Capstone Project 2 : Final Report

1. Define the Problem

Recommendation engine is a very broad topic. There are lots of algorithms dealing with this ideal. The goal of this project is to build a movie recommendation system for users based on the MovieLens data set (download link).

User Cold-start Problem
The cold-start problem is often seen in Recommender Systems because methods such as collaborative filtering rely heavily on past user-item interactions. Due to the lack of item information of movies and user personal information, the popularity model will be applied for new users.

Recommendation for Known Users
In this project, a user-based collaborative filtering method will be addressed with applying different similarity algorithms and dimension reduction.

2. Prepare Data for Consumption

2.1 Gather the Data

MovieLens data set download link

The datasets describe ratings and free-text tagging activities from MovieLens, a movie recommendation service. 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. Users were selected at random for inclusion. All selected users had rated at least 20 movies.

2.2. Prepare Data for Consumption

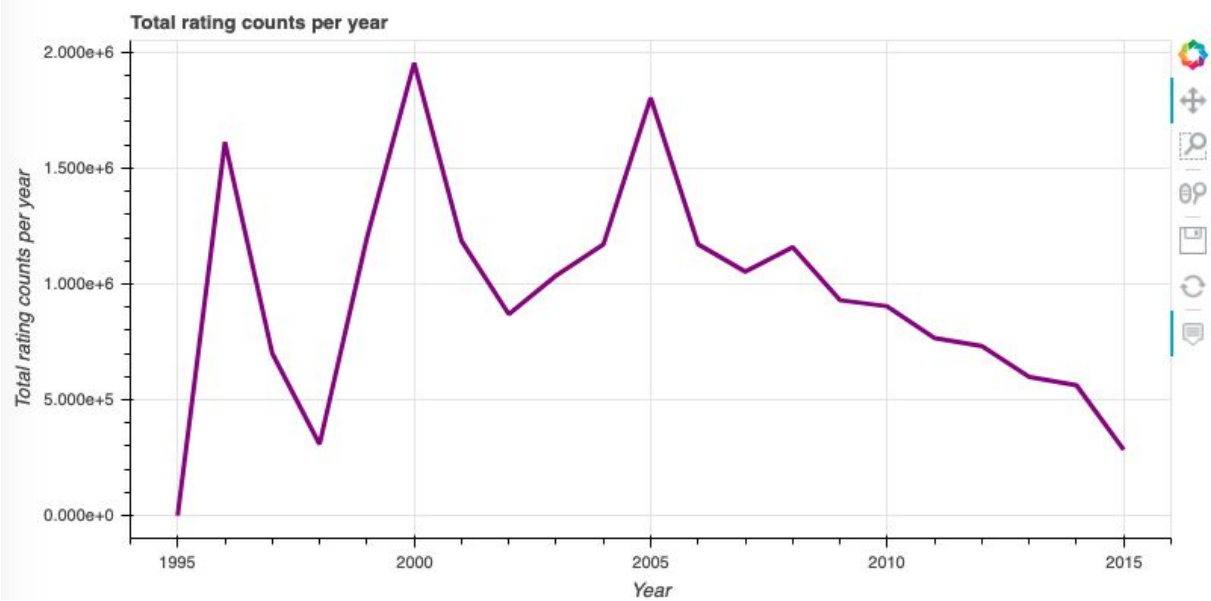
1. Load original CSV files.
2. Split title and year into separate columns. Convert year to datetime.
3. Categorize genres properly: split strings into boolean columns per genre.
4. Modify the rating timestamp: from universal seconds to datetime year.

