Group members:
Alireza Mahinparvar - 014295437
Satya Sai Roopa Sree Chiluvuri - 016005795
Yashwanth Kumar Nelakuditi - 015918604
Shanmukh Krishna - 016005743

Collaborative filtering - Singular value decomposition project

Project description

The rapid growth of data collection has led to a new era of information. Recommendation Systems are a type of information filtering systems as they improve the quality of search results and provide items that are more relevant to the search item or are related to the search history of the user. In this project, we'll be building a baseline Movie Recommendation System using TMDB 5000 Movie Dataset. For us the idea is to calculate the similarities between the movies in the dataset using similarity measures like cosine or pearson's correlation. But with this we will face issues like scalability and sparsity. One way to handle this is to leverage a latent factor model to find the similarities between the items and users.

This is the point where Singular Value Decomposition (SVD) comes to the rescue. SVD decreases the dimension of the utility matrix by extracting its latent factors. Essentially, we map each user and each item into a latent space with dimension r. Therefore, it helps us better understand the relationship between users and items as they become directly comparable.

DB Schema, Data statistics, indexes used

To talk about this dataset I should say that our dataset includes two separate sets of dataset.
One was rating_small.csv which includes 4 columns of data:

1. **userid** - a unique identifier for each user
2. **movieid** - a unique identifier for each movie
3. **rating** - rating of a movie given by a user
4. **timestamp** - time at which this rating has been given. we don't concentrate on this feature for this project.

By looking into the data we can see that the range of userId is from 0 to 671(the distribution of userId shows in figure1) and rating is from 0 to 5(the distribution of Rating shows in figure2). Scatter matrix and correlation plot shows that movieid and timestamp have the highest correlation among others . From Figure 3 and Figure 4 we can see that timestamp and movieid has 0.5 correlation while the others are 0 or 0.1

