CS582 Project 1

Recommender system and embedding Machine learning

Cong Trang Truong – 612731
Khoa Nam Nguyen – 612744
Thanh Nhan Nguyen – 612745
Thai Trung Phan – 612819
Thai Binh Duong Nguyen – 612700
Anh Thong Tran – 612723

Professor: **Anthony Sander**

Aug 2, 2021

Contents

Abstract

Objective is to learn recommendation systems techniques, and practice embedding and MLOps. A recommendation system filters the data using different algorithms and recommends the most relevant movies to users. In practice, movie recommendation system are of three types:

- Demographic: suggest movies to every user based on the popularity of the movie
- Content Based Similarity: suggest movies to user based on the movies they watched (using movie metadata, such as title, overview, tagline ...)
- Collaborative Filtering: suggest movies from various users based on user similarities

1 Problem Statement

The data used in this study is called: "TMDb Movies Dataset" and was obtained from Kaggle[1]. The dataset consists of information about 10,000+ movies collected from The Movie Database (TMDb) including user ratings, budgets and revenues and much more

This project aims to build a movie recommendation system which is served via Restful APIs using various techniques such as demographic, content based similarity, and collaborative filtering. For all APIs, applied k=10 candidate movies as an acceptable default value.

1.1 Demographic: aka Trending now

In this section, we produce a generic API that shows a user a list of movies based on user rating score, as call as Top IMDB Movies.

1.1.1 Dataset

The data used in the section can be obtain from the TMDb Movies Dataset[2]. The file name is tmdb_movies_data.csv

1.1.2 Problem

Firstly, we think that the movies can be easily filtered and sorted by their respective ratings, specially data from column vote_count and vote_average.

However, using rating as a score has a caveat. The popularity of a movies is a important point. Because if a movie which is voted by 100 users has 9.5 rating point will be considered “better” than a movie which is voted by 10,000 users has 9.0 rating point.

1.1.3 Solution

To solve the problem, let's use the result of weighted rating formula as a score. It is described as follows:

$$WeightedRating(WR) = (\frac{v}{v+m}.r) + (\frac{m}{v+m}.c)$$

. In the formula above:

1. v is the number of votes for the movies
2. m is the minimum votes required to be listed in the chart
3. r is the average ratings of the movie
4. c is the mean vote of all movies

We already have the values of *v(vote_count)* and *r(vote_average)*. We now need to calculate the values of c and r.

We can easily determine value of C by using the pandas *.mean()* function:

```
c = data['vote_average'].mean()
> 5.9749217743419845
```

From the above output, we can see that the average rating of a movie is around 5.8 on a scale of 10.

Next, we can calculate the value of *v*. In a data column, basically a quantile is a line that spilt the data into groups that have same number of data point. In this report, assume that there is 100 groups, we get a line at group 95, which is called 95th quantile. We can easily calculate it by using the pandas *.quantile()* function:

```
m = data['vote_count'].quantile(0.95)
> 1025.75
```

We can observe that value of 95th quantile is 1025.75.

After m is calculated, we can filter the movies of which *vote_count* are greater or equal m

```
rating_movies = data.copy().loc[data['vote_count'] >= m]
> (544, 21)
```

From the above output, only 544 / 10,000 movies have *vote_count* that is bigger than m. It means there are only 5% data after filtering. Next, we create a new function named *.calc_weighted_rating()* to determine the weighted rating of each movie, and then calculate the weighted rating score for each movie:

```
rating_movies['score'] = rating_movies.apply(lambda movie: calc_weighted_rati
```

Then, we can get the finally result by using the pandas function *.nlargest()*

```
rating_movies.nlargest(10, 'score')
```

	id	original_title	...	rating_score	imdb_id
4178	278	The Shawshank Redemption	...	8.033095	tt0111161
2875	155	The Dark Knight	...	7.869522	tt0468569
7269	238	The Godfather	...	7.822604	tt0068646
2409	550	Fight Club	...	7.786303	tt0137523
4177	680	Pulp Fiction	...	7.757735	tt0110912
4179	13	Forrest Gump	...	7.729396	tt0109830
629	157336	Interstellar	...	7.723911	tt0816692
1919	27205	Inception	...	7.717039	tt1375666
4949	122	The Lord of the Rings: The Return of the King	...	7.603584	tt0167260
630	118340	Guardians of the Galaxy	...	7.602512	tt2015381

Figure 1: Top 10 IMDB movies

1.2 Content Based: aka Because you watched X

1.2.1 EDA

1. Gathering Data

   Quickly testing if your object has the right type of data in it.

	id	imdb_id	popularity	budget	revenue	original_title	cast	...	production_companies	release_date	vote_count	vote_average	release_year	budget_adj	revenue_adj
0	135397	tt0369610	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	...	Universal Studios|Amblin Entertainment|Legenda...	6/9/2015	5562	6.5	2015	137999939.3	1.392446e+09
1	76341	tt1392190	28.419936	150000000	378436354	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...	...	Village Roadshow Pictures|Kennedy Miller Produ...	5/13/2015	6185	7.1	2015	137999939.3	3.481613e+08
2	262500	tt2908446	13.112507	110000000	295238201	Insurgent	Shailene Woodley|Theo James|Kate Winslet|Ansel...	...	Summit Entertainment|Mandeville Films|Red Wago...	3/18/2015	2480	6.3	2015	101199955.5	2.716190e+08
3	140607	tt2488496	11.173104	200000000	2068178225	Star Wars: The Force Awakens	Harrison Ford|Mark Hamill|Carrie Fisher|Adam D...	...	Lucasfilm|Truenorth Productions|Bad Robot	12/15/2015	5292	7.5	2015	183999919.0	1.902723e+09
4	168259	tt2820852	9.335014	190000000	1506249360	Furious 7	Vin Diesel|Paul Walker|Jason Statham|Michelle ...	...	Universal Pictures|Original Film|Media Rights ...	4/1/2015	2947	7.3	2015	174799923.1	1.385749e+09

Figure 2: Overview data

Get a concise summary of the dataframe. (object: 11 columns, numerical: 10 columns)