5. Check for NaN values.

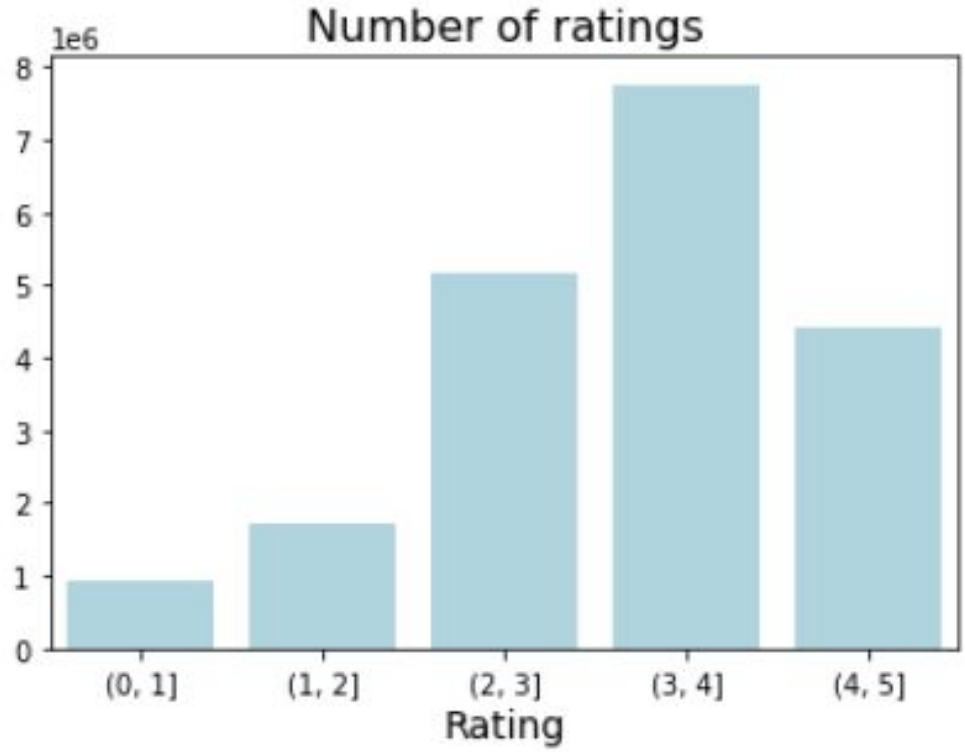
3. Perform Exploratory Analysis

Yearly trending of released movies

Yearly trending of total ratings



Bar chart of rating buckets



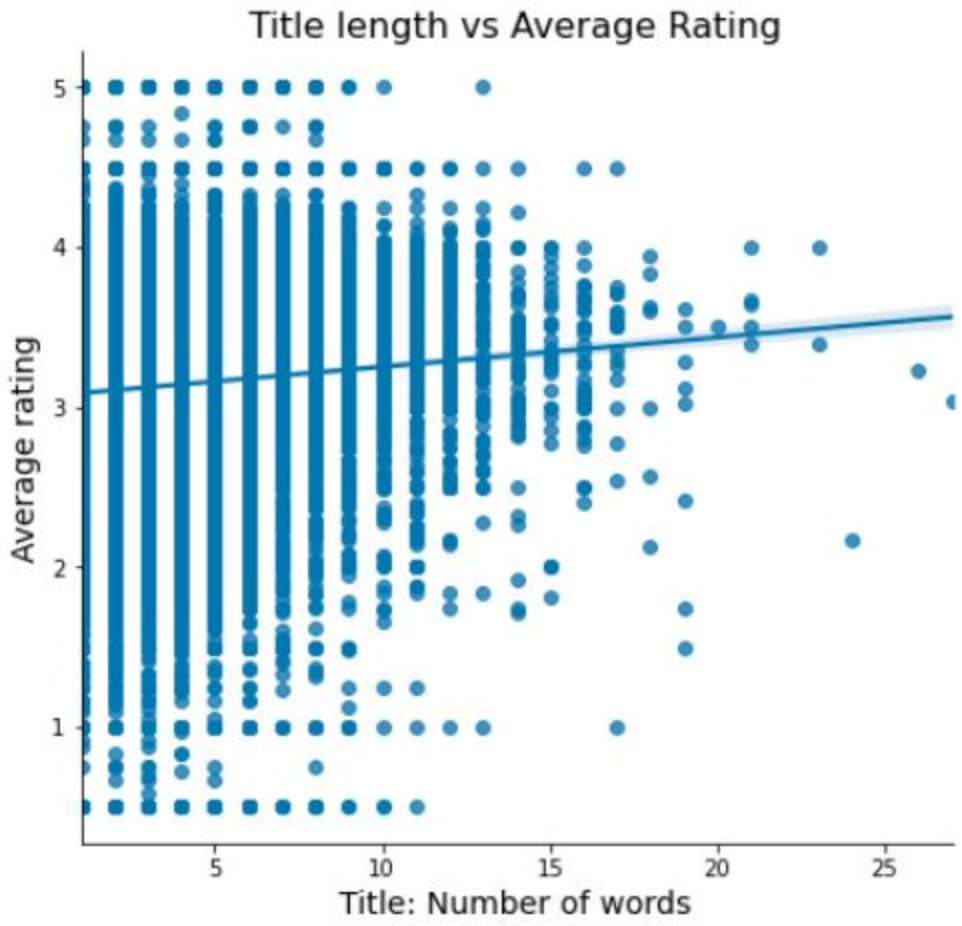
Percentage table:

```
# pecentage
df_ratings['rating_bins'].value_counts().sort_index(ascending=False)/len(df_ratings)
```

```
(4, 5]    0.221671
(3, 4]    0.388099
(2, 3]    0.258726
(1, 2]    0.085511
(0, 1]    0.045992
Name: rating_bins, dtype: float64
```

