PDS Challenge 3:
Matrix Completion for Recommendation Systems

Group 8

蔡長頤、葉宗翰
連亮凱、張峻魁

Video link:
https://youtu.be/VA-K0yFhUX0

Table of contents

A. Abstract

Recommender systems have become integral to the online experience, guiding users to discover products, movies, and music tailored to their preferences. Among various techniques, Singular Value Decomposition (SVD) stands out for its efficacy in building robust recommender systems. This report details our approach to utilizing SVD in predicting movie ratings using the extensive MovieLens20M dataset, a corpus of over 20 million ratings from diverse users.

B. Problem Introduction

In this challenge we need to predict what kind of movie a user likes, and build a recommender system. There are six datasets given in the challenge, which are tag, rating, movie, link, genome_scores, and genome_tags. We need to use SVD (singular value decomposition) to build the system and print the result in a csv file.

C. SVD:

SVD is a form of matrix factorization that decomposes a large matrix into three smaller matrices, capturing the essential information in a more manageable form.

At its core, SVD transforms a complex, potentially large matrix (such as a user-item rating matrix in recommender systems) into three simpler matrices:

1. User Matrix (U): This matrix represents the users and their relationship with underlying latent factors. These latent factors are abstract concepts that might represent user preferences or tendencies.
2. Singular Value Matrix (Σ): This diagonal matrix contains singular values, which quantify the importance of each latent factor. Higher singular values indicate more significant latent factors in explaining the interactions in the original matrix.
3. Movie Matrix (V^T): The transposed matrix of V represents the movies (or items) and their connection to the latent factors. It characterizes how each movie relates to the underlying factors identified in the decomposition.

The beauty of SVD lies in its ability to reduce the dimensionality of data while retaining the most meaningful aspects. By focusing on the principal singular values and their corresponding vectors in U and V^T, SVD approximates the original matrix with a much simpler form, preserving the core interactions between users and items.

D. Methodology:

The Singular Value Decomposition (SVD) algorithm is implemented in various Python libraries, each with its unique characteristics and use cases. The most notable implementations are in the Surprise library, which is often used for recommendation systems, and in other libraries such as NumPy, SciPy, and scikit-learn, which offer more general-purpose linear algebra and machine learning functionalities.

(1) SVD in Surprise Library:

- Purpose: Specifically tailored for building and analyzing recommender systems.
- Algorithm: Optimized for sparse datasets typically found in user-item interaction data.
- Features: Includes functionalities like collaborative filtering, where it can predict the rating a user would give to an item.
- Use Case: Best suited for scenarios like movie recommendations, product suggestions, etc., where the matrix is user-item ratings.
- Customization: Allows for tuning various parameters like the number of factors, epochs, learning rates, etc., to optimize for specific recommendation tasks.

(2) SVD in NumPy/SciPy:

- Purpose: General-purpose linear algebra operations.
- Algorithm: Designed to work with dense matrices and provides the full singular value decomposition.
- Features: Offers a straightforward implementation of SVD without additional functionalities specific to recommendation systems.
- Use Case: Useful in a broad range of applications from signal processing to data compression, where understanding the underlying structure of the data is essential.
- Customization: Limited in terms of tuning for specific tasks compared to the Surprise library.

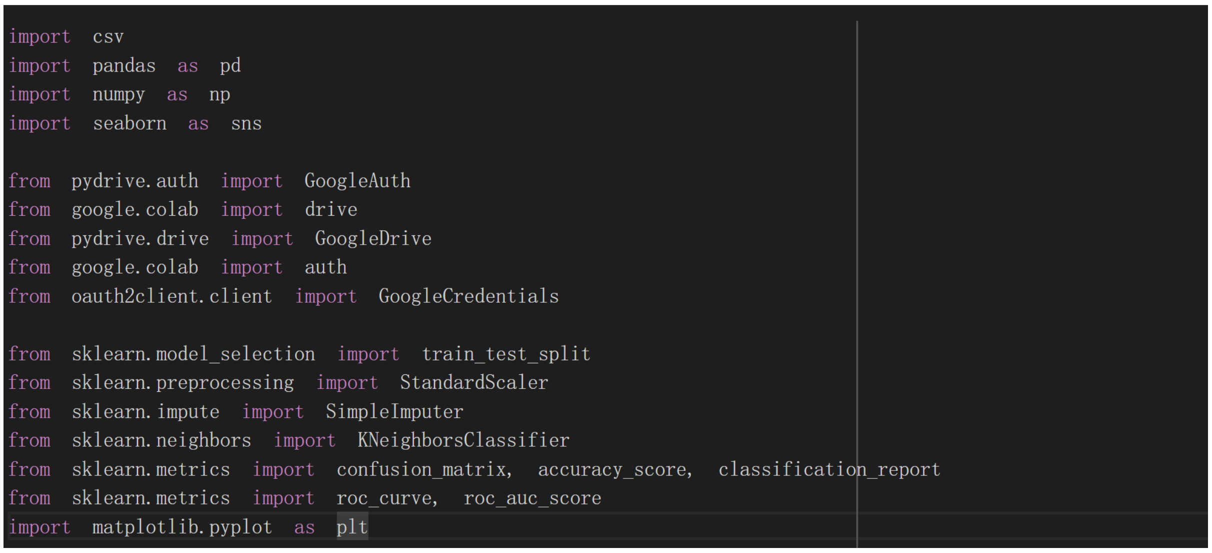
(3) SVD in scikit-learn:

- Purpose: Part of a broader machine learning library.
- Algorithm: Often used in the context of dimensionality reduction (e.g., Truncated SVD).
- Features: Integrated into a larger ecosystem of machine learning tools, making it useful for preprocessing steps in ML workflows.
- Use Case: Commonly used in text processing (like LSA for topic modeling), image compression, and as a preprocessing step in machine learning pipelines.
- Customization: Can be integrated with other scikit-learn tools for building comprehensive machine learning models.

As we can see, SVD in Surprise Library is the most suitable model foe our problem, therefore, we apply **SVD in Surprise Library** in our challenge.

E. Progress

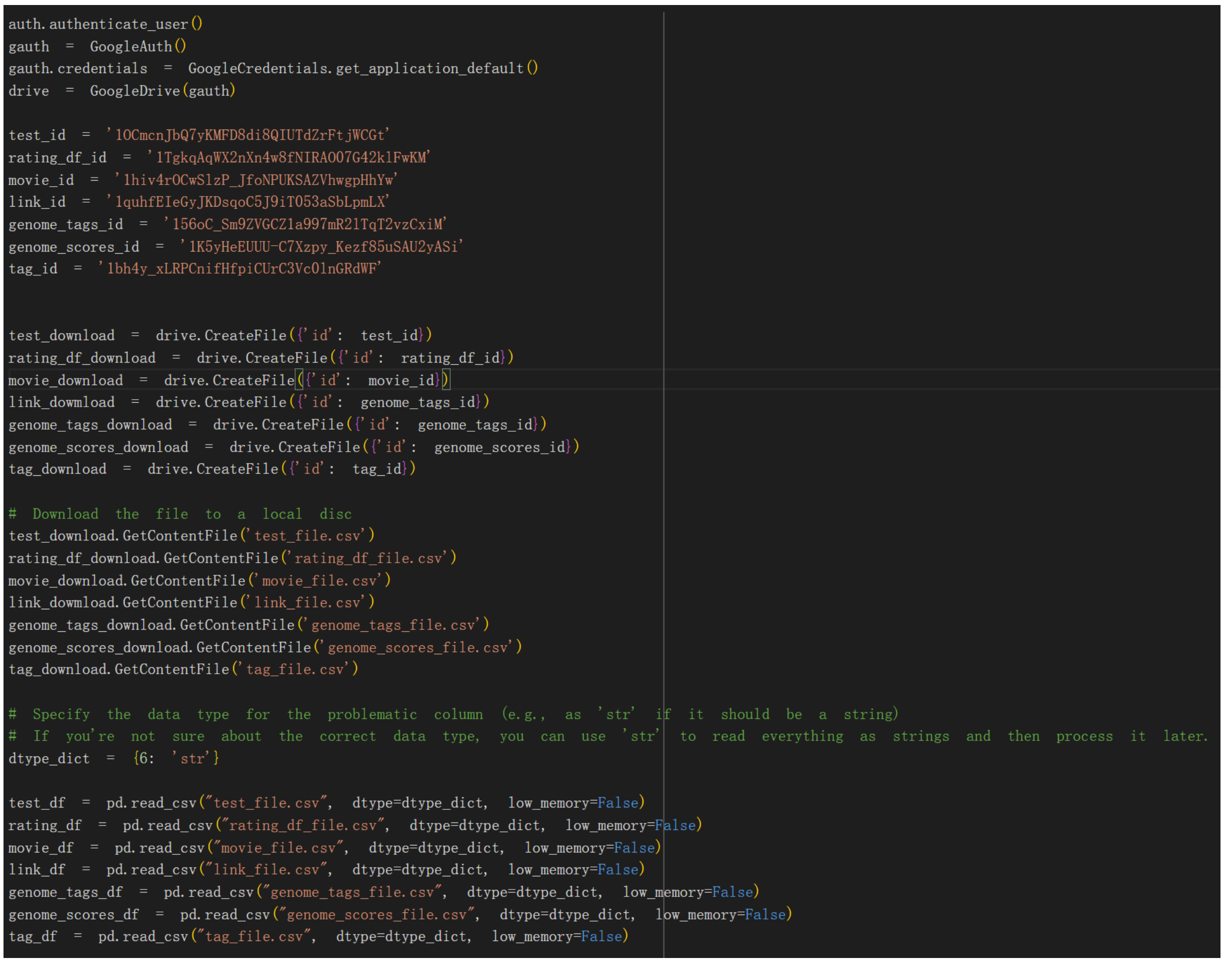
1. Import module and function



```python
import csv
import pandas as pd
import numpy as np
import seaborn as sns

from pydrive.auth import GoogleAuth
from google.colab import drive
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
```