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10866 entries, 0 to 10865
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    10866 non-null  int64
 1   imdb_id               10856 non-null  object
 2   popularity            10866 non-null  float64
 3   budget                10866 non-null  int64
 4   revenue               10866 non-null  int64
 5   original_title        10866 non-null  object
 6   cast                  10790 non-null  object
 7   homepage              2936 non-null   object
 8   director              10822 non-null  object
 9   tagline               8042 non-null   object
 10  keywords              9373 non-null   object
 11  overview              10862 non-null  object
 12  runtime               10866 non-null  int64
 13  genres                10843 non-null  object
 14  production_companies  9836 non-null   object
 15  release_date          10866 non-null  object
 16  vote_count            10866 non-null  int64
 17  vote_average          10866 non-null  float64
 18  release_year          10866 non-null  int64
 19  budget_adj            10866 non-null  float64
 20  revenue_adj           10866 non-null  float64
dtypes: float64(4), int64(6), object(11)
memory usage: 1.7+ MB
None
```


Figure 3: Colum info

View some basic statistical details like percentile, mean, std etc. of a data frame, that will help us chose right value when we need imputation missing data.

	id	popularity	budget	revenue	runtime	vote_count	vote_average	release_year	budget_adj	revenue_adj
count	10866.000000	10866.000000	1.086600e+04	1.086600e+04	10866.000000	10866.000000	10866.000000	10866.000000	1.086600e+04	1.086600e+04
mean	66064.177434	0.646441	1.462570e+07	3.982332e+07	102.070863	217.389748	5.974922	2001.322658	1.755104e+07	5.136436e+07
std	92130.136561	1.000185	3.091321e+07	1.170035e+08	31.381405	575.619058	0.935142	12.812941	3.430616e+07	1.446325e+08
min	5.000000	0.000065	0.000000e+00	0.000000e+00	0.000000	10.000000	1.500000	1960.000000	0.000000e+00	0.000000e+00
25%	10596.250000	0.207583	0.000000e+00	0.000000e+00	90.000000	17.000000	5.400000	1995.000000	0.000000e+00	0.000000e+00
50%	20669.000000	0.383856	0.000000e+00	0.000000e+00	99.000000	38.000000	6.000000	2006.000000	0.000000e+00	0.000000e+00
75%	75610.000000	0.713817	1.500000e+07	2.400000e+07	111.000000	145.750000	6.600000	2011.000000	2.085325e+07	3.369710e+07
max	417859.000000	32.985763	4.250000e+08	2.781506e+09	900.000000	9767.000000	9.200000	2015.000000	4.250000e+08	2.827124e+09

Figure 4: Describe dataset

Determine if any value in a data is Missing by heat map plot quickly

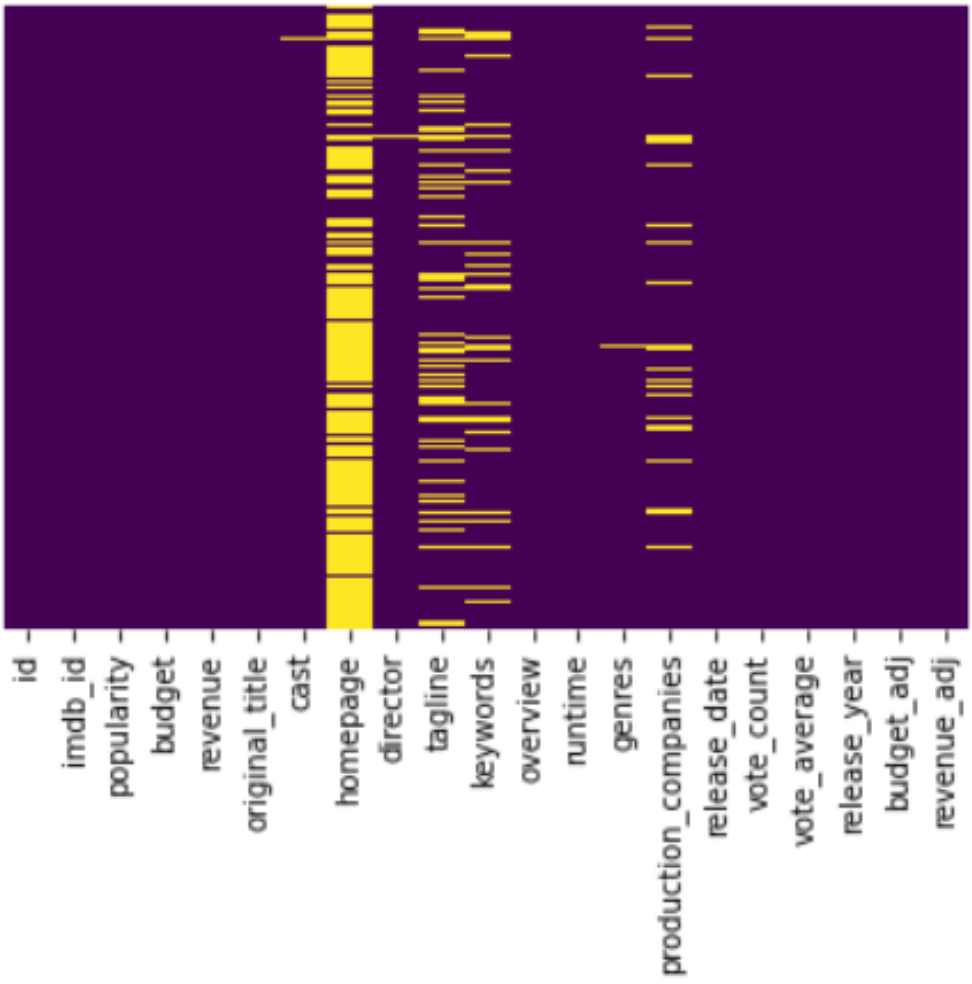


Figure 5: Overview missing data

There are 8 columns which is missing data on below figure, we know how many row per column which is missing data.

```
id                      0
imdb_id                10
popularity              0
budget                  0
revenue                 0
original_title          0
cast                   76
homepage             7930
director               44
tagline              2824
keywords             1493
overview                4
runtime                 0
genres                 23
production_companies  1030
release_date            0
vote_count              0
vote_average            0
release_year            0
budget_adj              0
revenue_adj             0
dtype: int64
```

Figure 6: Detail missing data

Pairwise correlation of all columns in the dataframe that will be useful for other prediction purpose, but We don't use it for this scope.

	id	popularity	budget	revenue	runtime	vote_count	vote_average	release_year	budget_adj	revenue_adj
id	1.000000	-0.014350	-0.141351	-0.099227	-0.088360	-0.035551	-0.058363	0.511364	-0.189015	-0.138477
popularity	-0.014350	1.000000	0.545472	0.663358	0.139033	0.800828	0.209511	0.089801	0.513550	0.609083
budget	-0.141351	0.545472	1.000000	0.734901	0.191283	0.632702	0.081014	0.115931	0.968963	0.622505
revenue	-0.099227	0.663358	0.734901	1.000000	0.162838	0.791175	0.172564	0.057048	0.706427	0.919110
runtime	-0.088360	0.139033	0.191283	0.162838	1.000000	0.163278	0.156835	-0.117204	0.221114	0.175676
vote_count	-0.035551	0.800828	0.632702	0.791175	0.163278	1.000000	0.253823	0.107948	0.587051	0.707942
vote_average	-0.058363	0.209511	0.081014	0.172564	0.156835	0.253823	1.000000	-0.117632	0.093039	0.193085
release_year	0.511364	0.089801	0.115931	0.057048	-0.117204	0.107948	-0.117632	1.000000	0.016793	-0.066256
budget_adj	-0.189015	0.513550	0.968963	0.706427	0.221114	0.587051	0.093039	0.016793	1.000000	0.646607
revenue_adj	-0.138477	0.609083	0.622505	0.919110	0.175676	0.707942	0.193085	-0.066256	0.646607	1.000000

Figure 7: Correlation data

Checking correlations is an important part of the exploratory data analysis process,Below heat map figure which used to decide which features affect the target variable the most

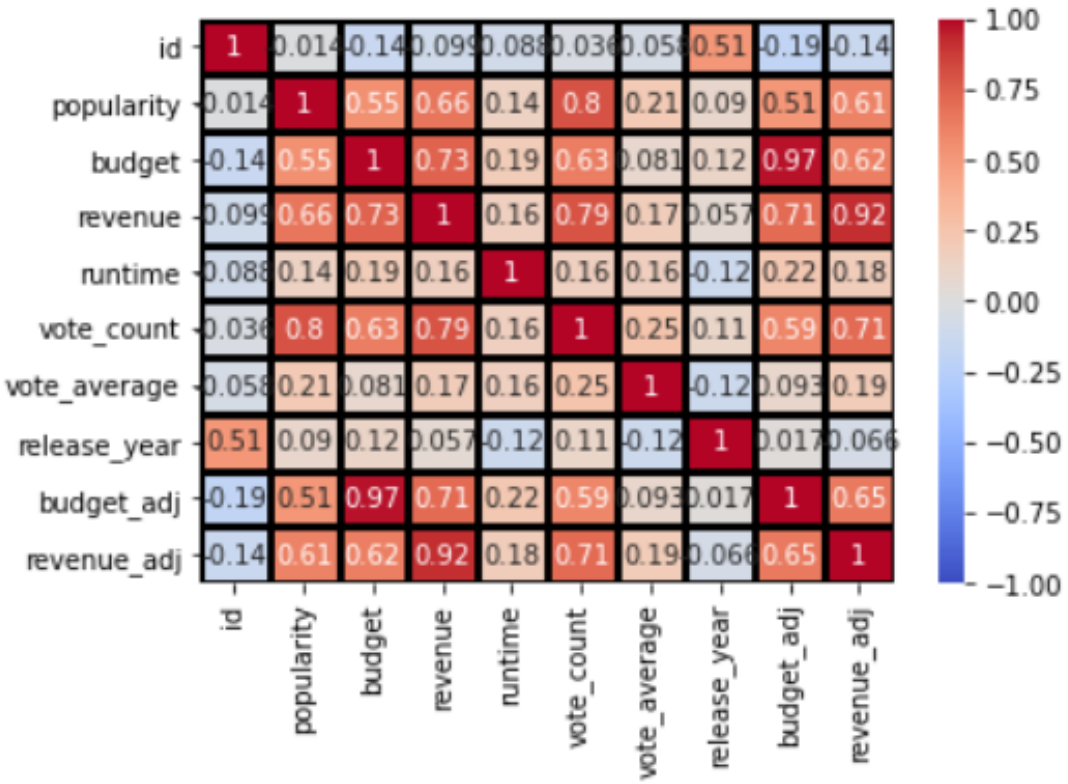


Figure 8: Heat map correlation

Conclusion: After exploring dataset, we chose a a set of features to represent each movie for content base filtering base on 3 columns below, but these colums is missing data, so we will need do data preparation step.

	original_title	overview	tagline
0	jurassic world	Twenty-two years after the events of Jurassic ...	The park is open.
1	mad max: fury road	An apocalyptic story set in the furthest reach...	What a Lovely Day.
2	insurgent	Beatrice Prior must confront her inner demons ...	One Choice Can Destroy You
3	star wars: the force awakens	Thirty years after defeating the Galactic Empi...	Every generation has a story.
4	furious 7	Deckard Shaw seeks revenge against Dominic Tor...	Vengeance Hits Home

2. Data Preparation
   Convert string to lower case and remove white space at the beginning and at the end of the string of 'original_title' column

```
data['original_title'] = data['original_title'].str.strip()
data['original_title'] = data['original_title'].str.lower()
```

   To get the recommended movies that are similar to a specific movie, we will calculate the similarity scores for all movies based on their descriptions. Because the description contains a lot of helpful keywords to get a better similarity score. The description is a combination of the 'overview' feature and 'tagline' features in our dataset.

