val_target = split_input_target(val_df)
    test_input, test_target = split_input_target(test_df)

    # Save all parts
    fold_dir = f"cv_splits/fold_{fold}"
    os.makedirs(fold_dir, exist_ok=True)

    train_df.to_csv(f"{fold_dir}/train.tsv", sep="\t", index=False)
    val_input.to_csv(f"{fold_dir}/val_input.tsv", sep="\t", index=False)
    val_target.to_csv(f"{fold_dir}/val_target.tsv", sep="\t", index=False)
    test_input.to_csv(f"{fold_dir}/test_input.tsv", sep="\t", index=False)
    test_target.to_csv(f"{fold_dir}/test_target.tsv", sep="\t", index=False)

    print(f" Fold {fold}:")
    print(f"  Train users: {len(train_users)}")
    print(f"  Val users: {len(val_users)} (input: {len(val_input)}, target:␣
↪{len(val_target)})")
    print(f"  Test users: {len(users_test)} (input: {len(test_input)}, target:␣
↪{len(test_target)})")
```

0.2 Pre-processing of Book-Crossing Dataset

For the Book-Crossing dataset we decided to keep only the books which had ratings equal or bigger than 6, since books don't have listening counts. And if an author's books received fewer than 2 ratings across all users, those books are excluded from the dataset.

0.2.1 1. Preprocess the Data

- Filter interactions to only include ratings **6**.
- Map gender values to standardized labels: `'female' → 'f'`, `'male' → 'm'`.
- Add a new column `binary_listen = 1` to indicate a positive interaction (possibly for implicit feedback).
- Remove unnecessary columns: `'Publisher'`, `'Book-Title'`, and `'Book-Author'`.
- Convert `user_id` to string type to ensure consistency across the pipeline.

0.2.2 2. Prepare for Cross-Validation

- Extract all **unique user IDs**.
- Set up **5-fold user-based cross-validation** with shuffling and fixed random seed for reproducibility.

0.2.3 3. For Each Fold (1 to 5)

a. Split Users

- Split users into:
  - **Train + Validation** (80% of users)
  - **Test** (20% of users)
- Further split the 80% into:
  - **Train** (60% of total users)
  - **Validation** (20% of total users)

0.2.4 4. Subset the DataFrame

- Create three datasets:
  - `train_df`: interactions of training users.
  - `val_df`: interactions of validation users.
  - `test_df`: interactions of test users.

0.2.5 5. Create Input/Target Splits for Validation and Test Sets

- For each user in `val_df` and `test_df`:
  - Skip users with $<$ **2 interactions**.
  - Shuffle their items.
  - Split 80% as **input** interactions, and 20% as **target**.
- Return two DataFrames: `*_input` and `*_target` for validation and test.

0.2.6 6. Save Each Fold to Disk

- Create a new directory for each fold: `book_folds/fold_{fold}`.
- Save the following files inside it:
  - `train.tsv` — Training data.
  - `val_input.tsv` — 80% of validation user interactions.
  - `val_target.tsv` — Remaining 20% of validation interactions.
  - `test_input.tsv` — 80% of test user interactions.
  - `test_target.tsv` — Remaining 20% of test interactions.

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
import os

df = pd.read_csv('book_dataset.tsv', sep='\t')

# Rename columns to match expected schema
df = df.rename(columns={
    'User-ID': 'user_id',
    'ISBN': 'item_id',
    'Book-Rating': 'rating',
    'Author-Gender-Guess': 'gender'
})

# Filter only books with rating >= 6
df = df[df['rating'] >= 6].copy()
# Map gender to 'f' and 'm'
df['gender'] = df['gender'].str.lower().map({'female': 'f', 'male': 'm'})
# Add binary_listen column
df['binary_listen'] = 1
# Drop unwanted columns
df = df.drop(columns=['Publisher', 'Book-Title', 'Book-Author'])

df['user_id'] = df['user_id'].astype(str)

users = df['user_id'].unique()
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Prepare output dir
os.makedirs("book_folds", exist_ok=True)

# Iterate over folds
for fold, (train_val_idx, test_idx) in enumerate(kf.split(users), start=1):
    users_train_val = users[train_val_idx]
    users_test = users[test_idx]
```

```
    # Further split train_val into train and val (60% / 20% of total users)
    n_total = len(users)
    n_train = int(0.6 * n_total)
    n_val = int(0.2 * n_total)

    train_users = users_train_val[:n_train]
    val_users = users_train_val[n_train:]

    # Get user interaction subsets
    train_df = df[df['user_id'].isin(train_users)].copy()
    val_df = df[df['user_id'].isin(val_users)].copy()
    test_df = df[df['user_id'].isin(users_test)].copy()

    def split_input_target(user_df):
        """Split each user's items into 80% input, 20% target."""
        input_rows = []
        target_rows = []
        for user_id, group in user_df.groupby('user_id'):
            if len(group) < 2:
                continue  # Skip users with fewer than 2 interactions
            items = group.sample(frac=1, random_state=42)
            split_point = int(len(items) * 0.8)
            input_rows.append(items.iloc[:split_point])
            target_rows.append(items.iloc[split_point:])
        return pd.concat(input_rows), pd.concat(target_rows)

    val_input, val_target = split_input_target(val_df)
    test_input, test_target = split_input_target(test_df)

    # Save
    fold_dir = f"book_folds/fold_{fold}"
    os.makedirs(fold_dir, exist_ok=True)

    train_df.to_csv(f"{fold_dir}/train.tsv", sep="\t", index=False)
    val_input.to_csv(f"{fold_dir}/val_input.tsv", sep="\t", index=False)
    val_target.to_csv(f"{fold_dir}/val_target.tsv", sep="\t", index=False)
    test_input.to_csv(f"{fold_dir}/test_input.tsv", sep="\t", index=False)
    test_target.to_csv(f"{fold_dir}/test_target.tsv", sep="\t", index=False)

    print(f" Fold {fold}:")
    print(f"  Train users: {len(train_users)}")
    print(f"  Val users: {len(val_users)} (input: {len(val_input)}, target:␣
↪{len(val_target)})")
    print(f"  Test users: {len(users_test)} (input: {len(test_input)}, target:␣
↪{len(test_target)})")
```