Recommending the Recommendation Engine

Mohit Khanna   
*Department of Computer Science*  
*University of Illinois at Urbana Champaign  
Champaign, IL, USA  
mkhanna2@illinois.edu*Shikhar Khanna  
*Department of Computer Science*  
*University of Illinois at Urbana Champaign  
Champaign, IL, USA  
shikhar2@illinois.edu*Soumya Nanda  
*Department of Computer Science  
University of Illinois at Urbana Champaign  
Champaign, IL, USA  
soumyan2@illinois.edu*

***Abstract—*** *Several approaches and algorithms, based on mathematical models and/or machine learning, exist for building recommendation systems. Which one(s) will work best for which domain is not obvious at all, and typically requires experimenting with multiple approaches. In this paper, we have tried to address this problem by building a tool that encapsulates several recommender algorithms and enabling the user to quickly try them out with their data and discover the approach that works best.*

*We tried our tool with publicly available datasets, specifically MovieLens, and could recommend the KNN algorithm as being the best recommender system for them.*

***Keywords—*** *Recommender system, K Nearest Neighbors, Collaborative Filtering, Alternating Least Squares, Singular Value Decompsition.*

# Introduction

With the explosion of information, countless product offerings and thousands, if not millions, of choices in today’s digital world, recommender systems have become vital, both for companies as well as for consumers. Recommendation systems seek to bring relevance to consumers by correlating their needs and interests with the product offerings from a vast pool of information that is practically impossible to navigate without any computational help.

People's tastes vary, but are generally stable over time. They tend to like things that are similar to other things they have liked in the past. However, their tastes and preferences also evolve over time, and they may or may not have similar taste as other people. Recommender systems try to exploit these behaviors and capture these patterns in data by learning from customer choices and predict what they might like now and in future.

# Background

There are a variety of approaches and implementation methods to choose when building a recommendation engine. The simplest method is the **Simple Popularity Model**, which recommends the most popular items as defined by the entire customer pool, to every user. This method is quick to implement and is stereotyped but lacks personalization. Next is **Content-Based**, which can be implemented either through a Classification algorithm or through Content-based algorithms. Classification algorithms generally do not work well with lot of features and are, therefore, not scalable. Content-based algorithms are used only when detailed metadata/tags of items are available. The third approach is **Collaborative Filtering**, which ignores item context and uses past user behavior to build patterns. Collaborative filtering can be of two types - User-User and Item-Item. User-User approach is used to recommend items liked by look-alike customers and Item-Item is used to recommend items that look-alike other items. E.g. for recommending “Users who are similar to you also liked …” we will use User-User and for recommending “Users who liked this item also liked …” we will use Item-Item. There is yet another technique in Collaborative Filtering called Dimensionality Reduction, wherein the User-Item-Rating matrix is reduced to a small set of ‘taste’ dimensions to make the computation more scalable and faster.

Which approaches are more appropriate than others depends on several factors such as the domain we are in, e.g. news, products, people (matchmaking), entertainment (music, movies), etc. An interesting property of the domain is whether to recommend new items or old ones that user has already experienced. New items are suitable for movies and books, while old ones may be suitable for groceries and music.

# Proposal

On researching the subject, we discovered vast arrays of custom recommendation engines like the friends recommendation system by Facebook, movies recommendation system by Netflix implemented by industries to suit their offerings and customer base. There are vast differences in the approaches of implementation, methodology and assumptions – either making the system limited to a small problem domain or delivering suboptimal performance due to “one size fits all” approach. The recommendation problem domain currently is flooded with custom approaches due to variation of data, variation of approach and variation of methodology. E.g. the dataset could be explicit customer ratings, transaction records or even user clicks. Similarly, approaches may vary too, as has been discussed in previous section. The methodologies have variations too – do you find similarity by K-nearest approach, SVD approach, Jaccard similarity, Cosine similarity or simply by calculating Euclidean or Manhattan distance?

As a result, a marketing company or an e-commerce shopping site that wants to exploit users’ shopping transactions data or product ratings data to generate recommendations for its users, or for that matter, any scenario that requires building a recommender system, will typically involve trying out different algorithms and methodologies, analyze the recommendations generated by each and determine which approach generated the “best” recommendations. Obviously, the best recommender in each situation will vary depending on the characteristics of the domain in question and the type of user/item data and ratings available.

Our project aims to simplify the above problem by developing an application that can serve as generic data mining application for collaborative filtering that leverages multiple implementations and is not restricted by specific kind of data, approach or methodology. The application user does not need to know about the mathematical or algorithmic suitability for the problem at hand. Instead, our application will run given input data through various recommendation approaches and figure out which approach generates the best recommendations and has the least “error” for a randomly chosen training dataset. Through a shell interface, the user will have a menu of recommender systems to select from, invoke one or more of them on demand, examine recommendations generated by each approach and then pick the one(s) with highest accuracy. The user will have saved the effort of understanding and implementing various recommender approaches by themselves, before finalizing on the most promising approach.

We have tried implementing the following approaches in our application that will each run on the same dataset and report the accuracy metric. The user will be advised to select the approach with the highest accuracy score.