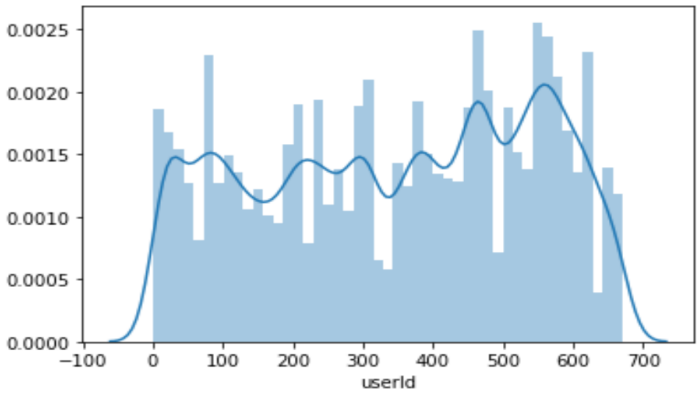


Figure1

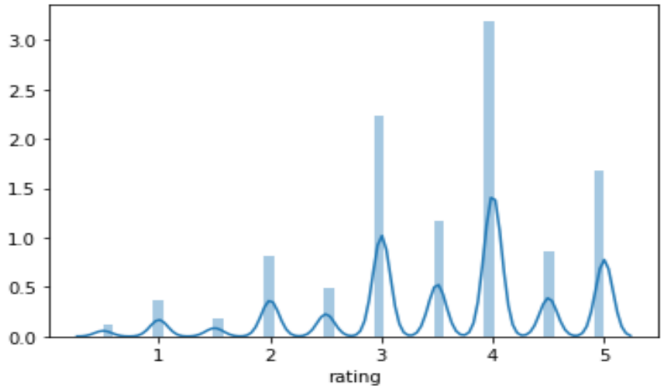


Figure2

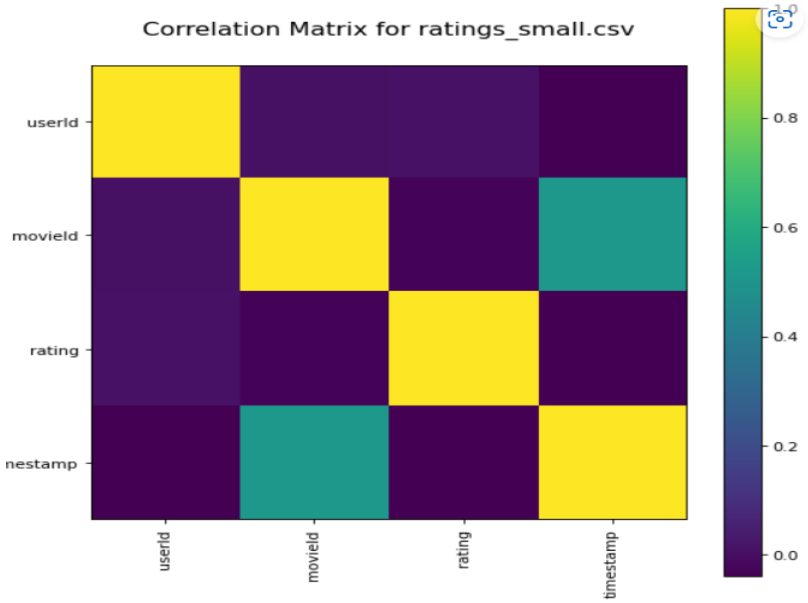


Figure3

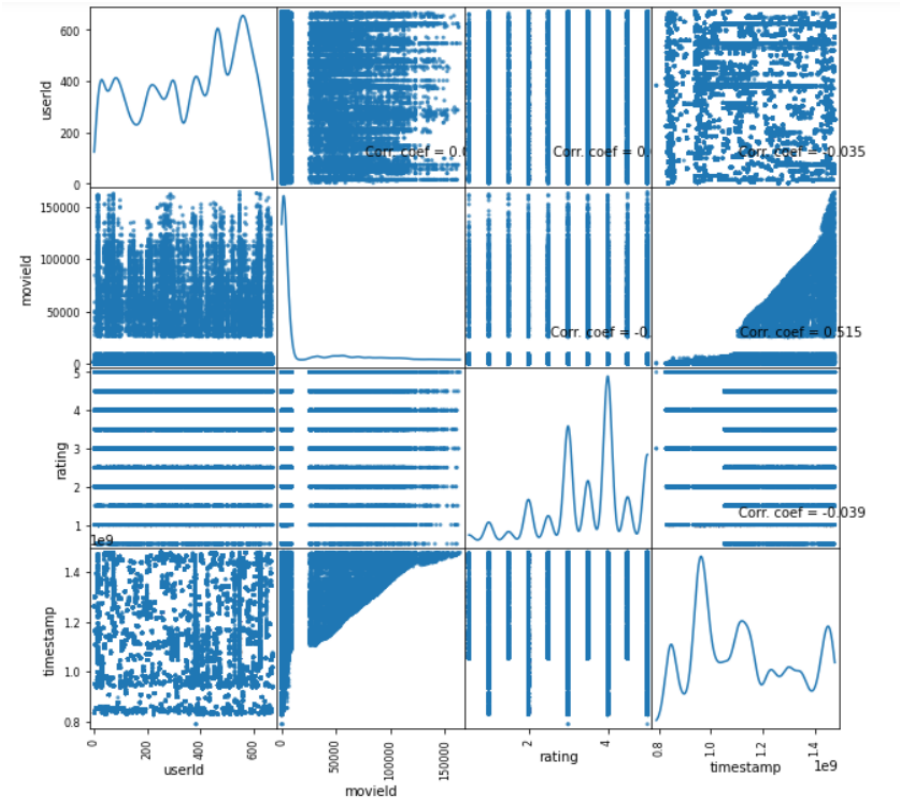
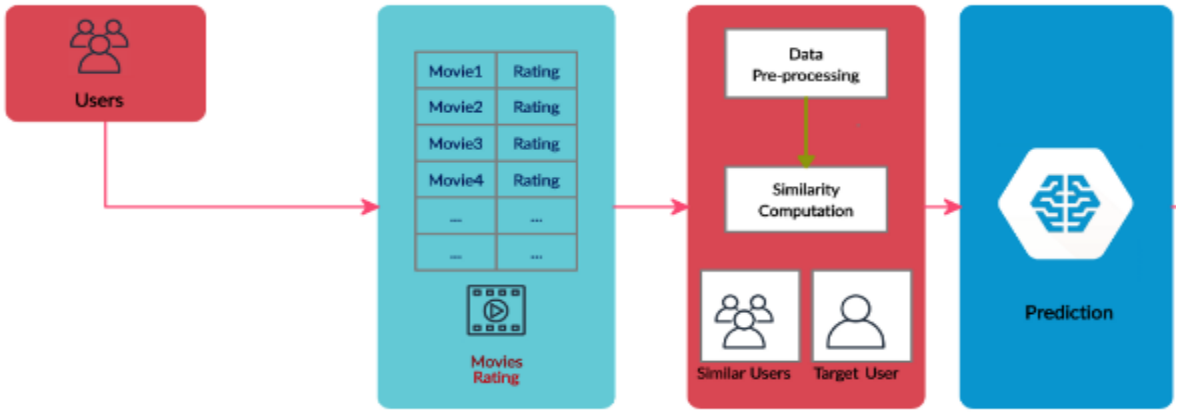


