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.*

I. 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.

II. 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.

III. 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.

IV. 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.

V. 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

```
~/Desktop/recommendation-engine/python — recomm_engine_shell
the command rehelp() offers a short introduction***
Search Engine cli> rehelp
CLI provides the functionality of triggering the commands
==== Recommend your recommendation engine

The command is          knn_on_movielens -u <userid> -n 3 -r <"yes"or"no">
Use this to get the rating on movielens data set using K-nearest neighbour algorithm

The command is          svd_on_movielens -u <userid> -n 3 -r <"yes"or"no">
Use this to get the rating on movielens data set using Singular value decomposition algorithm


The command is          svd_based_recommendation -u <userid> -n 3 -r <"yes"or"no">
Use this to get the rating on transaction data set using Singular value decomposition algorithm

==== Auto Completion feature

CLI comes with auto completion feature
Search Engine cli>
```

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

```
Search Engine cli> svd_based_recommendation -u 1 -n 10 -r yes
Generating new recommendation since the user requested refresh
Data import complete
Done predicting the ratings for all users and all items
The recommendations have been generated
For user id 1 top 10 predictions are:
1          red hot chili peppers
2                    the killers
3                   jack johnson
4                     schandmaul
5                dropkick murphys
6              the rolling stones
7                     korpiklaani
8                            moby
9                       eluveitie
10    die apokalyptischen reiter
Name: 1, dtype: object
Search Engine cli>
```

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

```
Search Engine cli>
Search Engine cli> knn_on_movielens -u 132 -n 15 -r no
Existing record found....generating recommendation from the records
For user id 132 top 15 predictions are:
1                                All Things Fair (1996)
2                          Very Natural Thing, A (1974)
3                             Walk in the Sun, A (1945)
4                                    Coldblooded (1995)
5                                  New Age, The (1994)
6                              King of New York (1990)
7                                    Mamma Roma (1962)
8                                 Late Bloomers (1996)
9                               Grass Harp, The (1995)
10                                   Small Faces (1995)
11                       Getting Away With Murder (1996)
12     Old Lady Who Walked in the Sea, The (Vieille q...
13                               Love and a .45 (1994)
14                                     Twin Town (1997)
15                             Kicked in the Head (1997)
Name: 132, dtype: object
Search Engine cli>
```

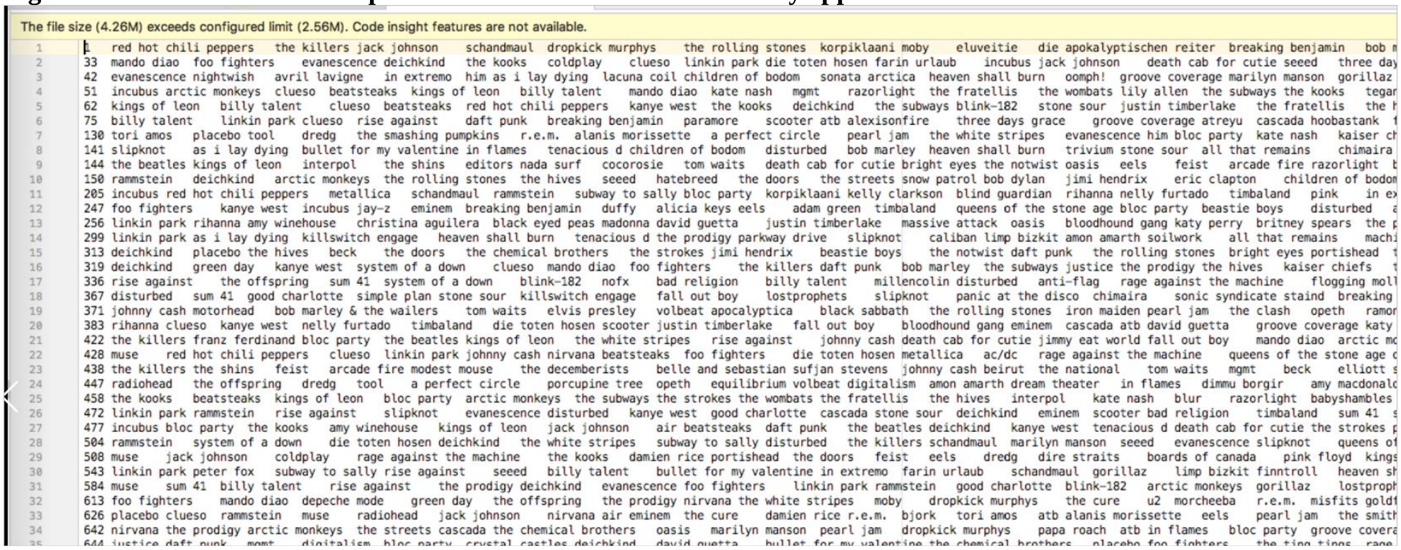
Figure 4: Application logs

```
