**FINAL REPORT**

FOR

**MINOR PROJECT**

**ON**

**Movie Recommendation Systems**

**BY**

**MIHIR AGARWAL**

**(03016404518)**

**MASTER OF COMPUTER APPLICATION**

**(Software Engineering)**

**ABSTRACT**

Recommender systems research has incorporated a wide variety of artificial intelligence techniques including machine learning, data mining, user modeling, case-based reasoning, and constraint satisfaction, among others. Recommender systems are tools for interacting with large and complex information spaces. They provide a personalized view of such spaces, prioritizing items likely to be of interest to the user. In most general terms, Recommendation systems are defined as the techniques used to predict the rating one individual will give to an item or social entity. These items can be books, movies, restaurants and things on which individuals have different preferences. These preferences are being predicted using two approaches first content-based approach which involves characteristics of an item and second collaborative filtering approaches which takes into account user's past behavior to make choices. In collaborative filtering, partners are chosen who will make recommendations because they share similar ratings history with the target user. One partner who have similar ratings to the target user may not be a reliable predictor for a particular item. So the past record of the partner of making a reliable recommendation also needs to be take into consideration which is dictated by trustworthiness of a partner. In order to keep track of past records of a recommender reputation systems comes into the picture those who actually assign reputation ratings to the partners.

**INTRODUCTION**

In today’s world, every customer is faced with multiple choices. For example, If I’m looking for a specific item without any specific idea of what I want, there’s a wide range of possibilities. I might waste a lot of time browsing around on the internet and trawling through various sites hoping to strike gold. I might look for recommendations from other people. But if there was a site or app which could recommend me based on what my preferences are, that would be a massive help. Instead of wasting time on various sites, I could just get 10 recommendations tailored to my taste. This is what recommendation engines do and their power is being harnessed by most businesses these days. From Amazon to Netflix, Google to Goodreads, recommendation engines are one of the most widely used applications of [Machine Learning techniques](https://courses.analyticsvidhya.com/courses/introduction-to-data-science-2?utm_source=blog&utm_medium=RecommendationEnginesfromScratcharticle).

**A recommendation engine filters the data using different algorithms and recommends the most relevant items to users. It first captures the past behaviour of a customer and based on that, recommends products which the users might be likely to buy.[1]**

**PROBLEM STATEMENT**

This project is based on recommendation system that recommends different movies to the users. This project aims to calculate the similarities between different users and then recommend movie to them as per the ratings given by the different users of similar tastes. This will provide a precise recommendation to the user. This project is based on providing recommendations based on the similarity of the movies and the similarities in the user preferences.

The first thing we would need to collect is the preferences of the different people. These preferences will possibly be the ratings (as in case of a Movie Recommender System) of different users who have rated some movies in the past. The ratings will be on a scale of 1 to 5 where 1 depicts strong dislike and 5 suggest a strong liking for the movie. A rating of 3-4 would mean average opinion on that movie. After the data has been collected, a technique has to be devised to present similarities between different users and generate a better prediction for new users.[1][2]

**HARDWARE REQUIREMENTS**

1. **CPU**

Core i3, Core i5, and Core i7 systems are all suitable for a Machine Learning and Data Science projects but Core **i7**systems are more preferable as they are more **powerful** with high **performance**.

1. **STORAGE**

It is recommended to go for a System which has both SSD & HDD storage. However you shouldn’t go for a System with just HDD storage as a lot of issues may arise by going for a HDD storage for running projects with high datasets. It requires minimum of 256 GB storage but it is preferable to have 1 TB or more storage capacity for efficient running of projects.

1. **OS**

For more steady and smooth execution of machine learning projects pick **MacOS** (MacBook Air or MacBook Pro) else you can also use Windows **OS to run the projects. I**t all depends on your choice.