* KNN (K Nearest Neighbors)
* Alternating Least Squares (ALS) / Singular Value Decomposition (SVD)
* Jaccard similarity
* Cosine similarity
* Centered Cosine similarity
* Euclidean / Manhattan similarity.

**Movie Lens**is a classic dataset suitable for investigating recommender systems, and we plan to use it.  We are curious to learn and discover which approach(es) will work best for this domain and why.  Investigating and understanding this question is part of our learning plan through this project. We are planning to run the ml-latest dataset from Group Lens which contains 26,000,000 ratings across 270,000 users and 45,000 movies. We shall be leveraging Spark to perform this heavy computation. We are also planning to run the algorithms on the customer transaction/interactions dataset which will give us a nice baseline criterion to evaluate the performance of the algorithms on different types of datasets.

# Implementation

The recommendation engine has been built in Python and supports multiple approaches for doing recommendation, as discussed above. The application has two modes of operation:

* Batch mode allows a dataset in the form of user-item rating matrix to be fed as input, and the application runs the algorithms and generates top item recommendations for every user in the dataset and saves them.
* Interactive (shell) mode allows user to specify the dataset, specify the algorithm and specify a particular user (in the data) and the application returns top recommendations for the particular user based on the chosen algorithm. The recommendations are either looked up from the previously pre-computed results, or are computed afresh, based on the parameter passed in.

We have implemented 3 algorithms: K-Nearest Neighbors, Singular Value Decomposition and Alternating Least Squares to compute recommendations, given a user-item rating matrix as input. We have developed this in Python using the following datasets as examples:

1. Movie Lens Small dataset with ~100,000 ratings on 9000 movies by 700 users. Given a user with their movie ratings, system recommends top X movies for the user.
2. Item Purchase Transactions ("http://www.salemmarafi.com/wp-content/uploads/2014/04/lastfm-matrix-germany.csv") Given a user with their item purchases, system recommends top X items for purchase by the user.

After gaining experience with these datasets, we processed the **Movie Lens Full** dataset with 26 million ratings on 45,000 movies by 270,000 users using the computing power of **Spark** cluster.

To leverage the computing resource of a multiple machines we even extended the application with the support of running it on Spark distributed platform. We made use of Spark ALS technique to create recommendations for a really large dataset. The results of those computations have been shared at the end of this paper.

# Challenges & Resolution

During development, we had several issues in implementing an optimized solution to achieve an efficient performance. Given the computational complexity, we were concerned about the wait time before the user sees the results each time; we therefore decided to implicitly generate the recommendation for every user and save it in a flat file. This feature will allow the user to refer the recommendations from the last run and for every user, even though user might be interacting with it for a one-time purpose. The advantage of this feature is that (given the underlying data hasn’t changed significantly), next time the user has an option to pick recommendation directly from the flat file instead of having it run again, saving time and resources.

Another challenge was to give user some metric to test various recommendation methodologies and pick the one most suitable to the type of recommendation problem and nature of the data. Generally, recommendation is an empirically defined problem which needs user evaluation, thus making it difficult for us to evaluate results. To test the relevance of the recommendation, we created a control group of artificial users and populated them with a few test choices to evaluate whether their recommendations made sense or not. This helped to ensure that the implementation is on correct track. For the end-user, we implemented a more robust and standard metric to compare the performance of each methodology and pick the one that gives the least error. Our choice of metric is RMSE which gets calculated for each of the recommendation methodologies we have developed.

We tried with the best possible way to optimize the computation and make the user-facing interface as simple as possible. To exploit the power of distributed processing we are making use of Spark ALS for which we had to deal with problems setting up cluster on local development environment. We used these findings when running the application on an actual Spark cluster.

VI. Evaluation

We used the Movie Lens small dataset to do a head-to-head comparison with an aim to identify which recommendation technique works the best for this dataset. We experimented with 3 different algorithms (KNN, SVD, ALS). We got the best results from KNN which has the least RMSE value of 0.4893. Thus for this dataset the machine learning model beat the mathematical modeling algorithms of SVD and ALS. Since the Movie Lens large dataset was run only using Spark ALS technique we don’t have the comparison result for that dataset.

# VII Conclusion

While several recommendation systems have been built and are finding application in a variety of industries, in our research we did not find much being done in the way of building a recommender for recommendation engines! This paper was our first attempt at tackling this problem. While we were able to test our recommenders with a few datasets only, we can easily extend the analysis by processing more publicly available datasets and compare the performance of various recommender approaches. The real value, however, of our work is for users to try it out for data in their specific domains and automatically be recommended the recommender algorithm that performs the best for their domain.