Import the module we need. For example, pandas, numpy, and seaborn for data processing; google.colad and pydrive.dive for importing the testing and training dataset from drive; scikit_learn for prediction.

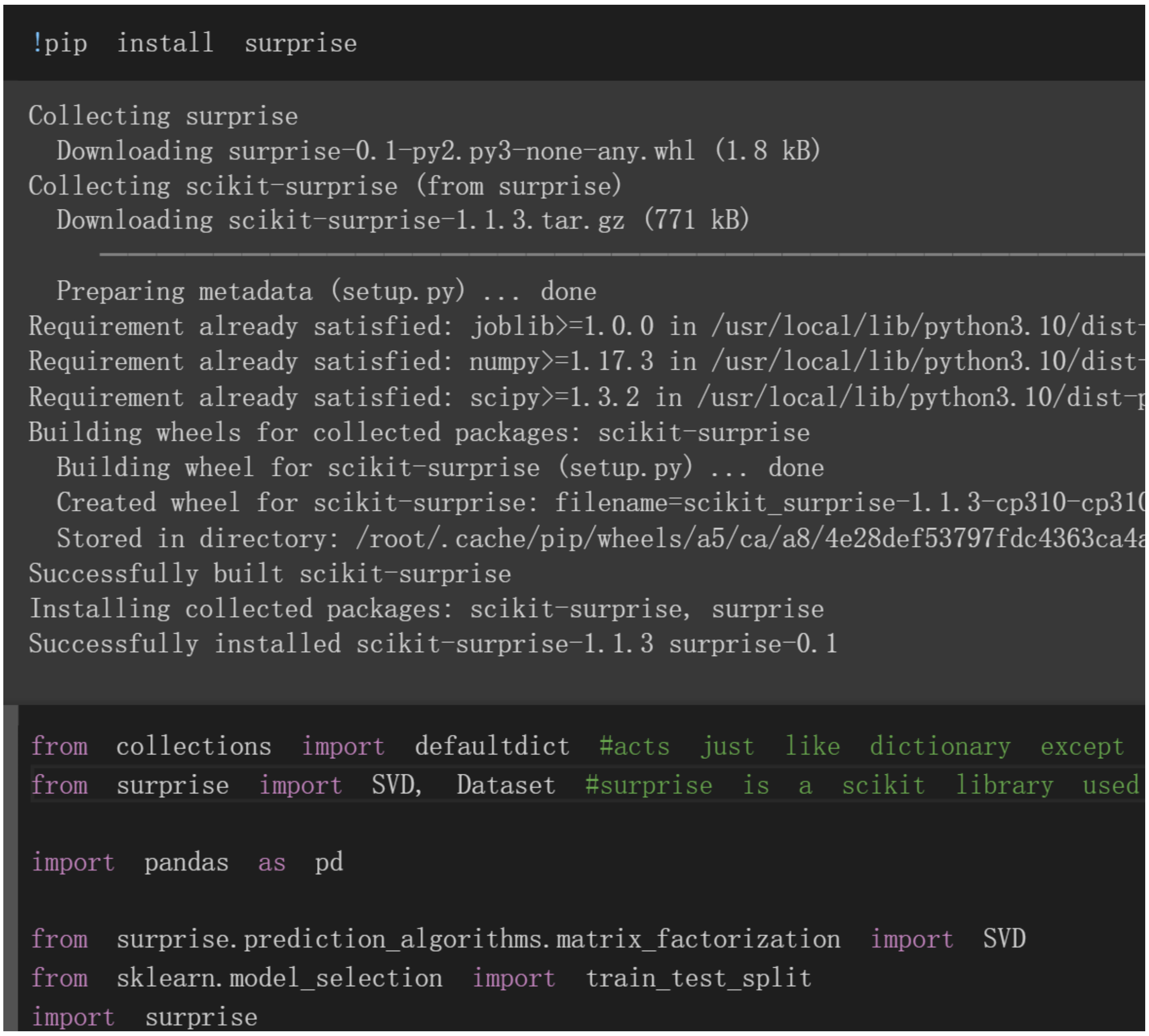
2. Reading the dataset



```python
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

test_id = '1OCmcnJbQ7yKMFD8di8QIUTdZrFtjWCGt'
rating_df_id = '1TgkqAqWX2nXn4w8fNIRAO07G42k1FwKM'
movie_id = '1hiv4rOCwSlzP_JfoNPUKSAZVhwgpHhYw'
link_id = '1quhfEIeGyJKDsqoC5J9iT053aSbLpmLX'
genome_tags_id = '156oC_Sm9ZVGCZ1a997mR21TqT2vzCxiM'
genome_scores_id = '1K5yHeEUUU-C7Xzpy_Kezf85uSAU2yASi'
tag_id = '1bh4y_xLRPCnifHfpiCUrC3Vc01nGRdWF'


test_download = drive.CreateFile({'id': test_id})
rating_df_download = drive.CreateFile({'id': rating_df_id})
movie_download = drive.CreateFile({'id': movie_id})
link_download = drive.CreateFile({'id': genome_tags_id})
genome_tags_download = drive.CreateFile({'id': genome_tags_id})
genome_scores_download = drive.CreateFile({'id': genome_scores_id})
tag_download = drive.CreateFile({'id': tag_id})

# Download the file to a local disc
test_download.GetContentFile('test_file.csv')
rating_df_download.GetContentFile('rating_df_file.csv')
movie_download.GetContentFile('movie_file.csv')
link_download.GetContentFile('link_file.csv')
genome_tags_download.GetContentFile('genome_tags_file.csv')
genome_scores_download.GetContentFile('genome_scores_file.csv')
tag_download.GetContentFile('tag_file.csv')

# Specify the data type for the problematic column (e.g., as 'str' if it should be a string)
# If you're not sure about the correct data type, you can use 'str' to read everything as strings and then process it later.
dtype_dict = {6: 'str'}

test_df = pd.read_csv("test_file.csv", dtype=dtype_dict, low_memory=False)
rating_df = pd.read_csv("rating_df_file.csv", dtype=dtype_dict, low_memory=False)
movie_df = pd.read_csv("movie_file.csv", dtype=dtype_dict, low_memory=False)
link_df = pd.read_csv("link_file.csv", dtype=dtype_dict, low_memory=False)
genome_tags_df = pd.read_csv("genome_tags_file.csv", dtype=dtype_dict, low_memory=False)
genome_scores_df = pd.read_csv("genome_scores_file.csv", dtype=dtype_dict, low_memory=False)
tag_df = pd.read_csv("tag_file.csv", dtype=dtype_dict, low_memory=False)
```