1. **GPU**

This is the most important aspect for running Machine Learning projects. GPU enables parallel processing. Without GPU the process might take lot of time for execution. But with it, the System can perform the same task in much lesser time. **NVIDIA**  [GeForce 10 series](https://www.nvidia.com/en-us/geforce/products/) & the [RTX 20 Series](https://www.nvidia.com/en-us/geforce/gaming-laptops/20-series/) are the best for the systems. You can also go for [**AMD Radeon**](https://www.amd.com/en/graphics/radeon-rx-graphics).

1. **RAM**

It would be more advantageous and considerably speedier to keep huge data sets. 16 GB is extremely my primary suggestion if large applications else 8 GB is also fine.

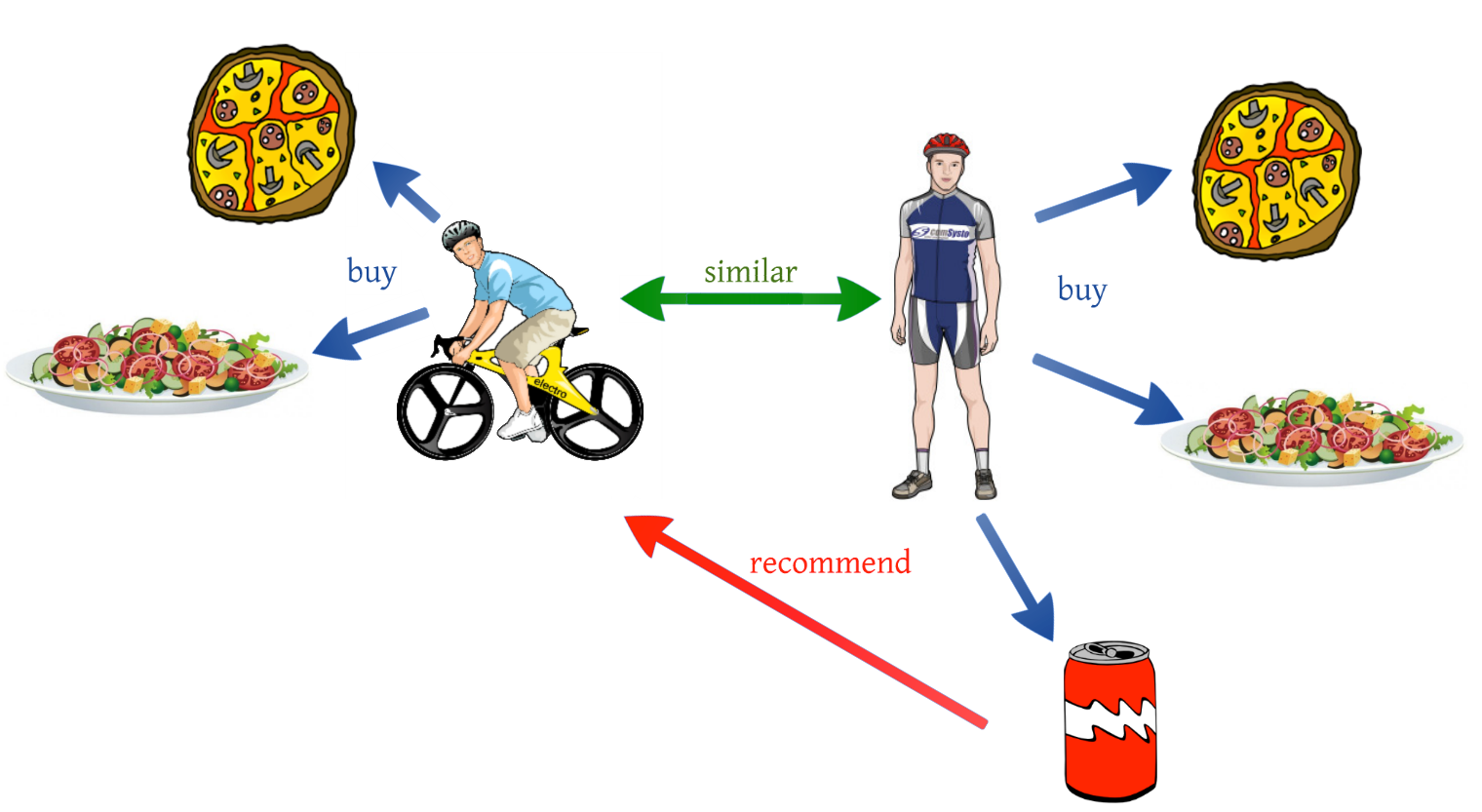
**SOFTWARE REQUIREMENTS**

1. **Python** - Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.
2. **Jupyter Notebook** – Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

