**PROGRAMMING ASSIGNMENT 2**

**K-NEAREST NEIGHBORS AS RECOMMENDER SYSTEM**

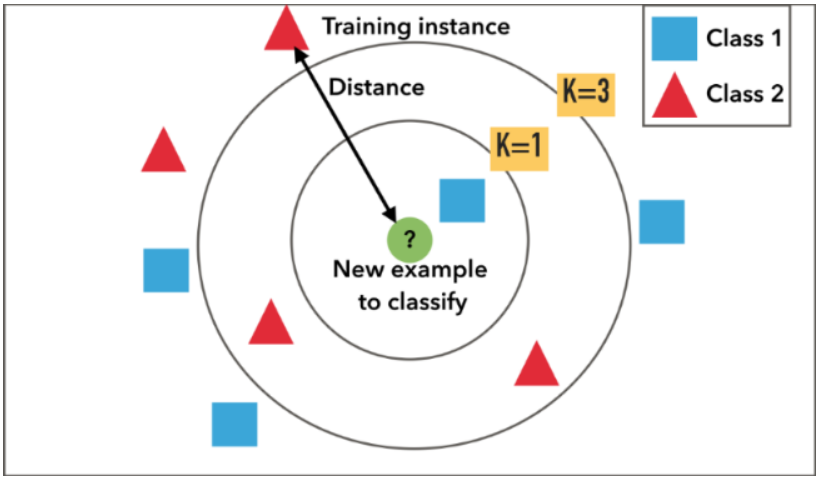
**Algorithm:**

**What is a Recommender System?**

Recommender system aims to predict the interests of the users and tries to recommend items that are quite likely to them.

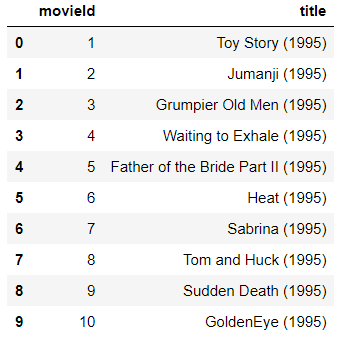
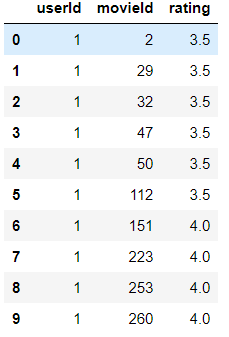
**KNN as Recommender System**

K-Nearest Neighbors is the perfect baseline model to implement the Recommendation system. KNN doesn’t make any assumptions on the underlying distribution of data, it solely depends on the similarity feature. When KNN makes an inference about a movie, it calculates similarity metric of the movie with all other movies in the dataset and sorts them according to their similarities and returns the Top K-Nearest Neighbors as the most similar movies.

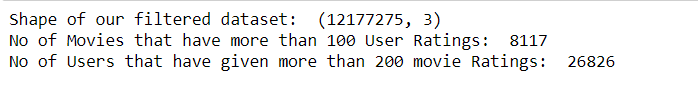


**Step by Step Implementation:**

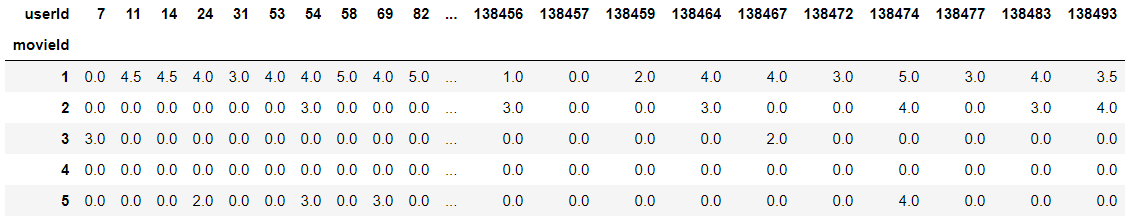
* Loading movies.csv and ratings.csv and converting them into data frames

