Capstone Project 1

Movie Recommendation System

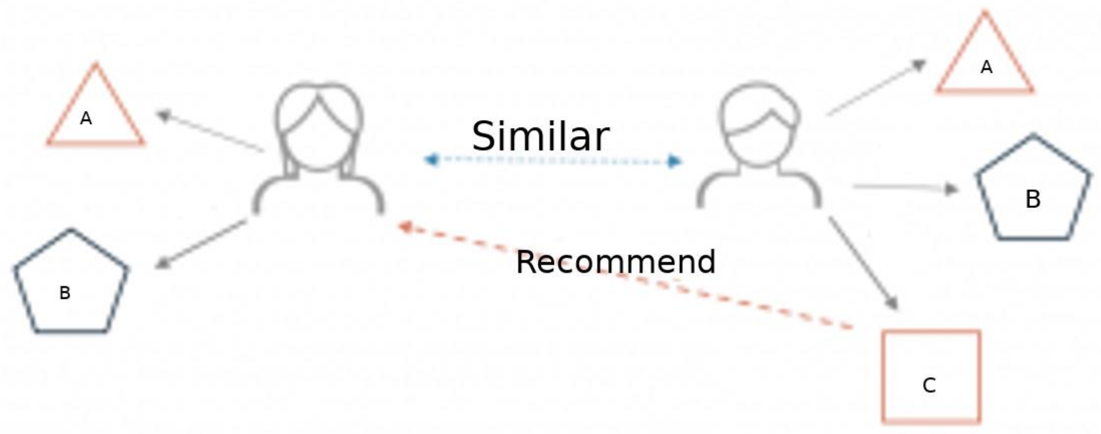
K MAHESH | Mentor – Sumit Dutta

MOVIE RECOMMENDATION SYSTEM

1. Introduction:

What is a Recommender system?

A recommender system is a simple algorithm whose aim is to provide the most relevant information to a user by discovering patterns in a dataset. The algorithm rates the items and shows the user the items that they would rate highly. An example of recommendation in action is when you visit Amazon and you notice that some items are being recommended to you or when Netflix recommends certain movies to you.



Two users buy the same items A and B from an ecommerce store. When this happens the similarity index of these two users is computed. Depending on the score the system can recommend item C to the other user because it detects that those two users are similar in terms of the items they purchase.

Different types of recommendation engines:

The most common types of recommendation systems are **content based** and **collaborative filtering** recommender systems.

1. Content based recommended system:

It uses metadata such as genre, producer, and actor, musician to recommend items say movies or music. Such a recommendation would be for instance recommending Infinity War that featured Vin Diesel because someone watched and liked The Fate of the Furious. Similarly you can get music recommendations from certain artists because you liked their music. Content based systems are based on the idea that if you liked a certain item you are most likely to like something that is similar to it.

A Content-based recommender system tries to recommend items to users, based on their profile. The user's profile revolves around the user's preferences and tastes, or based on the user ratings.

2. Collaborative based recommended system:

In collaborative filtering the behaviour of a group of users is used to make recommendations to other users. Recommendation is based on the preference of other users. A simple example would be recommending a movie to a user based on the fact that their friend liked the movie.

There are two types of collaborative models **Memory-based** methods and **Model-based** methods. The advantage of memory-based techniques is that they are simple to implement and the resulting recommendations are often easy to explain. They are divided into two:

1. User-based collaborative filtering:

In this model products are recommended to a user based on the fact that the products have been liked by users similar to the user. For example if Derrick and Dennis like the same movies and a new movie comes out that Derick likes, then we can recommend that movie to Dennis because Derrick and Dennis seem to like the same movies.

2. Item-based collaborative filtering:

These systems identify similar items based on users' previous ratings. For example if users A,B and C gave a 5 star rating to books X and Y then when a user D buys book Y they also get a recommendation to purchase book X because the system identifies book X and Y as similar based on the ratings of users A,B and C.

Model-based methods are based on matrix factorization and are better at dealing with sparsity. They are developed using data mining, machine learning algorithms to predict users' rating of unrated items. In this approach techniques such as dimensionality reduction are used to improve the accuracy. Examples of such model-based methods include decision trees, rule-based models, Bayesian methods and latent factor models.

2. Problem Statement:

For building a recommender system from scratch, we face several different problems. Currently there are a lot of recommender systems based on the user information, so what should we do if the website has not gotten enough users. After that, we will solve the representation of a movie, which is how a system can understand a movie. That is the precondition for comparing similarity between two movies. Movie features such as genre, actor and director is a way that can categorize movies. But for each feature of the movie, there should be different weight for them and each of them plays a different role for recommendation. So we get these questions:

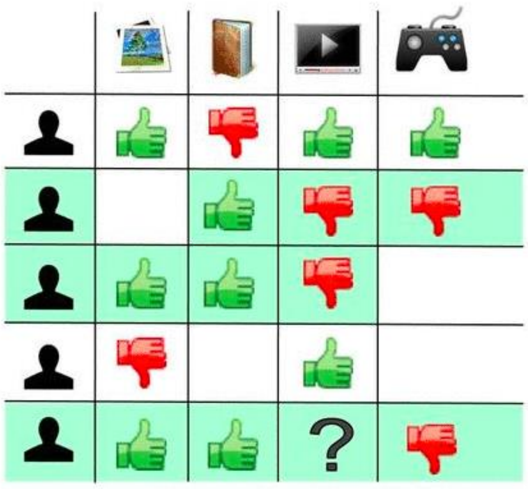
• How to recommend movies when there are no user information.

• What kind of movie features can be used for the recommender system?

. • How to calculate the similarity between two movies.

• Is it possible to set weight for each feature?

Collaborative filtering algorithms try to solve the prediction problem. In other words, we are given a matrix of i users and j items. The value in the ith row and the jth column (denoted by rij) denotes the rating given by user i to item j.



Matrix of i users and j items

Our job is to complete this matrix. In other words, we need to predict all the cells in the matrix that we have no data for. For example, in the preceding diagram, we are asked to predict whether user E will like the music player item. To accomplish this task, some ratings are available. For doing this we are using the **Matrix Factorization Technique** to predict the ratings, so that to complete this matrix.