```
data['overview'] = data['overview'].fillna('')
data['tagline'] = data['tagline'].fillna('')
data['description'] = data['overview'] + data['tagline']
```

   This is the dataframe ready for modeling:

	original_title	description
0	jurassic world	Twenty-two years after the events of Jurassic ...
1	mad max: fury road	An apocalyptic story set in the furthest reach...
2	insurgent	Beatrice Prior must confront her inner demons ...
3	star wars: the force awakens	Thirty years after defeating the Galactic Empi...
4	furious 7	Deckard Shaw seeks revenge against Dominic Tor...

1.2.2 Modeling

We will use TF-IDF(Term Frequency-Inverse Document Frequency) for vectorization. The TF-IDF score is determined by the frequency of a word existing in a document, and by the number of documents that it occurs.

First, we will use TfIdfVectorizer from scikit-learn to initialize a TF-IDF Vectorizer Object and remove the stop words such as 'a', 'the', 'an'. Because they don't contain any helpful information:

```
tfidf = TfidfVectorizer(analyzer='word', stop_words='english')
```

We will then create a TF-IDF matrix by fitting and transforming the data

```
tfidf_matrix = tfidf.fit_transform(data['description'])
```

Let's check the shape and sample feature names of tfidf_matrix

```
print(tfidf_matrix.shape)
print(tfidf.get_feature_names()[5000:5010])
```

```
(10866, 33682)
['carta', 'cartel', 'cartels', 'carter', 'carthage', 'carthief', 'cartier', 'carting', 'cartographer', 'cartons']
```

From this output, we can see that there are 33,682 different words in our dataset and we have 10,866 movies.

We can then define the index of a movie in our DataFrame from the original_title

```
movie_indices = pd.Series(data.index, index=data['original_title']).drop_dupl
```

```
original_title
jurassic world                    0
mad max: fury road                1
insurgent                         2
star wars: the force awakens      3
furious 7                         4
the revenant                      5
terminator genisys                6
the martian                       7
minions                           8
inside out                        9
dtype: int64
```

To reduce response time, we will save tfidf matrix and movie indices to .pkl files so that we can use them for later.

```
save_obj(tfidf_matrix, TFIDF_MATRIX_FILE)
save_obj(movie_indices, MOVIE_INDICES_FILE)
```

We can now use the consine similarity and calculate the dot product between each vector to get the similarity score between two movies. The formula of the cosine similarity is:

$$similarity = cos(\theta) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\| \|\boldsymbol{v}\|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \sqrt{\sum_{i=1}^{n} v_i^2}}$$

```
cosine_similar = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
(10866, 10866)
[0.00586399 1.         0.         ... 0.         0.04012636 0.        ]
```

The output shows a matrix of 10866x10866, that means each movie description cosine similarity score with every other movie description. Therefore, each movie will be a 1x10866 column vector where each column will have a similarity score with each movie.

Then we convert into a list of tuples where the first element is its index, and the second is the similarity score. Then we sort the list of tuples based on the similarity scores to get the list of cosine similarity scores for that specific movie with every other movies.

```
similar_scores = list(enumerate(cosine_similar[movie_indices[movie.strip().lo
similar_scores.sort(key=lambda x: x[1], reverse=True)
```

We then get the top movies of this list, skip the first movie because it's referred to itself and return the original titles of the top movies.

```
indices = [i[0] for i in similar_scores[1:(n + 1)]]
similar_movies = [movie_indices.keys().values[i].title() for i in indices]

return similar_movies
```

Here is the result of the most similar movies:

```
print(get_n_similar_movies('    jurassic world    ', 5))
```

```
['Jurassic Park', 'Dark Ride', 'Futureworld', 'The Lost World: Jurassic Park', "Trailer Park Boys: Don'T Legalize It"]
```

Figure 9: Result

1.3 Collaborative filtering using Singular Value Decomposition (SVD): aka Recommended for you

Collaborative filtering will look at similarities between different users rating and predict those rating of them on other movies.

1.3.1 Dataset

The data used in the study can be obtained from Rating dataset [3]. The file name is *ratings.csv*

1.3.2 EDA

The data consists of 100836 if entries, each containing 4 features:

- userId: Id of user
- movieId: Id of movie
- rating: rate value
- timestamp: the time that the user rating

There are 610 users and 9724 movies in this dataset. The general information of the dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   userId     100836 non-null  int64  
 1   movieId    100836 non-null  int64  
 2   rating     100836 non-null  float64
 3   timestamp  100836 non-null  int64  
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

Figure 10: General information

based on above information, the dataset do not have any missing values. All features is numerical, so we do not need to do any encode method.

The *timestamp* feature is not need for prediction so we drop this feature. Let see the range value of *rating* feature.

	userId	movieId	rating
count	100836.000000	100836.000000	100836.000000
mean	326.127564	19435.295718	3.501557
std	182.618491	35530.987199	1.042529
min	1.000000	1.000000	0.500000
25%	177.000000	1199.000000	3.000000
50%	325.000000	2991.000000	3.500000
75%	477.000000	8122.000000	4.000000
max	610.000000	193609.000000	5.000000

Figure 11: Describe of dataset

The range of *rating* feature is 0.5 to 5, it is good. Because there are some case the rating can have range such as 0 to 10. So for this dataset, we do not do any scale method on *rating* feature

Let see the distribution of rating on the dataset.

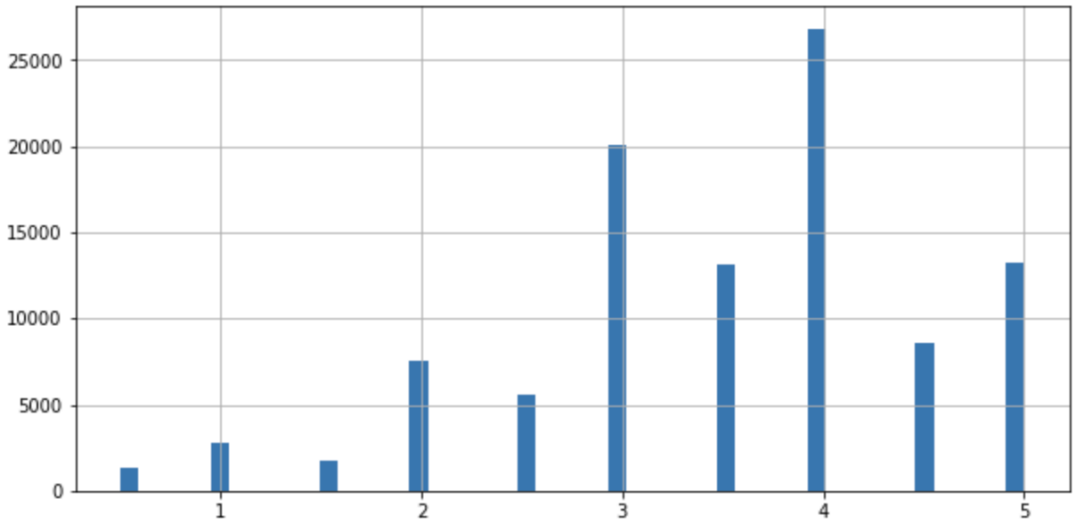


Figure 12: Rating distribution

Our purpose is find the most movies the user will like that mean the rating must larger than the median 2.5, so there is a lot of users rated larger than 2.5 that point out this dataset is suitable for us to go.

When look on the rating count of each movie

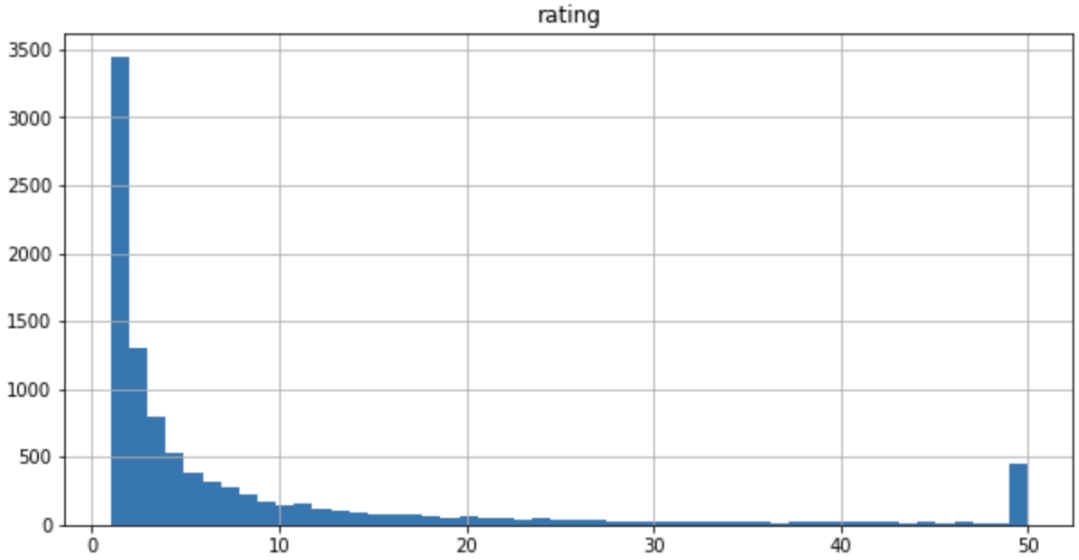


Figure 13: Rating count by movie

In this figure, we do small step for each values that larger than 50 will be set to it for easy to view. Based on the figure there are many movies were rated only once. it is more than 3400 in total 9724 movies. Let see below picture

	rating_count
count	9724.000000
mean	10.369807
std	22.401005
min	1.000000
25%	1.000000
50%	3.000000
75%	9.000000
max	329.000000

Figure 14: Rating count by movie describe

As we mention above there are a lot of movie have been rated once. Let see about number of rating of each user

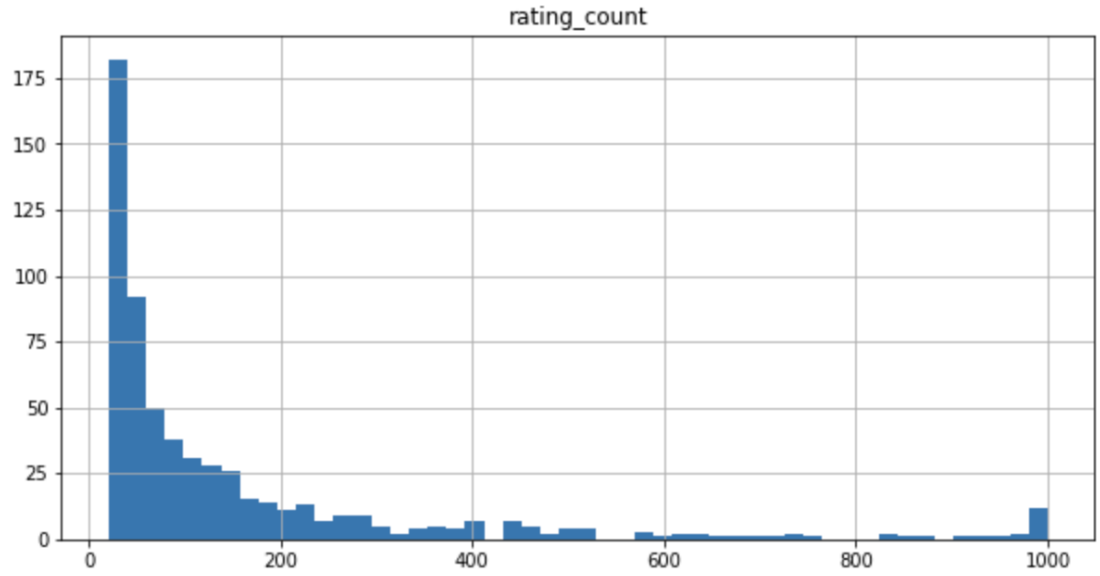


Figure 15: Rating count by user

We do small step to for each value larger than 1000 will be set to it for easy to view. There are a lot of users who have rated 20 times. Ratings count by users gradually decreases which means few users have rated many movies.

We think that what we need is the list of movies that the user maybe like, so we only care about those users that similar on their interest not on their do not like. So we remove those entries that have rating less than 2.5 to reduce time training

1.3.3 Singular Value Decomposition

The Singular-Value Decomposition, is a matrix decomposition method for reducing a matrix to its constituent parts in order to make certain subsequent matrix calculations simpler.

We use the implementation of *Surprise* library for SVD and we also using Grid Search to find the best hyper parameters

In previous section, we think about reduce the number of entries to improve speed, so let compare those result on same testset, first for learning full rating range from 0 to 5.

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.6518  0.6619  0.6637  0.6538  0.6434  0.6549  0.0073
Fit time          0.89    0.88    0.88    0.89    0.88    0.88    0.00
Test time         0.03    0.03    0.03    0.03    0.03    0.03    0.00
```

