Collaborative Filtering Recommendation of Personalized Projects to Users in GitHub

Fan Chen

Abstract— **GitHub is one of the most commonly used web-based code repository hosting service. Being able to recommend appropriate projects to users in GitHub opens completely new opportunities for project management. In this work, I developed a repository recommender system using Apache Spark and its ALS algorithm. The data is collected from GitHub API and a series of data preprocessing and preselection make sure the model avoid the problem of cold start. RMSE is used as evaluation matrix and hyperparameter tuning is implemented to improve the performance. The results showed a decent root mean square error (0.3173) as an output of the recommender model.**

Keywords—GitHub, Recommendation system, Spark, ALS

I. INTRODUCTION

GitHub is one of the most commonly used web-based code repository hosting service, with around 9 million users and 17 million public repositories [1]. Majority of projects hosted on GitHub are really small but, on the other hand, developers spend most of their time working in medium to large repositories. Developers can freely join and leave projects following their current needs and interests. Based on real data collected from GitHub we have tried to predict which developer will be interested in which project.

It is even more obvious for open source projects, as majority of contributors work only part-time, and many of them do it in spare time, next to their professional duties. The interest to the projects varies from people to people. For example, some people feel the project is interested or useful and they will choose star a repository. Some users will subscribe a repository because they want to keep themselves posted about any updates and discussion of the repository. Some users will folk and contribute to a repository after they decide to be a part of the project. Some users are the owner of repositories and they are in charge of the whole project.

Being able to predict who will be interested in which project for such big communities like the GitHub opens completely new opportunities for project management. We could for example invite people even before they find project by themselves or predict which project will attract enough high-quality developers and will eventually turn into a mature product. Also, users can discover their potential interesting projects efficiently.

Nowadays, personalized recommendation is very popular in retails and services. For open source community, it is also interesting to recommend appropriate projects to the users to activate their interest and improve their experience in the community. With the increase of data, building a recommendation system in scale up environment is necessary.

The rest of the paper is organized as follows. Related work is covered in Section 2. Section 3 gives an overview of the methodology used in this study. Section 4 presents the results of machine learning frameworks as well as the interpretation based on the top-performing algorithm. Section 5 outlines conclusion.

II. RELATED WORK

In the existed research of GitHub, Valerio et al.[1] present a meta-analysis of 93 research papers which addresses three main dimensions of those papers: i) the empirical methods employed, ii) the datasets they used and iii) the limitations reported. Results of their meta-analysis show some concerns regarding the dataset collection process and size, the low level of replicability, poor sampling techniques, lack of longitudinal studies and scarce variety of methodologies. In Eirini's [2] paper, their studies described the quality and properties of the data available from GitHub. They document the results of an empirical study aimed at understanding the characteristics of the repositories in GitHub and how users take advantage of GitHub's main features— namely commits, pull requests, and issues. Their results indicate that mining GitHub for research purposes should take various potential perils into consideration. For example, the majority of the projects are personal and inactive; that GitHub is also being used for free storage and as a Web hosting service; and that almost 40% of all pull requests do not appear as merged, even though they were. I confirmed it by collecting the repo's data from GitHub API and decided to changed my topic from predict merging of pull request to recommendation system due to the availability of merged data.

From the perspective of analysis of users and repos in GitHub, Fragkiskos [3] statistically analyzed GitHub data, discretized values of features via k-means algorithm, which will be helpful in my feature engineering, and finally they applied apriori algorithm via weka in order to find out association rules. Having assumed that project success could be measured by the cardinality of downloads they kept only the rules which had as right par a download cardinality higher than a threshold of 1000 downloads. The results provide interesting insight in the GitHub ecosystem and showed the success rules for GitHub projects. From the paper, I can know which kind of feature I can generate for users and repositories to calculate the similarity of recommendation model.