3. Description of Dataset:

This section describes the various data cleaning and data wrangling methods applied on the Movie lens datasets to make it more suitable for further analysis. The following sections are divided based on the procedures followed:

Cleaning:

The Movielens data are contained in six files, genome-scores.csv, genome-tags.csv, links.csv, movies.csv, ratings.csv and tags.csv. It contains 20000263 ratings and 465564 tag applications across 27278 movies. These data were created by 138493 users between January 09, 1995 and March 31, 2015. This dataset was generated on October 17, 2016.

The dataset had a lot of features which had 0s for values it did not possess. These values were converted to NaN. These NaN values are filled in two ways according to the type of the data

1. Categorical type was filled with the mode values.
2. Numerical type was filled with mean values.

Removing Unnecessary Features:

This process was done in different ways

Number of missing values:

If a particular feature has more than 50% of missing values in it then most of the times that particular feature will not play any significant role in learning the model. But in my dataset I am not at all having a single feature with more than 50 % of missing values. But there are some features with fewer NaNs. Those features are tag, year and tmdbId but I am not removing those features.

In this Movie lens dataset I am having almost 12 features. But I am not using all those 12 features whichever the features are important to me for solving the recommendation problem, I am using those features only.

Significance:

Movie Lens dataset bases its recommendations on input provided by users of the website, such as movie ratings. For each user, Movie Lens predicts how the user will rate any given movie on the website. Based on these predicted ratings, the system recommends movies that the user is likely to rate highly.

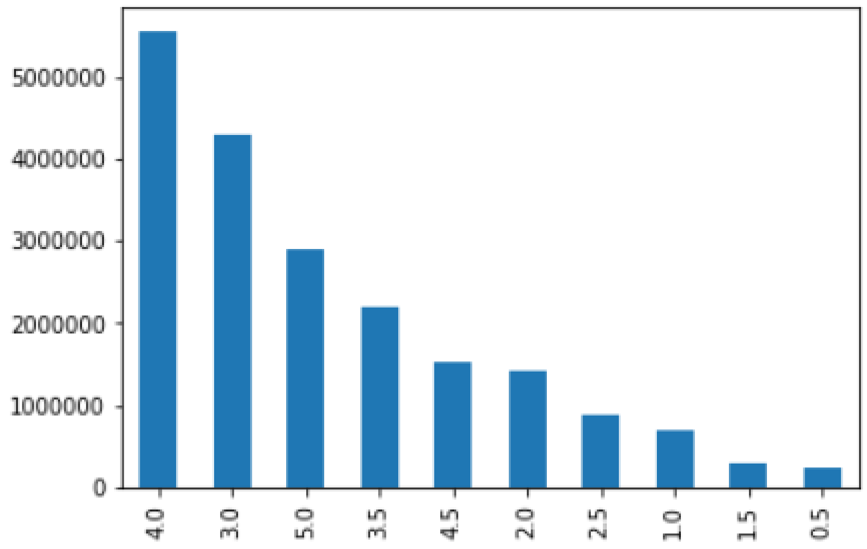
Outliers:

For features like Ratings, whose rating-value is > 5 or negative values, those values are called as Outliers and removing those values will help a lot in predicting the accurate output. And here in this case there are no outliers present in this dataset.

4. INITIAL FINDINGS FROM EXPLORATORY AALYSIS:

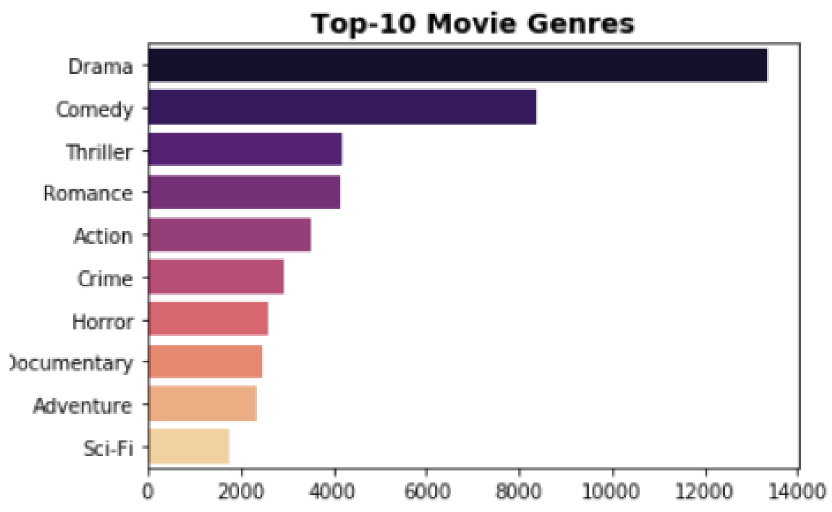
Data storytelling is about communicating your insights effectively, giving your data a voice. Data storytelling is a methodology for communicating information, tailored to a specific audience, with a compelling narrative.

1. Counting the Ratings:



From the above bar graph, we are counting the ratings given by the users for the movies. Most of the users are given rating value as 4. It seems to be good. And the least rating value given by the users are 0.5. From this we conclude that the rating tends to be relatively positive (>3). This may be due to the fact that unhappy customers tend to just leave instead of making efforts to rate. We can keep this in mind - low rating movies mean they are generally really bad

2. Top 10 Movie Genres:



Genre column has various genres associated with that movie but we need to process on each genre separately and hence we will split the genre with the '|' operator and then use explode so that every distinct genre will be in their own row. This will ignore null or empty genres if present in the dataset and if for some reason you want them to persist then you can go for explode outer as it will also store null or empty values.

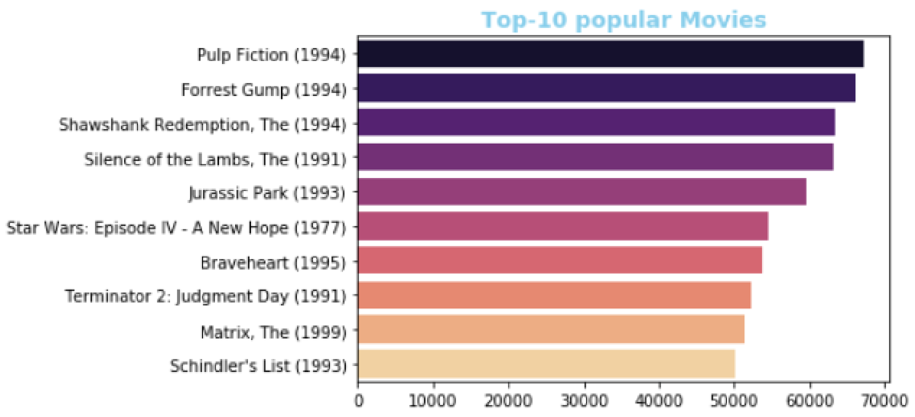
We have to group the movies by their genre and counted the number of rows to know how many movies are present in different genres. As we can see here drama won the race by some distance.