20180224085348.708;7fff9e3e73c0;svd_on_transaction_data.py;generate_ratings;I;0;All recommendations generated are written to '/Users/mohitkhanna/Desktop/recommendation-engine/pyt
hon/models/../records/svd_based_recommendation.txt' in '2.08547028303' minutes
20180224085348.708;7fff9e3e73c0;svd_on_transaction_data.py;get_user_rating;I;0;Fetching the records for user '1'
20180224085348.851;7fff9e3e73c0;svd_on_transaction_data.py;get_user_rating;I;0;Records successfully fetched for user '1' in '2.08785066605' minutes
20180224085542.409;7fff9e3e73c0;model_based_reccomendation.py;do_knn_on_movie_lens;I;0; Starting KNN model based recommendation engine
20180224085542.409;7fff9e3e73c0;model_based_reccomendation.py;do_knn_on_movie_lens;I;0;Since the refresh is not requested and a local copy of record is found hence rendering the
response
20180224085542.410;7fff9e3e73c0;model_based_reccomendation.py;get_user_rating;I;0;Fetching the records for user '132'
20180224085542.493;7fff9e3e73c0;model_based_reccomendation.py;get_user_rating;I;0;Records successfully fetched for user '132' in '0.00140521526337' minutes
20180224085700.103;7fff9e3e73c0;svd_on_movie_lens.py;do_svd_on_movie_lens;I;0; Starting SVD based recommendation engine
20180224085700.594;7fff9e3e73c0;svd_on_movie_lens.py;load_data;I;0;Successfully imported movie lens data in '0.00817803144455' minutes
20180224085706.393;7fff9e3e73c0;svd_on_movie_lens.py;do_svd_on_movie_lens;I;0; Starting SVD based recommendation engine
20180224085706.826;7fff9e3e73c0;svd_on_movie_lens.py;load_data;I;0;Successfully imported movie lens data in '0.00722468296687' minutes
20180224085731.010;7fff9e3e73c0;svd_on_movie_lens.py;do_svd;I;0;Data prediction completed in '0.410279186567' minutes
20180224085732.895;7fff9e3e73c0;svd_on_movie_lens.py;get_top_recommendations;I;0;Generated top recommendations and loaded into dataframe in '0.441708866755' minutes
20180224085732.900;7fff9e3e73c0;svd_on_movie_lens.py;read_item_names;I;0;Movie Item names imported in '0.441781349977' minutes
20180224085825.144;7fff9e3e73c0;svd_on_movie_lens.py;do_svd_on_movie_lens;I;0; Starting SVD based recommendation engine
20180224085825.647;7fff9e3e73c0;svd_on_movie_lens.py;load_data;I;0;Successfully imported movie lens data in '0.00838693380356' minutes



20180224085849.927;7fff9e3e73c0;svd_on_movie_lens.py;do_svd;I;0;Data prediction completed in '0.413053667545' minutes
20180224085851.813;7fff9e3e73c0;svd_on_movie_lens.py;get_top_recommendations;I;0;Generated top recommendations and loaded into dataframe in '0.44449026982' minutes
20180224085851.817;7fff9e3e73c0;svd_on_movie_lens.py;read_item_names;I;0;Movie Item names imported in '0.444555719694' minutes
20180224085905.472;7fff9e3e73c0;svd_on_movie_lens.py;do_svd_on_movie_lens;I;0; Starting SVD based recommendation engine
20180224085905.472;7fff9e3e73c0;svd_on_movie_lens.py;do_svd_on_movie_lens;I;0;Since the user requested new recommendations hence resubmitting the job
20180224085925.225;7fff9e3e73c0;svd_on_movie_lens.py;do_svd_on_movie_lens;I;0; Starting SVD based recommendation engine
20180224085925.681;7fff9e3e73c0;svd_on_movie_lens.py;load_data;I;0;Successfully imported movie lens data in '0.0075970808665' minutes
```