Here is a research of recommendation engine based on spark. In Lin's et al. [4] study, they developed a product recommender system using Apache Spark and its ALS algorithm. They applied the Alternating Least Squares (ALS) matrix factorization method, which is an advanced version of collaborative filtering method. They utilized Amazon book reviews and product review metadata to train the recommendation model and used root mean squared error (RMSE) to evaluate the model performance. This is the paper I mainly focus on in my work. I am going to reproduce it and apply ALS method in my recommendation system.

In Xu's [5] work, in order to alleviate the sparsity problem of recommender systems meanwhile increase their accuracy and diversity in complex contexts, they propose a hybrid recommendation method based on social network using matrix factorization technique. In this method, they cluster users and consider a variety of complex factors such as user similarity, item similarity and user relationship, etc. The simulation results of two benchmark data sets (MovieLens and Book-Crossing) show that their method achieves superior performance to existing methods. This paper offer an advanced method to improve recommendation model by integrating different information together. I can deploy the methodology in my model when I have time to further improve the recommendation system.

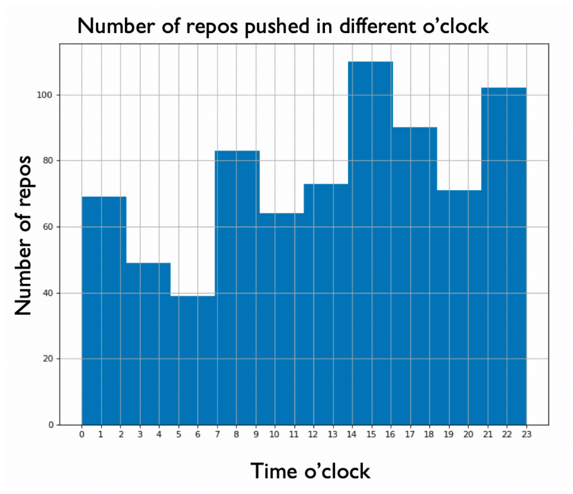
III. METHODOLOGY

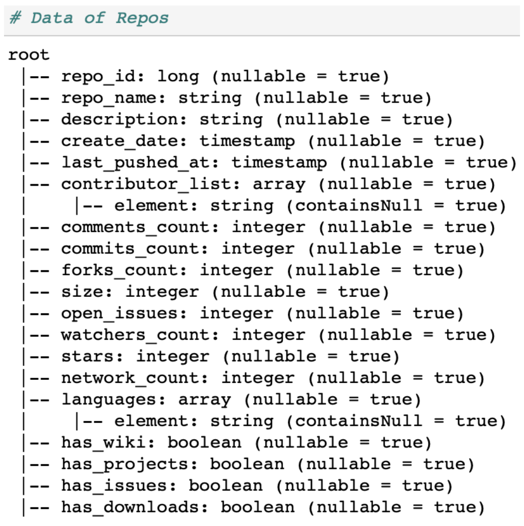
A. *Overvirew of Methodology*

The analysis process consists of data collection, data preprocessing and exploration, modelling, evaluation. This is very common process in data mining. These steps will be elaborated in the following.

B. *Data Collection*

The dataset was collected by the whole team from GitHub API. In the process, we dealt with the issue of API rate limit and search queries. It includes a subset of public repositories whose latest pushing date is after 2019. Our team did the preliminary analysis on number of pushed repositories each hour within a day, and figure I shows that most push activities happen between 7 a.m. and 11.59 p.m.





```
# Data of Repos

root
 |-- repo_id: long (nullable = true)
 |-- repo_name: string (nullable = true)
 |-- description: string (nullable = true)
 |-- create_date: timestamp (nullable = true)
 |-- last_pushed_at: timestamp (nullable = true)
 |-- contributor_list: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- comments_count: integer (nullable = true)
 |-- commits_count: integer (nullable = true)
 |-- forks_count: integer (nullable = true)
 |-- size: integer (nullable = true)
 |-- open_issues: integer (nullable = true)
 |-- watchers_count: integer (nullable = true)
 |-- stars: integer (nullable = true)
 |-- network_count: integer (nullable = true)
 |-- languages: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- has_wiki: boolean (nullable = true)
 |-- has_projects: boolean (nullable = true)
 |-- has_issues: boolean (nullable = true)
 |-- has_downloads: boolean (nullable = true)
```