Drama is the most commonly occurring genre with almost half the movies identifying itself as a drama film. Comedy comes in at a distant second with 25% of the movies having adequate doses of humour. Other major genres represented in the top 10 are Thriller, Romance, Action, Crime, Horror, Documentary, Adventure and Sci-Fi.

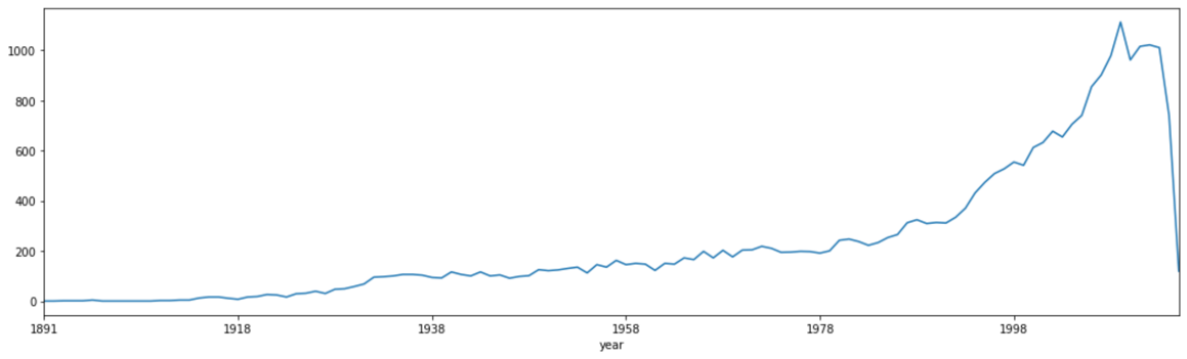
We will consider only those themes that appear in the top 10 most popular genres.

3. Top 10 Popular Movies:

There are almost 27278 movies available with us. From which these are the top 10 popular movies we shown in the above graph. And we notice that Pulp Fiction movie which was released in the year 1994, this is the most popular movie among all the movies. That means this movie is watched by the most number of the users, and then followed by Forrest Gump, Shawshank Redemption etc. From this graph, we conclude that these are the top 10 movies watched by the users.

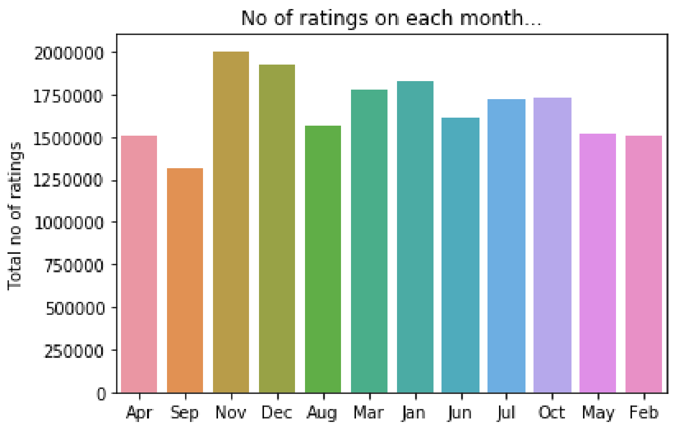


4. Number of movies released by the year:



From The Movie lens Dataset there are 27278 movies available with us. From that movies we are looking at the number of movies released by the year. We notice that there is a sharp rise in the number of movies starting the 1990s decade.

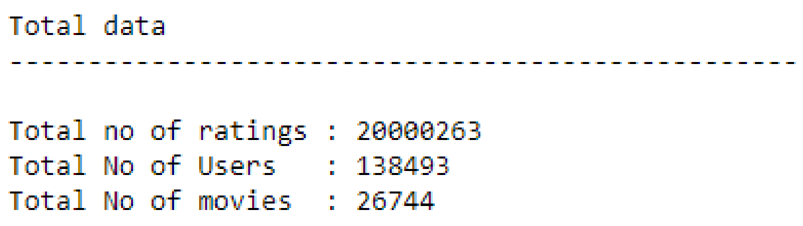
5. Number of ratings on each month:



From this we wanted to understand that the no of ratings differ by each month. Here, X-axis will be Month and Y-axis will be Total no of ratings. We'll notice that actually September month is far fewer than the other months. And also we noticed that lots of ratings are given by the November month.

5. EXPLORATORY DATA ANALYSIS:

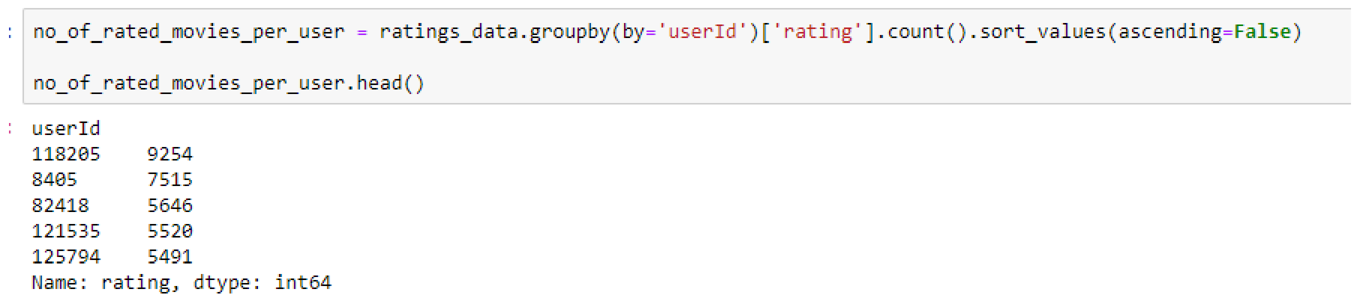
Basic Statistics (#Ratings, #Users, and #Movies):



```
Total data
--------------------------------------------------

Total no of ratings : 20000263
Total No of Users   : 138493
Total No of movies  : 26744
```

By seeing the above information I'm just understanding that how many ratings, users and movies are there in my dataset. I've almost 20000263 ratings given by the 138493 users on 26744 movies. And also this gives me very high level overview.