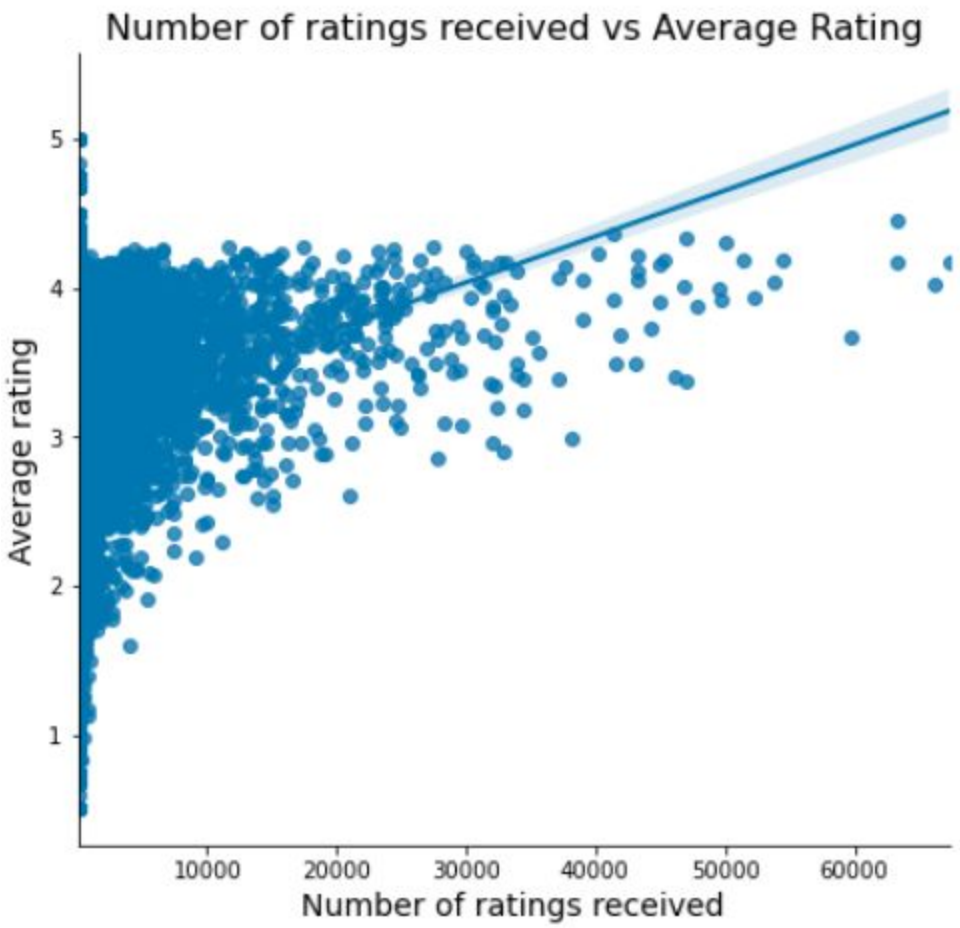
This is good news that the majority of the ratings fall into range 3+ ~ 5. This means most of the audience love the movies on the website.

