PROJECT REPORT

Subject: CMPE-256

Title: Github Repository Analysis and Recommendation System

Animesh Grover
animesh.grover@sjsu.edu

Mustafa Bandukwala
mustafa.bandukwala@sjsu.edu

Venkatesh Devale
venkatesh.devale@sjsu.edu

Guide
Prof. Magdalini Eirinaki

Table of contents

1 INTRODUCTION

1.1 Motivation

GitHub today is the most popular version control system, where all the open source projects start like TensorFlow by Google, BootStrap by Twitter, React by Facebook to name a few. A lot of projects are created on a daily basis with various programming languages and frameworks used in it. This data gives us lot of key insights on what will be the new languages trend, or which languages are used the most around the world. This also helps us in knowing the popularity of not only the language but the frameworks built on top of it.

With this great foundation, it is overwhelming to know that there exists so many programming languages which beginner has to learn. We also care whether the programming language we are currently studying will be popular or not and also do not know which other programming languages we should learn to keep up with the real world.

Considering the above-mentioned challenges we came up with three major use cases that we can work on:

- We decided that we can recommend top three programming languages to the user provided his current language that he can learn based on the repository and language dataset we have.
    - This use case actually solves the challenge of not knowing which language to study further. As GitHub is the current and most up to date data for the software programmers our recommendation will help user in learning the next top three popular programming languages or technologies.

- New users face a lot of challenging issues while working on some project using some programming language, they need a helping hand which can guide them to overcome these challenges, generally we go to StackOverflow, but nowadays we also check github repositories for some related work in that domain and star that repository to follow it further.
    - We see that we can use the state-of-the-art algorithms to provide recommendations to a user with top-k repositories based on the stars that multiple users have given to multiple repositories.
    - This will help them to explore new repositories which are similar to the ones they have already starred or may like in future.

1.2 Objective

We aim to solve the above-mentioned use cases by recommending popular languages for the given input language and provide repository recommendations to the user based on his starred repositories. Further we also do repository and language graph analysis.

2 SYSTEM DESIGN AND IMPLEMENTATION DETAILS

2.1 Algorithms considered

Various algorithms considered for recommendation model are:

- SVD
- KNNBasic
- Co-Clustering
- SlopeOne

All of these algorithms are from surprise library generally used for building and analyzing recommender systems. We will discuss the parameters chosen for each algorithms and their respective outcomes.

- **SVD**: The famous SVD algorithm popularized during the Netflix Prize is equivalent to Probabilistic Matrix Factorization when baselines are not used. The following parameter values were considered for best performance.
  - **n_factors**: 100(the number of factors)
  - **n_epochs**: 20(The number of iteration of the SGD procedure)
  - **biased**: True(whether to use baselines or biases)
  - **init_std_dev**: 0.1(the standard deviation of the normal distribution)
  - **lr_all**: 0.005(the learning rate for all parameters)

SVD was tested once with and without k-cross fold validation.

- **KNNBasic**: KNNBasic is a basic collaborative filtering algorithm where the actual number of neighbors that are aggregated to compute an estimation is necessarily equal to k. The following parameter values were considered for best performance.
  - **k**: **40**(the max number of neighbors to take into account for aggregation.
  - **min_k: 1**(The minimum number of neighbors to take into account for aggregation)
  - **sim_options: {'Cosine', 'Pearson'}** (a dictionary of options for the similarity measure)

- **Co-Clustering**: It is a collaborative filtering algorithm based on co-clustering. In co-clustering, clusters are assigned using a straightforward optimization method, much like k-means. The following parameter values were considered for best performance.
  - **n_cltr_u: 3**(number of user clusters)
  - **n_cltr_i: 3(**number of item clusters)
  - **n_epochs: 20**(number of iteration of the optimization loop)

- **SlopeOne**: SlopeOne is a simple yet accurate collaborative filtering algorithm. It is a straightforward implementation of the SlopeOne algorithm proposed by Daniel Lemire and Anna Maclachlan during Netflix competition under the paper "Slope one predictors for online rating-based collaborative filtering".