Number of movies rated by a user:



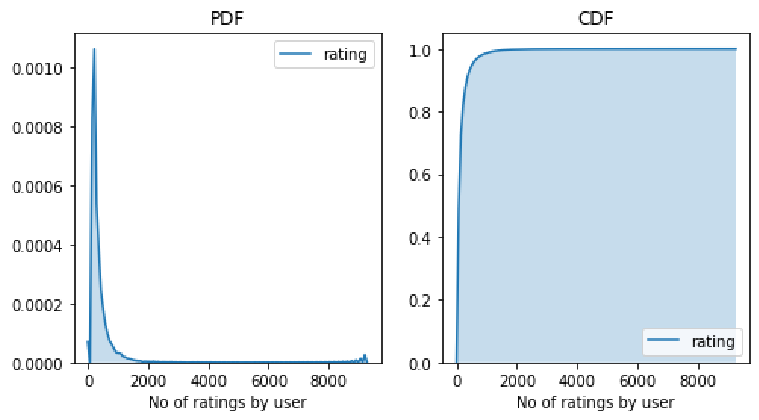
```
: no_of_rated_movies_per_user = ratings_data.groupby(by='userId')['rating'].count().sort_values(ascending=False)

  no_of_rated_movies_per_user.head()
```

```
: userId
  118205    9254
  8405      7515
  82418     5646
  121535    5520
  125794    5491
  Name: rating, dtype: int64
```

The above data says that the number of movies rated by a given user. If you see the above user Id with 118205 gave 9254 movies that seems to be very large to me for a single user to give. Likewise for every user how many movies he rated with corresponding count we calculated.

Finding PDF & CDF for no of ratings per user:

We plotted CDF & PDF here, we quickly noticed that the peak here tells us that, most of the users rate only a few movies. But there are few users here giving lots of rating. And if you look at the CDF also, almost 90% of people gave very few ratings.



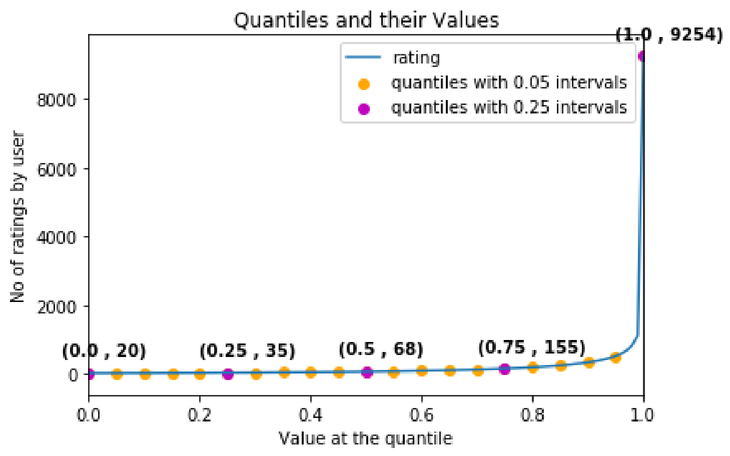
```
no_of_rated_movies_per_user.describe()
```

```
count    138493.000000
mean        144.413530
std         230.267257
min          20.000000
25%          35.000000
50%          68.000000
75%         155.000000
max        9254.000000
Name: rating, dtype: float64
```

From the above data, we observe that the average number of movies that are rated per user is about 144. This shows that most of the users rated lot of movies. Max number of ratings are 9254 and minimum number of ratings are 20. And if you see the median number of movie rated by a user are 68. That means almost 50% of customers have rated more than 68 movies.

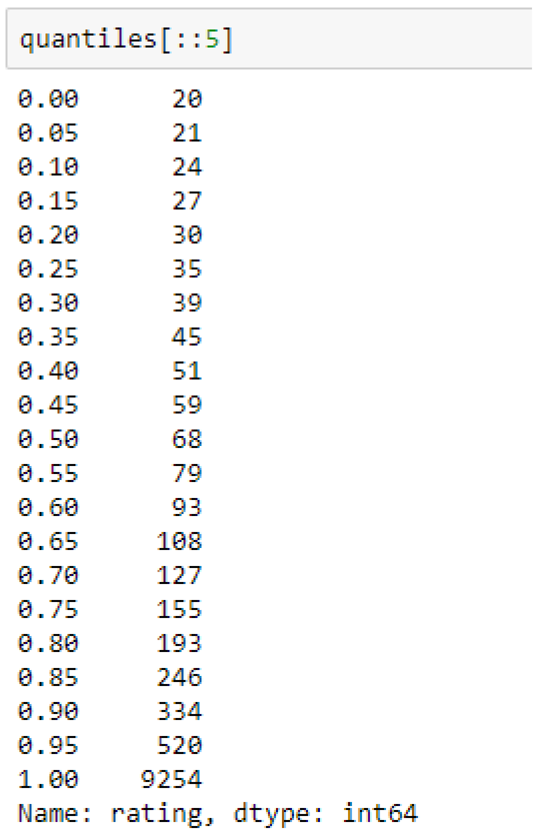
We thought by looking at PDF & CDF, we're not able to get it so well. So we went on understanding about percentiles.

Let's get all the percentile values.



If you notices, what's happening here is, each of the violet circle represents 0.25 intervals. And also we plotted yellow circle it represents 0.05 interval. At 0.25 percentile there are 35 movies rated by the users. And if you closely observe, at 0.95 percentile is also quite low, only 100% percentile is very large.

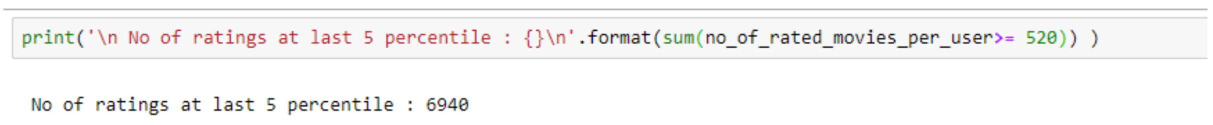
We actually printed those values.



```
quantiles[::5]
```

```
0.00      20
0.05      21
0.10      24
0.15      27
0.20      30
0.25      35
0.30      39
0.35      45
0.40      51
0.45      59
0.50      68
0.55      79
0.60      93
0.65     108
0.70     127
0.75     155
0.80     193
0.85     246
0.90     334
0.95     520
1.00    9254
Name: rating, dtype: int64
```

At 0.95 percentile also, it is showing 520 movies, there are 5% of users who rated more than 520 movies.