**DATASET**

MovieLens 1M movie ratings. Stable benchmark dataset. 1 million ratings from 6000 users on 4000 movies.Each user has more than 20 ratings. The ratings for each movie are from 1 to 5. It also contains features of various users and movies.This dataset is randomly divided into 2 parts: the training set and the test set. For each user, the training set contains 90% of the user’s ratings. The rest 10% ratings build up the test set. Collaborative filtering is trained based on the training set and algorithm evaluation is carried out based on the test set. [3]

# **COLLABORATIVE FILTERING RECOMMENDER SYSTEM**



**Figure 1 Collaborative Filtering (source : Internet)**

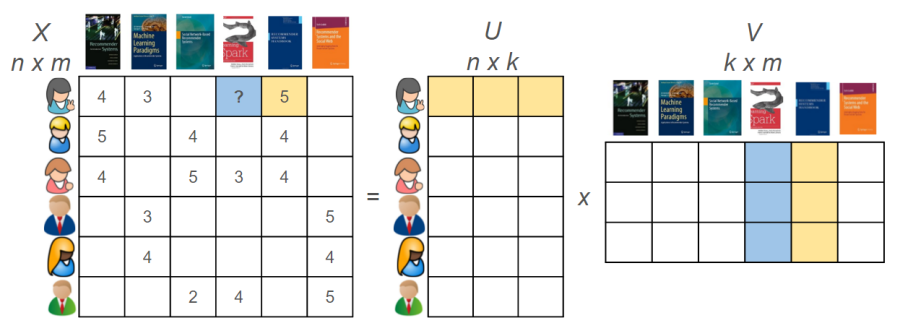
**Collaborative filtering (CF) is a technique used by**[**recommender systems**](https://en.wikipedia.org/wiki/Recommender_system)**.**

**Collaborative filtering is a method of making automatic**[**predictions**](https://en.wikipedia.org/wiki/Prediction)**(filtering) about the interests of a**[**user**](https://en.wikipedia.org/wiki/End_user)**by collecting preferences or**[**taste**](https://en.wikipedia.org/wiki/Taste_(sociology))**information from**[**many users**](https://en.wikipedia.org/wiki/Crowdsourcing)**(collaborating). The underlying assumption of the collaborative filtering approach is that if a person *A* has the same opinion as a person *B* on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person.[4]**

# **Matrix Factorization**

# Matrix factorization is used to factorize a matrix, i.e. to find out two (or more) matrices such that when you multiply them, you’ll get back the original matrix.

Matrix factorization can be used to discover features underlying the interactions between two different kinds of entities. And one obvious application is to predict ratings in collaborative filtering—in other words, to recommend items to users.



**Figure 2 Matrix Factorization (source : Internet)**

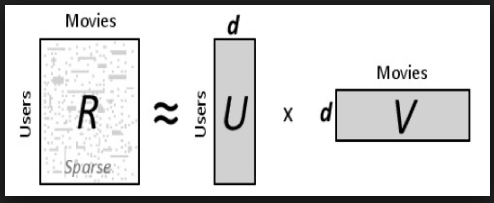
**Why perform matrix factorization?**

One advantage of employing matrix factorization for recommender systems is the fact that it can incorporate implicit feedback—information that’s not directly given but can be derived by analyzing user behavior—such as items frequently bought or viewed.

Using this capability we can estimate if a user is going to like a movie that they never saw. And if that estimated rating is high, we can recommend that movie to the user, so as to provide a more personalized experience.