Picture 1 The numnber of repos pushed in different o'clock   Picture 2 Data of users

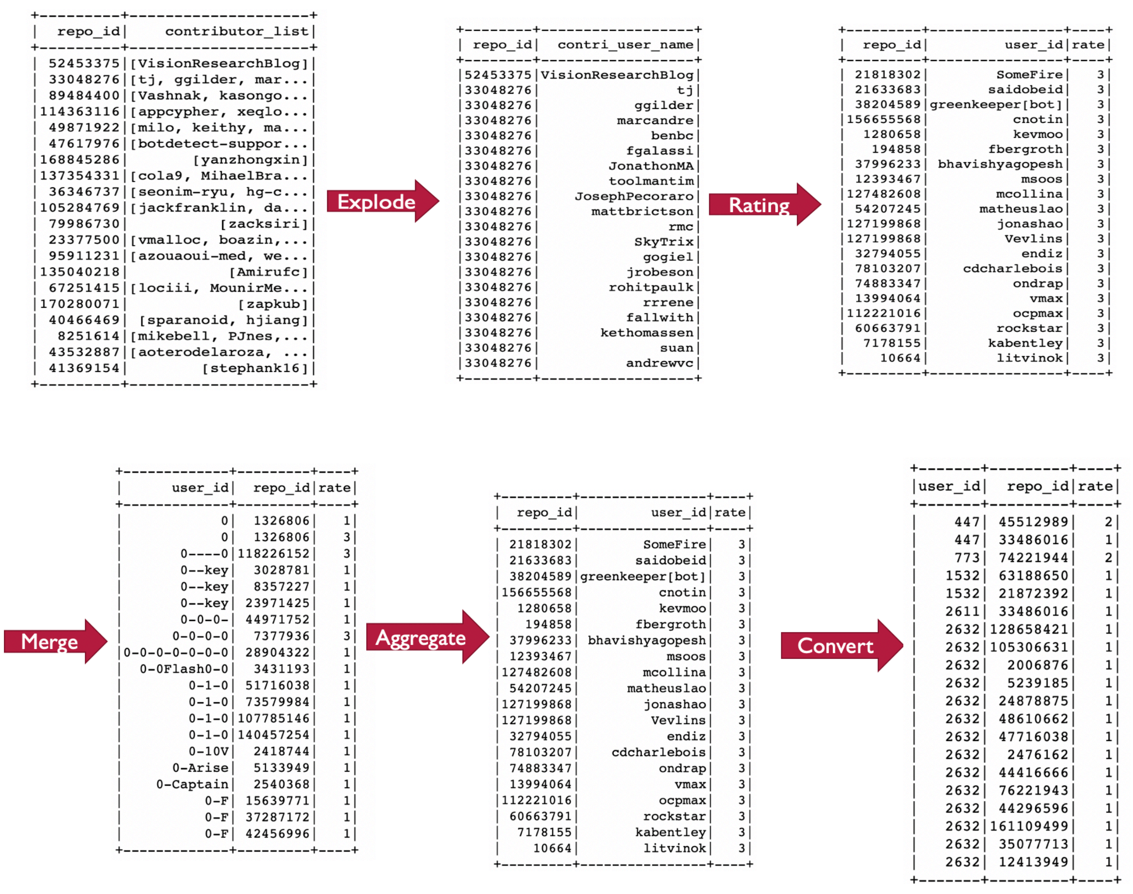
Therefore, we did the sampling by randomly select a 20-minutes time slot each time within the last 40 days between 7 a.m. and 11.59 p.m., and retrieve all the repositories pushed within that time slot. There are two conditions which are applied in search queries to ensure retrieved samples has sufficient information to avoid empty repositories: 1) the repository has at least 1 fork. 2) the repository has at least 1 star.