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



The file size (4.26M) exceeds configured limit (2.56M). Code insight features are not available.

```
1   red hot chili peppers   the killers jack johnson    schandmaul  dropkick murphys    the rolling stones  korpiklaani moby    eluveitie   die apokalyptischen reiter  breaking benjamin   bob m
33  mando diao  foo fighters    evanescence deichkind   the kooks   coldplay    clueso  linkin park die toten hosen farin urlaub    incubus jack johnson    death cab for cutie seeed   three day
42  evanescence nightwish   avril lavigne   in extremo  him as i lay dying  lacuna coil children of bodom   sonata arctica  heaven shall burn   oomph!  groove coverage marilyn manson  gorillaz
51  incubus arctic monkeys  clueso  beatsteaks  kings of leon   billy talent    mando diao  kate nash   mgmt    razorlight  the fratellis   the wombats lily allen  the subways the kooks   tegar
62  kings of leon   billy talent    clueso  beatsteaks  red hot chili peppers   kanye west  the kooks   deichkind   the subways blink-182   stone sour  justin timberlake   the fratellis   the h
75  billy talent    linkin park clueso  rise against    daft punk   breaking benjamin   paramore    scooter atb alexisonfire    three days grace    groove coverage atreyu  cascada hoobastank  f
130 tori amos   placebo tool    dredg   the smashing pumpkins   r.e.m.  alanis morissette   a perfect circle    pearl jam   the white stripes   evanescence him bloc party  kate nash   kaiser ch
141 slipknot    as i lay dying  bullet for my valentine in flames   tenacious d children of bodom   disturbed   bob marley  heaven shall burn   trivium stone sour  all that remains    chimaira
144 the beatles kings of leon   interpol    the shins   editors nada surf   cocorosie   tom waits   death cab for cutie bright eyes the notwist oasis   eels    feist   arcade fire razorlight  b
150 rammstein   deichkind   arctic monkeys  the rolling stones  the hives   seeed   hatebreed   the doors   the streets snow patrol bob dylan   jimi hendrix   eric clapton    children of bodom
205 incubus red hot chili peppers   metallica   schandmaul  rammstein   subway to sally bloc party  korpiklaani kelly clarkson  blind guardian  rihanna nelly furtado   timbaland   pink    in ex
247 foo fighters    kanye west  incubus jay-z   eminem  breaking benjamin   duffy   alicia keys eels    adam green  timbaland   queens of the stone age bloc party  beastie boys    disturbed   a
256 linkin park rihanna amy winehouse   christina aguilera  black eyed peas madonna david guetta    justin timberlake   massive attack  oasis   bloodhound gang katy perry  britney spears  the p
299 linkin park as i lay dying  killswitch engage   heaven shall burn   tenacious d the prodigy parkway drive   slipknot    caliban limp bizkit amon amarth soilwork    all that remains    machi
313 deichkind   placebo the hives   beck    the doors   the chemical brothers   the strokes jimi hendrix    beastie boys    the notwist daft punk   the rolling stones  bright eyes portishead  t
319 deichkind   green day   kanye west  system of a down    clueso  mando diao  foo fighters    the killers daft punk   bob marley  the subways justice the prodigy the hives   kaiser chiefs   t
336 rise against    the offspring   sum 41  system of a down    blink-182   nofx    bad religion    billy talent    millencolin disturbed   anti-flag   rage against the machine    flogging moll
367 disturbed   sum 41  good charlotte  simple plan stone sour  killswitch engage   fall out boy    lostprophets    slipknot    panic at the disco  chimaira    sonic syndicate staind  breaking
371 johnny cash motorhead   bob marley & the wailers    tom waits   elvis presley   volbeat apocalyptica    black sabbath   the rolling stones  iron maiden pearl jam   the clash   opeth   ramon
383 rihanna clueso  kanye west  nelly furtado   timbaland   die toten hosen scooter justin timberlake   fall out boy    bloodhound gang eminem  cascada atb david guetta    groove coverage katy
422 the killers franz ferdinand bloc party  the beatles kings of leon   the white stripes   rise against    johnny cash death cab for cutie jimmy eat world fall out boy    mando diao  arctic mo
428 muse    red hot chili peppers   clueso  linkin park johnny cash nirvana beatsteaks  foo fighters    die toten hosen metallica   ac/dc   rage against the machine    queens of the stone age c
438 the killers the shins   feist   arcade fire modest mouse    the decemberists    belle and sebastian sufjan stevens  johnny cash beirut  the national    tom waits   mgmt    beck    elliott s
447 radiohead   the offspring   dredg   tool    a perfect circle    porcupine tree  opeth   equilibrium volbeat digitalism  amon amarth dream theater   in flames   dimmu borgir    amy macdonald
458 the kooks   beatsteaks  kings of leon   bloc party  arctic monkeys  the subways the strokes the wombats the fratellis   the hives   interpol    kate nash   blur    razorlight  babyshambles
472 linkin park rammstein   rise against    slipknot    evanescence disturbed   kanye west  good charlotte  cascada stone sour  deichkind   eminem  scooter bad religion    timbaland   sum 41  s
477 incubus bloc party  the kooks   amy winehouse   kings of leon   jack johnson    air beatsteaks  daft punk   the beatles deichkind   kanye west  tenacious d death cab for cutie the strokes p
504 rammstein   system of a down    die toten hosen deichkind   the white stripes   subway to sally disturbed   the killers schandmaul  marilyn manson  seeed   evanescence slipknot    queens of
508 muse    jack johnson    coldplay    rage against the machine    the kooks   damien rice portishead  the doors   feist   eels    dredg   dire straits    boards of canada    pink floyd  kings
543 linkin park peter fox   subway to sally rise against    seeed   billy talent    bullet for my valentine in extremo  farin urlaub    schandmaul  gorillaz    limp bizkit finntroll   heaven sh
584 muse    sum 41  billy talent    rise against    the prodigy deichkind   evanescence foo fighters    linkin park rammstein   good charlotte  blink-182   arctic monkeys  gorillaz    lostproph
613 foo fighters    mando diao  depeche mode    green day   the offspring   the prodigy nirvana the white stripes   moby    dropkick murphys    the cure    u2  morcheeba   r.e.m.  misfits goldf
626 placebo clueso  rammstein   muse    radiohead   jack johnson    nirvana air eminem  the cure    damien rice r.e.m.  bjork   tori amos   atb alanis morissette   eels    pearl jam   the smith
642 nirvana the prodigy arctic monkeys  the streets cascada the chemical brothers   oasis   marilyn manson  pearl jam   dropkick murphys    papa roach  atb in flames   bloc party  groove covera
```

