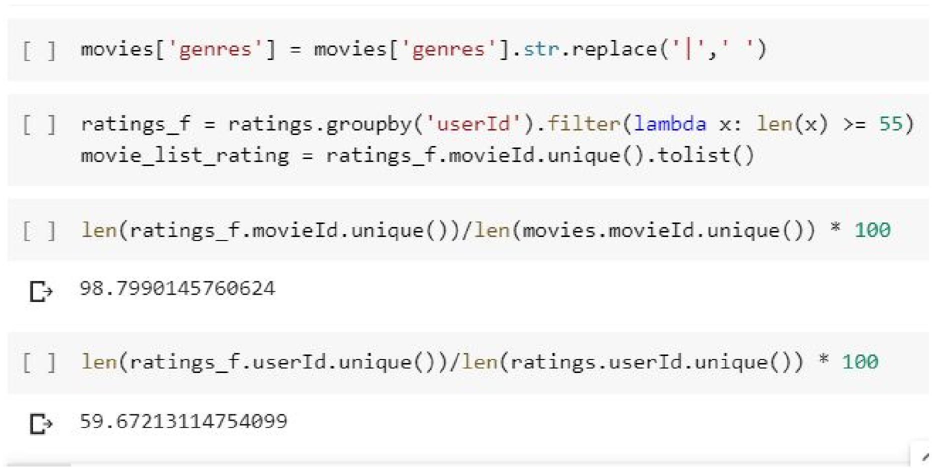
Faii In-on 6138020

Movie Recommendation System

1. Data preprocessing
    - Remove “|” from Genre attribute
    - Select only users that rated more than 55 movies
    - Remove the movie with no rating
    - Result after pre-processing, we got 59% of users and 98% of movies remained in dataset to proceed in the next step.



```
[ ] movies['genres'] = movies['genres'].str.replace('|',' ')
```

```
[ ] ratings_f = ratings.groupby('userId').filter(lambda x: len(x) >= 55)
    movie_list_rating = ratings_f.movieId.unique().tolist()
```

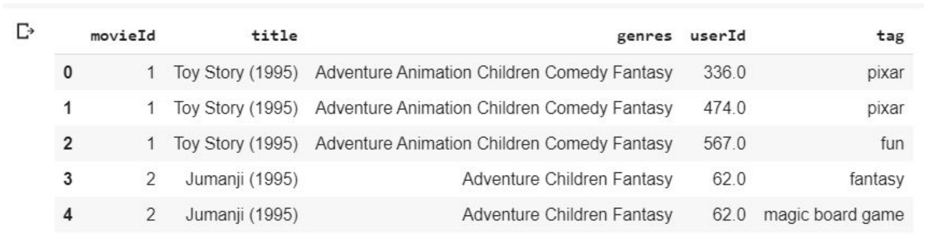
```
[ ] len(ratings_f.movieId.unique())/len(movies.movieId.unique()) * 100
```

```
98.7990145760624
```

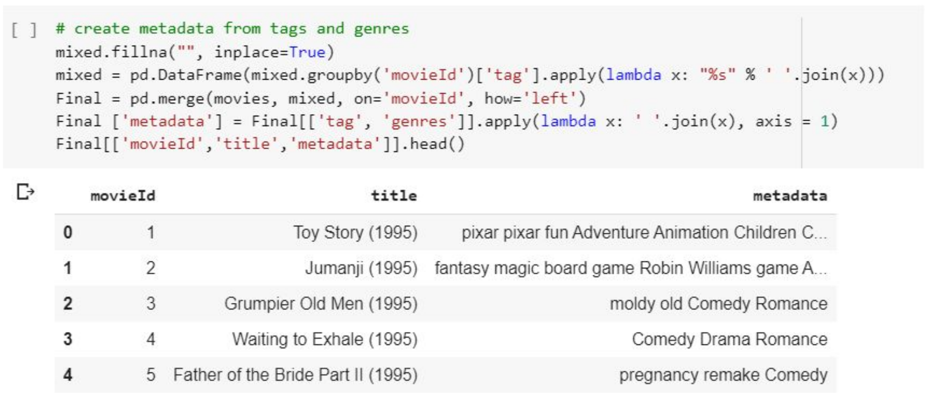
```
[ ] len(ratings_f.userId.unique())/len(ratings.userId.unique()) * 100
```

```
59.67213114754099
```

- Remove “time stamp”
- Create new “metadata” by combining “Genres” and “Tag”



	movieId	title	genres	userId	tag
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy	336.0	pixar
1	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy	474.0	pixar
2	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy	567.0	fun
3	2	Jumanji (1995)	Adventure Children Fantasy	62.0	fantasy
4	2	Jumanji (1995)	Adventure Children Fantasy	62.0	magic board game