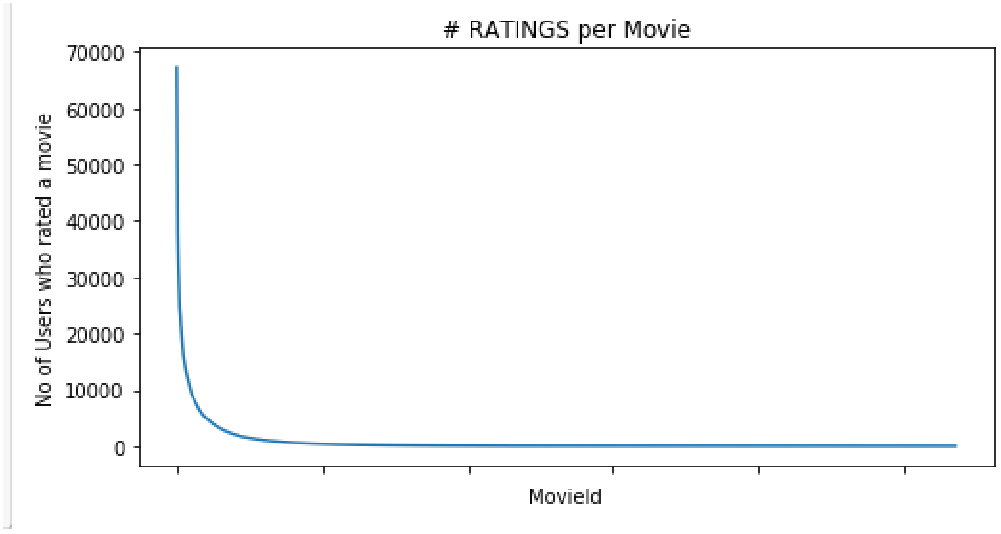
```
print('\n No of ratings at last 5 percentile : {}\n'.format(sum(no_of_rated_movies_per_user>= 520)) )
```

```
No of ratings at last 5 percentile : 6940
```

We also saw that No of ratings at last 5% percentile are 6940 movies. But this gave us good sense on how many ratings that each user has provide.

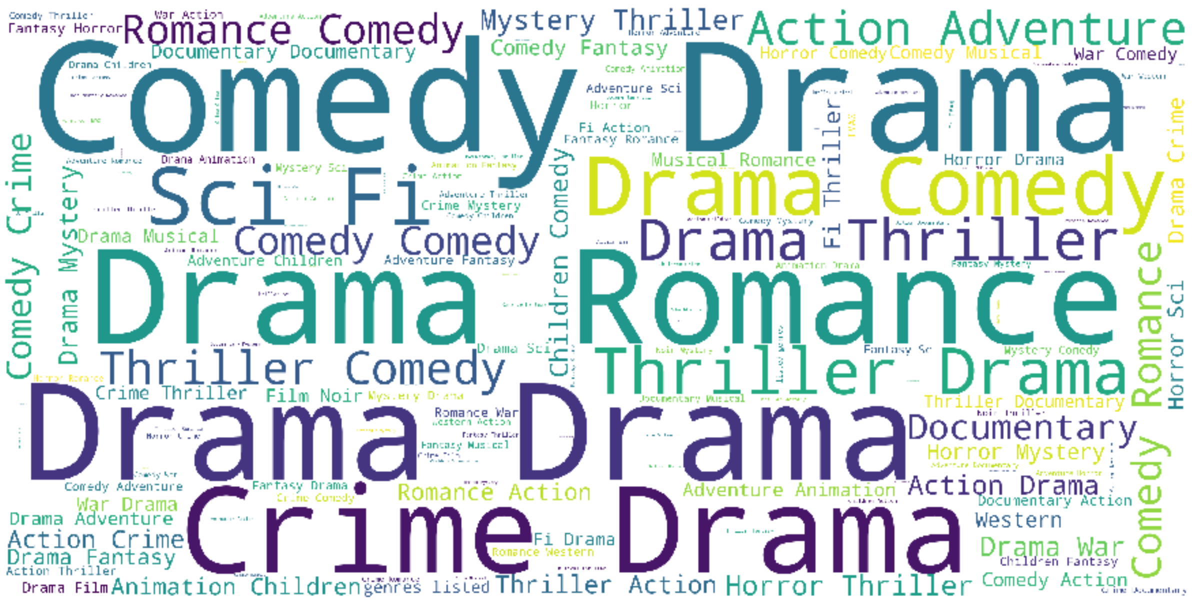
Analysis of rating of a movie given by a user:

For a given movie let's find the no of users who rated a movie. Because there will be some movies like titanic which are liked by millions of people across the world and hence there will be millions of the rating for a movie like that. But there are some other movies which are liked by very few of them.