Figure 4

Workflow



The above picture clearly shows the workflow of this project.

1. There are users who provide the rating to the movies with unique Ids.
2. All these ratings by the users are stored in the form of tables with userId and movieId and corresponding rating along with the timestamp of the rating recorded.
3. We use this data - preprocess it and use a similarity metric to find the similarities between the items as we are focusing on item-based filtering.
4. After this is done we use the similarities developed by our algorithm in this case SVD++ and predict the rating of a movie by a user which is not present in the dataset.

Tools used (refs and goal)

For this project, we have mainly used three python packages These are listed below:

1. Numpy
2. Pandas
3. Pickle
4. Surprise

Numpy: This library is mainly used for performing operations involving arrays. In this project, this library is mainly used while experimenting with SVD in the numpy package. Basically, we have used the SVD from the surprise package and we wanted to compare its performance to that with the Numpy. But for reasons this was not implemented and as we have a lot to do with the package surprise

Pandas: This package is used for file handling or to read the datasets which we will use to perform more operations.

Pickle: This package is mainly used to save the ML models that were trained. We choose to save these models because if we want to run those models to train again, some of the models will run up to 4 hrs which is a very long time. So, to make use of the already trained models we are saving them using pickle.

Surprise: This is the heart of this project since all the models that we have developed in this project are using the methods present in this package. We mainly tried to exploit the matrix factoring model named SVD and SVD++ which is an extended version of the former. In addition to these libraries, out of curiosity we have experimented with other prediction algorithms like KNNBasic, BaseLineOnly and a few others which will be explained later in this document.

Algorithms implemented

Collaborative filtering:

As we all know collaborative filtering basically finds the similarity between the items and give out the most similar item. The two most common types of collaborative filtering are

1. **User-based filtering**:these systems recommend products to a user that similar users have liked. For measuring the similarity between two users we can either use pearson correlation or cosine similarity. As you can see blow as example each row represent users and each column is movie and each cell represents rating user gives to cell

In above we see for finding similarity for user E we need to use user B,C,D because A, F does not share rating common with E

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You	Similarity(i, E)
A	2		2	4	5		NA
B	5		4			1	
C			5		2		
D		1		5		4	
E			4			2	1
F	4	5		1			NA

Figure 1

The below table shows D and E are different because their similarity is -1. From this part we can rate movies that were not rated by user E using similarities from other users(figure 3).One main issue is that users' preference can change over time.

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You	Similarity(i, E)
A	2		2	4	5		NA
B	5		4			1	0.87
C			5		2		1
D		1		5		4	-1
E			4			2	1
F	4	5		1			NA

Figure 2

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You	Similarity(i, E)
A	2		2	4	5		NA
B	5		4			1	0.87
C			5		2		1
D		1		5		4	-1
E	3.51*	3.81*	4	2.42*	2.48*	2	1
F	4	5		1			NA

Figure 3

2. **Item-based filtering**: Item-based CF recommends items based on their similarity with the items that the target user rated. Likewise, the similarity can be computed with Pearson Correlation or Cosine Similarity. In item based recommendation we will fill the blanks vertically and not horizontally like user based. It solves the problem posed by dynamic user preference because it is more static however this method has scalability and sparsity issues.

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You
A	2		2	4	5	2.94*
B	5		4			1
C			5		2	2.48*
D		1		5		4
E			4			2
F	4	5		1		1.12*
Similarity	-1	-1	0.86	1	1	

Figure 5

- **Singular Value Decomposition:**

As mentioned earlier for this project we are mainly focusing on the performance of SVD present in the surprise package. The famous SVD algorithm was popularized by Simon Funk during the Netflix Prize. In linear algebra, the singular value decomposition (SVD) is a factorization of a real or complex matrix.