- **Performance**: For performance we used RMSE which measures the differences between values predicted by a model or an estimator and the values observed.
  - **SVD without k-cross validation: 0.1318**
  - **SVD with k-cross validation: 0.6399**
  - **KNNBasic with cosine similarity: 0.0041**
  - **KNNBasic with pearson similarity: 0.0011**
  - **Co-Clustering: 1.7508**
  - **SlopeOne: 1.4244**

So, we finally chose **SVD without k-cross** validation for our recommendation model.

2.2 Technologies & Tools used

- **Language:** Python, Google's BigQuery
- **Frameworks:** pandas, numpy, sklearn, scipy, surprise, matplotlib, igraph
- **Tools:** Jupyter Notebook, Kaggle Kernel, Gephi

2.3 Workflow

Figure 1: Recommend next top-3 languages to user that can be learnt

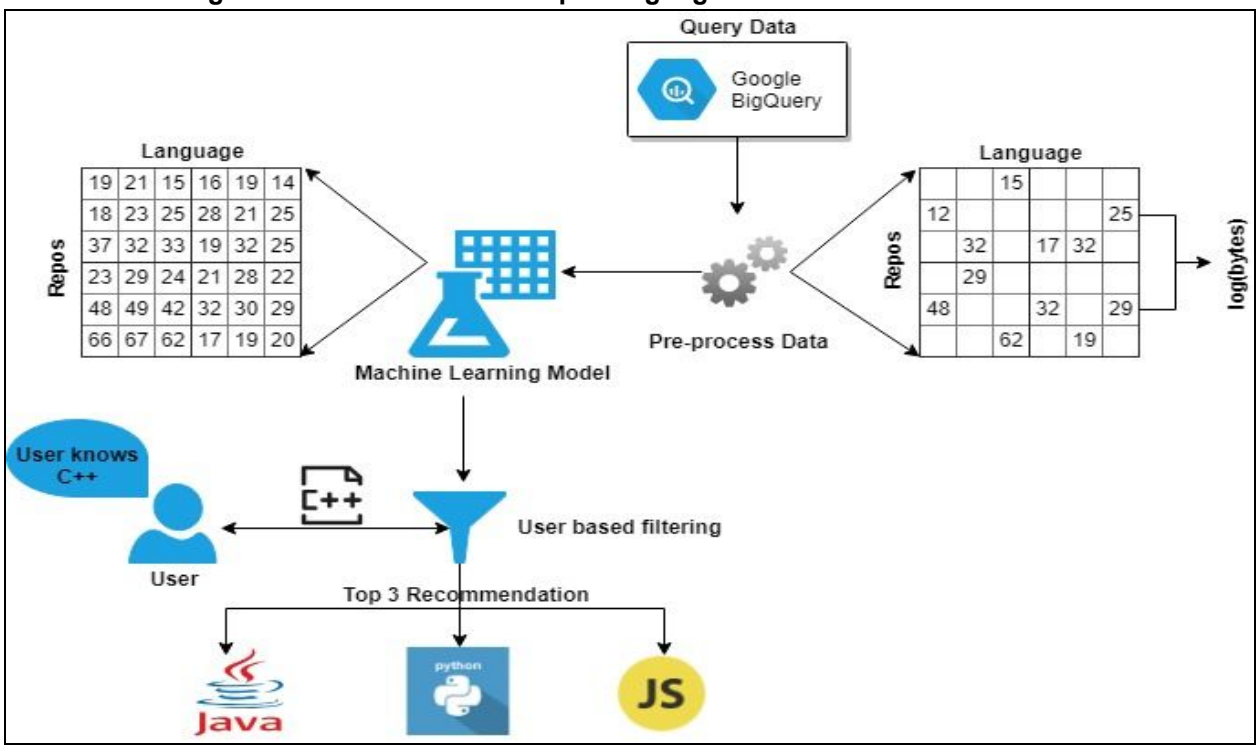
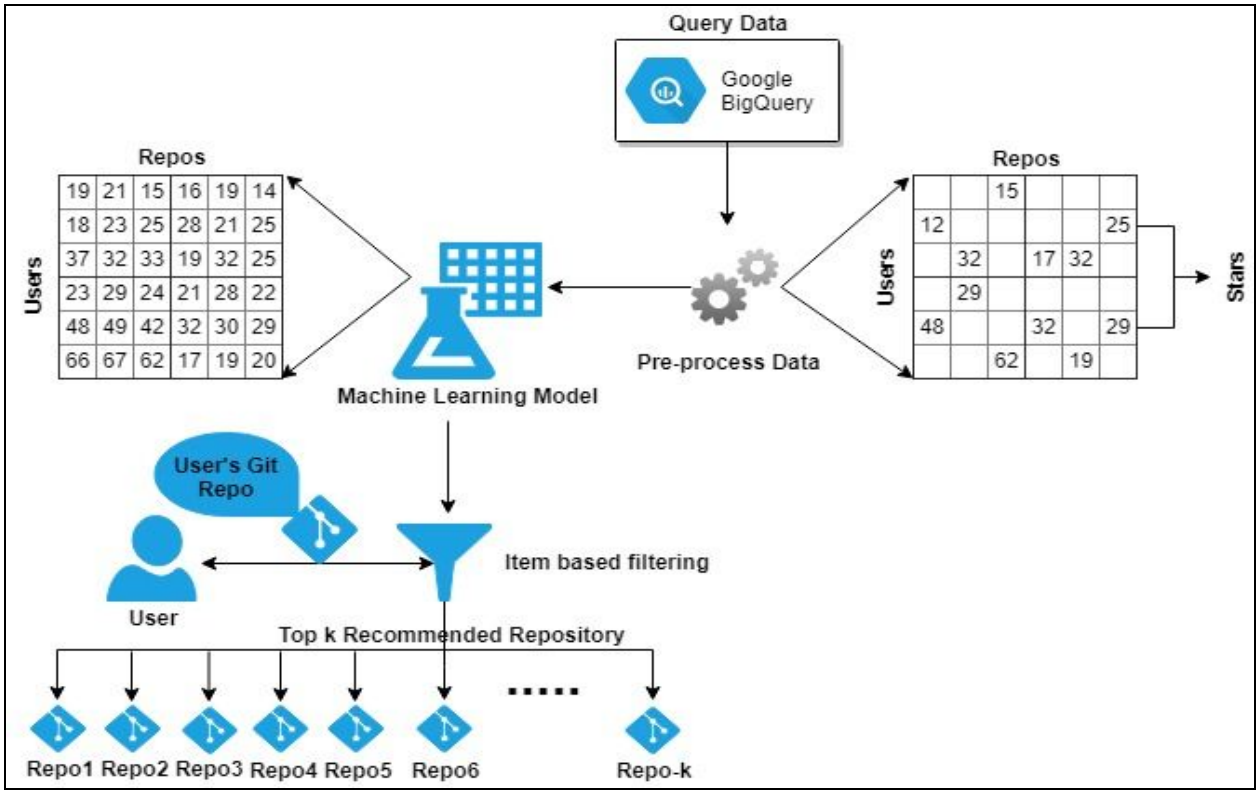
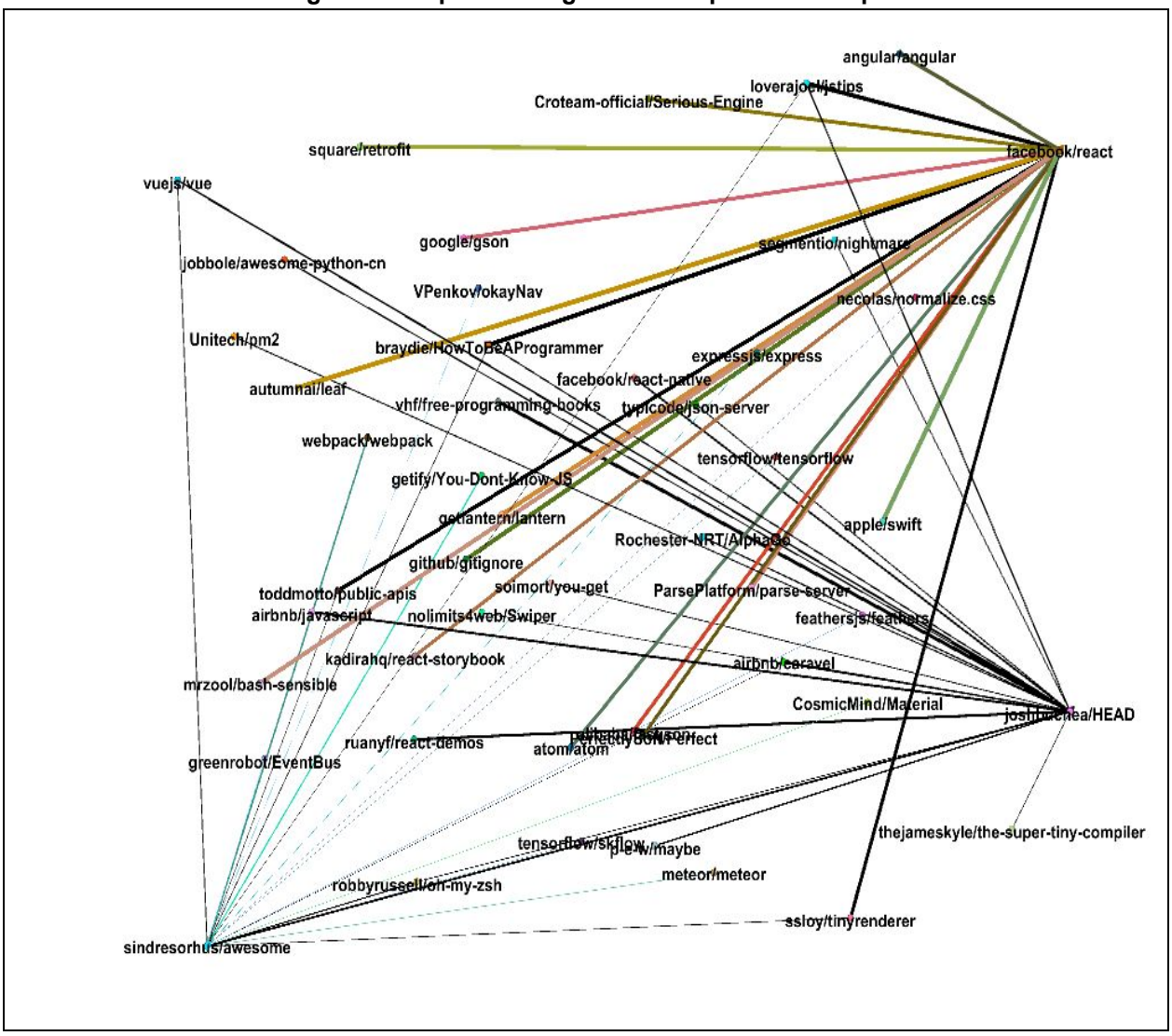