For example, two users might give high ratings to a certain movie if they both like the actors/actresses of the movie, or if the movie is a thriller movie, which is a genre preferred by both users.

Hence, if we can discover these kinds of latent features (like genre or actors and directors), we should be able to predict a rating with respect to a certain user and a certain item, because the features associated with the user should match with the features associated with the item.



**Figure 3 Decomposition of Matrix into User and Movies matrix (source : Internet)**

**The mathematics of matrix factorization**

Let’s quickly take a look at the mathematics behind matrix factorization. Firstly, we have a set U of users, and a set D of items. Let R of size |U|\*|D| be the matrix that contains all the ratings that the users have assigned to the items. Also, we can assume that we’d like to discover |K| latent features. Our task, then, is to find two matrices P=|U|\*K and Q=|D|\*K such that their product approximates R.



In this way, each row of P would represent the strength of the associations between a user and the features. Similarly, each row of Q would represent the strength of the associations between an item and the features. To get the prediction of a rating of an item dj, we can calculate the dot product of the two vectors:



Now we have to find a way to obtain P and Q. One way to approach this problem is to first initialize the two matrices with some values, calculate how ‘different’ their product is to M, and then try to minimize this difference iteratively. Such a method is called **gradient descent**, which is aimed at finding a local minimum of the difference.

The difference here, usually called the error between the estimated rating and the real rating, can be calculated with the following equation for each user-item pair:

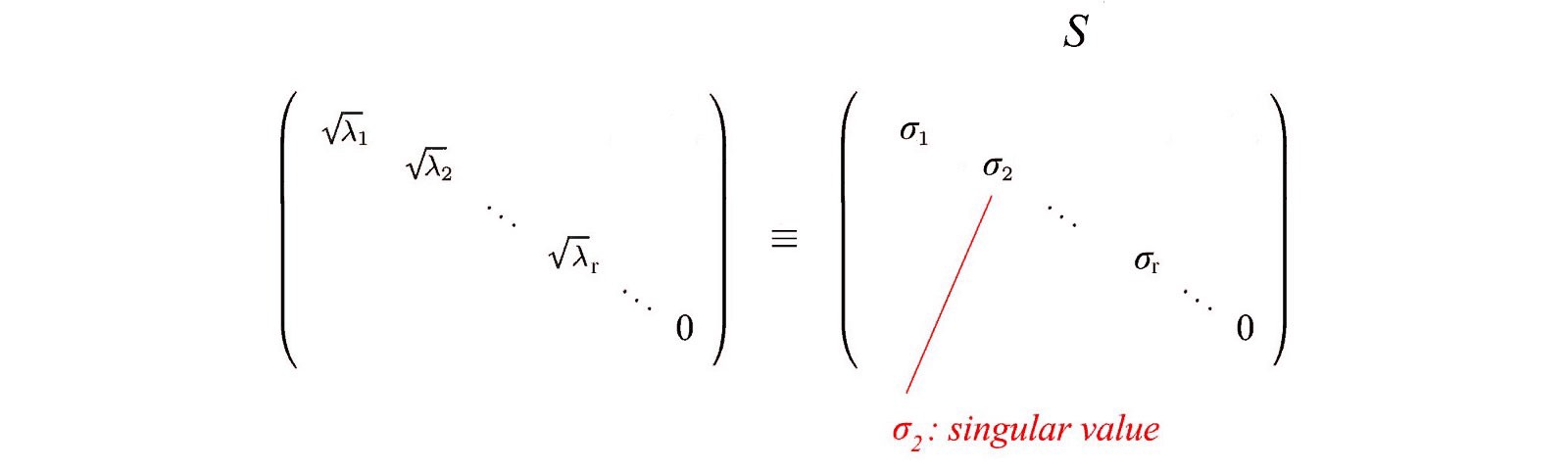
[5]

# **Singular Value Decomposition**

SVD states that **any** matrix *A* can be factorized as:



where *U* and *V* are orthogonal matrices with orthonormal eigenvectors chosen from ***AAᵀ***and ***AᵀA***,respectively. ***S*** is a diagonal matrix with *r* elements equal to the root of the positive eigenvalues of *AAᵀ* or *Aᵀ A*(both matrices have the same positive eigenvalues anyway). The diagonal elements are composed of singular values.