In this part we download the csv files by their id first, then download the csv files to a local disk by the “GetContentFile” function. After downloading the csv files, we use the “read_csv” function to import the data in the six csv files. Also, we set the date type to “string” as default.

3. **Install and import Surprise**



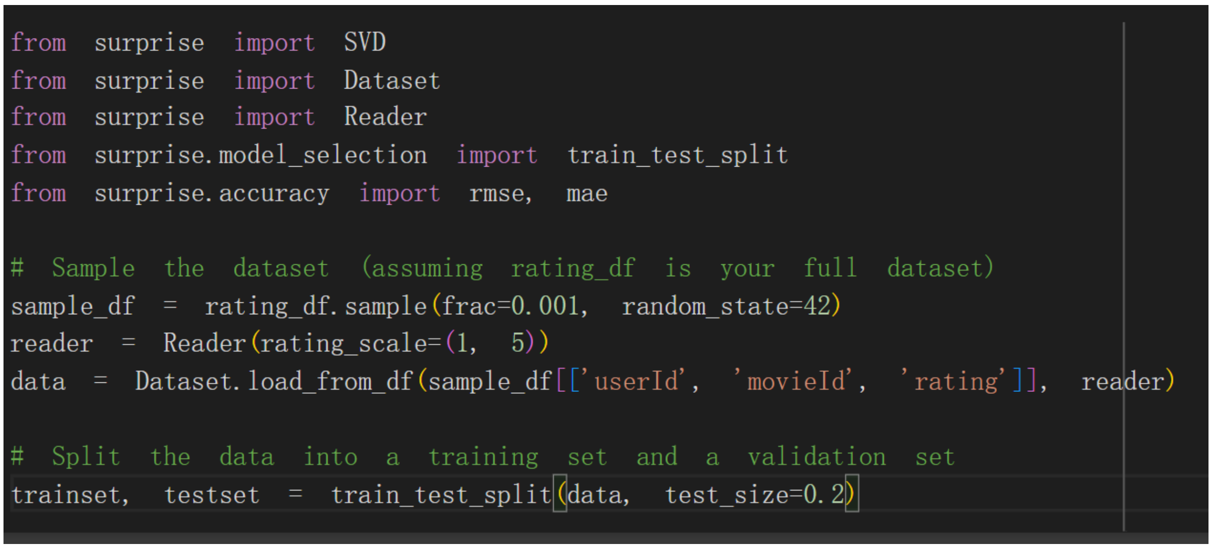
```
!pip  install  surprise
```

```
Collecting surprise
  Downloading surprise-0.1-py2.py3-none-any.whl (1.8 kB)
Collecting scikit-surprise (from surprise)
  Downloading scikit-surprise-1.1.3.tar.gz (771 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
  Preparing metadata (setup.py) ... done
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-p
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.3-cp310-cp310
  Stored in directory: /root/.cache/pip/wheels/a5/ca/a8/4e28def53797fdc4363ca4a
Successfully built scikit-surprise
Installing collected packages: scikit-surprise, surprise
Successfully installed scikit-surprise-1.1.3 surprise-0.1