Figure 6: Project structure of Python code

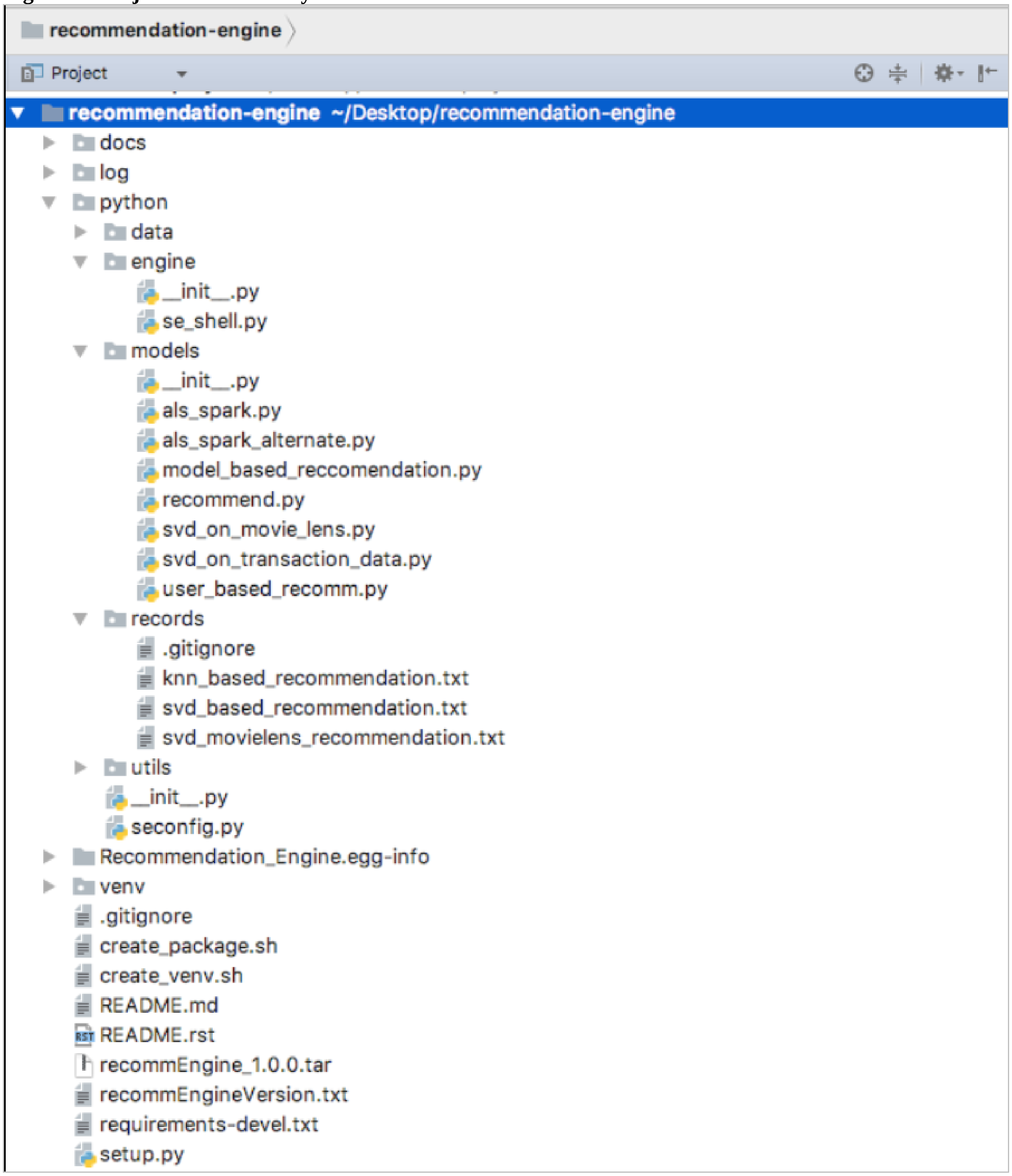


Figure 7: DAG for Spark computation

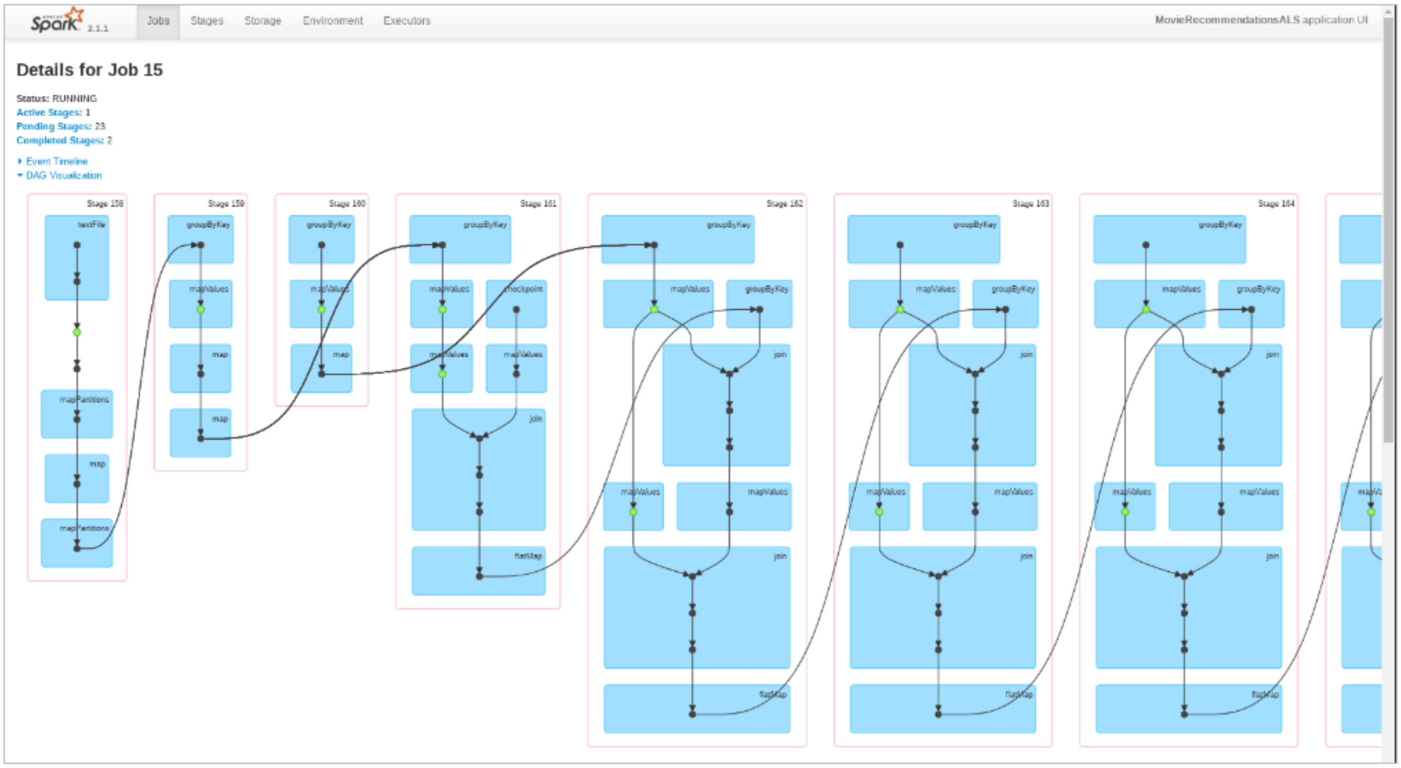


Figure 8: Spark Executors