Figure 2: Recommend top-k repositories to user based on the stars



2.4 Visualization

Figure 3: Graph showing relationship between repositories



- The graph contains top-n recommended repositories for 3 users(randomly chosen). These repositories are recommended based on the number of stars they have received. Out of all the recommended repositories to every user, the one with the highest weightage of star count is selected as a central node and the remaining repos are linked to it. Edges are drawn based on the weightage of star of each repository from the central repository.

3 EXPERIMENTS / PROOF OF CONCEPT EVALUATION

3.1 Dataset Used

The dataset used in this project are provided by Google Bigquery which are also available on Kaggle. The dataset contains all the relevant data related to the code and commits made to the 2.8 million repositories since 2008 which are hosted on github. There are total 9 tables that Github has in the Github Repos dataset, they are as follows: commits, content, files, languages, licenses, sample_commits, sample_files, sample_contents, sample_repos.

We used languages and sample_repo for the project to meet our requirement for the two use cases we are trying to work with:

- **languages**: Programming languages by repository as reported by GitHub's API
- **sample_repos**: GitHub repositories with more than 2 stars received during Jan-May 2016.

These datasets provide useful insight as to what languages are highly rated per repository and also reflects which languages are popular and its relation with other languages based on the repository.

3.2 Data preprocessing decisions

All the data used for the recommendation in this project were all acquired by executing bigquery from Google as this dataset is only available and released through this method. We are using queries which already exclude any null and 0 values and we only pull features which are relevant and are required for our learning and prediction algorithms. For example, when we pull the watch count for the repositories, the bigquery we use only returns repositories and watch count with more than 2 stars received. And we also further limit this data by only getting the repositories which received these stars during January to May of 2016 so we don't get an overwhelming amount of data which our system will not be able to handle.

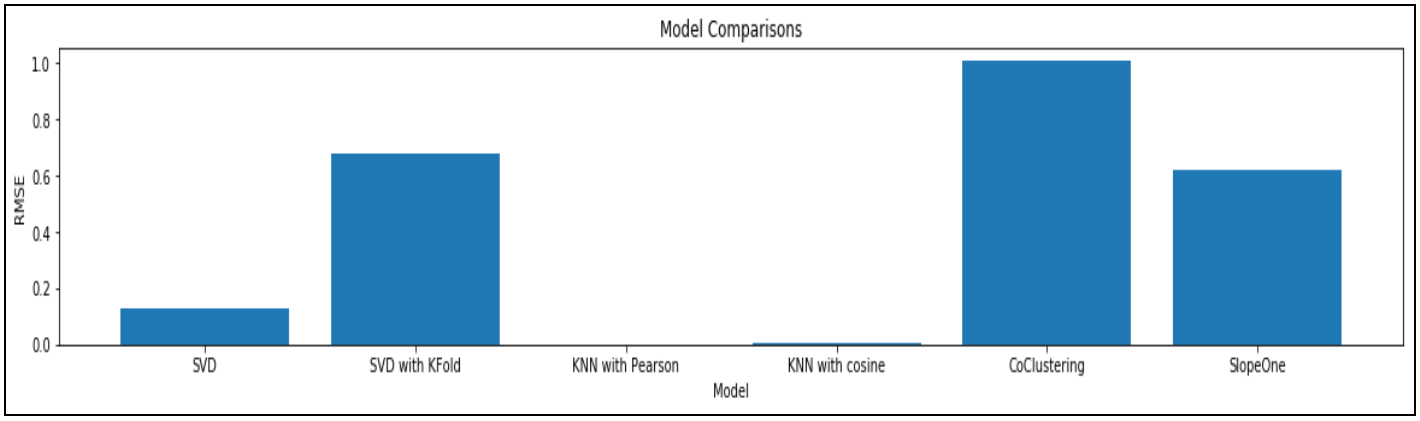
The watch_count for each repository had a big range with one repository with the maximum value of more than 90,000 while mean watch count of all other repository was around 2300. We found the best way to normalize was to divine the watch_count of each repository by the mean so that the data makes more sense and more accurate recommendations could be made by our algorithms. Similarly, for the first use case, when recommending languages we take the log of the bytes of each language used in each repository not only to normalize the data but to also consider each languages weight in each repository.

3.3 Methodology followed

For testing and evaluating different models/algorithms, we have used a sample of 1000 training data. Based on this data, the test set is built using the user and repository combination which is not in the training set data. This was then used to test the accuracy of different prediction algorithms. Based on the best RMSE score for the algorithms tested, we finalized our model and used it to predict rating and recommended top repositories.

3.4 Graphs showing different parameters / algorithms evaluated

Figure 4: Comparison between performance of the different models



Based on these comparisons, KNN with Pearson similarity was the best and accurate model to use followed by KNN with cosine similarity and then SVD.