```

```
from  collections  import  defaultdict  #acts  just  like  dictionary  except
from  surprise  import  SVD,  Dataset  #surprise  is  a  scikit  library  used

import  pandas  as  pd

from  surprise.prediction_algorithms.matrix_factorization  import  SVD
from  sklearn.model_selection  import  train_test_split
import  surprise
```

use pip install to install surprise and import SVD for further process.

4. Data preprocessing for SVD



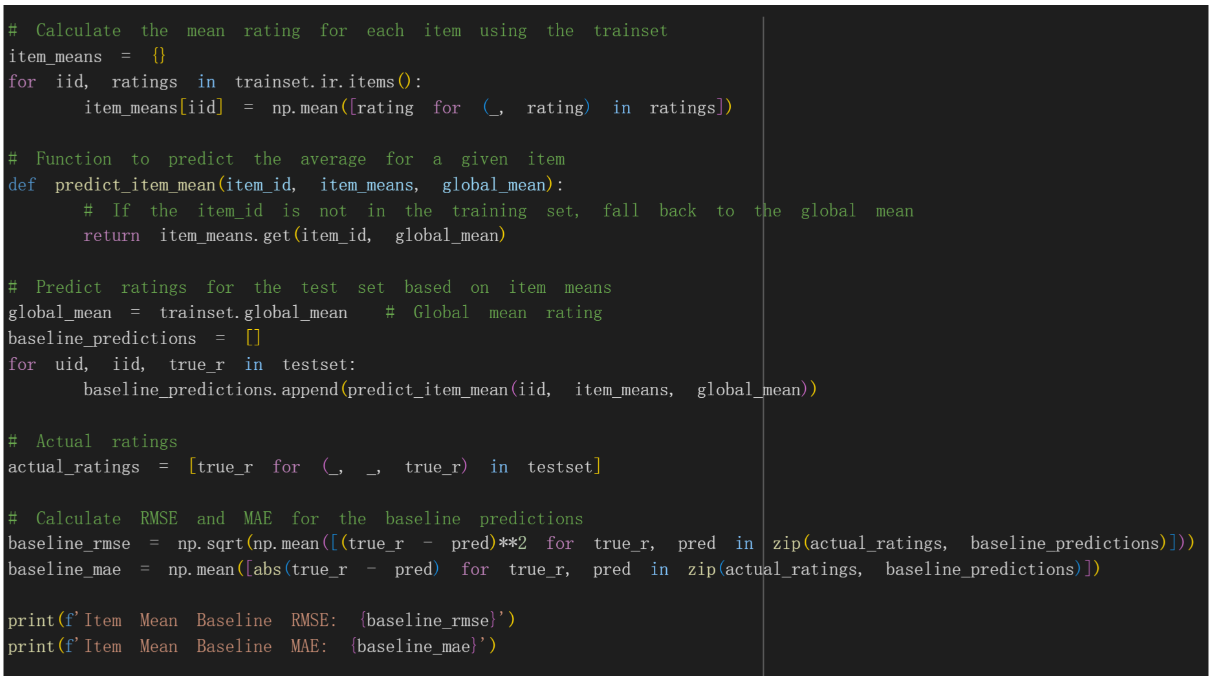
```
from surprise import SVD
from surprise import Dataset
from surprise import Reader
from surprise.model_selection import train_test_split
from surprise.accuracy import rmse, mae

# Sample the dataset (assuming rating_df is your full dataset)
sample_df = rating_df.sample(frac=0.001, random_state=42)
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(sample_df[['userId', 'movieId', 'rating']], reader)

# Split the data into a training set and a validation set
trainset, testset = train_test_split(data, test_size=0.2)
```