After the sampling process, we finally obtain over 40,000 repository samples. Besides retrieving the repositories, we also used GitHub API to retrieve user information of all users which were contributors, subscribers, owners and stargazers to the sample repositories.

C. *Data Processing & Exploration*

From the GitHub behaviors, we can know there are four levels of interest the users have in repos: Create, contribute, subscribe and star. "Create" means the user own the repo, which shows the highest level of interest. "Contribute" means the user contribute to the repo and belongs to one part of the repo. "Subscribe" means the user bookmark the repo and will get the notification of issues and update. "star" means the user bookmark the repo but won't receive the notification.

In the repo table, we have owner, contributor list, subscriber list and stargazer list. We need to explode the these list to the repos. After explode the table, we will get a completed table of users and contributors. Based on the different level of relationship, we assign different points to the levels: Create - 4, Contribute - 3, Subscribe - 2, Star - 1. This rating strategy successfully convert behavior information into explicit feedback to the repos.



```
+---------+--------------------+
| repo_id|    contributor_list|
+---------+--------------------+
| 52453375|[VisionResearchBlog]|
| 33048276|[tj, ggilder, mar...|
| 89484400|[Vashnak, kasongo...|
|114363116|[appcypher, xeqlo...|
| 49871922|[milo, keithy, ma...|
| 47617976|[botdetect-suppor...|
|168845286|       [yanzhongxin]|
|137354331|[cola9, MihaelBra...|
| 36346737|[seonim-ryu, hg-c...|
|105284769|[jackfranklin, da...|
| 79986730|          [zacksiri]|
| 23377500|[vmalloc, boazin,...|
| 95911231|[azouaoui-med, we...|
|135040218|           [Amirufc]|
| 67251415|[lociii, MounirMe...|
|170280071|            [zapkub]|
| 40466469| [sparanoid, hjiang]|
|  8251614|[mikebell, PJnes,...|
| 43532887|[aoterodelaroza, ...|
| 41369154|         [stephank16]|
+---------+--------------------+
```

```
+--------+------------------+
| repo_id|   contri_user_name|
+--------+------------------+
|52453375|VisionResearchBlog|
|33048276|                tj|
|33048276|           ggilder|
|33048276|         marcandre|
|33048276|             benbc|
|33048276|          fgalassi|
|33048276|        JonathonMA|
|33048276|        toolmantim|
|33048276|    JosephPecoraro|
|33048276|       mattbrictson|
|33048276|               rmc|
|33048276|           SkyTrix|
|33048276|            gogiel|
|33048276|          jrobeson|
|33048276|        rohitpaulk|
|33048276|            rrrene|
|33048276|          fallwith|
|33048276|        kethomassen|
|33048276|              suan|
|33048276|          andrewvc|
+--------+------------------+
```

```
+---------+----------------+----+
|  repo_id|         user_id|rate|
+---------+----------------+----+
| 21818302|        SomeFire|   3|
| 21633683|       saidobeid|   3|
| 38204589|greenkeeper[bot]|   3|
|156655568|          cnotin|   3|
|  1280658|          kevmoo|   3|
|   194858|       fbergroth|   3|
| 37996233| bhavishyagopesh|   3|
| 12393467|           msoos|   3|
|127482608|         mcollina|   3|
| 54207245|       matheuslao|   3|
|127199868|        jonashao|   3|
|127199868|         Vevlins|   3|
| 32794055|           endiz|   3|
| 78103207|     cdcharlebois|   3|
| 74883347|          ondrap|   3|
| 13994064|            vmax|   3|
|112221016|          ocpmax|   3|
| 60663791|        rockstar|   3|
|  7178155|        kabentley|   3|
|    10664|        litvinok|   3|
+---------+----------------+----+
```

```
+-------------+---------+----+
|      user_id|  repo_id|rate|
+-------------+---------+----+
|            0|  1326806|   1|
|            0|  1326806|   3|
|       0----0|118226152|   3|
|       0--key|  3028781|   1|
|       0--key|  8357227|   1|
|       0--key| 23971425|   1|
|       0-0-0-| 44971752|   1|
|      0-0-0-0|  7377936|   3|
|0-0-0-0-0-0-0| 28904322|   1|
|   0-0Flash0-0|  3431193|   1|
|        0-1-0| 51716038|   1|
|        0-1-0| 73579984|   1|
|        0-1-0|107785146|   1|
|        0-1-0|140457254|   1|
|        0-10V|  2418744|   1|
|      0-Arise|  5133949|   1|
|    0-Captain|  2540368|   1|
|          0-F| 15639771|   1|
|          0-F| 37287172|   1|
|          0-F| 42456996|   1|
+-------------+---------+----+
```