To be specific, Singular Value Decomposition of an mxn complex matrix M is a factorization of the form $M = U\Sigma V^T$,

Where U is mxm complex unitary matrix

Σ is mxn rectangular diagonal matrix

V^T is conjugate transpose of V

The singular value decomposition can be explained in another way as follows.

A non-negative real number σ is a singular value of the matrix M if and only if there exist unit-length vectors u in K^m and v in K^n such that,

$Mv = \sigma u$ and $Mu = \sigma v$

In any singular value decomposition, $M = U\Sigma V^T$

The algorithm that is coming from the surprise package has been built upon referencing this paper. The prediction equation used by this library is

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

In the above equation μ is global average, b_u is item bias, b_i is user bias, and $q_i^T p_u$ is useritem interaction.

If the user is unknown then the bias b_u and factors p_u are assumed to be zero. The same applies for i with b_i and q_i
To estimate all the unknowns, we minimize the regularized squared error:

$$\sum_{r_{ui} \in R_{train}} \left(r_{ui} - \hat{r}_{ui}\right)^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2\right)$$

The learning algorithm used for this model is *Stochastic Gradient Descent (SGD).*
For more information on this algorithm please refer to this doc.

SVD++
The SVD++ is an extended version of the SVD. The main addition in this is taking into account the implicit ratings. The prediction is calculated as

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j\right)$$

Where the y_j terms are a new set of item factors that capture implicit ratings. Here, an implicit rating describes the fact that a user u rated an item j, regardless of the rating value.

In SVD++ we do the same thing if the user is unknown as we did in SVD i.e., bias b_u and factors p_u are assumed to be zero. The same applies for i with b_i and q_i

Evaluation method(s)
The metrics that are used to measure the performance of the models are RMSE and MAE.

- **Root Mean Squared Error (RMSE)**

The formula to calculate this type of error is as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

There are no outliers in the dataset that we used so it is safe to use this measure as RMSE is prone to be affected by outliers.

- **Mean Absolute Error (MAE)**

The formula to calculate the MAE is as follows:

$$MAE = \frac{1}{N}\sum |predicted - actual|$$

Where N is the total number of records present in the dataset.

List of models developed

We have mainly focused on working with SVD and SVD++ algorithms and comparing their performance. For this we have developed multiple prediction models.

1. Firstly, we have developed direct models of SVD and SVD++ without any hyperparameter tuning.
2. Next, we have hyper parameters tuned i.e., using **GridSearchCV** provided by surprise we have passed a paramgrid of hyper parameters and developed a grid for each SVD and SVD++ algorithms.
3. After that, using the best performing parameters which are obtained by using RMSE and MAE and evaluation metrics we have developed the best SVD and best SVD++ algorithms.

In addition to this, out of curiosity and interest we tried to explore other prediction algorithms present in the surprise package. These algorithms are listed below.

- **KNNBasic**
- **BaseLineOnly**
- **NMF**
- **KNNWithMeans**

Results

Due to the limited computing capacity, we have used a shortened version of the dataset which named as **ratings_small.csv** in the [TMDB 5000 Movie Dataset | Kaggle.]

Below are the results of the basic versions of the SVD and SVD++ using 5 fold cross validation.

Applying SVD

```
svd = SVD()
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8951  0.9034  0.8949  0.9067  0.8873  0.8975  0.0069
MAE (testset)     0.6909  0.6945  0.6880  0.6964  0.6828  0.6905  0.0048
Fit time          0.80    0.87    0.81    0.93    0.93    0.87    0.06
Test time         0.11    0.13    0.10    0.12    0.15    0.12    0.02
```

Applying SVD++

```
svdpp = SVDpp()
cross_validate(svdpp, data, measures=['RMSE', 'MAE'], cv=5, verbose=1)
```

```
Evaluating RMSE, MAE of algorithm SVDpp on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8826  0.8866  0.8946  0.8814  0.8864  0.8863  0.0046
MAE (testset)     0.6736  0.6802  0.6842  0.6764  0.6822  0.6793  0.0038
Fit time          45.19   46.28   48.22   49.75   49.18   47.73   1.73
Test time         5.54    10.16   5.81    6.13    6.76    6.88    1.69
```

So, using the basic versions we have obtained the below final result.