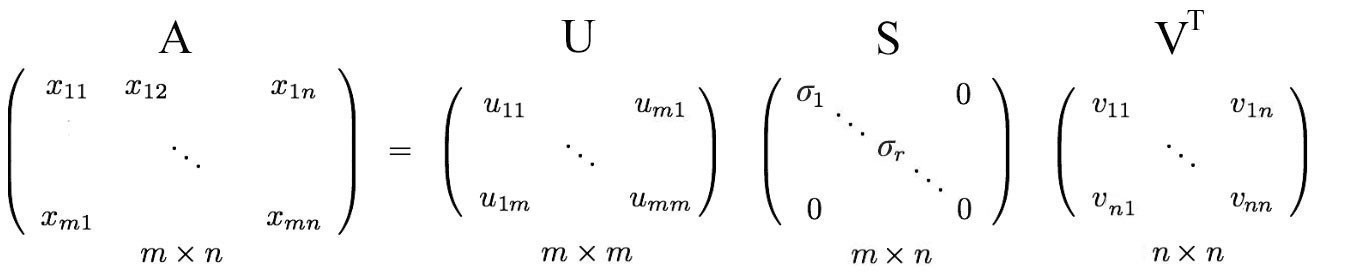
**Figure 4 Sigma Matrix (source : Internet)**

i.e. an m× n matrix can be factorized as:









**Figure 5 Decomposition of Matrix into Sub matrices (source : Internet)**

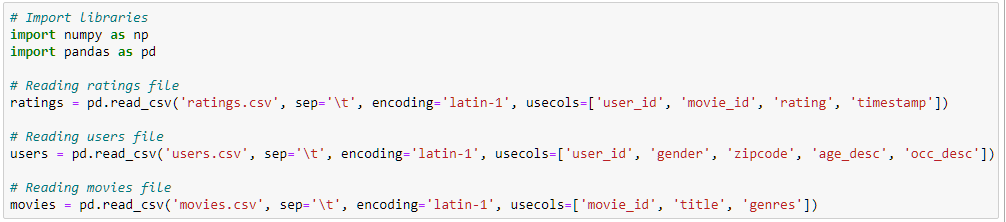
We can arrange eigenvectors in different orders to produce *U* and *V*.SVD helps in giving eigenvectors of the input matrix. The technique is used generally where eigenvectors are of interest to us. PCA (Principal Component Analysis) is one classic example.

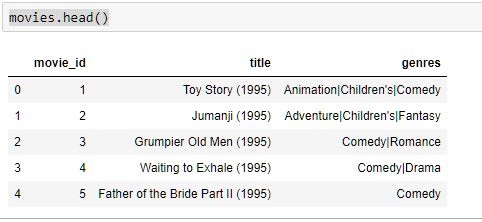
In the case of SVD, it doesn’t assume anything about missing values. So you need to give some missing value imputation for SVD. This might bring in unnecessary noise. But if your ratings matrix is not too sparse, SVD might produce better results.[6]

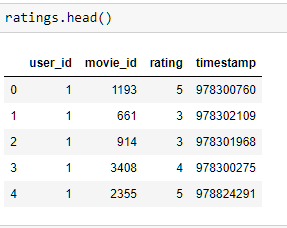
**BUILDING THE RECOMMENDER SYSTEM USING MATRIX FACTORIZATION(SVD)**

**Loading the Dataset**

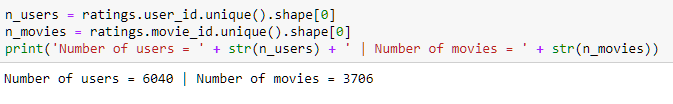
Loading three datasets ‘movies.csv’,’ratings.csv’ & ‘users.csv’