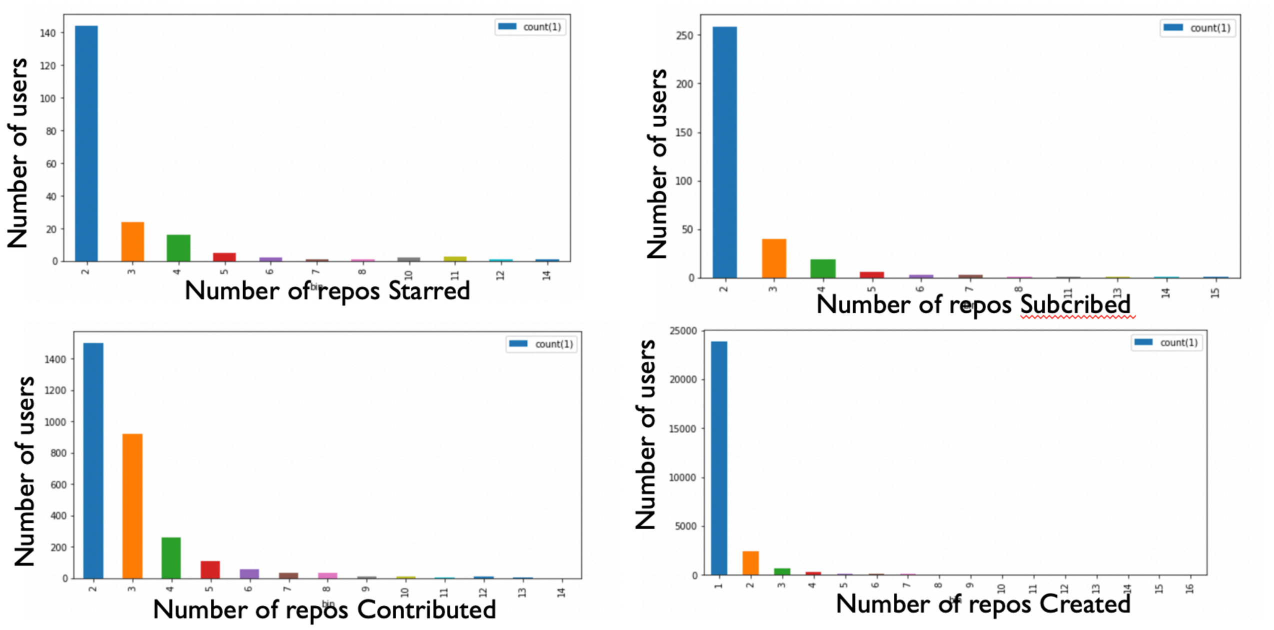
```
+---------+----------------+----+
|  repo_id|         user_id|rate|
+---------+----------------+----+
| 21818302|        SomeFire|   3|
| 21633683|       saidobeid|   3|
| 38204589|greenkeeper[bot]|   3|
|156655568|          cnotin|   3|
|  1280658|          kevmoo|   3|
|   194858|       fbergroth|   3|
| 37996233| bhavishyagopesh|   3|
| 12393467|           msoos|   3|
|127482608|         mcollina|   3|
| 54207245|       matheuslao|   3|
|127199868|        jonashao|   3|
|127199868|         Vevlins|   3|
| 32794055|           endiz|   3|
| 78103207|     cdcharlebois|   3|
| 74883347|          ondrap|   3|
| 13994064|            vmax|   3|
|112221016|          ocpmax|   3|
| 60663791|        rockstar|   3|
|  7178155|        kabentley|   3|
|    10664|        litvinok|   3|
+---------+----------------+----+
```

```
+-------+---------+----+
|user_id|  repo_id|rate|
+-------+---------+----+
|    447| 45512989|   2|
|    447| 33486016|   1|
|    773| 74221944|   2|
|   1532| 63188650|   1|
|   1532| 21872392|   1|
|   2611| 33486016|   1|
|   2632|128658421|   1|
|   2632|105306631|   1|
|   2632|  2006876|   1|
|   2632|  5239185|   1|
|   2632| 24878875|   1|
|   2632| 48610662|   1|
|   2632| 47716038|   1|
|   2632|  2476162|   1|
|   2632| 44416666|   1|
|   2632| 76221943|   1|
|   2632| 44296596|   1|
|   2632|161109499|   1|
|   2632| 35077713|   1|
|   2632| 12413949|   1|
+-------+---------+----+
```