3.5 Analysis of results

Using KNN with both Pearson and cosine similarity were recommending same repositories to all user and on further investigation, it was found that these model do not work well with the type of dataset and the type of methodology which should be followed.
Therefore, **SVD** was selected as our algorithm for prediction and recommending top repositories.

4 Discussion & Conclusions

4.1 Decisions made

- We have used matrix factorization collaborative filtering to achieve the recommender system that we have aimed to build. We made this decision because we have users who have starred only a few repositories and not all, hence the user-repository matrix relationships were very sparse.
- Though KNN gave us very low RMSE we chose SVD because KNN recommended same repositories to all the users, while SVD gave us better logical recommendations.

4.2 Difficulties faced

- It was a big challenge to get the ‘stars’ feature for each of the repositories that we have in the dataset that too with each user if he has starred that repository or not. We tried using the GitHub’s API to hit and get the desired results but we are allowed only to do 60 ‘GET’ requests per hour and hence we could not follow this approach.
- We faced problems installing igraph library in Kaggle as dependency package(cairo) was missing and we faced authentication issues while installing dependency package in kaggle’s environment. Due to this, we implemented Graph analysis on our local system.
- We tried using GCP but could not get the Kaggle’s dataset in GCP due to authentication restrictions, so we had to reside with Kaggle for our processing.
- Further, Kaggle’s kernel is loaded with very limited resources, it restarted each time we processed 5 lakhs or more records so we had to limit our data to implement our use cases.

4.3 Things that worked

- We designed the system using Google's BigQuery so we did receive the updated data every time we queried, our helper functions implementation is very robust as the implementation uses new updated data each time hence this worked very well for us.
- The matrix factorization implementation using SVD worked pretty well with a very low RMSE compared to Slope One and Co Clustering recommendation algorithms.

4.4 Things that didn't work well

- We aimed to get the current data using GitHub's API but could not get it due to the GitHub APIs restricted access, so this approach did not worked for us
- KNN implementation with 'cosine' and 'pearson' similarities gave us same repository recommendations for any user you pick, so it did not worked as we expected.

