Game Recommendation using Multiobjective Evolutionary Optimization

Project for Evolutionary Algorithms course

Jakub Bartczuk

June 2020

1 Introduction

To use machine learning methods for recommender systems the evaluation of recommendation procedure needs to be defined as optimization problem that captures this procedure's essential properties.

There exist various metrics commonly used for capturing these properties, but every such method has its drawbacks.

Because recommendation evaluation metrics can have inconsistent aims, this problem can be naturally formulated as multiobjective optimization. For such problems population-based methods like some evolutionary algorithms can be used to not only find *a* solution, but to find diverse set of nondominated solutions.

The following problem tackes problem of recommending Steam games. For evaluation two metrics are used, and three methods for recommendation are combined to jointly optimize these metrics.

2 Datasets

Two datasets were chosen for building recommenders:

- first dataset contains information about Steam game purchases and amount of time each users spent playing their games.
- second dataset has game metadata: genre, tag and description data is used for building recommendations.

Metadata information on genres, tags and descriptions were used to build features for games: these columns are essentially text columns, so TF-IDF was used to build vectors.

3 Recommenders

Two classical approaches to recommendations use item-item similarity and user-user similarity. These approaches are complementary - user-based system can be thought of a baseline, but it fails for users that didn't rate many items. On the other hand item-based recommender can work with minimal data (and alleviate the so-called cold start problem), but it will lack novelty.

User-user recommender considers sequences of items that were rated by similar users, and combines them using user similarity.

Item-item recommender for a given set of items and their ratings provides items that are most similar to rated items, with scores weighted by the similarity.

The project also uses a simpler version of item-item recommender, where an average of features of rated items is calculated, and items similar to these average are returned.

4 Evaluation metrics

We will use two ranking metrics to evaluate the recommender.

4.1 recall@k

It is essentially false negative rate for top k items. For a sequence I of ratings and $\hat{I}$ of predicted items

$recall@k(I, \hat{I}) = \frac{1}{min(k,|I|)} |I \cap \hat{I}|$

4.2 Kendall's τ

The second metric is used to measure how similar are orderings of I and $\hat{I}$.

For a pair of zipped of items from I, I'

$((i_k, i'_k)\ (i_j, i'_j))$ is said to be *concordant* if their signs are not zero and coincide: $sign(i_k - i_j) = sign(i'_k - i'_j)$. It is said to be *disconcordant* if the signs are not equal.

Kendall rank correlation coefficient for two rankings with n_c concordant pairs and n_d disconcordant pairs is defined as

$\tau = \frac{n_c - n_d}{C^n_2}$

Where C^n_2 is the number of pairs from n-element set.

4.3 Tradeoffs

Since recall measures false negative rate, increasing number of recommended items will never decrease it. On the other hand adding recommendations changes ranking, so by adding noisy entries it might decrease τ.

5 Problem definition

We have three recommenders R_u, R_c, R_m, user, item-based and mean item-based respectively.

For each user's item list I each recommender outputs predicted item list $R(I)$.

The recommenders are combined with weighted average - for parameter vector

$$w = (w_u, w_c, w_m)$$

We have predicted ratings

$$\hat{I} = w_u R_u(I) + w_c R_c(I) + w_m R_m(I)$$

Thus the optimized function is

$$F : \mathbb{R}^3_+ \to \mathbb{R}^2$$

$$F(w) = [recall@10(\hat{I}), \tau(\hat{I})]$$

More precisely domain of F is 3d simplex, $S = \{w \in \mathbb{R}^3 : \sum w_i = 1, w_i \geq 0\}$

6 Optimization Algorithms

The following optimization algorithms were used:

- NSGAII
- UNSGAIII
- MOEA/D

The first algorithm was implemented by author [1], and for two other implementations from *pymoo*[2] library were used.

[1] multiobjective.py
[2] pymoo.org

6.1 NSGAII

NSGAII with basic mutation and no crossover. This is the simplest method so we use population of size 1000.

6.2 UNSGAIII

An Evolutionary Many-Objective Optimization Algorithm Using Reference-point Based Non-dominated Sorting Approach

Algorithm that is an improved version of NSGA.

Key differences with NSGAII:

- using *reference directions* (NSGA-III)
- using tournament method for parent selection

Reference directions were introduced to sidestep the problem with multidimensional optimization: in high dimensions Pareto front size grows exponentially, and standard methods from NSGAII might become less meaningful, computationally more expensive and harder to visualize/evaluate.

To address this, a set of diverse points is chosen at start, and optimization for subpopulations proceeds to make it closer to these *reference points*.

6.3 MOEA/D

MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition

MOEA/D is a multiobjective optimization algorithm that uses **decomposition** into scalar subproblems.

The scalar subproblems are chosen so that optimizing them separately yields good approximation of Pareto Frontier, at least when subpopulations that optimize them are not too similar.

6.3.1 Decomposition

MOEA/D redefines optimization problem as separate single ojective optimization problems: for given λ it weights objectives of $f(x)$.

An example of such approach is scalarization: $g(x) = \lambda^T f(x)$

Chosen weight vectors $\Lambda = (\lambda^i)_{i<k}$ represent tradeoffs between optimizing different objectives (with respect to a given reference point).