Does the length of the title influence the average rating?



Correlation coefficient: 0.066
We got a very small positive correlation coefficient which means **a**s the number of words in the title increases, so does the average rating.

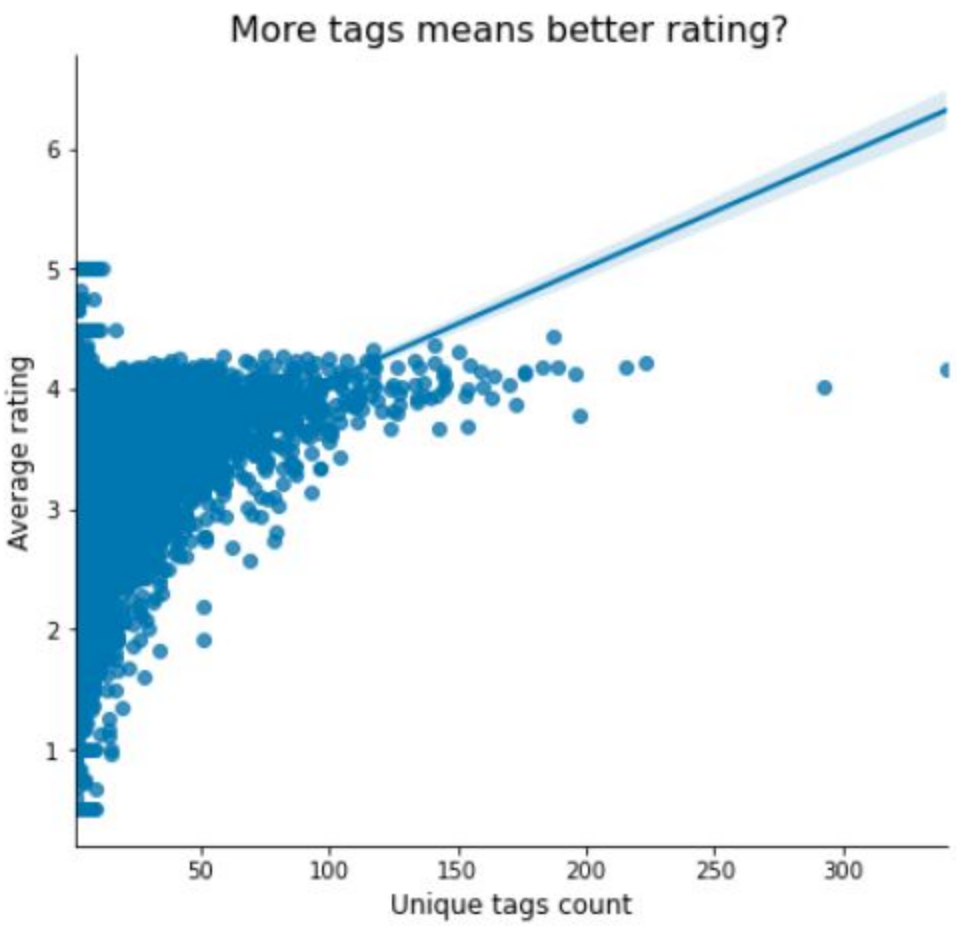
How does the average rating of a movie relate to the number of ratings it has received?