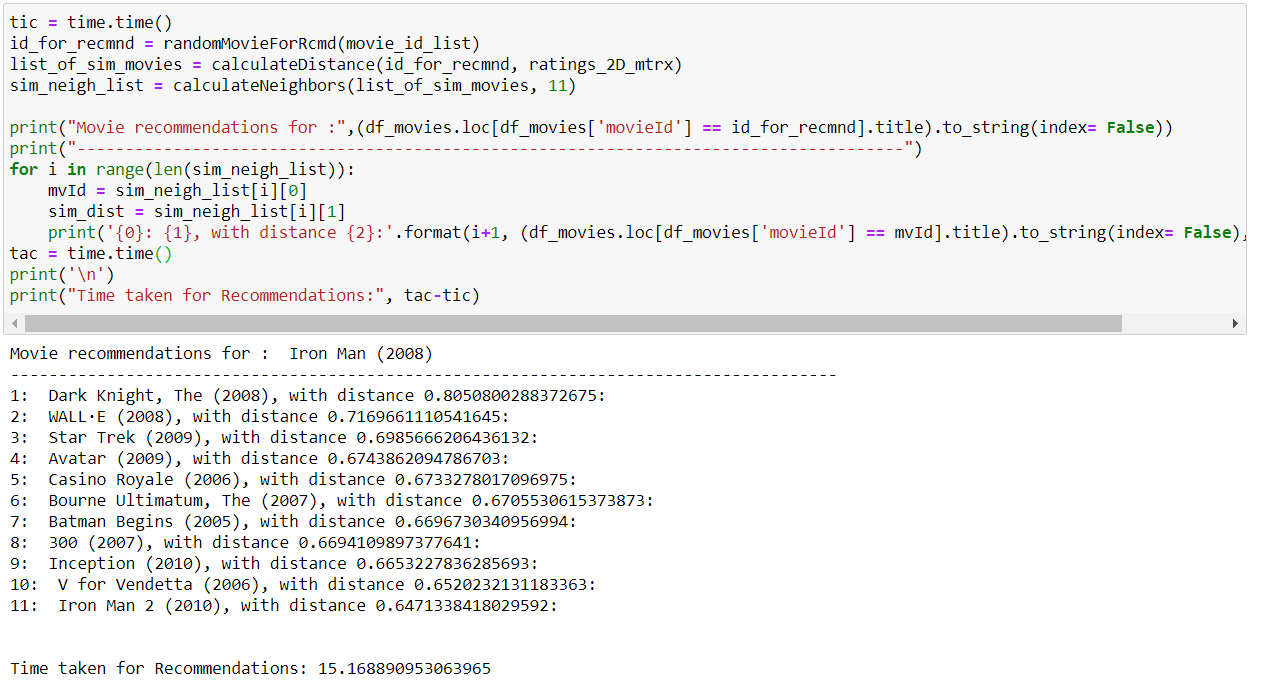
* For statistical significance removed users who rated less than 200 movies and removed movies that got rating less than 100. Final dataset size after filtering is



* Created the pivot table for the userId, movieId and ratings



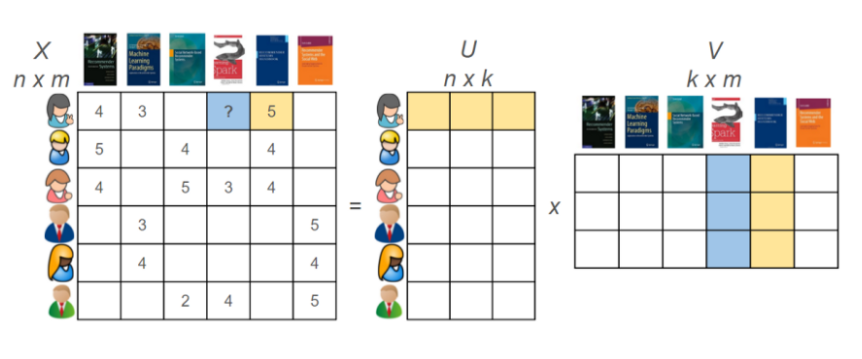
* Implemented K-Nearest Neighbors with distance metric as Cosine Similarity and displayed Movie Recommendations for Iron Man, calculated Cosine Similarity of each row with the target row.



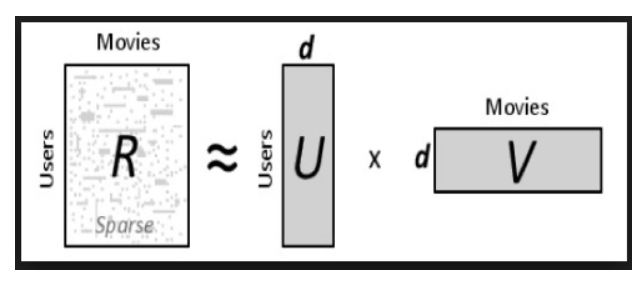
* Used distance metric as Cosine-Similarity because in higher dimensions the vectors are all most equidistant to the vector that needs recommendation, so Euclidean Distance will not be a great metric when it comes to higher dimensions.