Figure 16: SVD full cross validation

And the result of learning only rating larger or equal 2.5

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.6530  0.6582  0.6560  0.6514  0.6591  0.6556  0.0029
Fit time          0.86    0.84    0.87    0.85    0.87    0.86    0.01
Test time         0.06    0.03    0.03    0.03    0.03    0.03    0.01
```

Figure 17: SVD with remove less than 2.5 cross validation

Both result are similar and the RMSE is small so the prediction reliable

1.4 Collaborative filtering using Neural Network: aka Recommended for you

Now we will continue the report by using the popular Keras framework (integrated into tensorflow) to build a recommender system. We will take advantage of the power of a modern computation framework like Keras to implement the recommender with minimal code. We will build a true neural network and see how it compares to the collaborative filtering approach.

we can get started with some imports and reading in the ratings.csv file, which is where the data for this task comes from.

	userId	movieId	rating
0	1	1	4.0
1	1	3	4.0
2	1	6	4.0
3	1	47	5.0
4	1	50	5.0
...	...	...	...
100831	610	166534	4.0
100832	610	168248	5.0
100833	610	168250	5.0
100834	610	168252	5.0
100835	610	170875	3.0

Figure 18: Reading the data rating.scv

The data is tabular and consists of a user ID, a movie ID, and a rating (there's also a timestamp but we won't use it for this task). Our task is to predict the rating for a user/movie pair, with the idea that if we had a model that's good at this task then we could predict how a user would rate movies they haven't seen yet and recommend movies with the highest predicted rating.

Our dataset also includes movies.csv that is a listing of movies and their associated genres. We do not actually need this for the model but it's useful to know about.

To get a better sense of what the data looks like, we can turn it into a table by selecting the top 15 users/movies from the data and joining them together. The result shows how each of the top users rated each of the top movies.

movieId	1	50	110	260	296	318	356	480	527	589	593	1196	2571	2858	2959
userId															
68	2.5	3.0	2.5	5.0	2.0	3.0	3.5	3.5	4.0	3.5	3.5	5.0	4.5	5.0	2.5
182	4.0	4.5	3.5	3.5	5.0	4.5	5.0	3.5	4.0	2.0	4.5	3.0	5.0	5.0	5.0
249	4.0	4.0	5.0	5.0	4.0	4.5	4.5	4.0	4.5	4.0	4.0	5.0	5.0	4.5	5.0
274	4.0	4.0	4.5	3.0	5.0	4.5	4.5	3.5	4.0	4.5	4.0	4.5	4.0	5.0	5.0
288	4.5	NaN	5.0	5.0	5.0	5.0	5.0	2.0	5.0	4.0	5.0	4.5	3.0	NaN	3.5
307	4.0	4.5	3.5	3.5	4.5	4.5	4.0	3.5	4.5	2.5	4.5	3.0	3.5	4.0	4.0
380	5.0	4.0	4.0	5.0	5.0	3.0	5.0	5.0	NaN	5.0	5.0	5.0	4.5	NaN	4.0
387	NaN	4.5	3.5	4.5	5.0	3.5	4.0	3.0	NaN	3.5	4.0	4.5	4.0	4.5	4.5
414	4.0	5.0	5.0	5.0	5.0	5.0	5.0	4.0	4.0	5.0	4.0	5.0	5.0	5.0	5.0
448	5.0	4.0	NaN	5.0	5.0	NaN	3.0	3.0	NaN	3.0	5.0	5.0	2.0	4.0	4.0
474	4.0	4.0	3.0	4.0	4.0	5.0	3.0	4.5	5.0	4.0	4.5	5.0	4.5	3.5	4.0
599	3.0	3.5	3.5	5.0	5.0	4.0	3.5	4.0	NaN	4.5	3.0	5.0	5.0	5.0	5.0
603	4.0	NaN	1.0	4.0	5.0	NaN	3.0	NaN	3.0	NaN	5.0	3.0	5.0	5.0	4.0
606	2.5	4.5	3.5	4.5	5.0	3.5	4.0	2.5	5.0	3.5	4.5	4.5	5.0	4.5	5.0
610	5.0	4.0	4.5	5.0	5.0	3.0	3.0	5.0	3.5	5.0	4.5	5.0	5.0	3.5	5.0

Figure 19: Visualization of the variables: movieId, userId and rating

We use Min-max scaling for normalize the values of rating feature that will finally range from 0 to 1, which help Machine Learning algorithm perform well. The result of the transformation is now shown as follows:

movieId	1	50	110	260	296	318	356	480	527	589	593	1196	2571	2858	2959
userId															
68	0.444444	0.555556	0.444444	1.000000	0.333333	0.555556	0.666667	0.666667	0.777778	0.666667	0.666667	1.000000	0.888889	1.000000	0.444444
182	0.777778	0.888889	0.666667	0.666667	1.000000	0.888889	1.000000	0.666667	0.777778	0.333333	0.888889	0.555556	1.000000	1.000000	1.000000
249	0.777778	0.777778	1.000000	1.000000	0.777778	0.888889	0.888889	0.777778	0.888889	0.777778	0.777778	1.000000	1.000000	0.888889	1.000000
