Welcome To Our

Personalized Recommendation Systems

Overview

This Python script implements a movie recommendation system using Genetic Programming (GP) with the DEAP library. It leverages the MovieLens dataset to train and evaluate a model that recommends movies based on user ratings, movie genres, and other features. The system uses parallel processing, caching, and early stopping to optimize performance.

Code Structure and Functionality

1. Imports and Setup

Libraries: Uses random, operator, numpy, pandas, pickle, os, sklearn, collections, deap, multiprocessing, logging, and time for data handling, GP, and parallel processing.
Logging: Configured with logging.basicConfig to output info, warnings, and errors to the console, suitable for Google Colab.

2. Data Loading (load_data)

Purpose: Loads MovieLens dataset (movies.csv, ratings.csv).
Implementation: Reads CSV files into pandas DataFrames, logs the number of movies and ratings, and handles file not found or other errors.
Output: Returns movies_df and ratings_df.

3. Train-Test Split (user_train_test_split)

Purpose: Splits ratings data per user into training and test sets (default: 80% train, 20% test).
Implementation:
Caches split data to /content/train_test_split.pkl to avoid recomputation.
Groups ratings by userId, skips users with <5 ratings, and uses train_test_split from sklearn with a fixed random seed (42).
Logs and caches the split.
Output: Returns train_df and test_df as pandas DataFrames.

4. Feature Maps (build_feature_maps)

Purpose: Builds feature dictionaries for GP model input.
Implementation:
Caches feature maps to /content/feature_maps.pkl.
Computes:
user_avg: Mean rating per user.
item_avg: Mean rating per movie.
item_count: Number of ratings per movie.
item_genres: Genres per movie from movies_df.
user_item_genre_scores: Genre-based scores per user-movie pair.
candidate_items_per_user: Unseen movies per user.
Merges ratings with genres, computes user-genre preferences, and logs the process.
Output: Returns a tuple of feature maps.

5. Recommendation Function (recommend)

Purpose: Generates top-N movie recommendations for a user using a GP individual.
Implementation:
Compiles the GP individual into a function using toolbox.compile.
Vectorizes inputs (u_avg, i_avg, i_count, g_score) using NumPy arrays.
Computes scores, handles errors by falling back to a loop-based approach, and returns the top-N movie IDs.
Output: List of top-N movie IDs.

6. Fitness Evaluation (evaluate_individual, evaluate)

Purpose: Evaluates a GP individual's performance using precision, recall, and F1 score.
Implementation:
Iterates over a subset of test users, generates recommendations, and computes metrics:
Precision: Hits / 10 (N=10 recommendations).
Recall: Hits / number of test items.
F1: Harmonic mean of precision and recall.
Returns average F1 and per-user metrics.
evaluate wraps evaluate_individual for parallel processing.

7. Genetic Programming Setup

Primitive Set (pset):
Inputs: u_avg, i_avg, i_count, g_score.
Operators: np.add, np.subtract, np.multiply, safe_div (division with small constant to avoid zero division), np.tanh, np.abs.
DEAP Configuration:
Creates FitnessMax and Individual classes.
Registers tools for individual creation, population initialization, compilation, evaluation, selection (selTournament), crossover (cxOnePoint), and mutation (mutUniform).
Limits tree size to 17 nodes using staticLimit.

8. Early Stopping (EarlyStopping)

Purpose: Stops evolution if no fitness improvement occurs for a specified number of generations (patience).
Implementation:
Tracks the best fitness and generation.
Stops if no improvement after patience generations (default: 3).
Logs stopping condition.

9. Custom Evolutionary Algorithm (custom_eaSimple)

Purpose: Runs GP evolution with crossover, mutation, and early stopping.
Implementation:
Initializes population, evaluates fitness, and updates a Hall of Fame (hof).
Performs selection, crossover (probability: cxpb), and mutation (probability: mutpb) per generation.
Re-evaluates invalid individuals, updates population, and logs stats.
Stops early if EarlyStopping triggers.
Output: Returns final population and logbook.

10. Main Execution

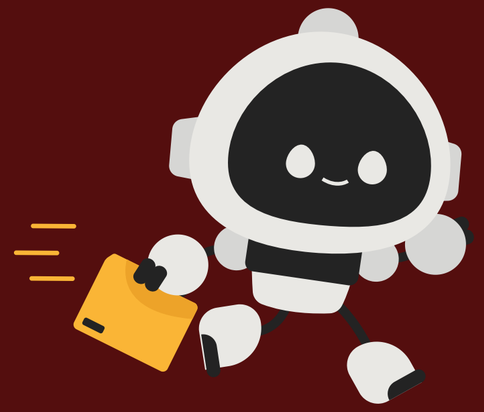
Steps:

Loads data, splits it, and builds feature maps.
Selects test users and sets up parallel evaluation with multiprocessing.Pool.
Initializes population (25 individuals), Hall of Fame, and statistics (avg, max fitness).
Runs custom_eaSimple for up to 8 generations with crossover probability 0.5 and mutation probability 0.3.
Closes the process pool.

Outputs:

Best GP individual and its F1 score.
Total runtime.
Recommendations for a fixed user (default: userId=2).
Evaluation metrics (precision, recall, F1) for all test users and the fixed user.

Strengths
Modularity: Functions are well-separated, reusable, and documented.
Optimization:
Caching (train_test_split, feature_maps) reduces recomputation.
Parallel evaluation with multiprocessing.Pool speeds up fitness computation.
Early stopping prevents unnecessary iterations.
Robustness:
Error handling for file loading, caching, and vectorized scoring.
Logging provides detailed execution insights.
Feature Engineering: Incorporates user averages, item averages, rating counts, and genre-based scores for richer recommendations.



Conclusion

The script effectively implements a GP-based movie recommendation system with robust data processing, feature engineering, and optimization techniques. While it performs well for the MovieLens dataset, improvements in scalability, portability, and evaluation metrics could enhance its applicability to larger or more diverse datasets.