**Matrix Factorization:**

Matrix Factorization means, factorizing a matrix in to two different matrices such that if we multiply those two matrices, we get the original matrix. It is used to find the underlying features of different entities. For example, using matrix factorization we can recommend an item to the user. The reason why we perform matrix factorization is it incorporates implicit feedback. It analyses the behavior of the user although there is no information regarding it. It derives its analysis based on the movies watched and rated high, or the products bought etc.,



In the matrix factorization technique, we discover latent features with the help of these latent features we will be able to predict the user ratings that are not available for certain products and based on those ratings we can provide Recommendations for the user on that product.



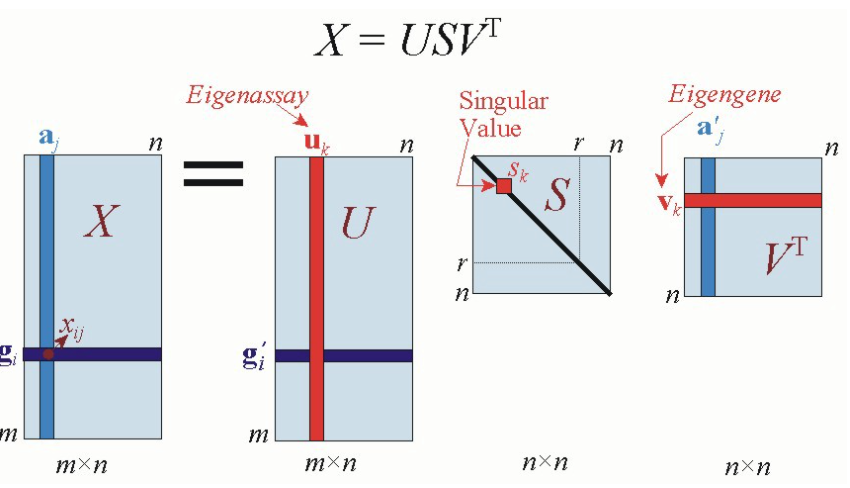
Below are different types of matrix factorization techniques:

* LU Matrix Decomposition
* QR Matrix Decomposition
* Cholesky Decomposition
* Singular Value Decomposition

For this Recommender System I used Singular Value Decomposition as Matrix Factorization Technique

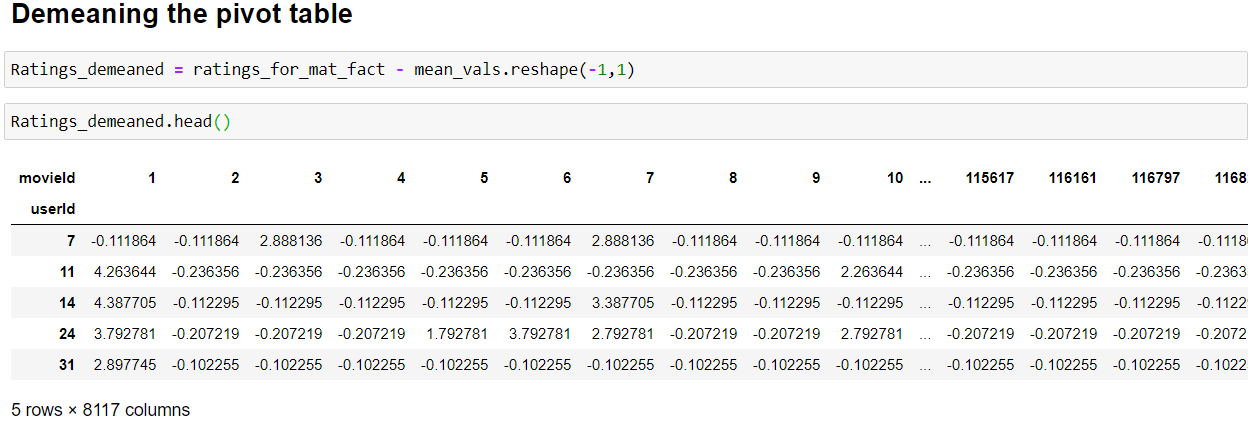
**Singular Value Decomposition (SVD):**

Basically, SVD states that any matrix can be represent in the form of *U S Vt* where U and V are orthogonal matrices and S is a diagonal matrix