Picture 3 Data Processing

However, sometimes one user can have several behaviors to the same repository simultaneously. For example, one user can be the owner of a repo but also contributor. In this case, the user will have two ratings for one repository, which are 3 and 4. To solve the problem, we need to aggregate the user and repositories, and pick up the highest score as the final rating. In the previous case, we choose 4 as the final rating for the customer to the repo. At last, user ID need to be converted into numerical variable.

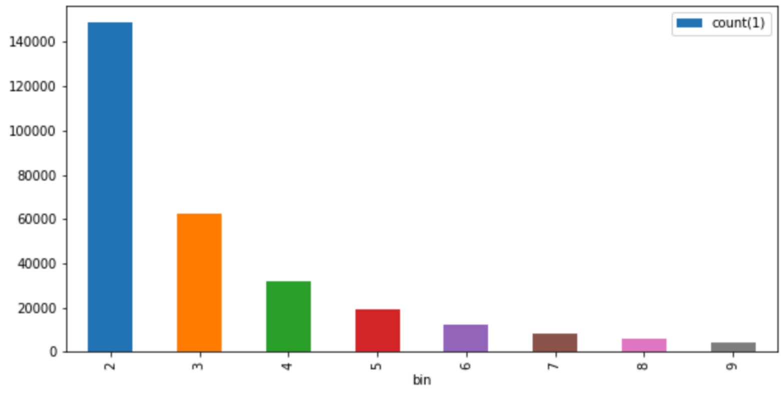
The picture 3 shows the data processing.



Picture 4 the number of users against number of repos

First, we are going to explore the distribution of stargazer, subscribers, contributors, and creators. The following pictures shows number of users against the number of repos which is starred/subscribed/contributed/created by the users. The four diagrams show the same trend that with the number of repos increase, the number of users decrease. Most of users will have one repository, therefore we drop bin one for some behaviors because the number is too large to compare. This makes sense because most of the users is not very activate in GitHub community.

To make sure our model has a good ability of prediction, we need to filter the user whose number of rating is less than 5. After fliting, we can see the frequency distribution from the following picture. The width of bin is 5. We can see, most of the customer have records of repositories between 10 and 25. The preselection process avoid the problem of cold start in the recommendation system.