# VIII Related Works

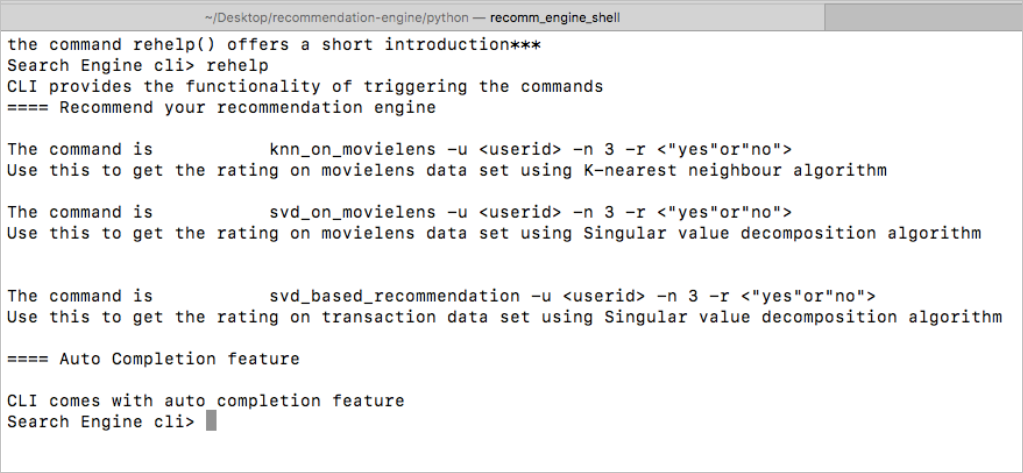
During the subject research we found Lab41 [4] is working on a similar initiative - “how to recommend data, tools, and programs to a software developer or data scientist. In large organizations, it can be difficult to know the variety of data sources and tools that have been curated. For an analyst starting a new project how does one choose among all the choices?” After doing the research they discovered that building a recommender system really “depends on the type of application you are analyzing and it can be difficult to determine which algorithm to try first for your dataset”. As part of the initiative the Circulo project created a community detection evaluation framework. The resulting quantitative measures can be used to drive experiments such as measuring algorithm efficacy against specific dataset types or comparing different algorithm execution results against the same dataset. The work is still in progress to find datasets suitable to evaluate recommender system algorithms.

##### References

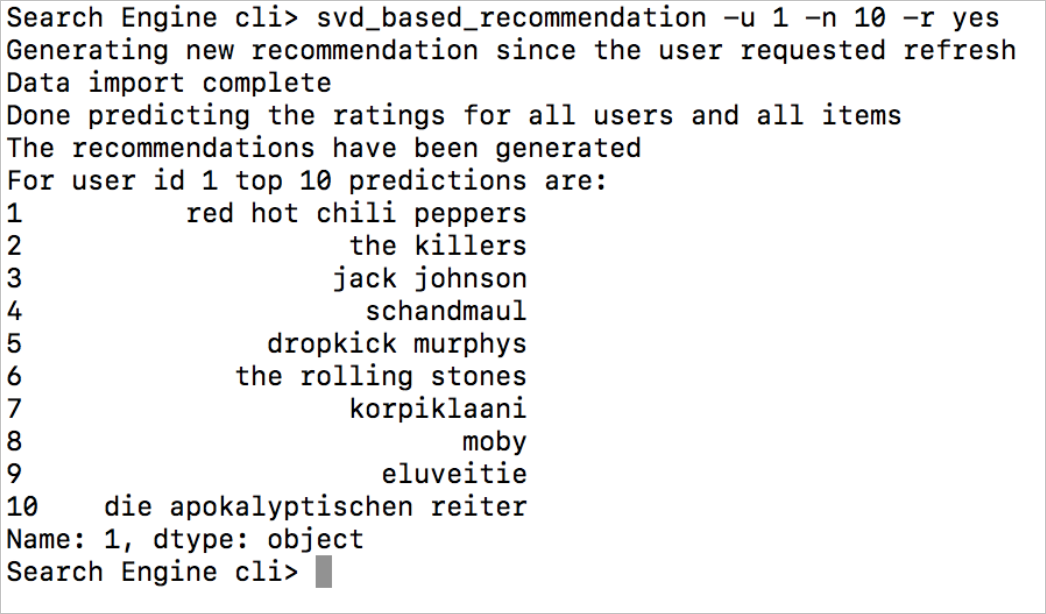
1. University of Minnesota, “Recommender Systems Specialization”, *https://www.coursera.org/specializations/recommender-systems*
2. Lab41, “9 Must-Have Datasets for Investigating Recommender Systems”, *https://www.kdnuggets.com/2016/02/nine-datasets-investigating-recommender-systems.html*
3. “Movie Lens dataset”,<https://grouplens.org/datasets/movielens/>
4. https://gab41.lab41.org/recommending-recommendation-systems-cc39ace3f5c1