**Count the number of unique users and movies**



We will format the ratings matrix to be one row per user and one column per movie. To do so, We will pivot ratings to get that and call the new variable Ratings

## 

## We will de-normalize the data (normalize by each users mean) and convert it from a dataframe to a numpy array.

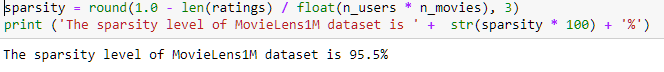
## 

## With ratings matrix properly formatted and normalized, We are ready to do some dimensionality reduction

## Model-Based Collaborative Filtering

Model-based Collaborative Filtering is based on matrix factorization (MF) which has received greater exposure, mainly as an unsupervised learning method for latent variable decomposition and dimensionality reduction. Matrix factorization is widely used for recommender systems where it can deal better with scalability and sparsity than Memory-based CF:

* The goal of MF is to learn the latent preferences of users and the latent attributes of items from known ratings (learn features that describe the characteristics of ratings) to then predict the unknown ratings through the dot product of the latent features of users and items.
* When you have a very sparse matrix, with a lot of dimensions, by doing matrix factorization, you can restructure the user-item matrix into low-rank structure, and you can represent the matrix by the multiplication of two low-rank matrices, where the rows contain the latent vector.
* You fit this matrix to approximate your original matrix, as closely as possible, by multiplying the low-rank matrices together, which fills in the entries missing in the original matrix.



## Support Vector Decomposition (SVD)

A well-known matrix factorization method is Singular value decomposition (SVD). At a high level, SVD is an algorithm that decomposes a matrix A into the best lower rank (i.e. smaller/simpler) approximation of the original matrix A. Mathematically, it decomposes A into a two unitary matrices and a diagonal matrix:

Where A is the input data matrix (users's ratings), U is the left singular vectors (user "features" matrix), Σ is the diagonal matrix of singular values (essentially weights/strengths of each concept), and VT is the right singluar vectors (movie "features" matrix). U and VT  are column orthonormal, and represent different things. U represents how much users "like" each feature and VT represents how relevant each feature is to each movie.

To get the lower rank approximation, We take these matrices and keep only the top k features, which can be thought of as the underlying tastes and preferences vectors.

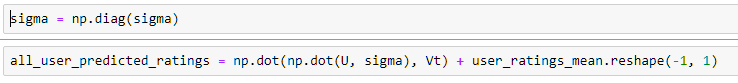
### **Setting Up SVD**

Scipy and Numpy both have functions to do the singular value decomposition. We are going to use the Scipy function svds because it let choose how many latent factors we want to use to approximate the original ratings matrix (instead of having to truncate it after)