Picture 5 number of users against number of repositories recorded(bin=5)

Also, we need to check the distribution of ratings. It's very clear that the with interest of repo increase, the number of ratings decrease. It makes sense because the higher rating it is, the more time the customer needs to spend on the repos. Most people will choose the easiest way - star the repository.

	rate	count(1)
0	1	5339863
1	2	265159
2	3	186322
3	4	10041

Table 1 rating frequency

D. Modelling

In order to recommend a product, we applied the Alternating Least Squares (ALS) matrix factorization method [4], which is an advanced version of collaborative filtering method. Collaborative filtering (CF) is a recommendation method based on user behavior similarity. CF provides a recommendation to that user or others who have similar tastes by measuring r_{ui}, which is an observation of item i from user u such as rating and implicit user behaviors. r_{ui} could be explicit ratings including reviews or implicit ratings such as browsing or purchase frequencies. Using k items rated by u that are most similar to i and similarity measure s_{ij} between item i and j, item-oriented CF calculates a predicted rating r_{ij}.

The Alternating least square (ALS) matrix factorization algorithm applies both explicit and implicit user feedback and denotes the strengths in observations of implicit feedback. The ALS method uses binary variable of the preference of user u to item i, pui, where

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases} \quad (1). \qquad c_{ui} = 1 + \alpha r_{ui} \quad (2). \qquad min_{x,y} \sum c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2) \quad (3).$$

Equation 1 means that if a user u rates or browses item i ($r_{ui} > 0$), this means that u likes i. Otherwise, there is no preference. For $p_{ui} = 1$, we can calculate confidence level c_{ui} , where c_{ui} increases according to r_{ui} and α is a constant and Hu's experiments demonstrate good results with $\alpha = 40$. Using the preference and confidence matrices, the algorithm finds latent factors of a user's preference to an item factored by x_u and y_i and recommends items. User and item factors are obtained from the objective function in Equation 3. where λ is data-dependent and determined by cross- validation. All possible u and i pairs should be calculated for unobserved data and the possible combinations of users and items could be easily over a few billion. The ALS matrix algorithm applies an efficient optimization process by fixing either user factors (x_u) or item factors (y_i) and recomputing the other factor alternately and achieves linear computing time.

The system provides recommendations to a user based on the highest predicted ratings using ALS developed on the Apache Spark. After data preprocessing and preselection, we get 5,801,385 samples. We randomly split the data into training set and test set with ratio of 3:7.

After splitting, we need to check the distribution of training set and test set to make sure they have a similar distribution. The result shows that random split works well.

```
+----+-------+
|rate|  count|
+----+-------+
|   1|4272238|
|   2| 211989|
|   3| 148982|
|   4|   8062|
+----+-------+
```

Table 2 training set.

```
+----+-------+
|rate|  count|
+----+-------+
|   1|1067625|
|   2|  53170|
|   3|  37340|
|   4|   1979|
+----+-------+
```

Table 3 test set

We train our data to get a model and provide personalized recommendations. We used PySpark ML library to process our data and applied the ALS algorithm to build a recommendation system.

E. Evaluation

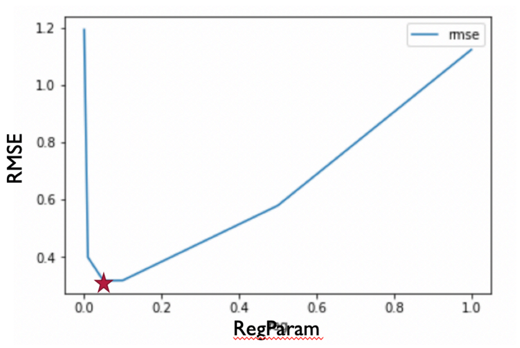
To evaluate our model, we use RMSE as our evaluation matrix. We used root mean squared error (RMSE) to evaluate the model performance. RMSE measures an average distance between the predicted rating and the actual rating. When we set the hyperparameter as defaulted value (RegParam=0.01), we can get 0.39872.