**Results and Screenshots**

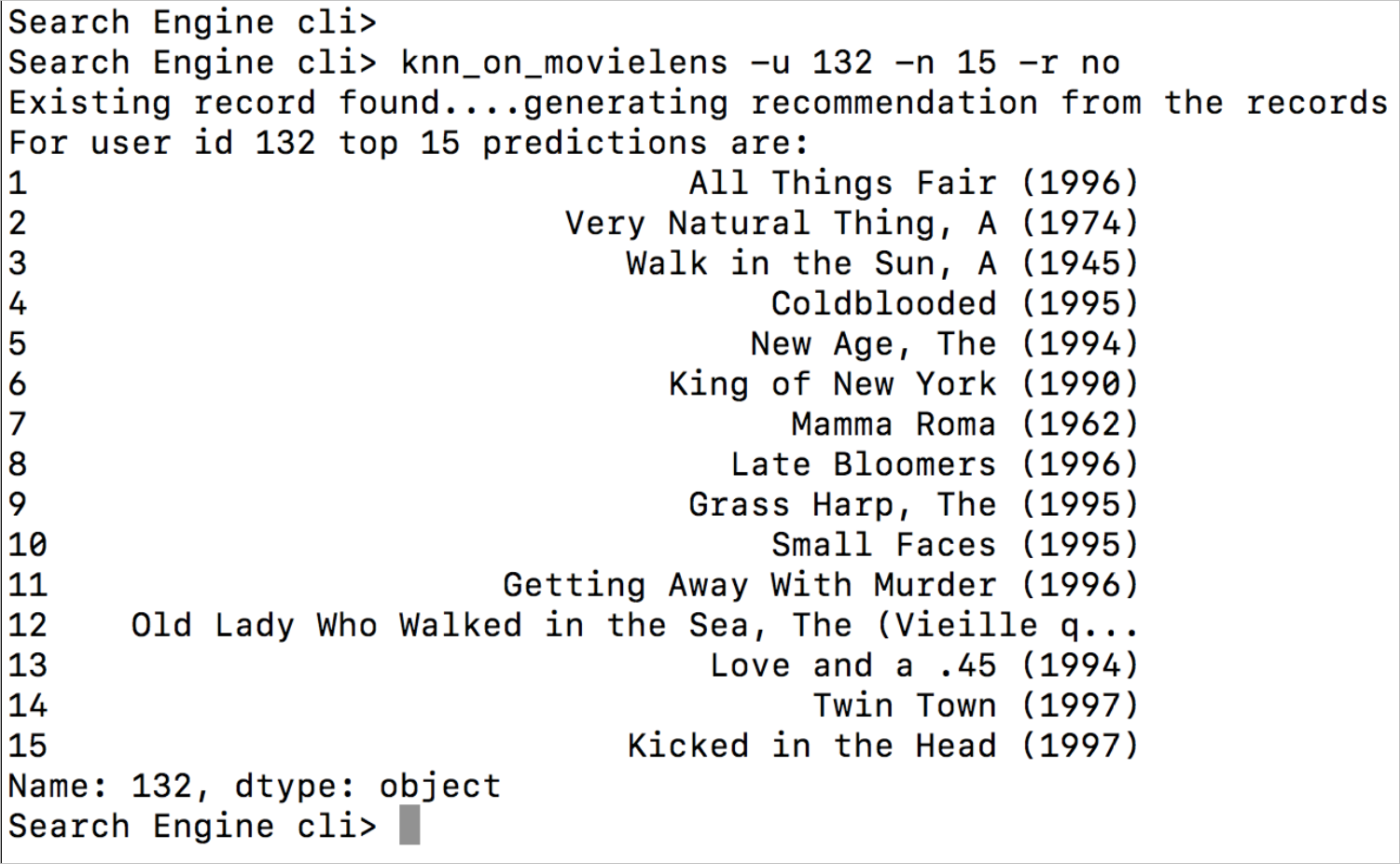
**Figure 1: Command-line interface of application**



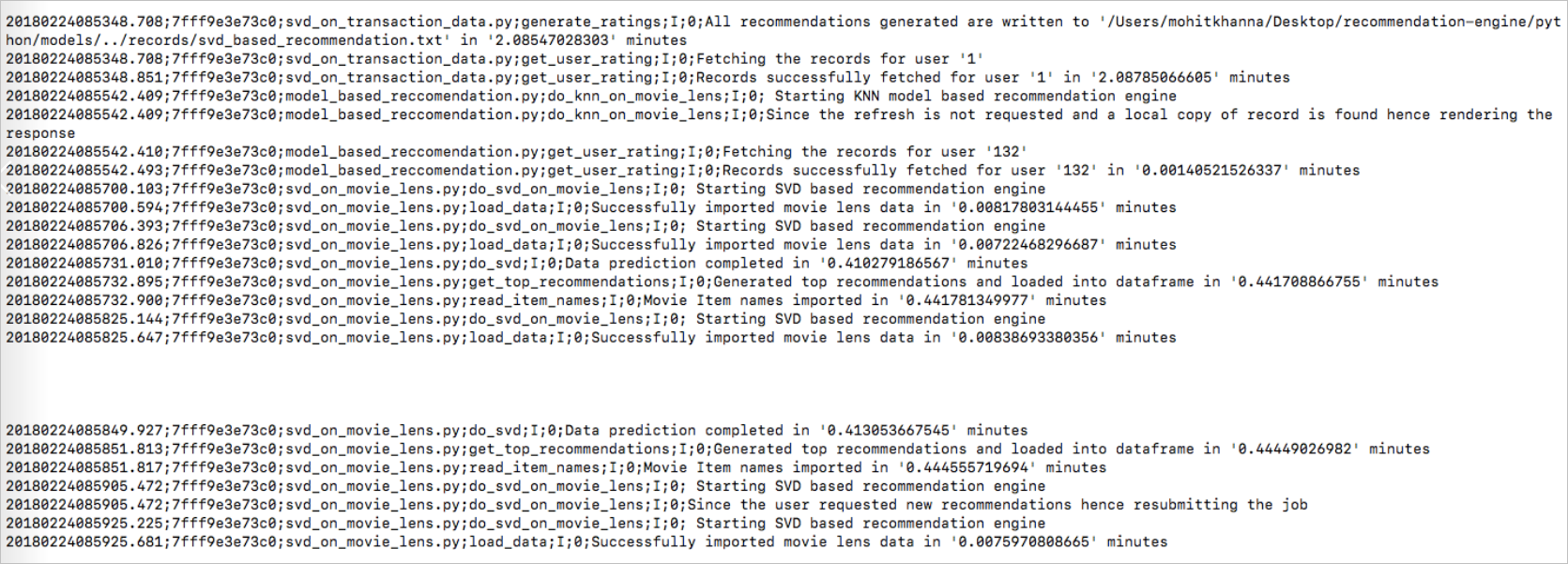
**Figure 2: SVD-based top 10 recommendation for user ID 1 computed afresh**