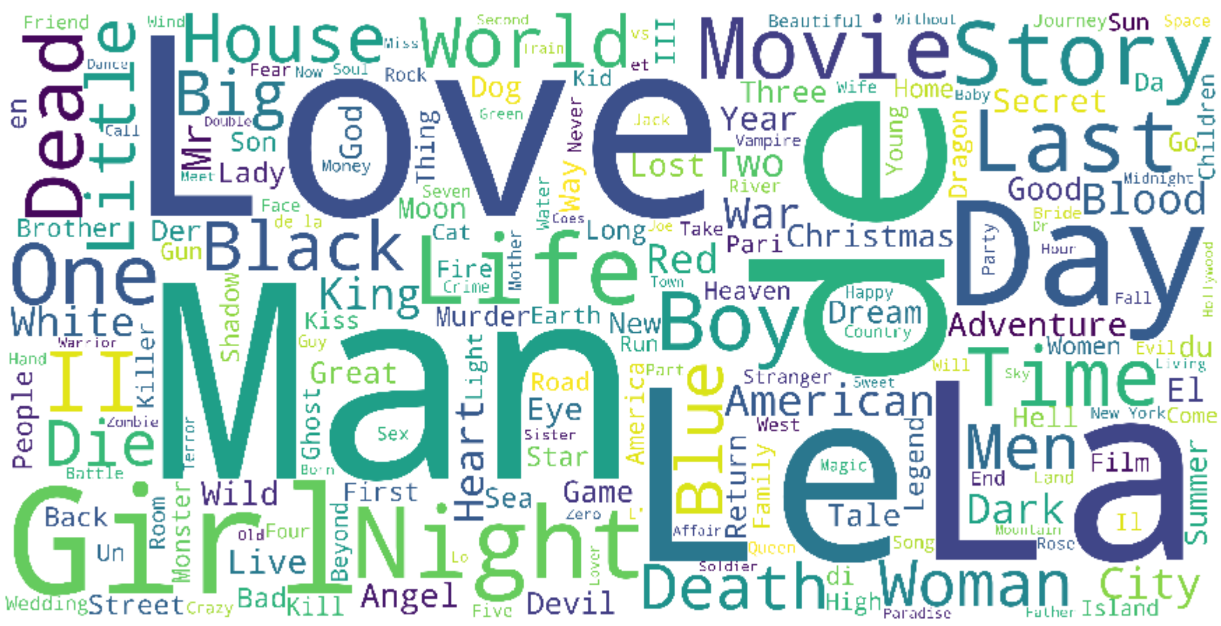
From the above fig, X-axis will be the Movieid, and Y-axis will be no of users and we plotted it. As you'll see there are very small no of movies which are rated by lots of people. But most of the movies rated by a fewer no of people.

Movie genres and overview wordclouds:



Drama is the most commonly used word in Movie genres. **Comedy** and **Romance** are also popular in Movie genres. Together with **Drama**, **Comedy** and **Romance**, these wordclouds give us a pretty good idea of the most popular genres.

Title and overview wordclouds:



Love is the most commonly used word in Movie titles. **Man** and **Girl** are also popular in Movie Blurbs. Together with **Love**, **Man** and **Girl**, these wordclouds give us a pretty good idea of the most popular themes present in movie.

6. IN-DEPTH ANLYSIS:

1. SVD Matrix Factorization for Recommendations in Python:

Matrix factorization is the breaking down of one matrix in a product of multiple matrices. It's extremely well studied in mathematics, and it's highly useful. There are many different ways to factor matrices, but singular value decomposition is particularly useful for making recommendations.

SVD in the context of recommendation systems is used as a collaborative filtering (CF) algorithm. For those of you who don't know, collaborative filtering is a method to predict a rating for a user item pair based on the history of ratings given by the user and given to the item. Most CF algorithms are based on user-item rating matrix where each row represents a user, each column an item. The entries of this matrix are ratings given by users to items.

What is singular value decomposition (SVD)?

At a high level, SVD is an algorithm that decomposes a matrix R into the best lower rank (i.e. smaller/simpler) approximation of the original matrix R. Mathematically, it decomposes R into a two unitary matrices and a diagonal matrix:

$$R = U\Sigma V^T$$

Where, R is user's ratings matrix,

U is the user "features" matrix,

Σ is the diagonal matrix of singular values (essentially weights),

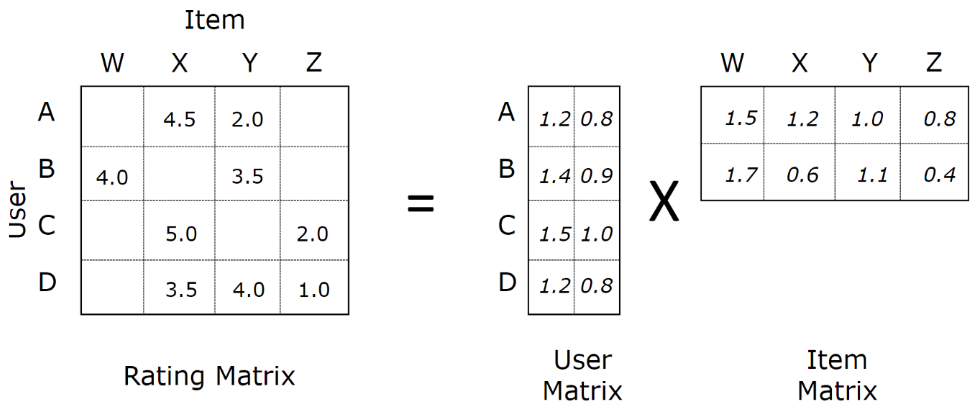
VT is the movie "features" matrix.

U and VT are orthogonal, and represent different things. U represents how much users "like" each feature and V 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 we think of as the underlying tastes and preferences vectors.

SVD and Matrix factorization:

SVD is a matrix factorization technique that is usually used to reduce the number of features of a data set by reducing space dimensions from N to K where K < N. For the purpose of the recommendation systems however, we are only interested in the matrix factorization part keeping same dimensionality. The matrix factorization is done on the user-item ratings matrix. From a high level, matrix factorization can be thought of as finding 2 matrices whose product is the original matrix.



Each item can be represented by a vector `*qi*`. Similarly each user can be represented by a vector `*pu*` such that the dot product of those 2 vectors is the expected rating

$$expected\ rating = \hat{r}_{ui} = q_i^T p_u$$

It is a form of factorization!!

`*qi*` and `*pu*` can be found in such a way that the square error difference between their dot product and the known rating in the user-item matrix is minimum

$$minimum(p,q) \sum_{(u,i)\epsilon K} \left(r_{ui} - q_i^T . p_u\right)^2$$

Regularization:

For our model to be able to generalize well and not over-fit the training set, we introduce a penalty term to our minimization equation. This is represented by a regularization factor λ multiplied by the square sum of the magnitudes of user and item vectors.

$$minimum(p,q) \sum_{(u,i)\epsilon K} \left(r_{ui} - q_i^T . p_u\right)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

To illustrate the usefulness of this factor imagine we have an extreme case where a low rating given by a user to a movie with no other rating from this user. The algorithm will minimize the error by giving `*qi*`a large value. This will cause all rating from this user to other movies to be very low. This is intuitively wrong. By adding the magnitude of the vectors to the equation, giving vectors large value will minimize the equation and thus such situations will be avoided.

Minimizing with Stochastic Gradient Descent (SGD):

The above equation is minimized using stochastic gradient descent algorithm. From a high level perspective SGD starts by giving the parameters of the equation we are trying to minimize initial values and then iterating to reduce the error between the predicted and the actual value each time correcting the previous value by a small factor. This algorithm uses a factor called learning rate γ which determines the ratio of the old value and the new computed value after each iteration. Practically, when using high γ one might skip the optimal solution whereas when using low γ values a lot of iterations are needed to reach optimal value.