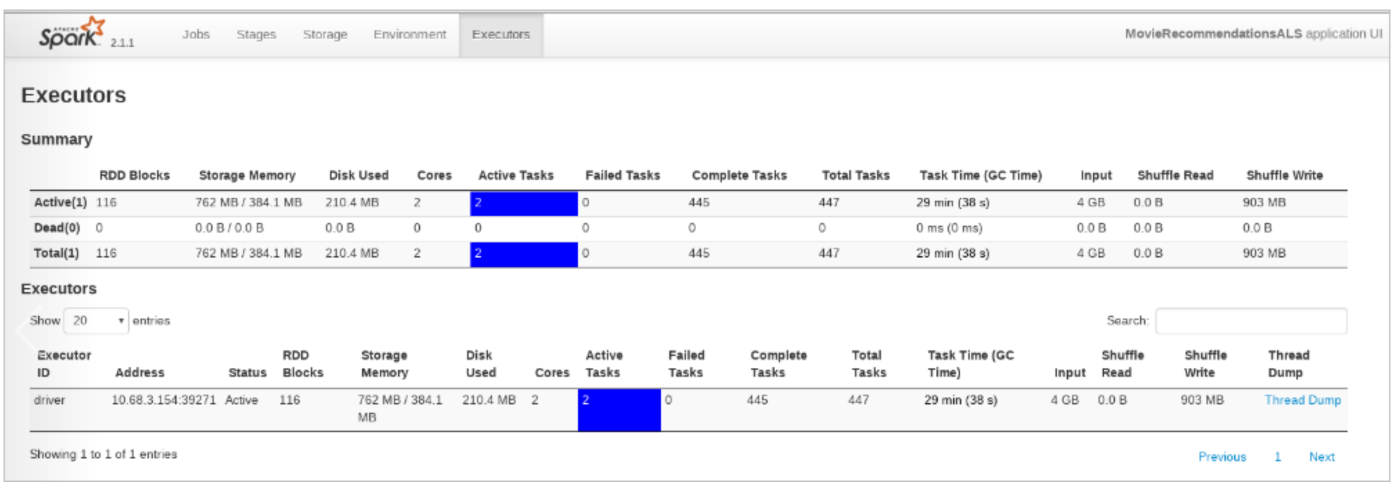
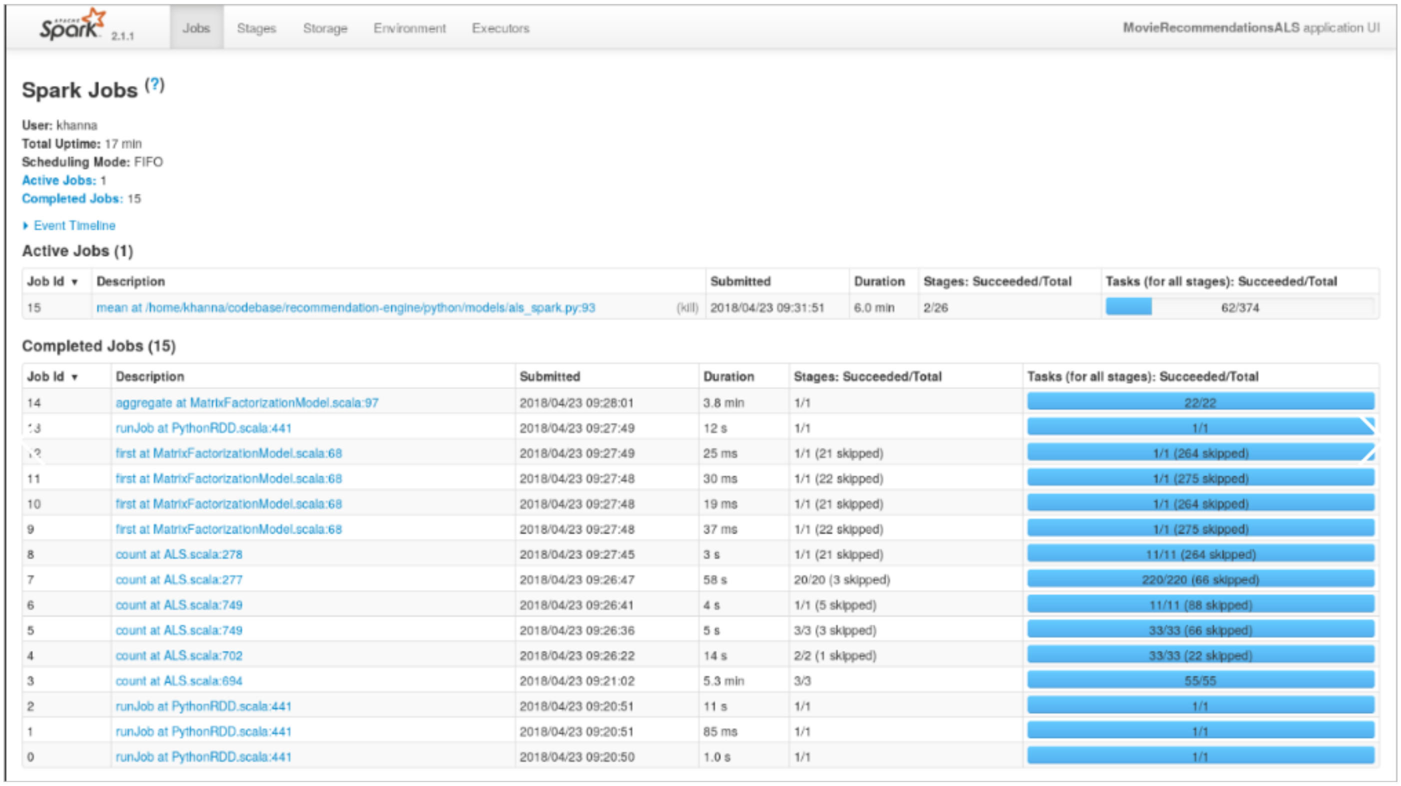


Figure 9: Spark Jobs


Spark 2.1.1 | Jobs | Stages | Storage | Environment | Executors | MovieRecommendationsALS application UI

Spark Jobs (?)