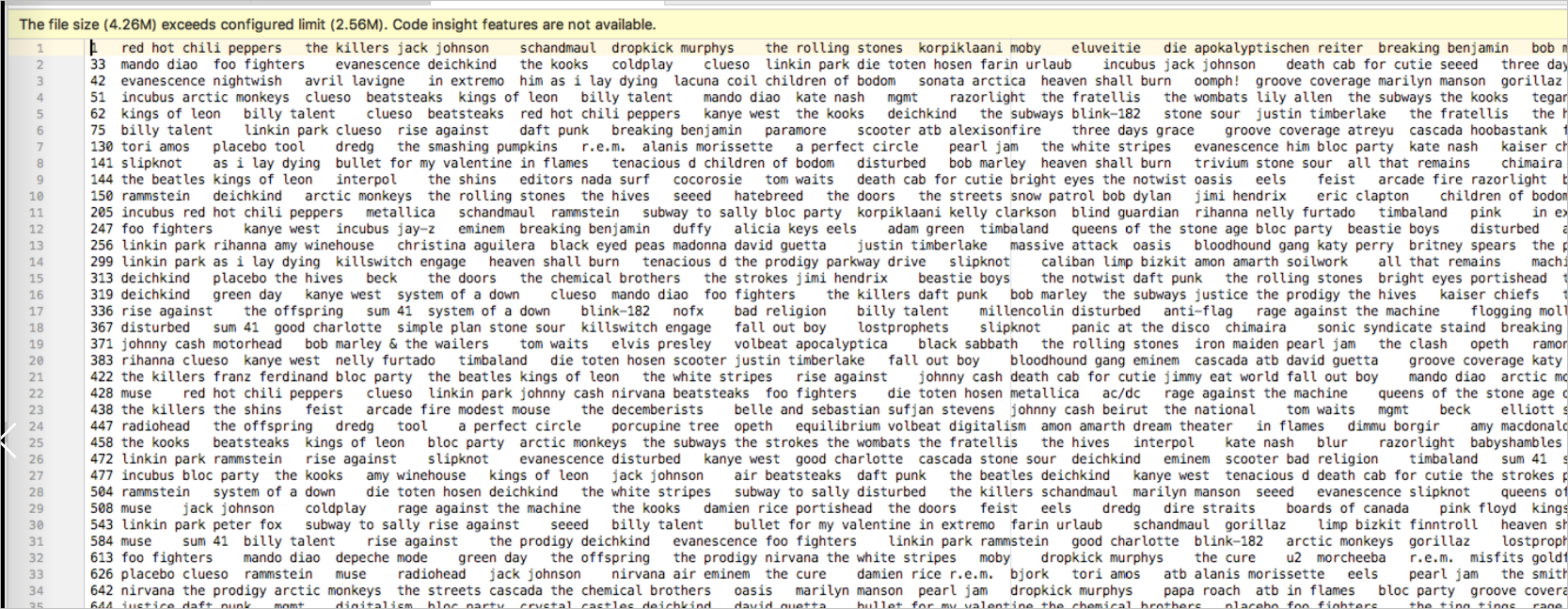
**Figure 3: KNN-based top 15 recommendations for user ID 132**