If I wanted to put this kind of system into production, I'd want to create a training and validation set and optimize the number of latent features (k) by minimizing the Root Mean Square Error. Intuitively, the Root Mean Square Error will decrease on the training set as '**k**' increases.

Making Movie Recommendations:

Finally, with the predictions matrix for every user, I can build a function to recommend movies for any user. All I need to do is return the movies with the highest predicted rating that the specified user hasn't already rated. Though I didn't use actually use any explicit movie content features (such as genre or title), I'll merge in that information to get a more complete picture of the recommendations.

2. Content Based Movie Recommendation:

The Content-Based Recommender relies on the similarity of the items being recommended. The basic idea is that if you like an item, then you will also like a "similar" item. It generally works well when it's easy to determine the context/properties of each item.

A content based recommender works with data that the user provides, either explicitly movie ratings for the Movie Lens dataset. Based on that data, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate.
In this project we are using two different approaches to build content based movie recommendation.

First Approach:

The concepts of **Term Frequency (TF)** and **Inverse Document Frequency (IDF)** are used in information retrieval systems and also content based filtering mechanisms (such as a content based recommender). They are used to determine the relative importance of a document / article / news item / movie etc.

Below is the equation to calculate the TF-IDF score:

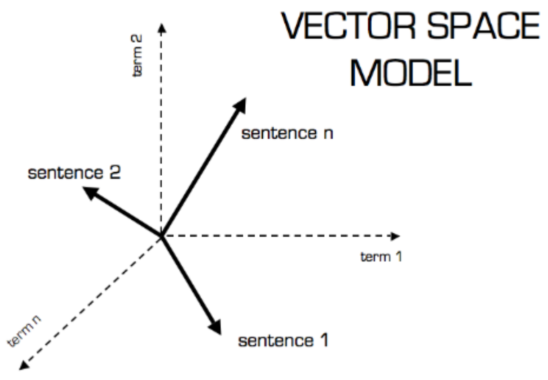
$$\mathrm{tfidf}_{i,j} = \mathrm{tf}_{i,j} \times \log\left(\frac{\mathrm{N}}{\mathrm{df}_i}\right)$$

$\mathrm{tf}_{i,j}$ = total number of occurences of i in j
df_i = total number of documents (speeches) containing i
N = total number of documents (speeches)

TF-IDF Equation

After calculating TF-IDF scores, how do we determine which items are closer to each other, rather closer to the user profile?

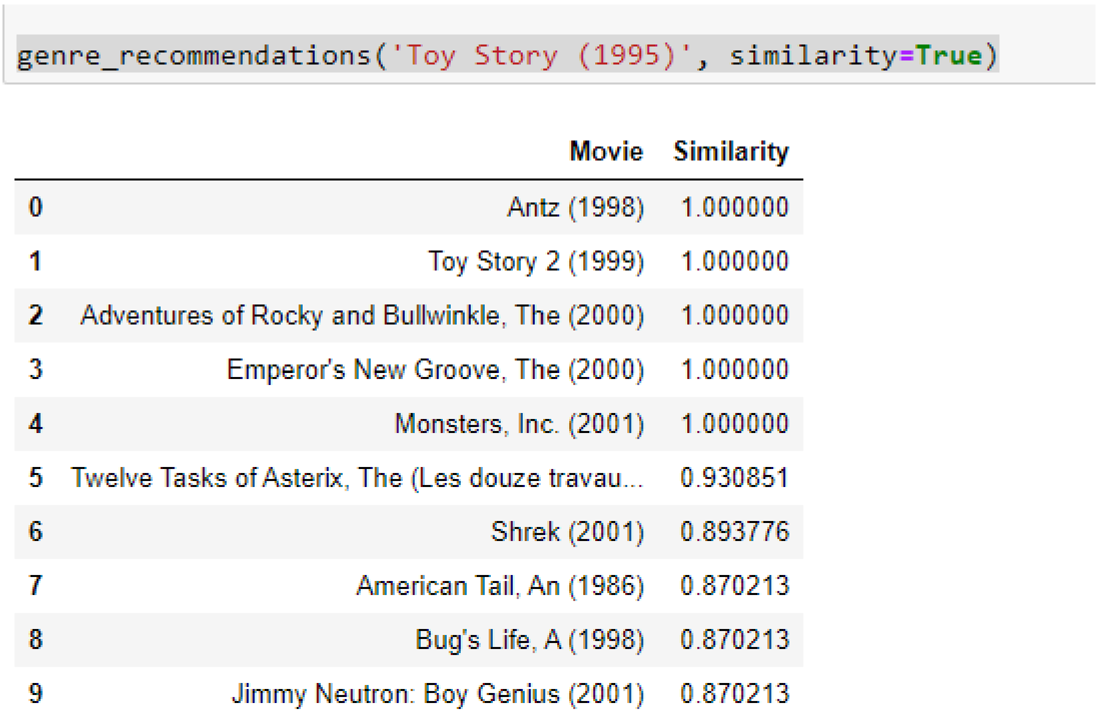
This is accomplished using the **Vector Space Model** which computes the proximity based on the angle between the vectors. In this model, each item is stored as a vector of its attributes (which are also vectors) in an **n-dimensional space** and the angles between the vectors are calculated to **determine the similarity between the vectors**. Next, the user profile vectors are also created based on his actions on previous attributes of items and the similarity between an item and a user is also determined in a similar way.



Vector Space Model

Sentence 2 is more likely to be using Term 2 than using Term 1. Vice-versa for Sentence 1. The method of calculating this relative measure is calculated by taking the cosine of the angle between the sentences and the terms. The ultimate reason behind using cosine is that the **value of cosine will increase with decreasing value of the angle** between which signifies more similarity. The vectors are length normalized after which they become vectors of length 1 and then the cosine calculation is simply the sum-product of vectors.