$$RMSE = \sqrt{\frac{\sum (r_{ui} - \hat{r}_{ui})^2}{k}}$$

To further improve root mean square error of our model, we need to do grid search for hyper parameter tuning. There are three parameters: rank, max iteration and regularization parameters. Rank is the number of latent variables in the model. MaxIter: Maximum iteration times. the algorithm generally converges to a promising solution within 10 iterations. RegParam: RegParam specifies the regularization parameter. Here we mainly focus on the tuning of regularization parameter.

	reg	rmse
0	0.0005	1.193938
1	0.0100	0.398724
2	0.0500	0.317228
3	0.1000	0.317980
4	0.5000	0.579120
5	1.0000	1.123761

Table 4. RMSE and RegParam



Picture 6 RMSE and RegParam

In the hyperparameter tuning, we use 3 cross validation. The result shows that when the RegParam equals 0.05, the model have a best performance. The RMSE in test set is 0.3173. It means the predicted rating has little difference between actual rating.

```
+-------+--------------------+
|repo_id|     recommendations|
+-------+--------------------+
|1231710|[[1104723, 6.2099...|
|3837696|[[8132786, 8.3066...|
|4189105|[[3822112, 3.8015...|
|5967893|[[1914405, 4.7893...|
|6116266|[[1624020, 5.7906...|
|6632698|[[8378708, 10.636...|
|7079500|[[6225013, 5.5033...|
|7533238|[[4879030, 5.5826...|
|7611307|[[5716644, 7.5900...|
|8003417|[[5668813, 6.2299...|
+-------+--------------------+
```

Table 5. Recommended users to repos

```
+-------+--------------------+
|user_id|     recommendations|
+-------+--------------------+
|   8592|[[110691676, 5.53...|
|  10817|[[48311711, 3.820...|
|  28088|[[26092173, 3.869...|
|  29194|[[171046539, 3.76...|
|  43302|[[101424202, 3.62...|
|  44358|[[127570975, 3.81...|
|  53691|[[52882030, 3.806...|
|  57039|[[52882030, 3.690...|
|  67376|[[26092173, 3.970...|
|  70097|[[48311711, 3.778...|
+-------+--------------------+
```

Table 6. Recommended repos to users

By our model, we can invite 10 best users who may be interested in the repositories. Also, we can recommend 10 best repositories to each user. The result can be deployed in GitHub to make a recommendation to users and repositories. With the low RMSE, the performance of the recommendation engine is good.

IV. Conclusion

In this work, I developed a repository recommender system using Apache Spark and its ALS algorithm. The data is collected from GitHub API and a series of data preprocessing is implemented to convert the behaviors of users to repos into ratings, which can be applied in recommendation engine. In addition, preselection make sure the model aviod the problem of cold start. RMSE is used as evaluation matrix and hyperparameter tuning is implemented to improve the performance. The results showed a decent root mean square error (0.3173) as an output of the recommender model.

Further study is also required to improve the model. For example, we can tune the hyperparameter of ranks to get the optimum perameter[5]. Also, we may consider how to recommende the repository to the new users who have little historic data.

References

[1] V. Cosentino, J. L. C. Izquierdo and J. Cabot, "Findings from GitHub: Methods, Datasets and Limitations," *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, Austin, TX, 2016, pp. 137-141.

[2] Kalliamvakou, Eirini, et al. "The promises and perils of mining GitHub." Proceedings of the 11th working conference on mining software repositories. ACM, 2014, pp. 92-101.

[3] Chatziasimidis, Fragkiskos, and Ioannis Stamelos. "Data collection and analysis of GitHub repositories and users." *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA)*. IEEE, 2015, pp. 1-6

[4] L. Chen, R. Li, Y. Liu, R. Zhang and D. M. Woodbridge, "Machine learning-based product recommendation using Apache Spark," *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation*, San Francisco, CA, 2017, pp. 1-6.

[5] Xu, Chonghuan. "A novel recommendation method based on social network using matrix factorization technique." *Information Processing & Management* 54.3, 2018, pp. 463-474.