User: khanna
Total Uptime: 17 min
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 15

▸ Event Timeline

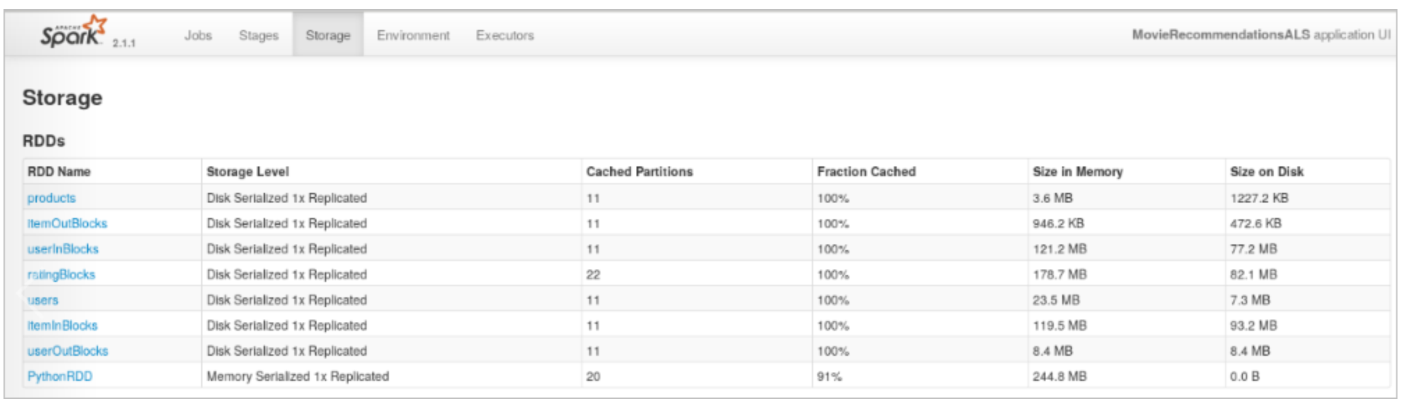
Active Jobs (1)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
15	mean at /home/khanna/codebase/recommendation-engine/python/models/als_spark.py:93 (kill)	2018/04/23 09:31:51	6.0 min	2/26	62/374

Completed Jobs (15)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
14	aggregate at MatrixFactorizationModel.scala:97	2018/04/23 09:28:01	3.8 min	1/1	22/22
13	runJob at PythonRDD.scala:441	2018/04/23 09:27:49	12 s	1/1	1/1
12	first at MatrixFactorizationModel.scala:68	2018/04/23 09:27:49	25 ms	1/1 (21 skipped)	1/1 (264 skipped)
11	first at MatrixFactorizationModel.scala:68	2018/04/23 09:27:48	30 ms	1/1 (22 skipped)	1/1 (275 skipped)
10	first at MatrixFactorizationModel.scala:68	2018/04/23 09:27:48	19 ms	1/1 (21 skipped)	1/1 (264 skipped)
9	first at MatrixFactorizationModel.scala:68	2018/04/23 09:27:48	37 ms	1/1 (22 skipped)	1/1 (275 skipped)
8	count at ALS.scala:278	2018/04/23 09:27:45	3 s	1/1 (21 skipped)	11/11 (264 skipped)
7	count at ALS.scala:277	2018/04/23 09:26:47	58 s	20/20 (3 skipped)	220/220 (66 skipped)
6	count at ALS.scala:749	2018/04/23 09:26:41	4 s	1/1 (5 skipped)	11/11 (88 skipped)
5	count at ALS.scala:749	2018/04/23 09:26:36	5 s	3/3 (3 skipped)	33/33 (66 skipped)
4	count at ALS.scala:702	2018/04/23 09:26:22	14 s	2/2 (1 skipped)	33/33 (22 skipped)
3	count at ALS.scala:694	2018/04/23 09:21:02	5.3 min	3/3	55/55
2	runJob at PythonRDD.scala:441	2018/04/23 09:20:51	11 s	1/1	1/1
1	runJob at PythonRDD.scala:441	2018/04/23 09:20:51	85 ms	1/1	1/1
0	runJob at PythonRDD.scala:441	2018/04/23 09:20:50	1.0 s	1/1	1/1

Figure 10: Spark RDDs


Spark 2.1.1 | Jobs | Stages | Storage | Environment | Executors | MovieRecommendationsALS application UI

Storage

RDDs

RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
products	Disk Serialized 1x Replicated	11	100%	3.6 MB	1227.2 KB
itemOutBlocks	Disk Serialized 1x Replicated	11	100%	946.2 KB	472.6 KB
userInBlocks	Disk Serialized 1x Replicated	11	100%	121.2 MB	77.2 MB
ratingBlocks	Disk Serialized 1x Replicated	22	100%	178.7 MB	82.1 MB
users	Disk Serialized 1x Replicated	11	100%	23.5 MB	7.3 MB
itemInBlocks	Disk Serialized 1x Replicated	11	100%	119.5 MB	93.2 MB
userOutBlocks	Disk Serialized 1x Replicated	11	100%	8.4 MB	8.4 MB
PythonRDD	Memory Serialized 1x Replicated	20	91%	244.8 MB	0.0 B

Figure 11: Spark Stages

176	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
175	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
174	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
173	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
172	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
171	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
170	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
169	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
168	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
167	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
166	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
165	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
164	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
163	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
162	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
161	flatMap at ALS.scala:1272 +details	Unknown	Unknown	0/11				
160	map at ALS.scala:1183 +details	Unknown	Unknown	0/22				
159	map at ALS.scala:1183 +details	Unknown	Unknown	0/22				
158	mapPartitions at ALS.scala:938 +details	Unknown	Unknown	0/22				