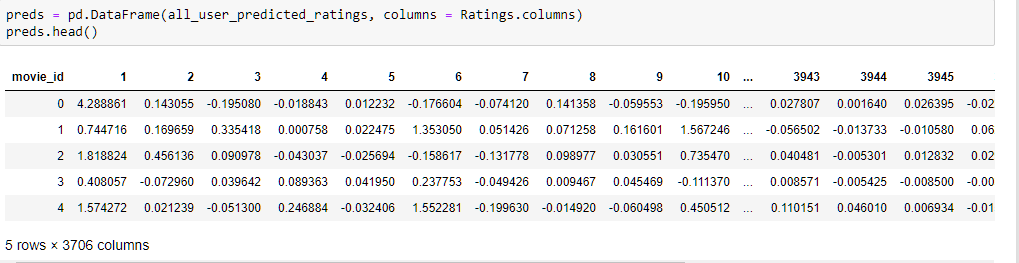


### **Making Predictions from the Decomposed Matrices**

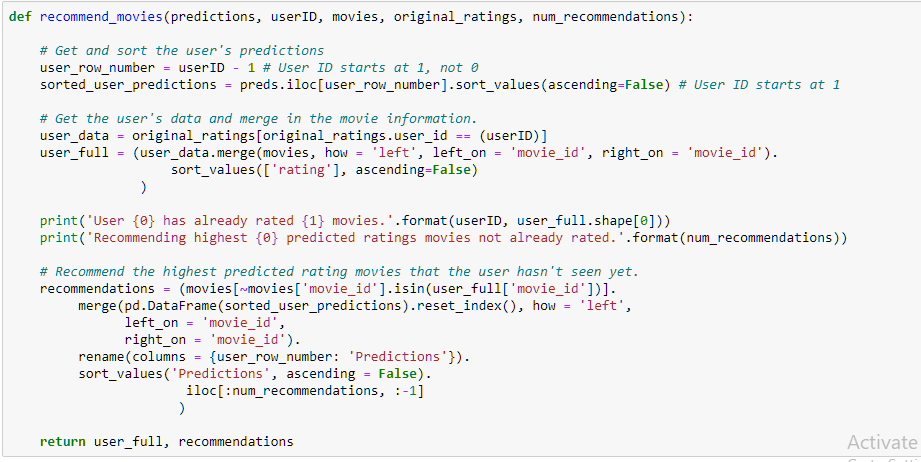
We will now make movie ratings predictions for every user. We can do it all at once by following the math and matrix multiply U, Σ, and VT back to get the rank k=50 approximation of A.



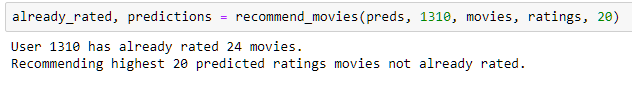
With the predictions matrix for every user, We can build a function to recommend movies for any user.



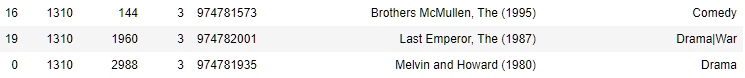
Now We write a function to return the movies with the highest predicted rating that the specified user hasn't already rated. Though We didn't use any explicit movie content features (such as genre or title), We'll merge in that information to get a more complete picture of the recommendations.



Let's try to recommend 20 movies for user with ID 1310.







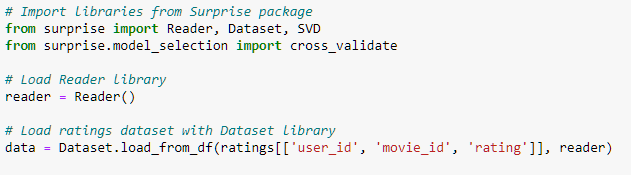


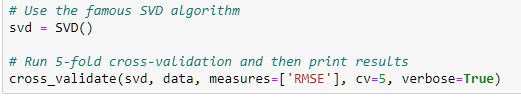
 

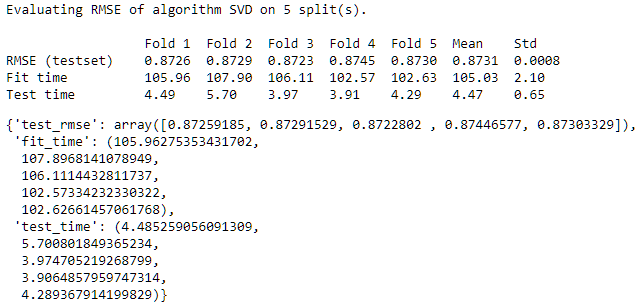
These look like pretty good recommendations. It's good that although we didn't actually use the genre of the movie as a feature, the truncated matrix factorization features "picked up" on the underlying tastes and preferences of the user. We've recommended some comedy, drama, and romance movies - all of which were genres of some of this user's top rated movies.

### **Model Evaluation**

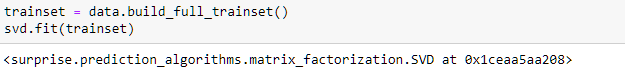
We will use the “[**Surprise**](https://pypi.python.org/pypi/scikit-surprise)” library that provides various ready-to-use powerful prediction algorithms including (SVD) to evaluate its RMSE (Root Mean Squared Error) on the MovieLens dataset. It is a Python scikit building and analyzing recommender systems.