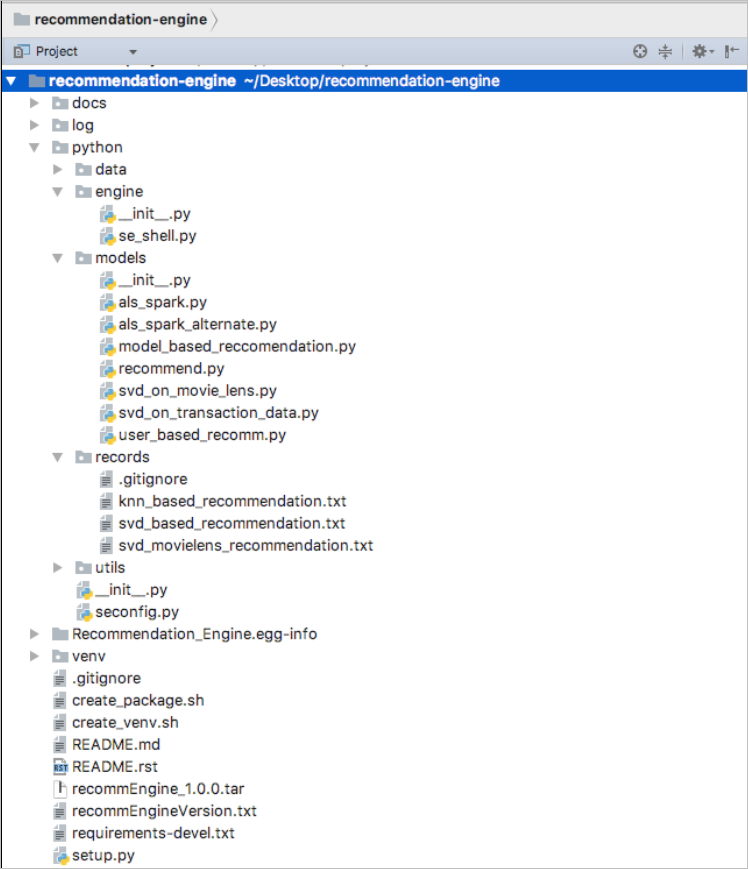
**Figure 4: Application logs**



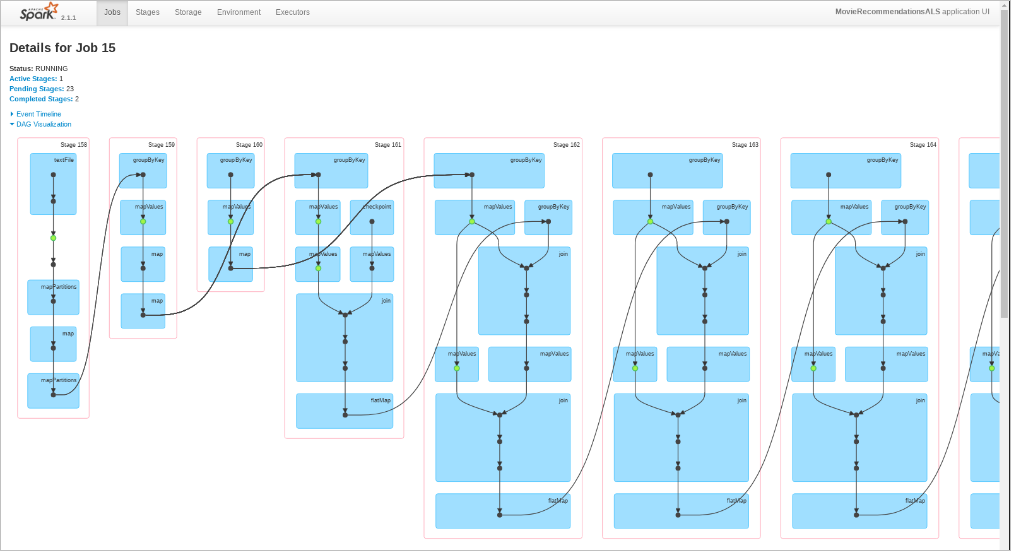
**Figure 5: Recommendations output created for all users and stored by application**



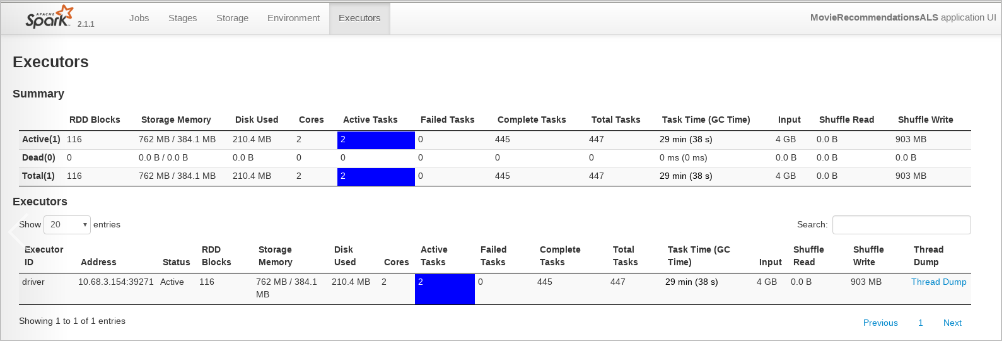
**Figure 6: Project structure of Python code**