	RMSE	MAE	Fit time	Test time
SVD	0.8975	0.6905	0.87	0.12
SVD++	0.8863	0.6793	47.73	6.88

After this we have hyperparameters tuned and found the best performing model for SVD and SVD++ individually. Using this there is a slight change in the errors.
Below are the results of the best performing models.

SVD

```
best_svd = grid_search.best_estimator['rmse']
cross_validate(best_svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8915  0.8813  0.8885  0.8953  0.8945  0.8902  0.0051
MAE (testset)     0.6889  0.6793  0.6873  0.6909  0.6874  0.6868  0.0040
Fit time          0.48    0.45    0.47    0.48    0.47    0.47    0.01
Test time         0.11    0.11    0.23    0.11    0.22    0.16    0.06
```

SVD++

```
best_svdpp = grid_searchpp.best_estimator['rmse']
cross_validate(best_svdpp, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVDpp on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8822  0.8844  0.8873  0.8867  0.8872  0.8856  0.0020
MAE (testset)     0.6790  0.6829  0.6844  0.6838  0.6803  0.6821  0.0021
Fit time          88.35   94.99   95.84   100.50  101.25  96.19   4.63
Test time         5.71    5.79    5.95    6.38    6.25    6.02    0.26
```

The above result can be summarized as below.

	RMSE	MAE	Fit time	Test time
SVD	0.8902	0.6868	0.47	0.16
SVD++	0.8856	0.6821	96.19	6.02

The below table shows the predictions of all the models developed. As we can see SVD++ with hyper parameter tuning has performed or predicted the rating which is very close to the actual rating.

	SVD	Hyper parameter tuning	SVD++	Hyper parameter tuning
Actual	2.5	2.5	2.5	2.5
Predict	2.3445	2.4324	2.4407	**2.5076**

These are results with respect to the other algorithms that are explored in the surprise package. So among all the models available SVD++ is the best performing model.

Algorithm	**RMSE**	**MAE**
SVDpp	**0.8954**	**0.6873**
BaselineOnly	0.8962	0.6932
SVD	0.9018	0.6951
KNNWithMeans	0.9291	0.7103
NMF	0.9606	0.7378
KNNBasic	0.9795	0.7533

More information can be found related to these models in this document. The git repository containing all the code related to all the models developed can be accessed here.

Lessons learned

1. This project helped the entire team to explore a new package called surprise. There are many popular packages which implement the SVD but we have chosen the surprise package to work.
2. In addition to the technical knowledge, the team has also learned how to work in a team or in other words how important it is to have team collaboration to drive the project to the end.
3. Work distribution was also important and the team has learned how to effectively split the work among all the team members.
4. Another important thing is that upon completion of this project,we realized that there is a lot to explore in this field and out of curiosity we have also explored other prediction models in the surprise package.

Conclusion

In conclusion, we have evaluated results by implementing various models such as SVD and SVD++ along with hyper parameter tuning to find the best predictor among them.

Open issues/future work

- SVD is a very popular technique and can be found in other libraries as well. So we can use the SVD present in other libraries and perform the same operations and can compare the results obtained by other SVDs
- The input used in this project is a dense matrix. This can be converted into a sparse matrix and perform the predictions and observe the results obtained.

Systems requirements to install and run the project:

1. **Install python**

You can install the latest python version from here.

2. **Installing jupyter**

Installation for python and Jupyter notebook is easy. By going to website below Download Python | Python.org you can install python software on your compute. Also by Following this website Project Jupyter | Installing Jupyter you can install Jupyter notebook That fits your system. Please make sure to install pip module on your command line as well by entering the command “python get-pip.py”.

3. **Installing libraries**

In your jupyter notebook please type the below commands to install the libraries.

- pip install surprise
- pip install pandas
- pip install numpy
- pip install pickle

Github link with the code and sample collection:

- **Code repository:** CMPE297_Special_topic-project/Project-RecommendationSystem.ipynb at main · Alireza-Mahinparvar/CMPE297_Special_topic-project
- **Dataset:** https://www.kaggle.com/code/erikbruin/movie-recommendation-systems-for-tmdb/data