(1) Sample the dataset:
The training dataset – ratings.csv is too big and exceeds the computation capacity of google collab. Therefore, we used `sample()` to sample a fraction of 0.001 of the dataset for further computation.

(2) Setting up a `Reader()` with a rating scale from 1 to 5.

(3) Split to train and test datasets:
Use the `train_test_split()` function to divide the the training data into trainset and validation set, and the size of testset is 0.2 of the original training dataset size.
This validation dataset is for further evaluation in our program.

5. Baseline prediction: Predict the item's average rating across all users

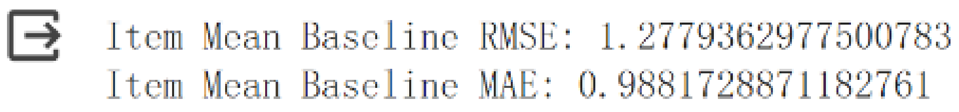


```
# Calculate the mean rating for each item using the trainset
item_means = {}
for iid, ratings in trainset.ir.items():
    item_means[iid] = np.mean([rating for (_, rating) in ratings])

# Function to predict the average for a given item
def predict_item_mean(item_id, item_means, global_mean):
    # If the item_id is not in the training set, fall back to the global mean
    return item_means.get(item_id, global_mean)

# Predict ratings for the test set based on item means
global_mean = trainset.global_mean  # Global mean rating
baseline_predictions = []
for uid, iid, true_r in testset:
    baseline_predictions.append(predict_item_mean(iid, item_means, global_mean))

# Actual ratings
actual_ratings = [true_r for (_, _, true_r) in testset]

# Calculate RMSE and MAE for the baseline predictions
baseline_rmse = np.sqrt(np.mean([(true_r - pred)**2 for true_r, pred in zip(actual_ratings, baseline_predictions)]))
baseline_mae = np.mean([abs(true_r - pred) for true_r, pred in zip(actual_ratings, baseline_predictions)])

print(f'Item Mean Baseline RMSE: {baseline_rmse}')
print(f'Item Mean Baseline MAE: {baseline_mae}')
```

The baseline prediction model is implemented as the item's average rating across all users.

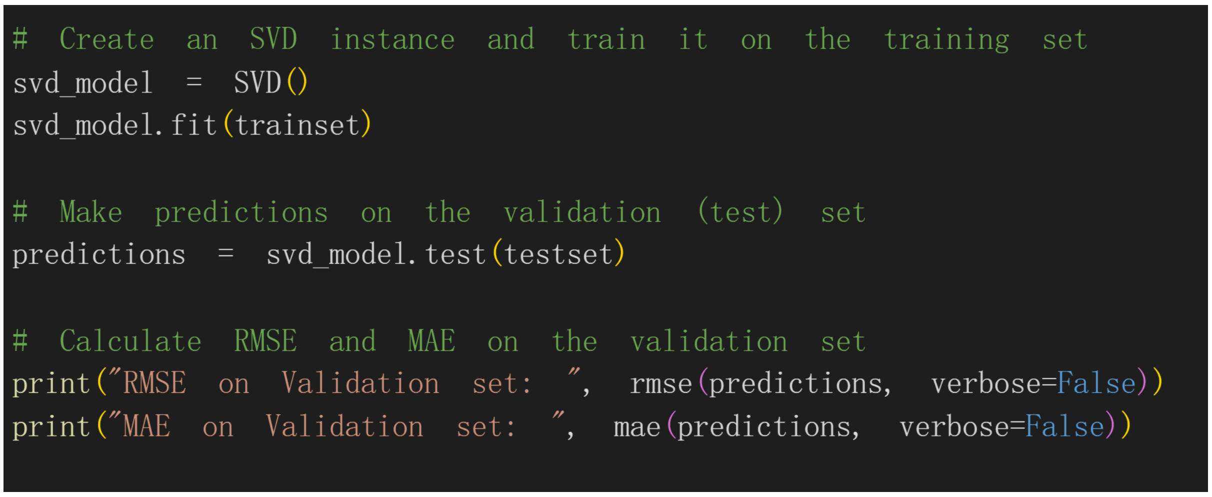
We also calculated the RMSE(root-mean-square error) and the MAE(mean absolute error) of the baseline prediction on the validation data.