274	0.777778	0.777778	0.888889	0.555556	1.000000	0.888889	0.888889	0.666667	0.777778	0.888889	0.777778	0.888889	0.777778	1.000000	1.000000
288	0.888889	NaN	1.000000	1.000000	1.000000	1.000000	1.000000	0.333333	1.000000	0.777778	1.000000	0.888889	0.555556	NaN	0.666667
307	0.777778	0.888889	0.666667	0.666667	0.888889	0.888889	0.777778	0.666667	0.888889	0.444444	0.888889	0.555556	0.666667	0.777778	0.777778
380	1.000000	0.777778	0.777778	1.000000	1.000000	0.555556	1.000000	1.000000	NaN	1.000000	1.000000	1.000000	0.888889	NaN	0.777778
387	NaN	0.888889	0.666667	0.888889	1.000000	0.666667	0.777778	0.555556	NaN	0.666667	0.777778	0.888889	0.777778	0.888889	0.888889
414	0.777778	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.777778	0.777778	1.000000	0.777778	1.000000	1.000000	1.000000	1.000000
448	1.000000	0.777778	NaN	1.000000	1.000000	NaN	0.555556	0.555556	NaN	0.555556	1.000000	1.000000	0.333333	0.777778	0.777778
474	0.777778	0.777778	0.555556	0.777778	0.777778	1.000000	0.555556	0.888889	1.000000	0.777778	0.888889	1.000000	0.888889	0.666667	0.777778
599	0.555556	0.666667	0.666667	1.000000	1.000000	0.777778	0.666667	0.777778	NaN	0.888889	0.555556	1.000000	1.000000	1.000000	1.000000
603	0.777778	NaN	0.111111	0.777778	1.000000	NaN	0.555556	NaN	0.555556	NaN	1.000000	0.555556	1.000000	1.000000	0.777778
606	0.444444	0.888889	0.666667	0.888889	1.000000	0.666667	0.777778	0.444444	1.000000	0.666667	0.888889	0.888889	1.000000	0.888889	1.000000
610	1.000000	0.777778	0.888889	1.000000	1.000000	0.555556	0.555556	1.000000	0.666667	1.000000	0.888889	1.000000	1.000000	0.666667	1.000000

Figure 20: The result of the transformation of Min-Max Scaller

The user/movie fields are currently non-sequential integers representing some unique ID for that entity. We also need them to be sequential starting at zero to use for modeling. We can use scikit-learn's LabelEncoder class to transform the fields.

Data prepossessing methods including feature scaling and normalization is done at this stage. Now we get to the model itself.

1.4.1 Using Dot Product only without any dense layers

The layers that we use in the model can be seen briefly as follows:

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 1, 50)        30500       input_1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 50)        486200      input_2[0][0]
__________________________________________________________________________________________________
reshape (Reshape)               (None, 50)           0           embedding[0][0]
__________________________________________________________________________________________________
reshape_1 (Reshape)             (None, 50)           0           embedding_1[0][0]
__________________________________________________________________________________________________
dot (Dot)                       (None, 1)            0           reshape[0][0]
                                                                 reshape_1[0][0]
==================================================================================================
Total params: 516,700
Trainable params: 516,700
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 21: the structure of layers in the model 1

We can build the model by computing the dot product between a user vector and a movie vector to get a predicted rating. The code is fairly simple, there isn't even a traditional neural network layer or activation involved.

Below is the graph showing learning curve of the trained model:

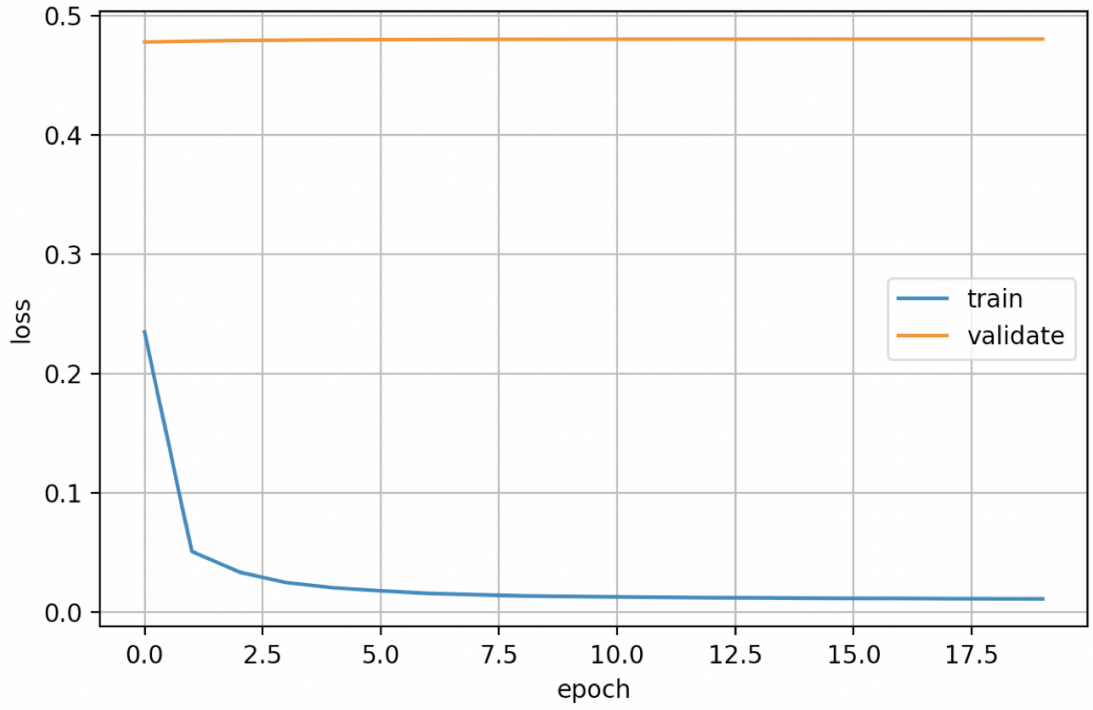


Figure 22: the learning curve of the model 1

By training the model this way, the model learned well as the loss curve of the train set goes down, but it generalizes pretty bad since the loss curve of the test set witnessed a stability.