I will be using the **Cosine Similarity** to calculate a numeric quantity that denotes the similarity between two movies. Since I have used the TF-IDF Vectorizer, calculating the Dot Product will directly give me the Cosine Similarity Score. Therefore, I will use sklearn's **linear_kernel** instead of cosine_similarities since it is much faster. I now have a pairwise cosine similarity matrix for all the movies in the dataset. The next step is to write a function that returns the 20 most similar movies based on the cosine similarity score.



```
genre_recommendations('Toy Story (1995)', similarity=True)
```

	Movie	Similarity
0	Antz (1998)	1.000000
1	Toy Story 2 (1999)	1.000000
2	Adventures of Rocky and Bullwinkle, The (2000)	1.000000
3	Emperor's New Groove, The (2000)	1.000000
4	Monsters, Inc. (2001)	1.000000
5	Twelve Tasks of Asterix, The (Les douze travau...	0.930851
6	Shrek (2001)	0.893776
7	American Tail, An (1986)	0.870213
8	Bug's Life, A (1998)	0.870213
9	Jimmy Neutron: Boy Genius (2001)	0.870213

Second Approach:

Suppose I watch a particular genre movie I will be recommended movies with respective to that specific genre. The Title, Year of Release, Director are also helpful in identifying similar movie content.

In this approach content of the product are already rated based on the user's preference (User Profile), while the genre of an item is an implicit features that it will be used to build item profile.

In movie lens dataset movies data consist of Movie Id, Title, Genres and Year of Release. First we split the genres feature and then we apply the one hot encoding on the genres feature.

Since keeping genres in a list format isn't optimal for the content-based recommendation system technique, we will use the One Hot Encoding technique to convert the list of genres to a vector where each column corresponds to one possible value of the feature. This encoding is needed for feeding categorical data. In this case, we store every different genre in columns that contain either 1 or 0. 1 shows that a movie has that genre and 0 shows that it doesn't. Let's also store this data frame in another variable since genres won't be important for our first recommendation system.

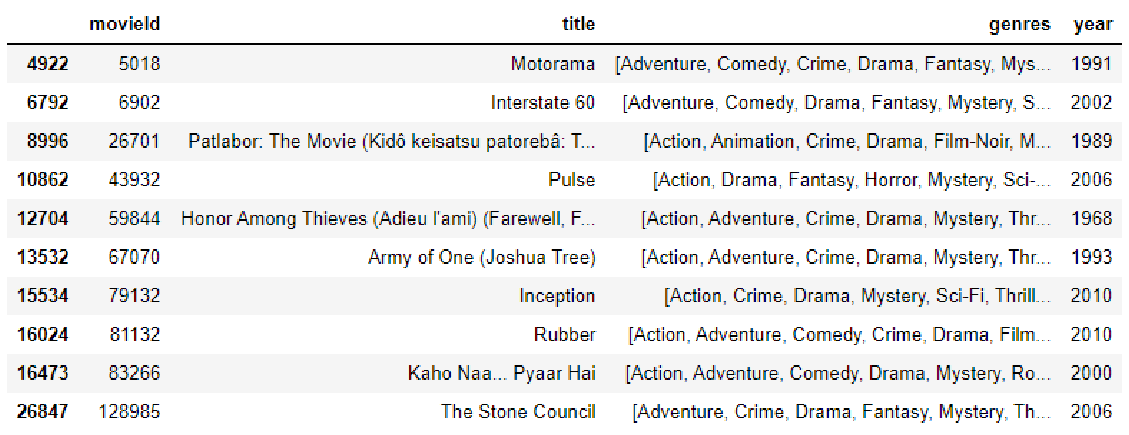
Now, let's take a look at how to implement **Content-Based** or **Item-Item recommendation systems**. This technique attempts to figure out what a user's favourite aspects of an item is, and then recommends items that present those aspects. In our case, we're going to try to figure out the input's favourite genres from the movies and ratings given.

Once we create the User’s input list, and then we’ll try to build the User Profile. First filtering out the movies by title. Dropping information we won’t use from the input list dataframe. If a movie you added in above isn't here, then it might not be in the original dataframe or it might spelled differently, please check capitalisation. And then filtering out the user’s input movie list from the original movies dataframe. Dropping unnecessary features due to save memory and avoid issues.

We’re going to turn each genre into weights. We can do this by using the input's reviews and multiplying them into the input's genre table and then summing up the resulting table by column. This operation is actually a dot product between a matrix and a vector, so we can simply accomplish by calling Pandas’ "dot" function.

Now let's get the genres of every movie in our original dataframe. Multiply the genres by the weights and then take the weighted average. And then sort our recommendations in descending order. So that we’ll get the top weighted average values. And based on that we’ll recommend the movies.

Final Recommendation table looks like this:



	movieId	title	genres	year
4922	5018	Motorama	[Adventure, Comedy, Crime, Drama, Fantasy, Mys...	1991
6792	6902	Interstate 60	[Adventure, Comedy, Drama, Fantasy, Mystery, S...	2002
8996	26701	Patlabor: The Movie (Kidô keisatsu patorebâ: T...	[Action, Animation, Crime, Drama, Film-Noir, M...	1989
10862	43932	Pulse	[Action, Drama, Fantasy, Horror, Mystery, Sci-...	2006
12704	59844	Honor Among Thieves (Adieu l'ami) (Farewell, F...	[Action, Adventure, Crime, Drama, Mystery, Thr...	1968
13532	67070	Army of One (Joshua Tree)	[Action, Adventure, Crime, Drama, Mystery, Thr...	1993
15534	79132	Inception	[Action, Crime, Drama, Mystery, Sci-Fi, Thrill...	2010
16024	81132	Rubber	[Action, Adventure, Comedy, Crime, Drama, Film...	2010
16473	83266	Kaho Naa... Pyaar Hai	[Action, Adventure, Comedy, Drama, Mystery, Ro...	2000
26847	128985	The Stone Council	[Adventure, Crime, Drama, Fantasy, Mystery, Th...	2006

Conclusion:

With content-based filtering as we saw above, content-based filtering is not practical, or rather, not very dependable when the number of items increases along with a need for clear and differentiated descriptions. To overcome all the issues discussed earlier, we can implement collaborative filtering techniques, which have proven to be better and more scalable.

We’ve seen that we can make good recommendations with raw data based collaborative filtering methods (neighbourhood models) and latent features based matrix factorization methods (factorization models).

Low-dimensional matrix recommenders try to capture the underlying features driving the raw data (which we understand as tastes and preferences). From a theoretical perspective, if we want to make recommendations based on people’s tastes, this seems like the better approach. This technique also scales **significantly** better to larger datasets, since we can actually approximate the SVD with gradient descent.

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.