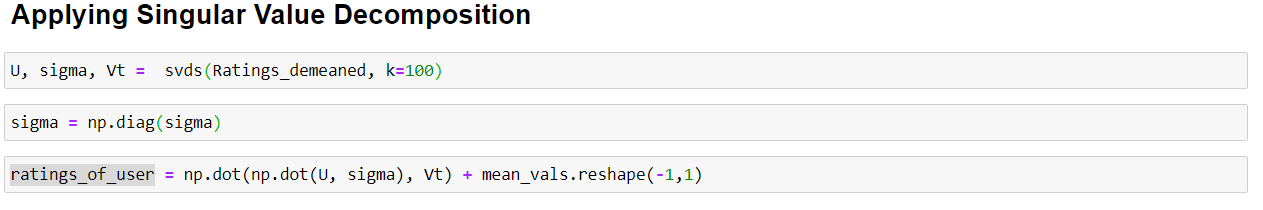


**Steps followed in Implementing Matrix Factorization:**

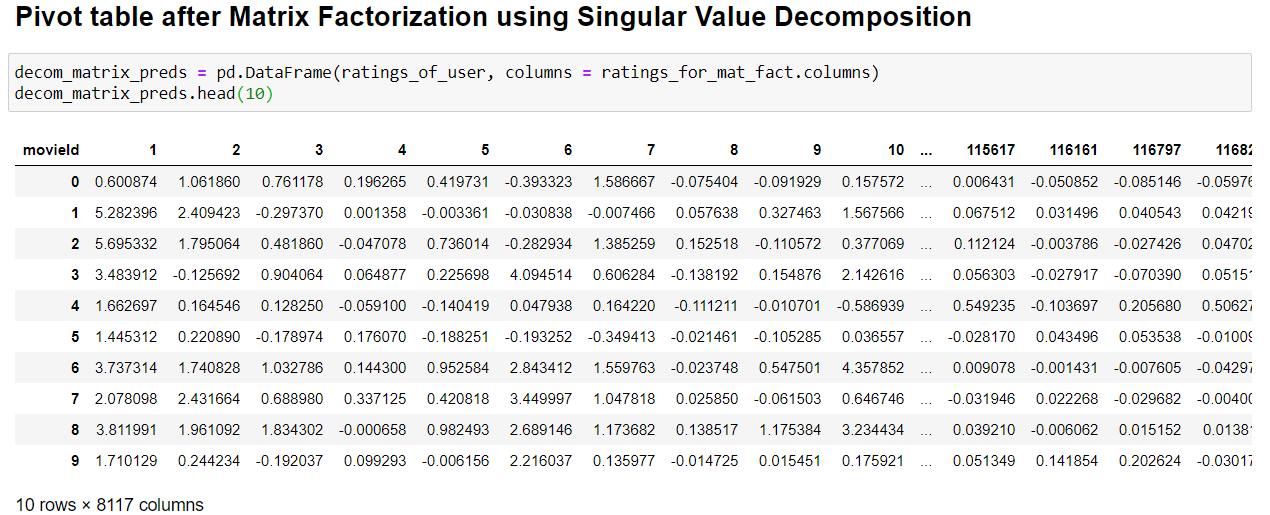
* Demeaned the pivot table with mean calculated for each row.