Completed Stages (42)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
182	join at /home/khanna/codebase/recommendation-engine/python/models/als_spark.py:92 +details	2018/04/23 09:33:10	5.3 min	33/33	294.3 MB		41.6 MB	278.3 MB
180	map at MatrixFactorizationModel.scala:138 +details	2018/04/23 09:32:55	15 s	11/11	3.6 MB		33.0 MB	41.6 MB
179	map at MatrixFactorizationModel.scala:138 +details	2018/04/23 09:31:51	1.1 min	22/22	270.9 MB			33.0 MB
157	aggregate at MatrixFactorizationModel.scala:97 +details	2018/04/23 09:28:01	3.8 min	22/22	14.5 MB			
156	runJob at PythonRDD.scala:441 +details	2018/04/23 09:27:49	12 s	1/1				
155	first at MatrixFactorizationModel.scala:68 +details	2018/04/23 09:27:49	17 ms	1/1	332.7 KB			
133	first at MatrixFactorizationModel.scala:68 +details	2018/04/23 09:27:48	12 ms	1/1	2.1 MB			
110	first at MatrixFactorizationModel.scala:68 +details	2018/04/23 09:27:48	10 ms	1/1	332.7 KB			
88	first at MatrixFactorizationModel.scala:68 +details	2018/04/23 09:27:48	26 ms	1/1	2.1 MB			
65	count at ALS.scala:278 +details	2018/04/23 09:27:45	3 s	11/11	238.9 MB		37.7 MB	
43	count at ALS.scala:277 +details	2018/04/23 09:27:42	3 s	11/11	242.4 MB		4.2 MB	
42	flatMap at ALS.scala:1272 +details	2018/04/23 09:27:39	3 s	11/11	120.4 MB		37.7 MB	4.2 MB
41	flatMap at ALS.scala:1272 +details	2018/04/23 09:27:36	3 s	11/11	129.6 MB		4.2 MB	37.7 MB
40	flatMap at ALS.scala:1272 +details	2018/04/23 09:27:33	3 s	11/11	120.4 MB		37.7 MB	4.2 MB
39	flatMap at ALS.scala:1272 +details	2018/04/23 09:27:30	3 s	11/11	129.6 MB		4.2 MB	37.7 MB

Performance Results

1.) Running Knn algorithm on movie lens small dataset

```
Perform knn on movielens dataset (small):
Logs:

20180423085307.513;7f366f067700;model_based_reccomendation.py;do_knn_on_movie_lens;I;0; Starting KNN model based recommendation engine
20180423085307.513;7f366f067700;model_based_reccomendation.py;do_knn_on_movie_lens;I;0;Since the user requested new recommendations hence resubmitting the job
20180423085308.294;7f366f067700;model_based_reccomendation.py;load_data;I;0;Successfully imported movie lens data in '0.0130230824153' minutes
20180423085840.027;7f366f067700;model_based_reccomendation.py;do_knn;I;0;Data prediction completed in '5.54190555016' minutes
20180423085844.032;7f366f067700;model_based_reccomendation.py;get_top_recommendations;I;0;Generated top recommendations and loaded into dataframe in '5.608643551
20180423085844.078;7f366f067700;model_based_reccomendation.py;read_item_names;I;0;Movie Item names imported in '5.60942070087' minutes
20180423085844.163;7f366f067700;model_based_reccomendation.py;do_knn_on_movie_lens;I;0;All recommendations generated are written to
'/home/khanna/codebase/recommendation-engine/python/models/../records/../records/knn_based_recommendation.txt' in '5.61083594958' minutes
20180423085844.163;7f366f067700;model_based_reccomendation.py;get_user_rating;I;0;Fetching the records for user '1'
20180423085844.322;7f366f067700;model_based_reccomendation.py;get_user_rating;I;0;Records successfully fetched for user '1' in '5.61348256667' minutes


RMSE:
Rmse values for doing model based recomm on movielens data is 0.4893037847705541
```

2.) Running SVD algorithm on movie lens small dataset

```
Perform SVD on movielens dataset (small):
Logs:

20180423090049.249;7f366f067700;svd_on_movie_lens.py;do_svd_on_movie_lens;I;0; Starting SVD based recommendation engine
20180423090049.250;7f366f067700;svd_on_movie_lens.py;do_svd_on_movie_lens;I;0;Since the user requested new recommendations hence resubmitting the job
20180423090049.966;7f366f067700;svd_on_movie_lens.py;load_data;I;0;Successfully imported movie lens data in '0.0119508345922' minutes
20180423090121.692;7f366f067700;svd_on_movie_lens.py;do_svd;I;0;Data prediction completed in '0.540707035859' minutes
20180423090125.740;7f366f067700;svd_on_movie_lens.py;get_top_recommendations;I;0;Generated top recommendations and loaded into dataframe in '0.608178035418' minut
20180423090125.747;7f366f067700;svd_on_movie_lens.py;read_item_names;I;0;Movie Item names imported in '0.60830026865' minutes
20180423090125.805;7f366f067700;svd_on_movie_lens.py;do_svd_on_movie_lens;I;0;All recommendations generated are written to
 '/home/khanna/codebase/recommendation-engine/python/models/../records/../records/svd_movielens_recommendation.txt' in '0.60925753514' minutes
20180423090125.805;7f366f067700;svd_on_movie_lens.py;get_user_rating;I;0;Fetching the records for user '1'
20180423090125.889;7f366f067700;svd_on_movie_lens.py;get_user_rating;I;0;Records successfully fetched for user '1' in '0.610656885306' minutes

RMSE:
Rmse values for doing svd based recomm on movielens data is 0.6050537100118739
```