Result is as below:



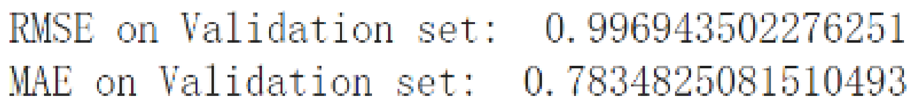
```
Item Mean Baseline RMSE: 1.2779362977500783
Item Mean Baseline MAE: 0.9881728871182761
```

6. **SVD**



```
# Create an SVD instance and train it on the training set
svd_model = SVD()
svd_model.fit(trainset)

# Make predictions on the validation (test) set
predictions = svd_model.test(testset)

# Calculate RMSE and MAE on the validation set
print("RMSE on Validation set: ", rmse(predictions, verbose=False))
print("MAE on Validation set: ", mae(predictions, verbose=False))
```

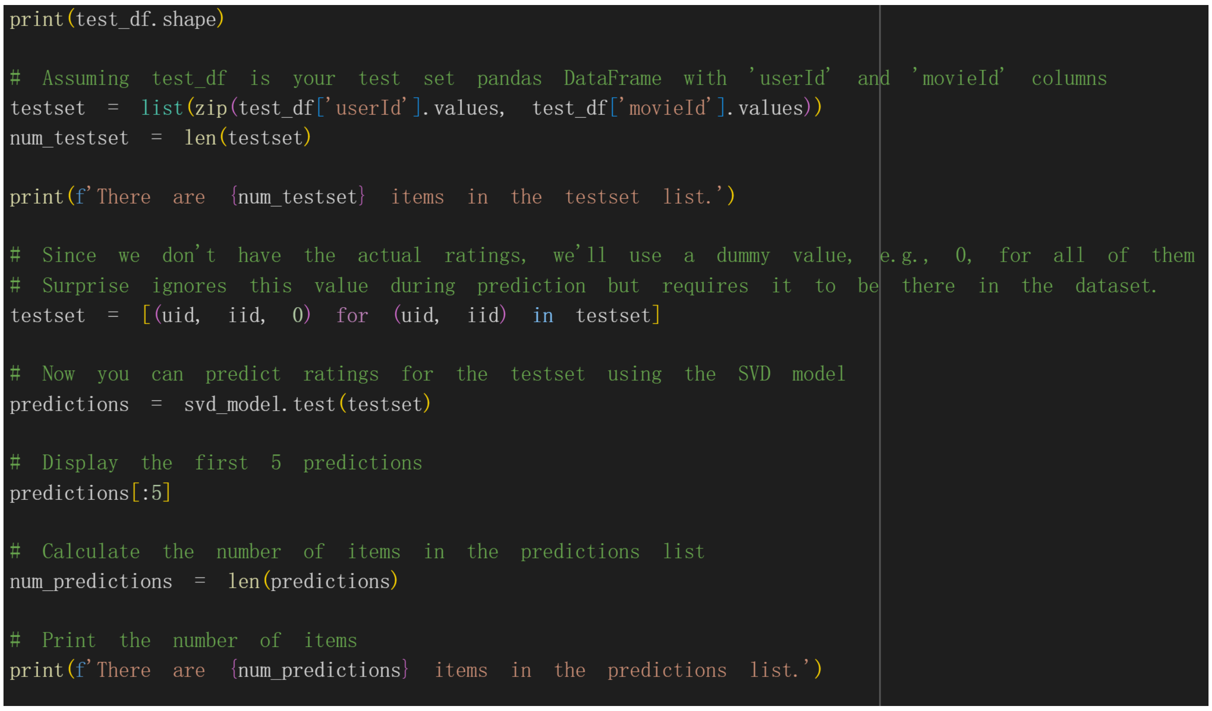
Finally, we applied SVD. We fit the SVD model to the trainset, then use the model to predict on the validation dataset. After the prediction, we also calculated the RMSE(root-mean-square error) and the MAE(mean absolute error) of the prediction.



```
RMSE on Validation set:  0.996943502276251
MAE on Validation set:  0.7834825081510493
```

Comparing to the RMSE and MAE of the Baseline prediction model and the SVD model, we can see that both RMSE and MAE are lowered. This indicates that the SVD model has better performance than the baseline model.