Correlation coefficient: 0.066

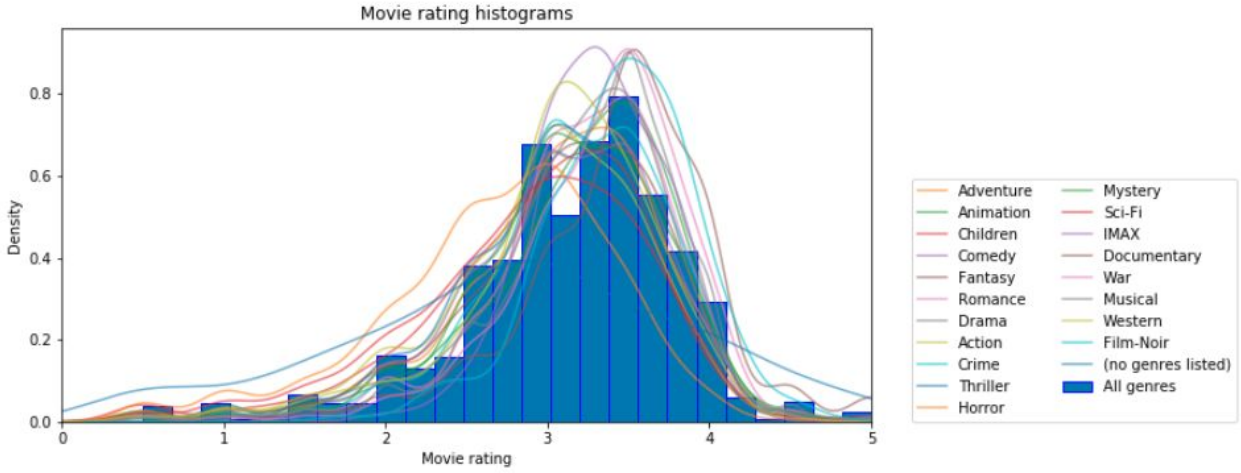
We got a very small positive correlation coefficient which means as the number of ratings received increases, so does the average rating.

How does more unique tags per movie mean the movie having better ratings?



Correlation coefficient: 0.26637361301113666

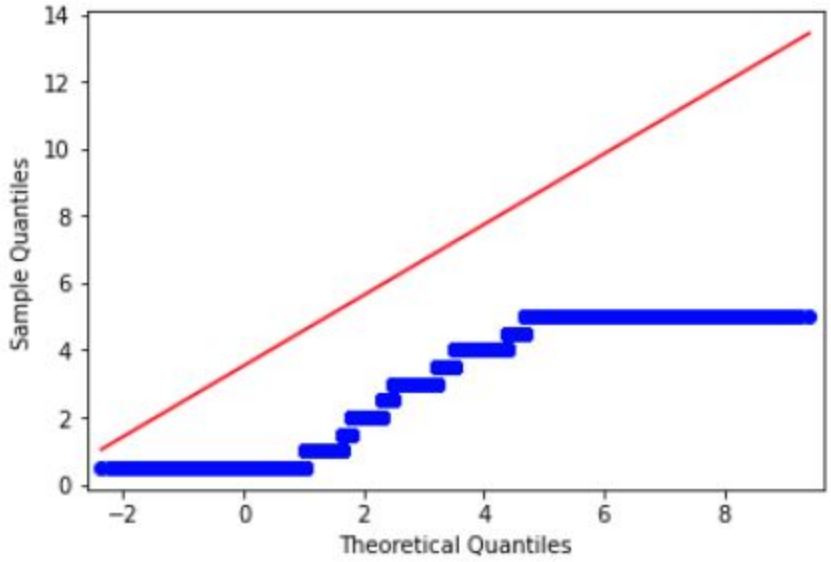
Rating distribution by genres



The plot shows us a left-skewed distribution for all genres and each genre follows the left-skewed shape as well.

Check Normality

```
import pylab
import statsmodels.api as sm
mu=np.mean(df_ratings['rating'].dropna())
sigma=np.var(df_ratings['rating'].dropna())
sm.qqplot(df_ratings['rating'], loc = mu, scale = sigma, line='s')
pylab.show()
```




```
from scipy.stats import kurtosis
from scipy.stats import skew
print("mean : ", np.mean(df_ratings['rating'].dropna()))
print("var  : ", np.var(df_ratings['rating'].dropna()))
print("skew : ",skew(df_ratings['rating'].dropna()))
print("excess kurtosis : ",kurtosis(df_ratings['rating'].dropna()))
```

```
mean :  3.5255287
var  :  1.1066805
skew :  -0.6553115248680115
excess kurtosis :  0.13746752211657665
```

1. we got negative skewness which means the mass of th distribution is concentrated on the right which fits the plog above

2. Kurtosis is a measure of the thickness of the tails of a distribution. Excess kurtosis = kurtosis - 3. Excess kurtosis is not ZERO telling us that the data is not normally distributed.