3.) Running Spark ALS algorithm on movie lens small dataset

```
Perfrom ALS using SPARK on movielens dataset(small)
LOGS:

20180423090556.611;7f5736e99700;als_spark.py;load_data;I;0;Successfully imported movie lens data in '6.35782877604e-07' minutes
20180423090604.698;7f5736e99700;als_spark.py;transform_data;I;0;Data ready for prediction in '0.134780100981' minutes
20180423090604.698;7f5736e99700;als_spark.py;do_cross_validation;I;0;Data splitted into test-train in '0.134789216518' minutes
20180423090631.740;7f5736e99700;als_spark.py;do_als;I;0;Predictions completed in '0.585487218698' minutes

Application output:
[rdd_217_0]
18/04/23 09:06:13 WARN Executor: 1 block locks were not released by TID = 64:
[rdd_218_0]
18/04/23 09:06:13 WARN Executor: 1 block locks were not released by TID = 65:
[rdd_217_0]
18/04/23 09:06:13 WARN Executor: 1 block locks were not released by TID = 66:
[rdd_218_0]
18/04/23 09:06:13 WARN Executor: 1 block locks were not released by TID = 67:
[rdd_3_0]
For rank 4 the RMSE is 0.947397387831
18/04/23 09:06:17 WARN Executor: 1 block locks were not released by TID = 82:
[rdd_3_0]
18/04/23 09:06:21 WARN Executor: 1 block locks were not released by TID = 143:
[rdd_449_0]
18/04/23 09:06:21 WARN Executor: 1 block locks were not released by TID = 144:
[rdd_450_0]
18/04/23 09:06:21 WARN Executor: 1 block locks were not released by TID = 145:
[rdd_449_0]
18/04/23 09:06:21 WARN Executor: 1 block locks were not released by TID = 146:
[rdd_450_0]
For rank 8 the RMSE is 0.957024708311
18/04/23 09:06:25 WARN Executor: 1 block locks were not released by TID = 162:
[rdd_3_0]
18/04/23 09:06:28 WARN Executor: 1 block locks were not released by TID = 223:
[rdd_681_0]
18/04/23 09:06:28 WARN Executor: 1 block locks were not released by TID = 224:
[rdd_682_0]
18/04/23 09:06:28 WARN Executor: 1 block locks were not released by TID = 225:
[rdd_681_0]
18/04/23 09:06:28 WARN Executor: 1 block locks were not released by TID = 226:
[rdd_682_0]
18/04/23 09:06:28 WARN Executor: 1 block locks were not released by TID = 227:
[rdd_3_0]
For rank 12 the RMSE is 0.954850413563
The best model was trained with rank 4
[((452, 1084), 3.1502539621683985), ((472, 1084), 3.912258081109531), ((529, 1084), 3.8555683869514152), ((605, 1084), 2.8234720170341068), ((547, 1084), 3.7294157984825467)]
```

4.) Running SVD algorithm on transaction dataset

```
Perfrom SVD on trasaction data
LOGS:


20180423091040.039;7f5736e99700;svd_on_transaction_data.py;generate_ratings;I;0; Starting SVD based recommendation engine
20180423091040.039;7f5736e99700;svd_on_transaction_data.py;generate_ratings;I;0;Since the user requested new recommendations hence resubmitting the job
20180423091040.207;7f5736e99700;svd_on_transaction_data.py;get_data;I;0;Successfully imported data from
'/home/khanna/codebase/recommendation-engine/python/models/../data/transaction_data.csv' in '0.00279709895452' minutes
20180423091040.211;7f5736e99700;svd_on_transaction_data.py;do_predictions;I;0;Data normalised in '0.00286293029785' minutes
20180423091040.593;7f5736e99700;svd_on_transaction_data.py;do_predictions;I;0;Data successfully decomposed into 3 singular matrix in with 80 iterations in '0.00923023223877' minutes
20180423091040.644;7f5736e99700;svd_on_transaction_data.py;do_predictions;I;0;Data prediction completed and loaded into dataframe in '0.0100844502449' minutes
20180423091339.367;7f5736e99700;svd_on_transaction_data.py;generate_ratings;I;0;All recommendations generated are written to
'/home/khanna/codebase/recommendation-engine/python/models/../records/svd_based_recommendation.txt' in '2.98880636692' minutes
20180423091339.368;7f5736e99700;svd_on_transaction_data.py;get_user_rating;I;0;Fetching the records for user '1'
20180423091339.616;7f5736e99700;svd_on_transaction_data.py;get_user_rating;I;0;Records successfully fetched for user '1' in '2.99294250011' minutes


RMSE:
Rmse values for doing svd on transaction data is 0.014703872830133597
```

5.) Running Spark ALS algorithm on movie lens large Dataset

```
Perfrom ALS using SPARK on movielens dataset(large)
LOGS:

20180423479113.100;7f5736q886765;als_spark.py;load_data;I;0;Successfully imported movie lens data in '6.35782877604e-07' minutes
201804234979119.112;7f5736q886765;als_spark.py;transform_data;I;0;Data ready for prediction in '11.679356108528' minutes
201804234981165.233;7f5736q886765;als_spark.py;do_cross_validation;I;0;Data splitted into test-train in '6.612891431960' minutes
201804234983167.456;7f5736q886765;als_spark.py;do_als;I;0;Predictions completed in '3.671471010178' minutes

For rank 12 the RMSE is 0.786541907245
The best model was trained with rank 2
[((897, 1110), 3.127852234334), ((321, 1110), 3.765467891345), ((112, 1110), 2.854319098671), ((423, 1110), 2.456791056718), ((900, 1110), 3.431090187167)]
```