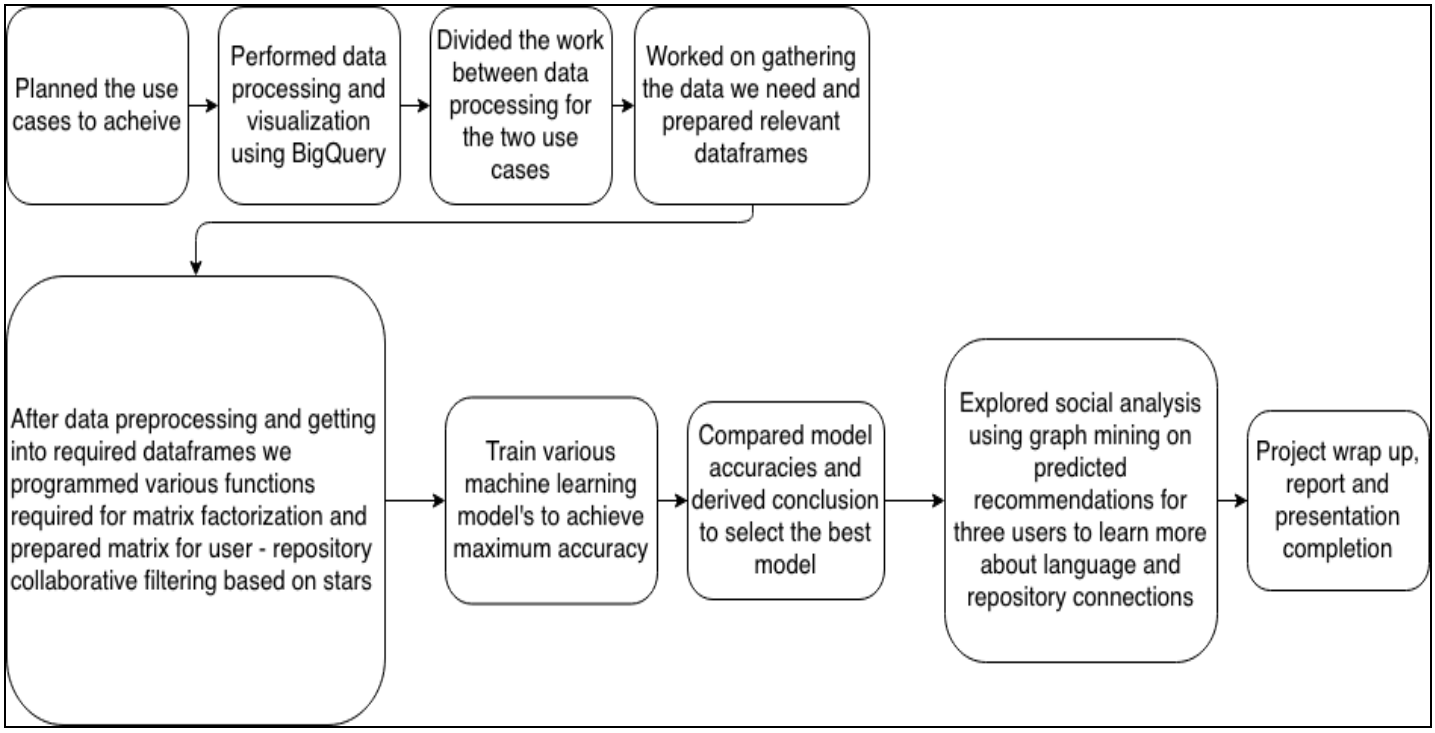
4.5 Conclusion

- We got to learn that large data and large computational power are two required aspects in building recommender systems.
- Matrix factorization works pretty well when you have user, item and rating combination, but when you have metadata related to repository or user you have to explore the data a lot and do a lot of data preprocessing, and then feature engineering with techniques like feature scaling, extraction etc.
- From our study we also found model selection and prediction is comparatively easy part compared to data gathering, exploration, preprocessing and extraction.

5 PROJECT PLAN / TASK DISTRIBUTION

5.1 Project Plan

Figure 5: Workflow of project plan



5.2 Task Distribution

Animesh Grover:

- Contributed in data visualization and building dataframes in the entire project.
- Understood and implemented Loss, Gradient and PCA functions for matrix factorization.
- Worked on matrix factorization to achieve first use case - Recommending top 3 languages given a language based on their popularity in repositories.

Mustafa Bandukwala:

- Contributed in data visualization and building dataframes in the entire project.
- Worked on third use case for Social analysis of recommended repositories.
- Developed helper functions to gather the data implemented filtering data functions with respect to repository searches based on the recommended languages.

Venkatesh Devale:

- Contributed in data visualization and building dataframes in the entire project.
- Implemented matrix factorization and top-k recommendation function for the second use case - recommending repositories to the user based on stars.
- Prepared required data input for further graphical repository - language analysis.

References

1. Surprise library documentation: https://surprise.readthedocs.io/en/stable/getting_started.html
2. Nick Becker's on Matrix Factorization: https://beckernick.github.io/matrix-factorization-recommender/
3. Lawrence Pang's analysis: https://www.kaggle.com/lpang36/recommend-languages-with-collaborative-filtering
4. Corey Ford's analysis: https://github.com/coyotebush/github-network-analysis
5. Luca Hammer on Gephi: https://medium.com/@Luca/guide-analyzing-twitter-networks-with-gephi-0-9-1-2e0220d9097d
6. GHRecommender System: https://medium.com/@andrey_lisin/building-recommender-system-for-github-a8108f0cb2bd
7. Quality Analysis of Open-Source Software: https://link.springer.com/chapter/10.1007%2F978-3-319-13734-6_6
8. Recommender Systems with Social Regularization: http://delivery.acm.org/10.1145/1940000/1935877/p287-ma.pdf?ip=130.65.254.14&id=1935877&acc=ACTIVE%20SERVICE&key=F26C2ADAC1542D74%2ED0BD0A8C52906328%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1542594783_a9907b2673956b2403369a2968c86ce4