For each of these λ we have a single objective function - this problem can be solved using classical Evolutionary Algorithms.

The usefulness of this approach is that gs are similar for similar λs, so we can reuse knowledge obtained optimizing using close λs.

6.3.2 Decomposition using Chebyshev approach

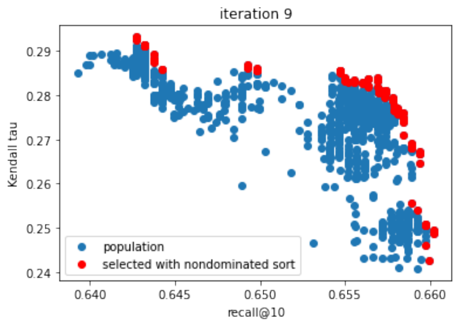
Original MOEAD uses so-called Chebyshev approach: for m objectives, vector of subproblem weights λ and a reference point $z \in R^m$ we define

$$g^{Ch}(x|\lambda, z) = \|\lambda_i^T(f(x) - z)\|_\infty = max_{i<m}\lambda_i|f_i(x) - z_i|$$

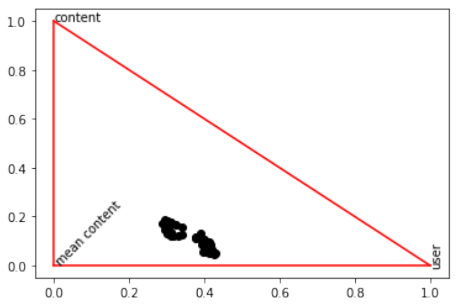
7 Results

We see that all three methods have found varied sets of solutions that illustrate tradeoffs between metrics. The first method illustrates the whole population and contrasts set of solutions selected with NSGAIII, and other methods show only small number of nondominated solutions.

7.1 NSGAII

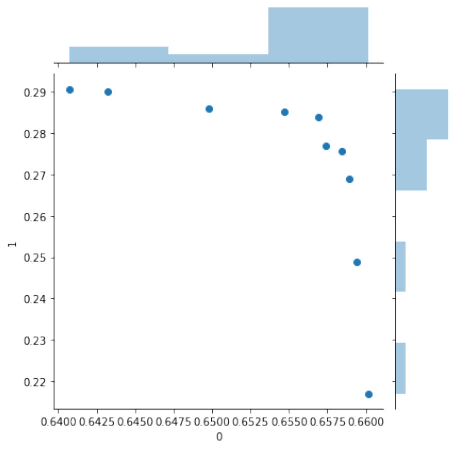


(a) Objectives - F(w)

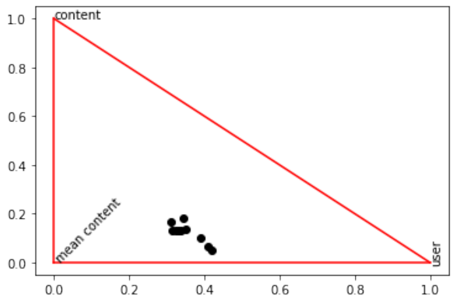


(b) Weights - w

7.2 UNSGAIII

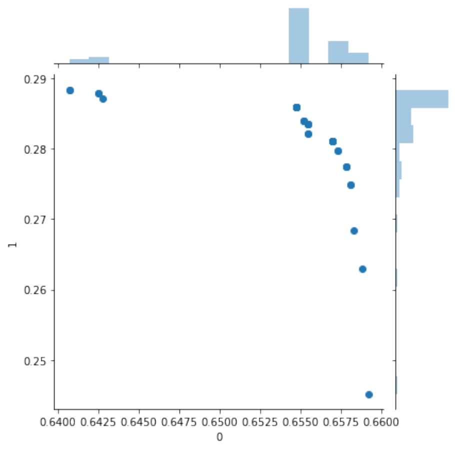


(a) Objectives - F(w)

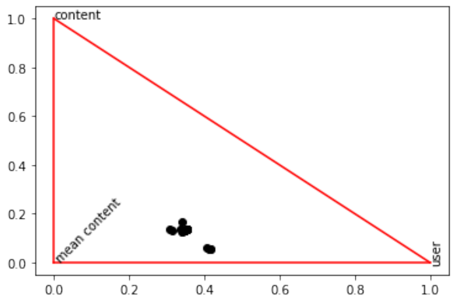


(b) Weights - w

7.3 MOEA/D



(a) Objectives - F(w)



(b) Weights - w

8 Comments, future steps

The code and notebooks with further details can be found on github [3]

The next step will be to implement online evaluation - provide users possibility to manually add games and their respective ranks.

From the machine learning perspective, it will be interesting to add more advanced methods for recommendation. For now only kNN methods were used, because they allow to easily add and evaluate new users; typically ML methods for recommenders like Matrix Factorization and Factorization Machines assume fixed user-item matrix, which means that user's ID is a feature.

Another interesting direction will be to consider another metric for recommendation diversity; the author considered adding this, but measuring diversity is complex topic and itself. Adding diversity would mean additional choices on how to calculate it.

[3] https://github.com/lambdaofgod/game_recommender