1.4.2 Using Neural Network with Dense Layers

Instead of taking the dot product of the embedding vectors, we just concatenated the embeddings together and stick a fully connected layer on top of them, which would make it a neural network. To modify the code, we just concatenated on the embedding layers, then add some dropout, insert a dense layer, and stick some dropout on the dense layer. Finally, we run it to examine its performance.

The layers that we use in the model can be seen briefly as follows:

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 1, 10)        6100        input_1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 10)        97240       input_2[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 10)           0           embedding[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 10)           0           embedding_1[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 20)           0           flatten[0][0]
                                                                 flatten_1[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 20)           0           concatenate[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 32)           672         dropout[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 32)           0           dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 16)           528         dropout_1[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 16)           0           dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 1)            17          dropout_2[0][0]
==================================================================================================
Total params: 104,557
Trainable params: 104,557
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 23: the structure of layers in the model 2

The main idea here is we are going to use embeddings to represent each user and each movie in the data.

Below is the graph showing learning curve of the trained model:

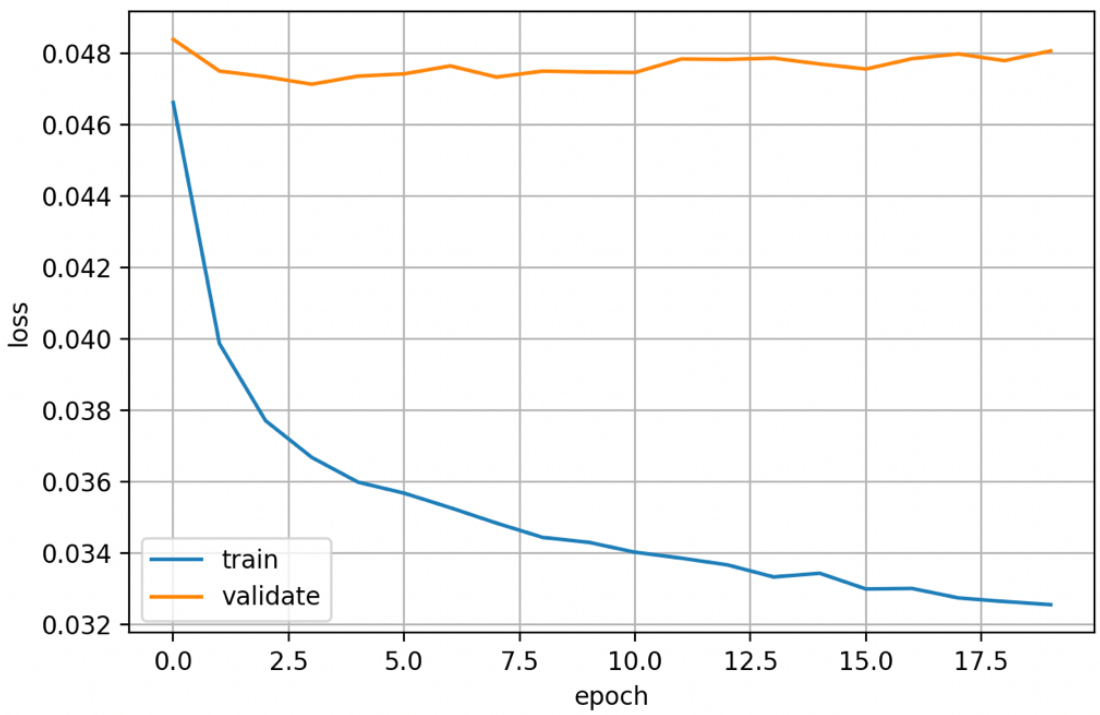


Figure 24: the learning curve of the model 2

The model that was trained by neural network with dense layers performed much better compared with the one without dense layers because the loss curve of the one with dense layers is just ranging around 0.048 which is way much smaller than the one without dense layers.

Let us fine tune the model with the value of the learning_rate decreasing gradually from 0.01, 0.001 to 0.0001 to find the best learning_rate for our algorithm:

1. Learning_rate = 0.01 and epoch = 20

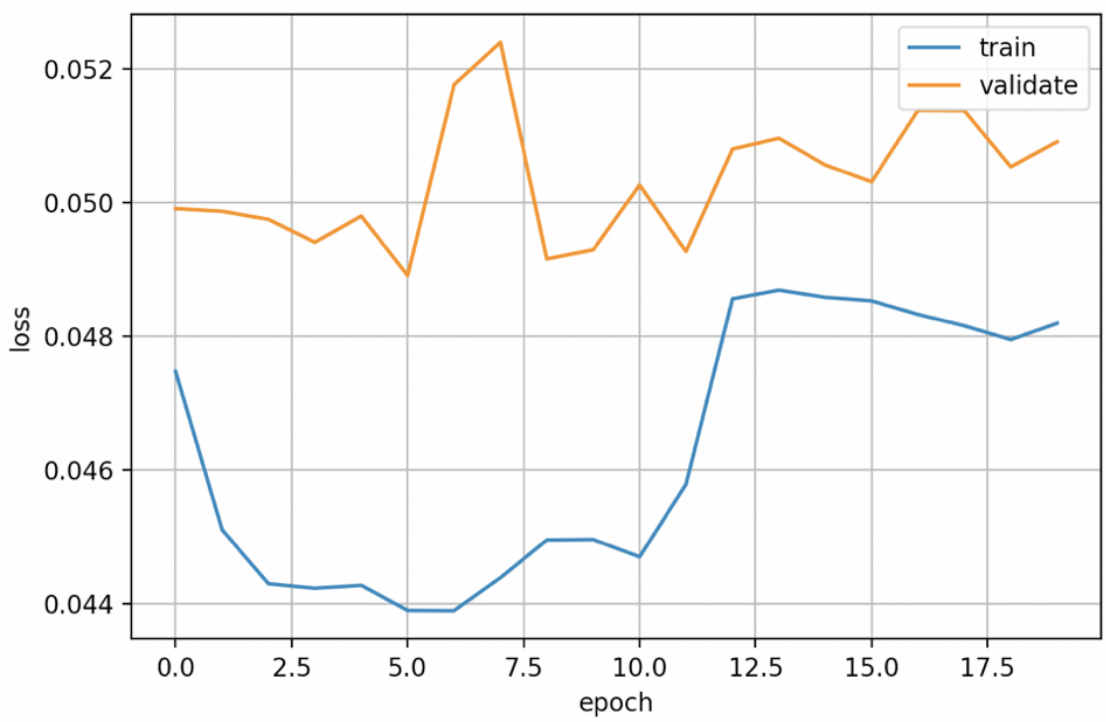


Figure 25: the learning curve of the model 2 with learning rate = 0.01, epoch = 20

Execution Time: 96.9178807735

As we can see from the chart, the two lines of training and validation go up, so learning rate = 0.01, epoch = 20 is not good for training the model.

2. Learning_rate = 0.001 and epoch = 20

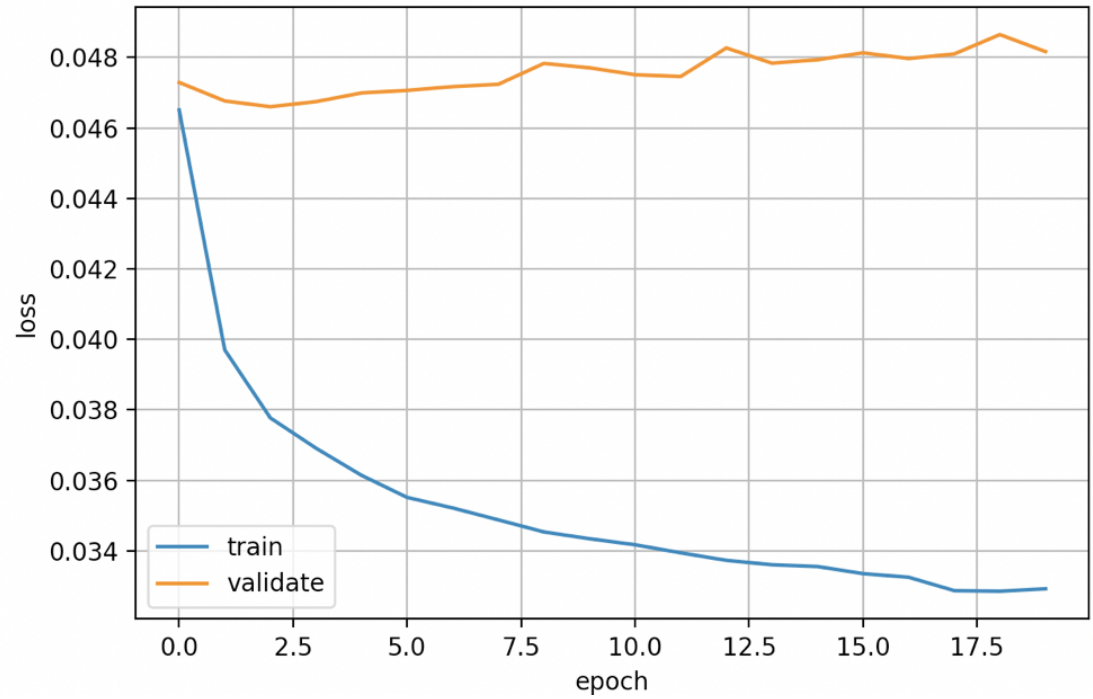


Figure 26: the learning curve of the model 2 with learning rate = 0.001, epoch = 20

Execution time: 101.03255558

As we can see from the chart, although the training line goes down gradually, but the more the epoch is, the more the loss we get in the validation line, which is not really good for our model with learning rate = 0.001, epoch = 20.

3. Learning_rate = 0.0001 and epoch = 20

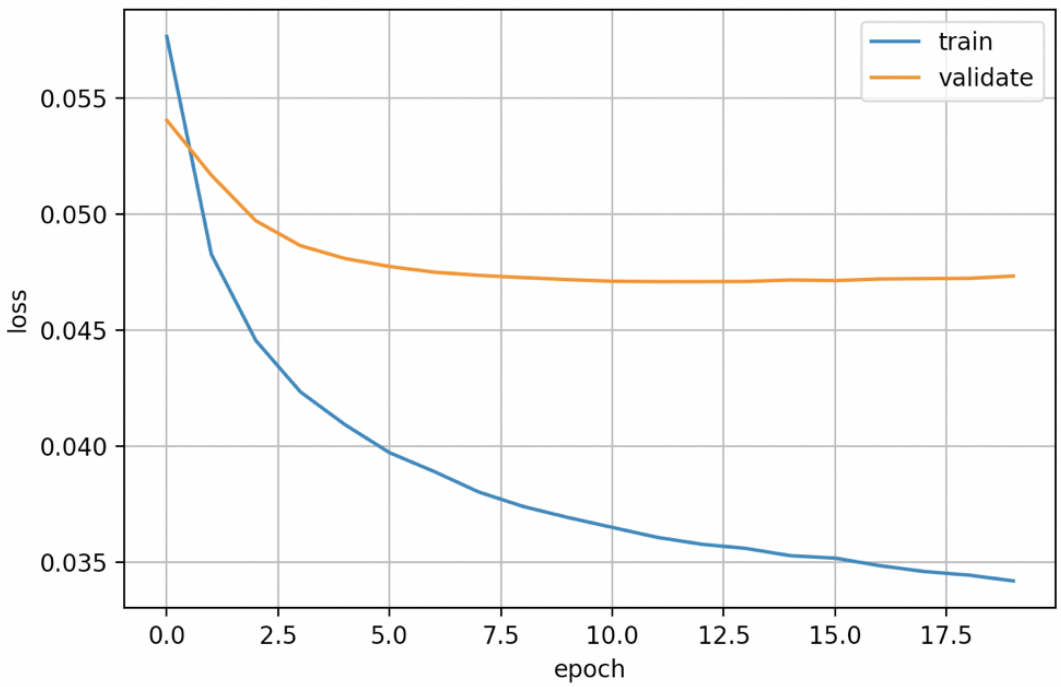


Figure 27: the learning curve of the model 2 with learning rate = 0.0001, epoch = 20

Execution time: 101.093003035

As we can see from the chart, the more the epoch is, the less the loss is in both training and validation line. One more important point is that when the epoch is greater than 6, the validation line does not go down any more but keep stable at about 0.047 in loss. Furthermore, we also know that the less the learning_rate is, the more the executive time is, so we should consider to set the right learning rate depending on our own model or our own dataset.