We are able to tell that the data is not normal by plot QQ plot and calculate skewness and excess kurtosis. However, let's still run a normality test to double check our conclusion.

By conducting Kolmogorov–Smirnov test with alpha = 0.05

```
stats.kstest(df_ratings['rating'].dropna(),'norm', args=(mu, sigma))
```

```
KstestResult(statistic=0.16570544657699904, pvalue=0.0)
```

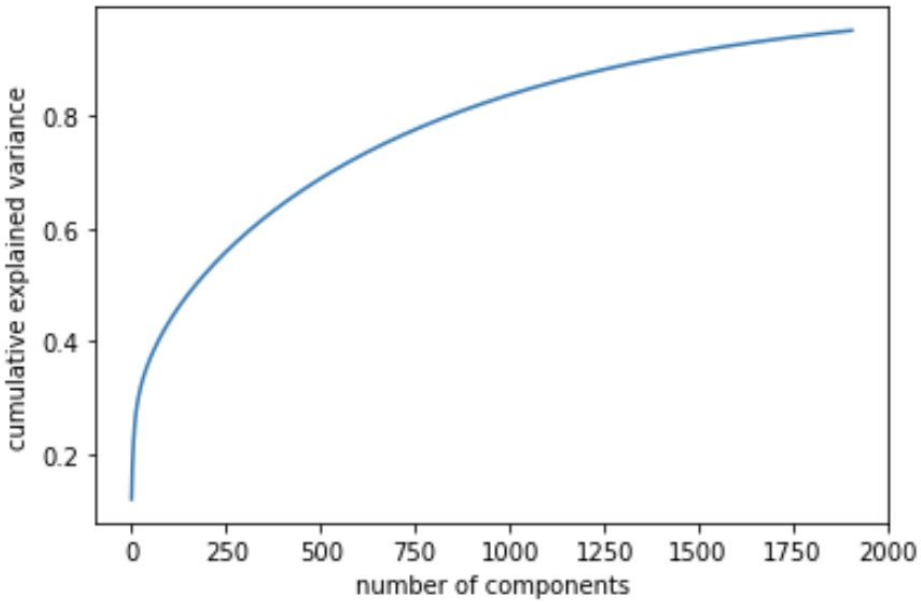
We rejected the null hypothesis due to p value < .05

4. Feature Engineering

4.1 Dimensionality Reduction - PCA

Because the Sklearn library NearestNeighbors(metric='correlation') can not handle the sparse matrix. The PCA technique has been applied to reduce the feature dimensions.

Before applying PCA, the original train dataframe has 4880 features which represent the ratings for each movie. I decided to keep 95% of the variance during the PCA transformation. Then I got a transformed train dataframe with 1906 features. More than half of the features are being reduced.



4.2 Dimensionality Reduction - NNMF

Lightfm handles

5. Evaluation metrics

There are lots of evaluation metrics to evaluate the model performance. For discrete predictions, precision, recall, Area Under Curve (AUC), and F1 score at top K recommended items are widely used. For continuous predictive outcomes, there are metrics like RMSE, MAE and etc.
In this project, the customized evaluation score formula below is derived at top K recommended items.

Evaluation score: $J(A,B) = |A \cap B| / \min(A, B)$

A is a movie list that each user watched
B is a recommended movie list for each user.

Evaluation score ranges from 0 (the lowest) to 1 (the highest).

We are interested in the items that show in both lists because we are targeting recommended items that known users would like to watch. So this formula will penalize the missing items from the recommended list but they are on the known user's list. For example, user i in the test dataset has 60 watched movies and only 30 of them on the recommended list (total 50 items). Then the evaluation score for user i = 30/60 = 0.5.