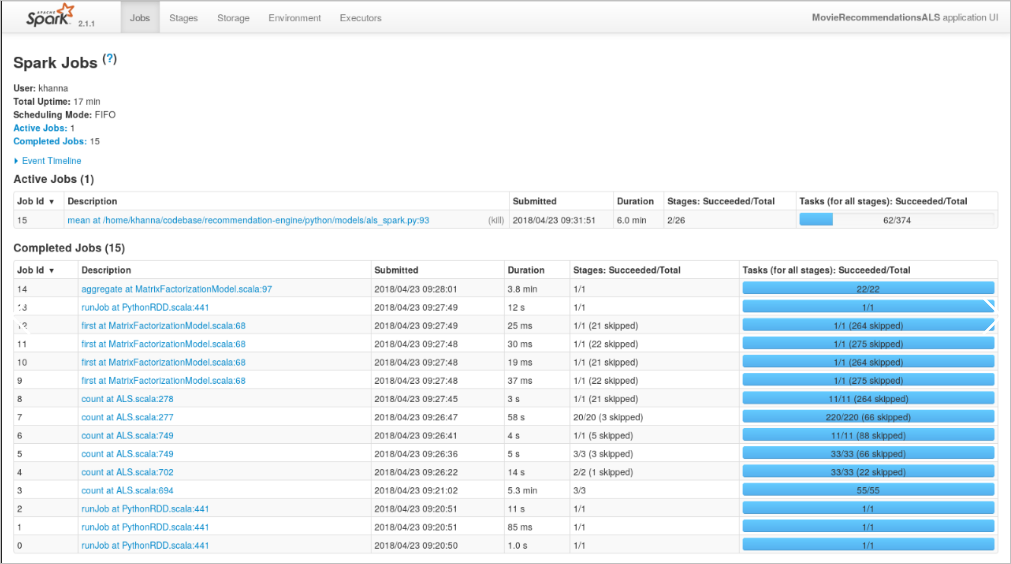
**Figure 7: DAG for Spark computation**



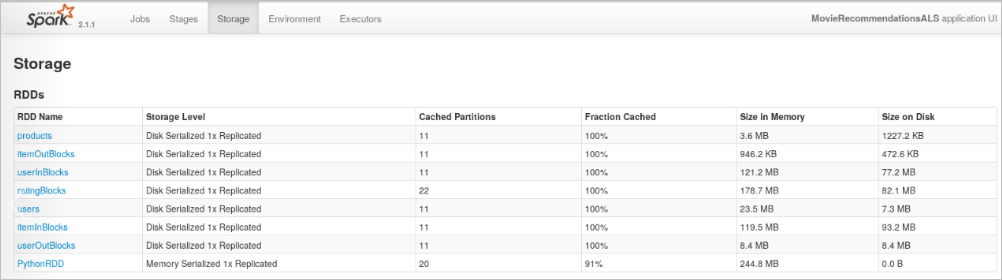
**Figure 8: Spark Executors**



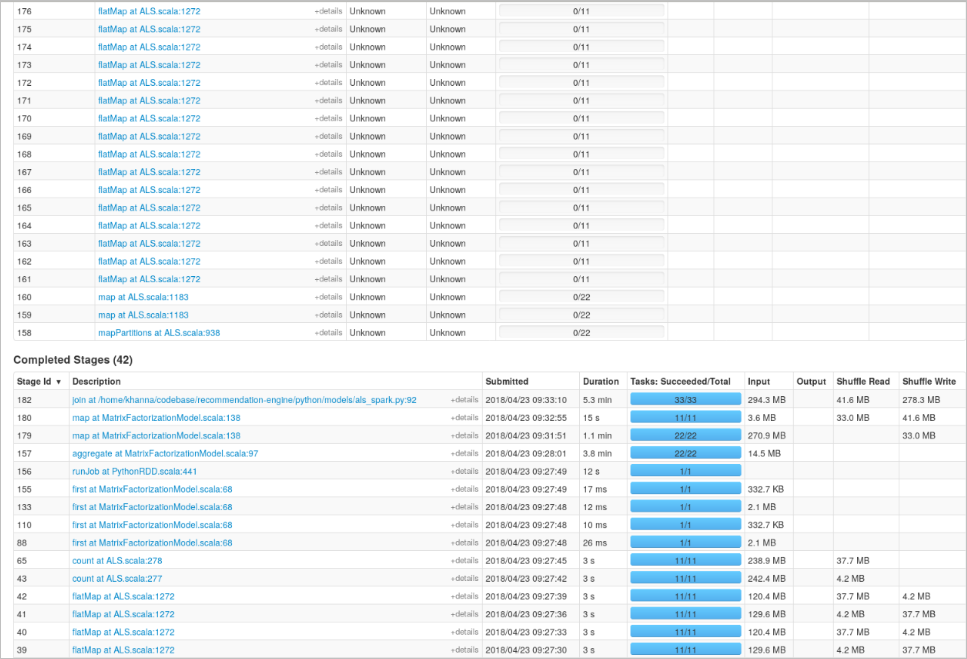
**Figure 9: Spark Jobs**



**Figure 10: Spark RDDs**

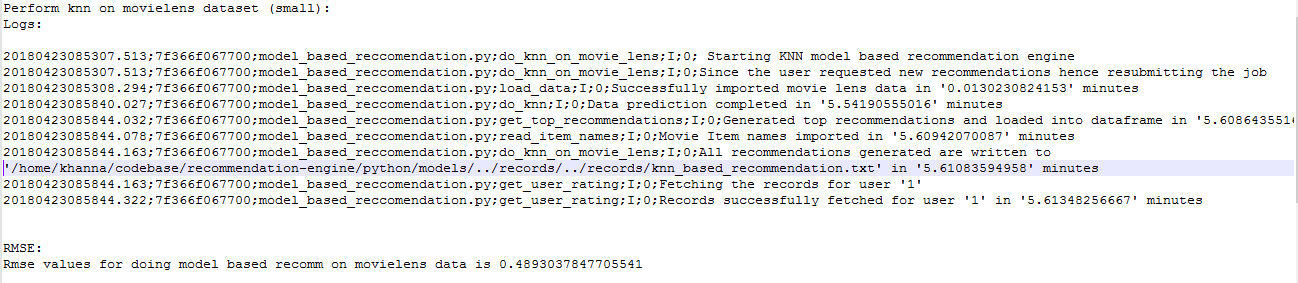


**Figure 11: Spark Stages**

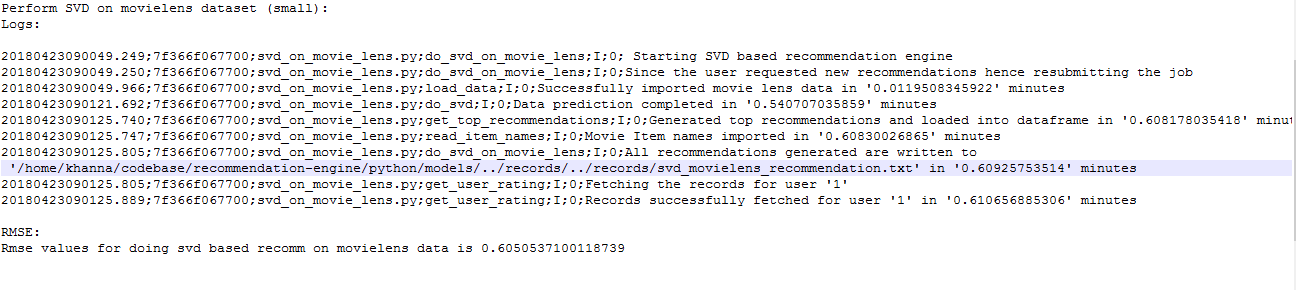