Let us try to train the model with the same learning_rate (0.0001), but with the epoch = 6 to examine the performance of the algorithm.

4. Learning Learning_rate = 0.0001 and epoch = 6

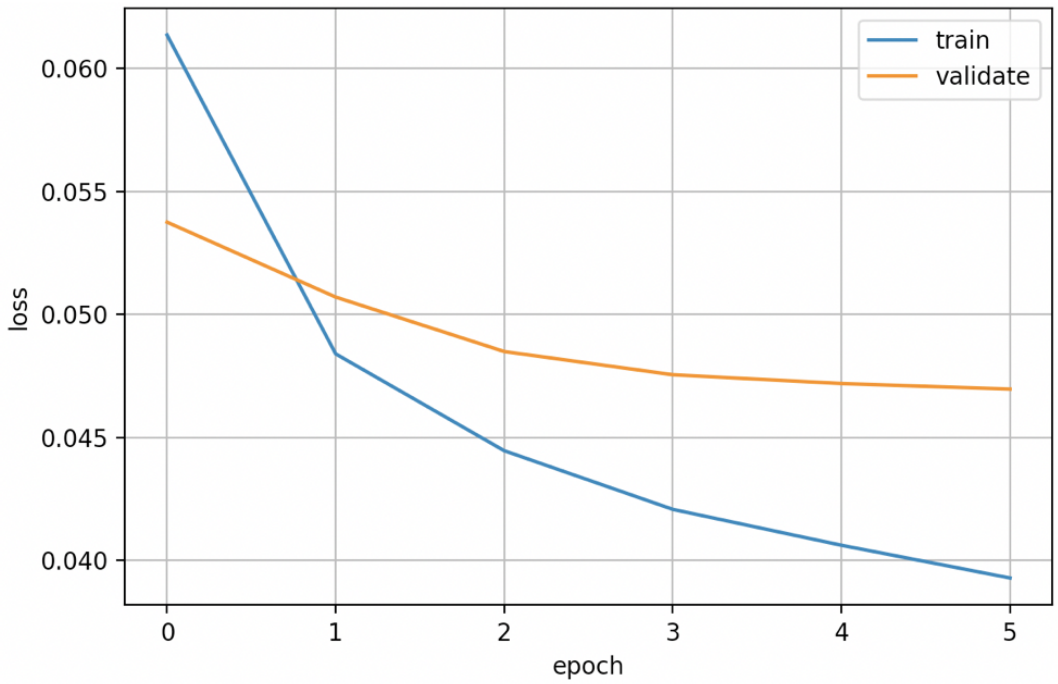
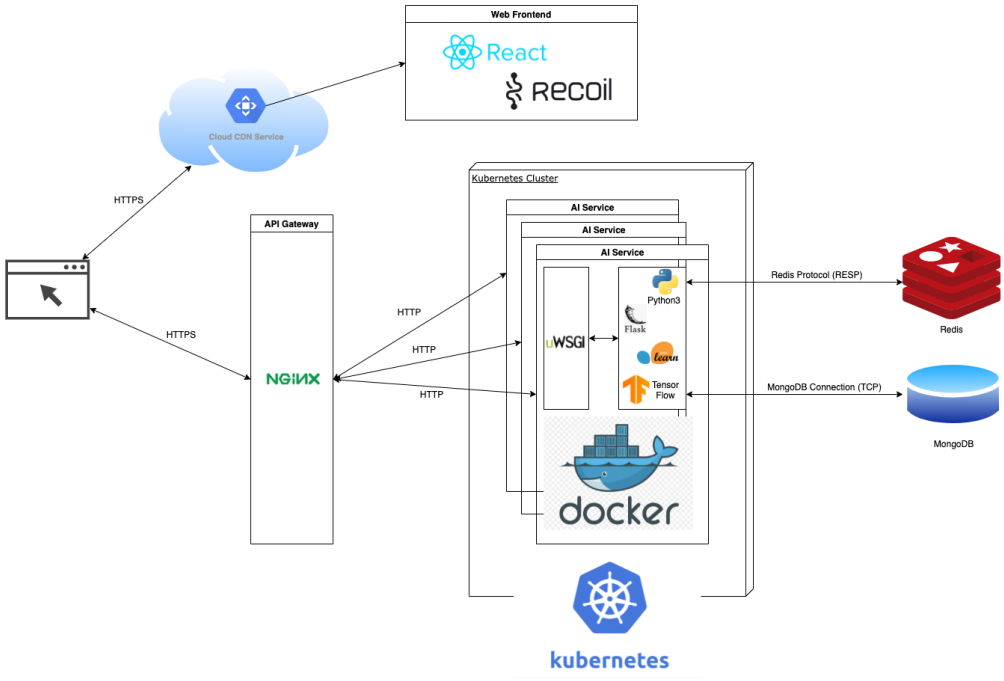


Figure 28: the learning curve of the model 2 with learning rate = 0.0001, epoch = 6

Executive time: 28.9866921902

As we can see from the chart, there is no difference between the previous chart and this chart in loss. However, the executive time now is much shorter than the previous one (learning rate = 0.0001, epoch = 20), which increase the performance of the algorithm in general.

1.5 Model serving



List of our APIs for our movie recommendation system:

1. Demographic

   - Get list of trending movies in recent 3 years and older than 3 years based on popularity (default: top=10, all genres)

     `http://localhost:8080/recommender/trend/now?genres=<string>&top=<number>`

   - Get list of trending movies based on popularity (default: top=10, all genres)

     `http://localhost:8080/recommender/trend/popular?genres=<string>&top=<number>`

   - Get list of trending movies based on IMDB rating scores (default: top=10, all genres)

     `http://localhost:8080/recommender/trend/rating?genres=<string>&top=<number>`

2. Content Based

   - Get list of the most similar movies for a movie (default: top=10)

     `http://localhost:8080/recommender/similar?imdbId=<string>&top=<number>`

   - Fit again TF-IDF matrix and movie indices when add new movie data

`http://localhost:8080/recommender/train/tfidf`

3. Collaborative filtering

   - Get predicted rating of a user for a movie

     `http://localhost:8080/recommender/predict/rating?userId=<number>&movieId=<number>`

   - Get list of recommended movies for a user (default: top=10, all genres, watched=false)

     `http://localhost:8080/recommender/user/<user_id>?genres=<string>&top=<number`

   - Train again predicted rating model when add new rating data 1.3

     `http://localhost:8080/recommender/train/rating`

4. Hybrid Recommendation

   - Get list of recommended movies for a user if the user have watched the movie (default: top=10)

     `http://localhost:8080/recommender/user/<user_id>/watched?imdbId=<string>&top=<number>`

5. Others

   - Get list of all user's ids

     `http://localhost:8080/user/list`

   - Get list of all watched movies of a user with pagination (default: all genres, page=1)

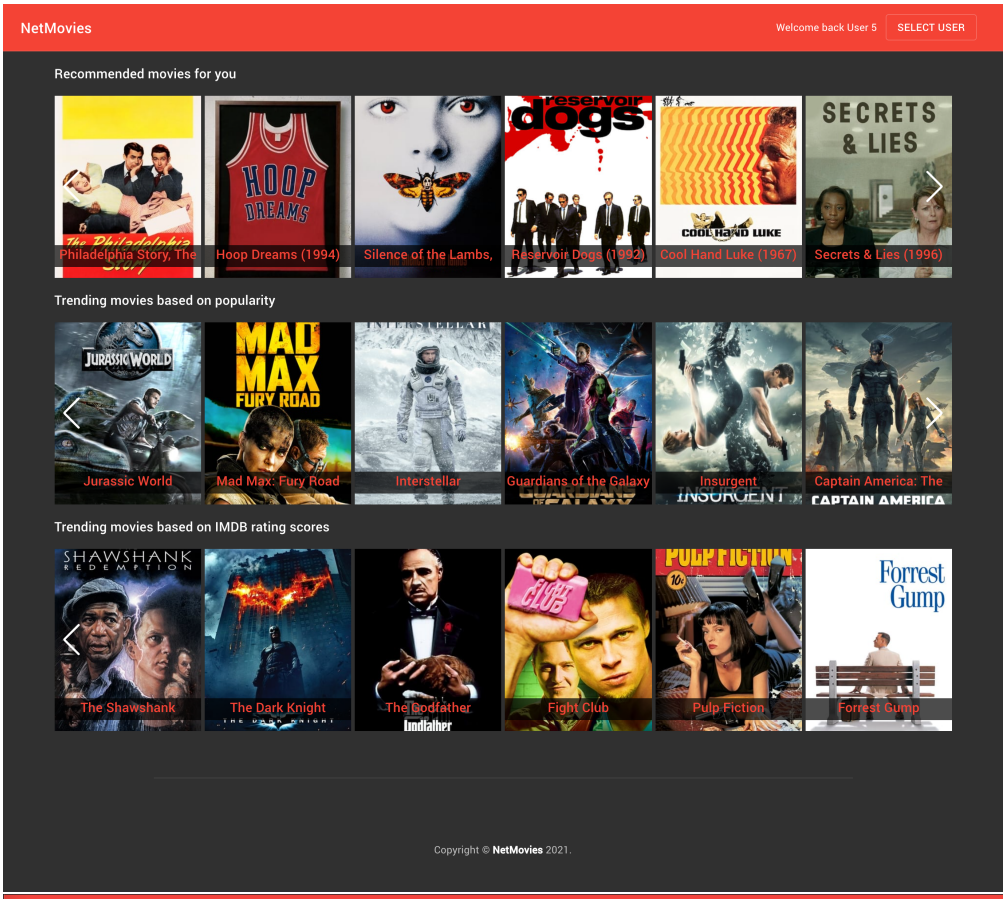
     `http://localhost:8080/user/<user_id>/watched?genres=<string>&page=<number>`

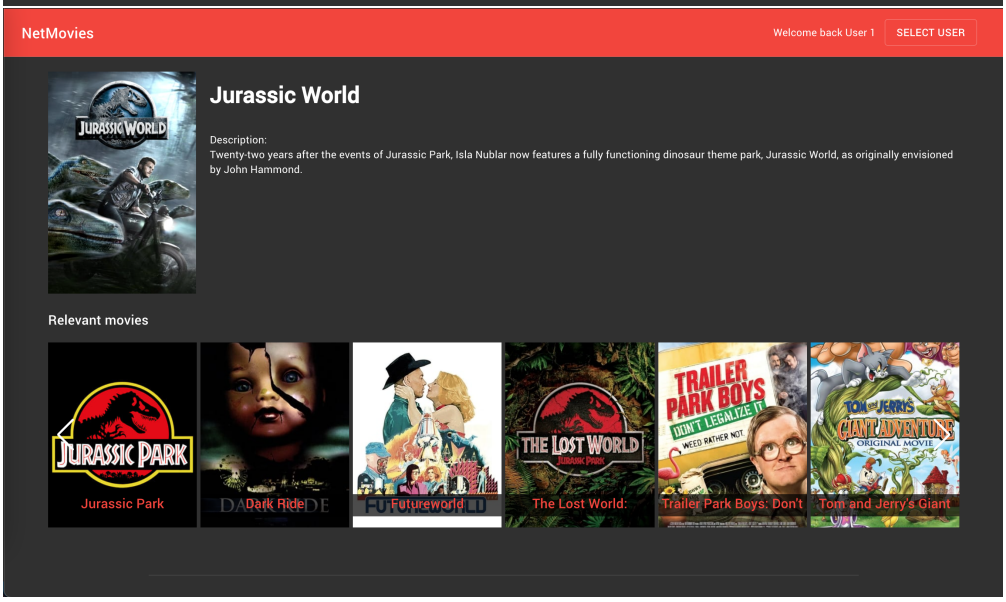
   - Clear cache APIs

     `http://localhost:8080/caching/delete/<:key>`
     `http://localhost:8080/caching/clear`

1.6 Bonus

1. Integrated CI/CD pipeline
2. Implemented Front-end User Interface





3. Implemented Monitor Redis Cache statistics

4. Hydrid recommendation

2 Conclusion

Every technique has its own advantages and disadvantages. Demographic is considered to be the simplest among the others, but Content Based, Collaborative and the Hybrid one give more personalised movie recommendations, thus enhance the recommendation process and lead to improved predictions. The biggest issue that recommendation systems facing probably is that they need a lot of data to effectively make recommendations.Through this project, we have learned recommender systems techniques, served it via a Restful API and practiced embedding and MLOps.

References

[1] J. Shakir, "Tmdb movies dataset." https://www.kaggle.com/juzershakir/tmdb-movies-dataset.

[2] "Tmdb movies dataset." https://www.kaggle.com/juzershakir/tmdb-movies-dataset.

[3] "Rating dataset." http://files.grouplens.org/datasets/movielens/ml-latest-small.zip.