We get a mean (RMSE)of 0.8731 which is pretty good. We will now train on the dataset and arrive at predictions.

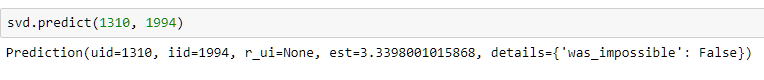


We'll pick again user with ID 1310 and check the ratings he has given.



Now we use SVD to predict the rating that User with ID 1310 will give to a random movie (let's say with Movie ID 1994).



For movie with ID 1994, We get an estimated prediction of **3.349**. The recommender system works purely on the basis of an assigned movie ID and tries to predict ratings based on how the other users have predicted the movie.

# **Issues with SVD-based Collaborative Filtering**

## Data sparsity

One typical problem caused by the data sparsity is the [cold start](https://en.wikipedia.org/wiki/Cold_start_(computing)) problem. As collaborative filtering methods recommend items based on users’ past preferences, new users will need to rate a sufficient number of items to enable the system to capture their preferences accurately, and thus provides reliable recommendations. Similarly, new items also have the same problem.

## Gray sheep

Gray sheep refers to the users whose opinions do not consistently agree or disagree with any group of people, and thus do not benefit from collaborative filtering.

## Scalability

As the numbers of users and items grow, traditional CF algorithms will suffer serious scalability problems. For example, with tens of millions of customers and millions of items, a CF algorithm with the complexity of O(n)is already too large.

## Synonyms

[Synonyms](https://en.wikipedia.org/wiki/Synonyms) refers to the tendency of a number of the same or very similar items to have different names or entries. Most recommender systems are unable to discover this latent association and thus treat these products differently.[6][7]

## CONCLUSION

In this notebook, I have attempted to build a model-based Collaborative Filtering movie recommendation system based on latent features from a low rank matrix factorization method called SVD. As it captures the underlying features driving the raw data, it can scale significantly better to massive datasets as well as make better recommendations based on user's tastes.

However, we still likely lose some meaningful signals by using a low-rank approximation. Specifically, there's an interpretability problem as a singular vector specifies a linear combination of all input columns or rows. There's also a lack of sparsity when the singular vectors are quite dense. Thus, SVD approach is limited to linear projections.

**RESULT**

The following Movie Recommender System is capable of providing movie recommendations according to users preferences and movie features and is also able to predict the ratings that will be provided by the user for that particular movie.

In the above case we have predicted movies for user 1310 and also we have predicted the ratings given by user having user Id 1310 on a particular movie having movie Id 1994. Since the value of RMSE is small so our predictions are likely close to the actual results and our system can give right predictions to the particular user according to the user preferences and movie features.

**REFERENCES**

1. [**https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/**](https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/)
2. [**https://pdfs.semanticscholar.org/2d04/c6d8508d426e859072b9141007c88f759969.pdf**](https://pdfs.semanticscholar.org/2d04/c6d8508d426e859072b9141007c88f759969.pdf)
3. [**https://grouplens.org/datasets/movielens/10m/**](https://grouplens.org/datasets/movielens/10m/)
4. [**https://en.wikipedia.org/wiki/Collaborative\_filtering**](https://en.wikipedia.org/wiki/Collaborative_filtering)
5. [**https://heartbeat.fritz.ai/recommender-systems-with-python-part-iii-collaborative-filtering-singular-value-decomposition-5b5dcb3f242b**](https://heartbeat.fritz.ai/recommender-systems-with-python-part-iii-collaborative-filtering-singular-value-decomposition-5b5dcb3f242b)
6. [**https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning/**](https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning/)
7. [**https://towardsdatascience.com/understanding-singular-value-decomposition-and-its-application-in-data-science-388a54be95d**](https://towardsdatascience.com/understanding-singular-value-decomposition-and-its-application-in-data-science-388a54be95d)