Project Proposal

Title: Parallel Implementation of a Recommendation System

About the project

The goal of this project is to parallelize the process of generating product recommendations to Amazon's users. Specifically, we aim to predict, as accurately as possible, the rating a user gives to a particular product based on Spark and OpenMP. If we are able to make accurate predictions, we can recommend products to users that they have not bought yet.

Methodology

The raw dataset that we use for this project is the "Amazon Product Data". This dataset contains 142.8 million product reviews, as well as the associated metadata from Amazon spanning May 1996 to July 2014. Therefore, the size of this dataset is considerable (over 100 GB), and it is not practical to fit all the data on a single machine and to make useful recommendations.

The two popular collaborative filtering system models that we will use are: Standard Collaborative Filtering Model (SCF), and Matrix Factorization (MF) optimized through Alternative Least Square (ALS).

The advantage of SCF model is that it is easy to understand and implement. However, this model suffers from the limitations that it is not computationally efficient and it does not handle sparsity well (i.e. It does not have accurate predictions if there are not enough reviews for a product).

In light of the limitations of SCF, we will proceed with matrix factorization, which is a more advanced technique that decomposes the original sparse matrix to lower-dimensional matrices incorporating latent vectors. These latent vectors may include higher-level attributes which are not captured by ratings for individual products.

In order to increase the performance, all models are running on a multi-node cluster, which is further optimized by increasing the number of threads on each node through OpenMP.

Language/Libraries

For our project, we'll use Intel MKL which is now packaged within Intel Distribution for Python, since it provides the highest speedup among all BLAS variants. By using Intel Distribution for Python, we can achieve OpenMP's multithreading performance, and yet enjoy the simplicity of Python.