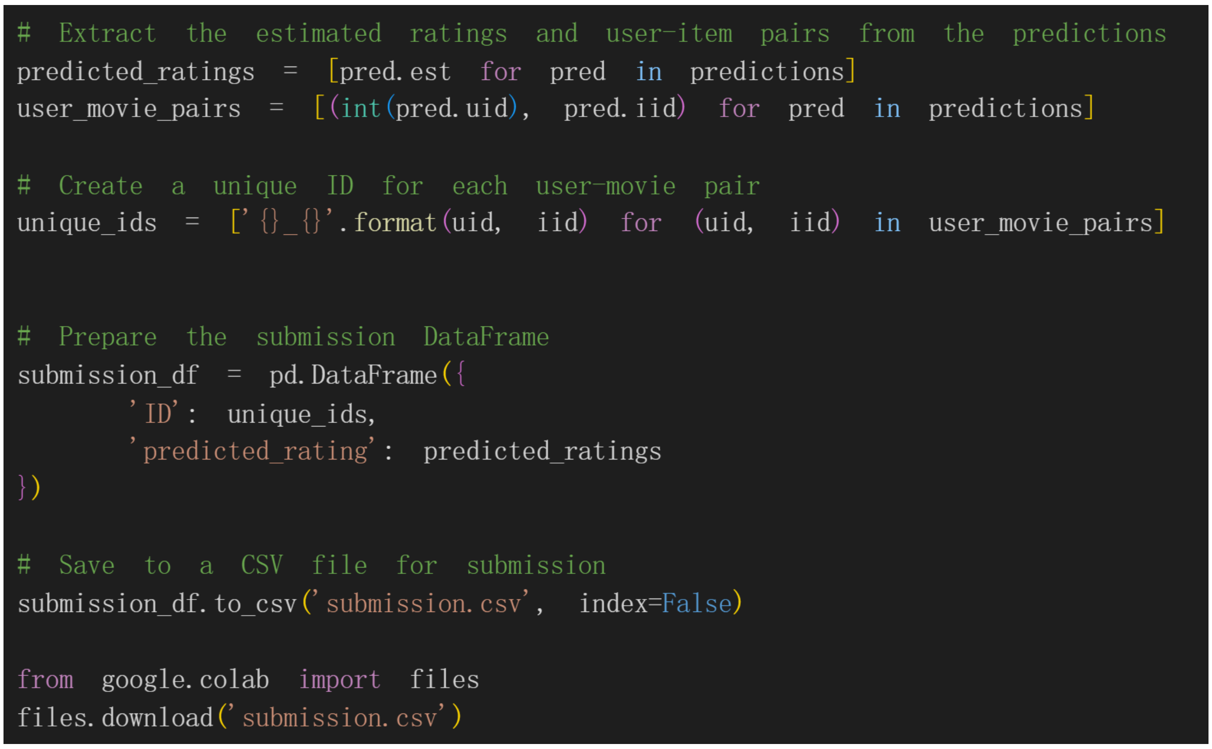
7. Prediction



```python
print(test_df.shape)

# Assuming test_df is your test set pandas DataFrame with 'userId' and 'movieId' columns
testset = list(zip(test_df['userId'].values, test_df['movieId'].values))
num_testset = len(testset)

print(f'There are {num_testset} items in the testset list.')

# Since we don't have the actual ratings, we'll use a dummy value, e.g., 0, for all of them
# Surprise ignores this value during prediction but requires it to be there in the dataset.
testset = [(uid, iid, 0) for (uid, iid) in testset]

# Now you can predict ratings for the testset using the SVD model
predictions = svd_model.test(testset)

# Display the first 5 predictions
predictions[:5]

# Calculate the number of items in the predictions list
num_predictions = len(predictions)

# Print the number of items
print(f'There are {num_predictions} items in the predictions list.')
```

After using the testset to test the precision of the model, we can use the SVD model to predict ratings of the testing dataset from the challenge.

8. Output



```python
# Extract the estimated ratings and user-item pairs from the predictions
predicted_ratings = [pred.est for pred in predictions]
user_movie_pairs = [(int(pred.uid), pred.iid) for pred in predictions]

# Create a unique ID for each user-movie pair
unique_ids = ['{}_{}'.format(uid, iid) for (uid, iid) in user_movie_pairs]


# Prepare the submission DataFrame
submission_df = pd.DataFrame({
        'ID': unique_ids,
        'predicted_rating': predicted_ratings
})

# Save to a CSV file for submission
submission_df.to_csv('submission.csv', index=False)

from google.colab import files
files.download('submission.csv')
```

Save the predicted result in submission.csv, then download the file to local directory.