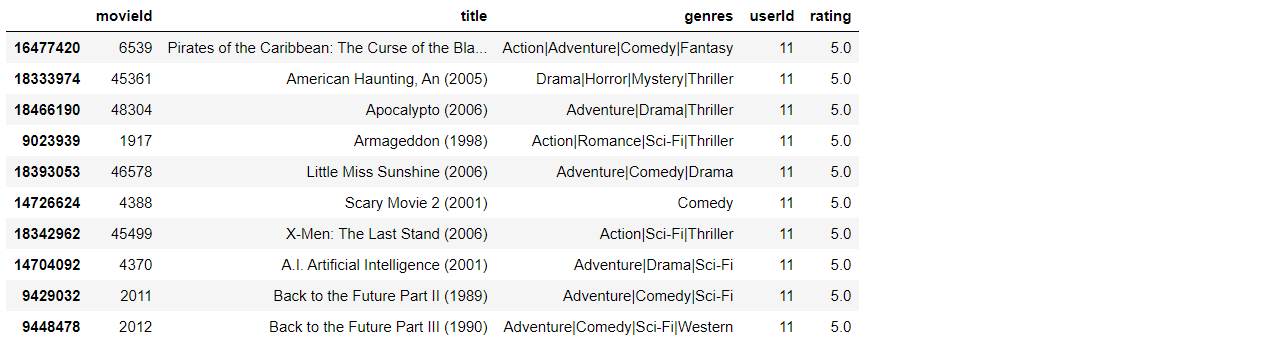
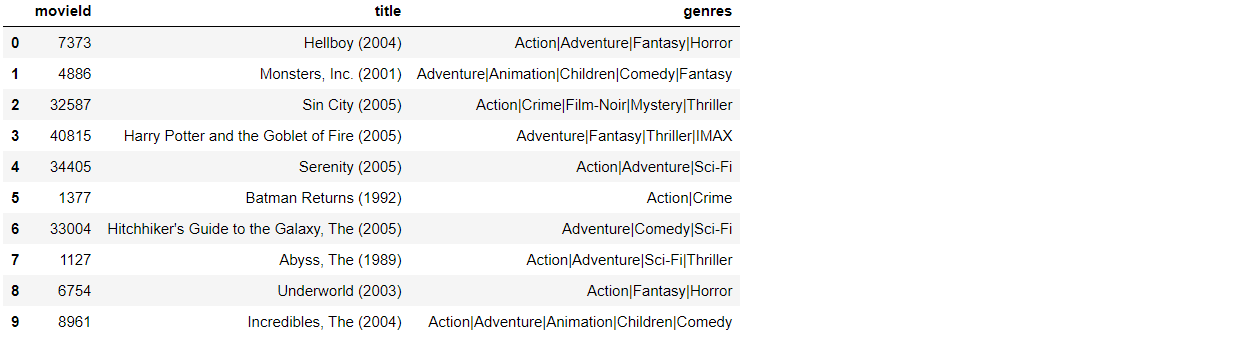
* Applied Singular Value Decomposition to the demeaned Data frame



* Decomposed predictions matrix



* recommendByMatrixFact method recommends based on the userId. If we see the highest rated movies of the user and their genres, if we compare them with predictions that we made from the decomposition matrix, they look similar

**Users top rated movies and their genres**  
**Recommendation of other movies for that User**

* We can observe that recommended movies to that specific user have similar kind of genres that he has given highest rating to.

**Maximum Dataset that my Recommender System can use:**

Maximum Dataset that my Recommender System can work with is 30k users and 9k movies that is (30000 X 9000) rating values

**Time Complexity of my Recommender System:**

Though my Recommender System operates in Constant Time as the matrix is a sparse matrix and most of the data are zeros, it takes more time while performing multiplication and addition operations on those zeroes which are trivial. This problem increases time complexity as the matrices get larger and larger. And in this case my Recommender System takes 25 seconds to give recommendations on an average.

**Performance of my Recommender System:**

My Recommendation System when given an UserId, recommends movies similar to the genres user likes. Though the Recommendation System did not use the genre of the movie as feature, the Matrix Factorization picked up on the underlying tastes and preferences. So, from this we can say that its performance is decent. But if we add a new movie to the dataset it will have a cold start as it will have less user ratings, so it is considered as trivial.

**Ways to scale up Recommender System to work with Large Datasets:**

If we can handle the sparse matrix data in an efficient way, then we can scale up the Recommender system to work with large datasets. As the sparse matrices are less significant and occupy more memory, so computations on such type of data will be time and memory consuming, handling them would increase the scope of working with large datasets.

**References:**

* <https://www.kdnuggets.com/2019/06/select-rows-columns-pandas.html>
* <https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/>
* <https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea>
* <https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1>
* <https://towardsdatascience.com/how-did-we-build-book-recommender-systems-in-an-hour-part-2-k-nearest-neighbors-and-matrix-c04b3c2ef55c>
* <https://towardsdatascience.com/how-did-we-build-book-recommender-systems-in-an-hour-the-fundamentals-dfee054f978e>
* <https://machinelearningmastery.com/sparse-matrices-for-machine-learning/>
* <https://beckernick.github.io/matrix-factorization-recommender/>
* <https://heartbeat.fritz.ai/recommender-systems-with-python-part-iii-collaborative-filtering-singular-value-decomposition-5b5dcb3f242b>
* <https://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/>
* <https://tryolabs.com/blog/introduction-to-recommender-systems/>
* <https://www.eecs.tufts.edu/~mhughes/courses/intro_ml/2019s/proj3.html>