On the other hand, this formula won't penalize the extra items on the recommendation list. For instance, user j in the test dataset has 30 watched movies and all of them are on the recommendation list (total 50 items), Then the evaluation score for user j = 30/30 = 1. We came up with this idea based on the assumption that extra recommended items won't bring a negative impact on users experience once the user's favourite items are covered.

In order to model an actual business environment, the two years data is used for training and the following full year data will be used as testing. The purpose is trying to gauge by using the two years' data how the model performs for recommending movies for the coming users (new or known).

6. Hyperparameter tuning and model selection

(1) KNN with cosine similarity
Applied grid search on:
nb = [5,10,15,20]
topk = [25, 50, 75]

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Now, Regular Cosine Similarity by the definition reflects differences in direction, but not the location. Therefore, using the cosine similarity metric does not consider for example

the difference in ratings of users. Adjusted cosine similarity offsets this drawback by subtracting respective user's average rating from each co-rated pair, and is defined as below:

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_u)(R_{u,j} - \bar{R}_u)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_u)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_u)^2}}.$$

（2） KNN with correlation similarity

Applied grid search on :

nb = [5,10,20]

topk = [50, 75]

$$r = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sqrt{\sum(x - \bar{x})^2 \sum(y - \bar{y})^2}}$$

（3） Lighfm

no_components = [32, 500, 1000]

Learning_rate = [0.1, 100， 10000]

Hyperparameter Tuning Summary Table

Model	Similarity	Dimension Reduction	Neighbors	Top K	Loss	no_components	epochs	alpha	learning_rate	evaluation score
KNN	cosine	None	5	25						0.3406
KNN	cosine	None	5	50						0.3406
KNN	cosine	None	5	75						0.3406
KNN	cosine	None	7	50						0.3855
KNN	cosine	None	10	25						0.4344
KNN	cosine	None	10	50						0.4344
KNN	cosine	None	10	75						0.4344
KNN	cosine	None	15	25						0.4876
KNN	cosine	None	15	50						0.4876
KNN	cosine	None	15	75						0.4876
KNN	cosine	None	20	25						0.525
KNN	cosine	None	20	50						0.525
KNN	cosine	None	20	75						0.525
KNN	pearson correlation	PCA	5	50						0.2684
KNN	pearson correlation	PCA	5	75						0.2684
KNN	pearson correlation	PCA	10	50						0.3451
KNN	pearson correlation	PCA	10	75						0.3451
KNN	pearson correlation	PCA	20	50						0.4192
KNN	pearson correlation	PCA	20	75						0.4192
lightfm		NNMF			warp	20	20	1.00E-05	10000	0.021
lightfm		NNMF			bpr	20	20	1.00E-05	10000	0.0093
lightfm		NNMF			warp	20	20	1.00E-05	1000000	0.0165
lightfm		NNMF			bpr	20	20	1.00E-05	1000000	0.0135

Overall the KNN model with Cosine Similarity has the best performance.

6.2 Model selection

Nearest Neighbor Model with cosine similarity outperforms other models. The final model selected as NearestNeighbors(metric = 'cosine', algorithm= 'brute') with 20 neighbors during prediction (model.kneighbors(df_train.iloc[i,:].values.reshape(1, -1), n_neighbors = 21))

7. Explanation

In general, the lighfm families have the lowest evaluation score. I think it probably due to the limited item information so that it's hard for lightfm to identify latent factors which connect both users and items via NNFM.

KNN overall performance better especially with cosine similarity. The correlation similarity tries to adjust the rating score for each user which is making sense because each user has a different rating standard. And the correlation similarity helps to normalize user scoring. This is useful if we are to predict movie ratings instead of doing top k movie recommendations.

KNN with cosine similarity sounds very simple but it really leverages the movies watched by different users.

8. Optimize and further work

1. A rolling window cross validation to check how does the model perform over time
2. Leverage content information to increase recommendation
3. Explore supervised models to predict user ratings