Measurement & Results - Parallel Probabilistic Matrix Factorization

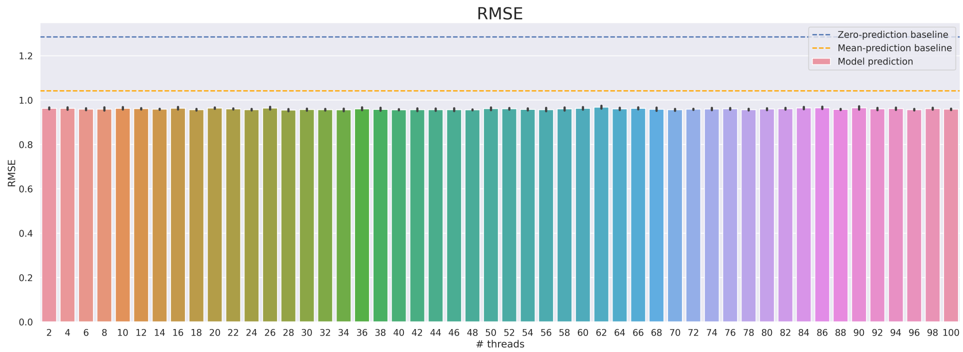
Anders Geil, Sol Park, Yinuo Jin

Overview

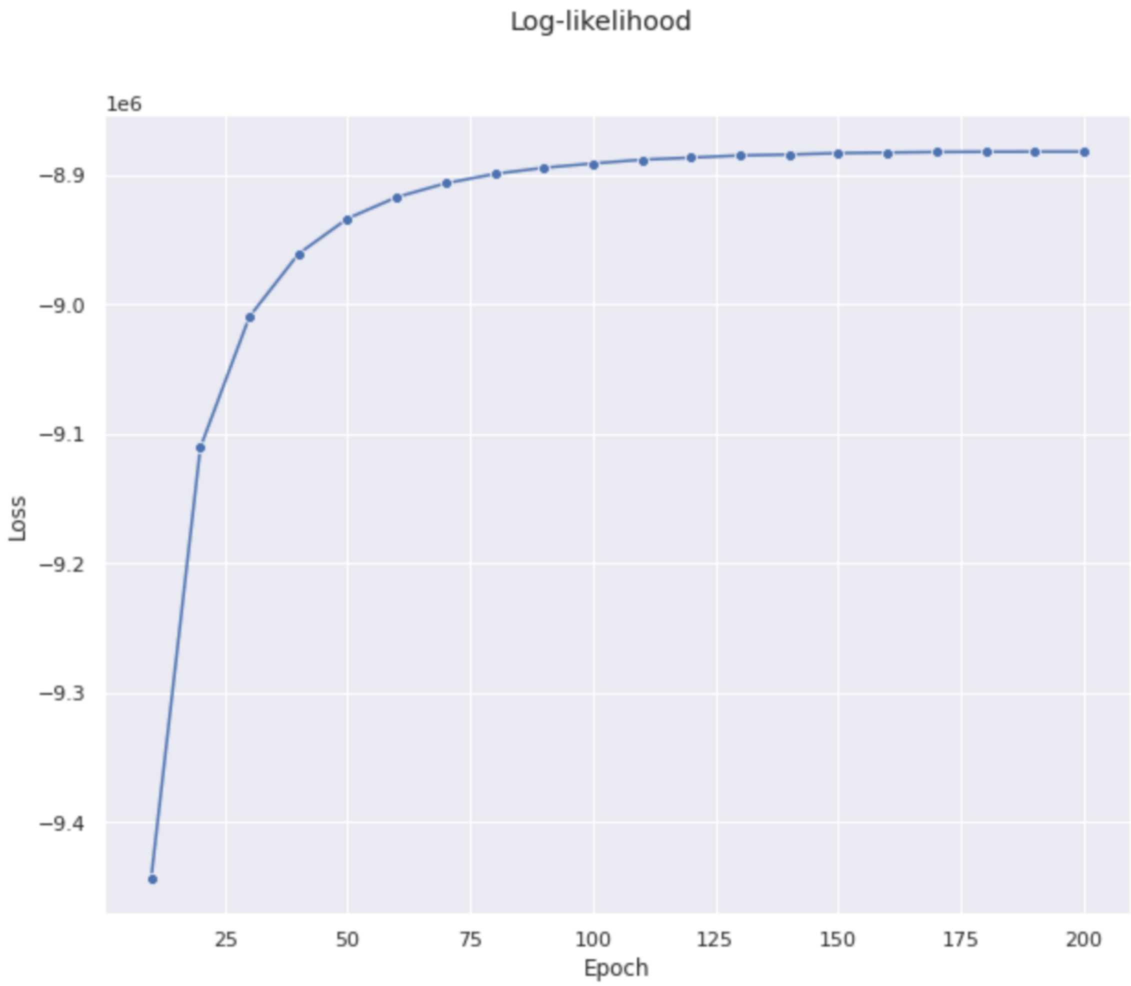
The main goal of our project is to provide a scalable implementation of probabilistic matrix factorization (PMF) to process large dataset efficiently. Therefore we benchmarked our model on the sample Movielens 100K dataset, with strong focus on two major factors: (1). the correctness of our implementation; (2). efficiency.

(1). Correctness

We splitted the dataset into mutually exclusive training and test set during our model-fitting phase. We applied the parameters learned from the training set to predict the unseen rating values in the dataset and compare with the ground-truth value. The following plot shows that our parallelized model consistently shows better Root-Mean-Squared Error (RMSE) in every single parallelization setup (from 2 threads to 100 threads) then two baseline predictions: (a). "zero" prediction (we always predict rating value as 0); (b). "mean" prediction (we always predict rating value as the average rating value in the test set):



The second indicator for correctness verification of our model is loss function. We calculated the joint probability of seeing the dataset and the model parameters ($\Pr(\mathrm{data}, \theta, \beta)$) for each 10 epochs during the model fitting step. We expect the loss function value should gradually converge towards 0. The following plot shows a sample loss vs. epoch measurement in a 200-epoch training with our example dataset (MovieLen 100K):



(2). Efficiency

We measured two round of comparisons to show the significant running time speedup with the aid of our parallelization architecture. First, we implemented and measured the sequential PMF in both C++ and Python, our C++ sequential script finished training in around 160s whereas the Python version takes over 800s.

Second, we benchmarked the parallelization performance on a 12-core Linux machine by incrementing two threads a time from 2 threads to 100. The following plot shows that, we could at best further improve the running time to around 2.5, which shows over 75X speedup with sequential version (C++), and over 320X speedup with python implementation.