**Performance Results**

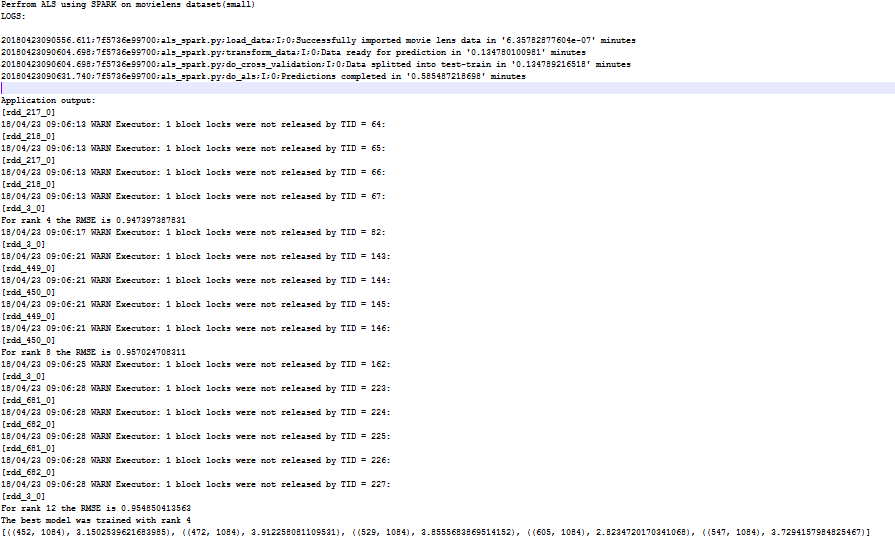
1. Running Knn algorithm on movie lens small dataset



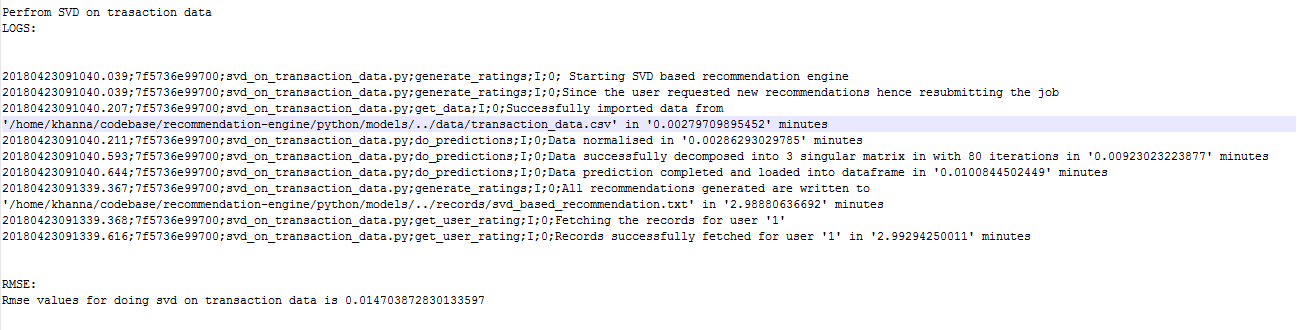
1. Running SVD algorithm on movie lens small dataset



1. Running Spark ALS algorithm on movie lens small dataset



1. Running SVD algorithm on transaction dataset



1. Running Spark ALS algorithm on movie lens large Dataset