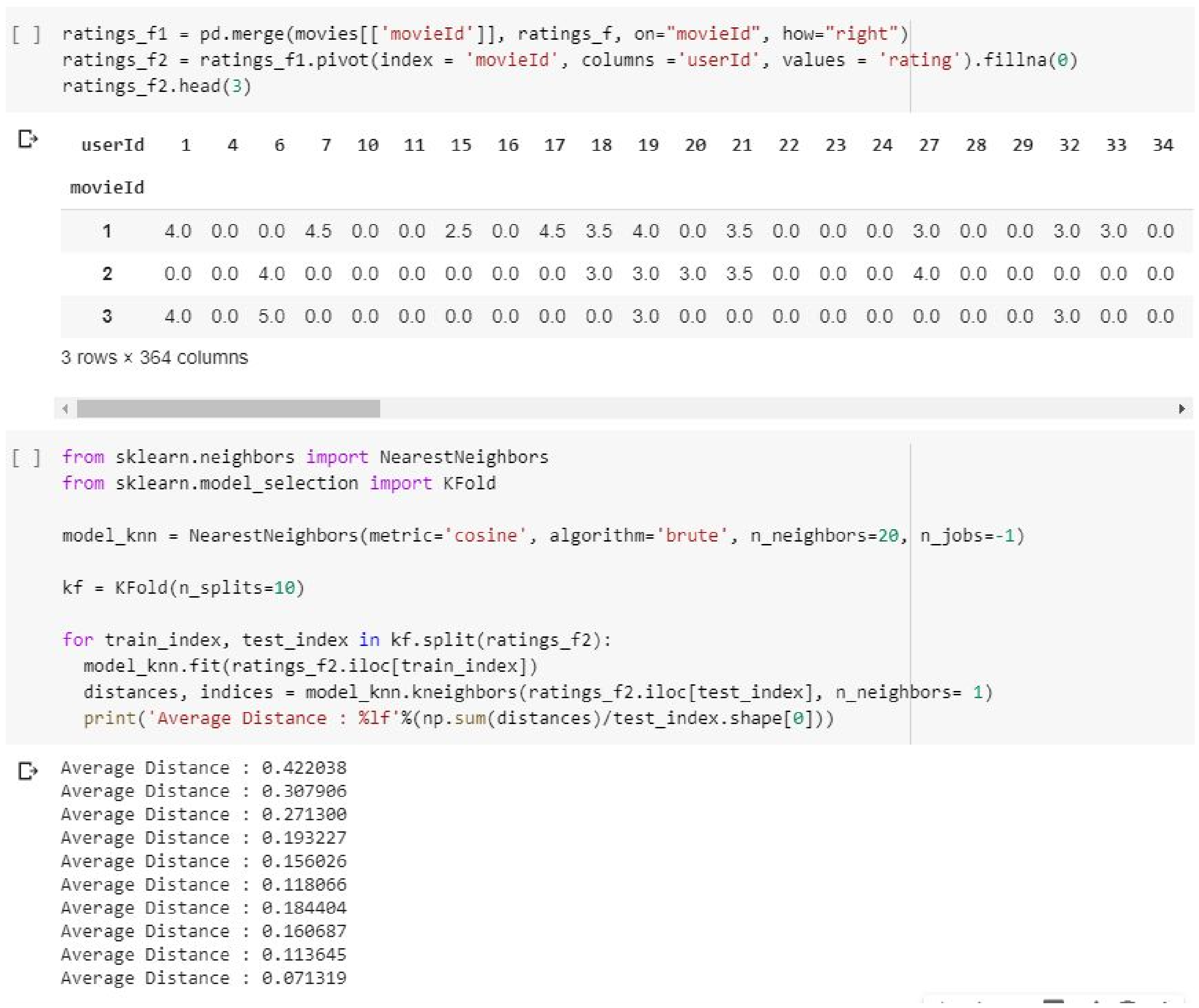
```
[ ] # create metadata from tags and genres
    mixed.fillna("", inplace=True)
    mixed = pd.DataFrame(mixed.groupby('movieId')['tag'].apply(lambda x: "%s" % ' '.join(x)))
    Final = pd.merge(movies, mixed, on='movieId', how='left')
    Final ['metadata'] = Final[['tag', 'genres']].apply(lambda x: ' '.join(x), axis = 1)
    Final[['movieId','title','metadata']].head()
```

	movieId	title	metadata
0	1	Toy Story (1995)	pixar pixar fun Adventure Animation Children C...
1	2	Jumanji (1995)	fantasy magic board game Robin Williams game A...
2	3	Grumpier Old Men (1995)	moldy old Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	pregnancy remake Comedy

2. Collaborative Filtering

2.1 Apply K-nearest on user’s rating

We use cosine similarity as a distance metric, and in order to compute nearest neighbor we used brute force search. Then, we compute the average distance.



```
[ ] ratings_f1 = pd.merge(movies[['movieId']], ratings_f, on="movieId", how="right")
    ratings_f2 = ratings_f1.pivot(index = 'movieId', columns ='userId', values = 'rating').fillna(0)
    ratings_f2.head(3)
```

userId	1	4	6	7	10	11	15	16	17	18	19	20	21	22	23	24	27	28	29	32	33	34
movieId																						
1	4.0	0.0	0.0	4.5	0.0	0.0	2.5	0.0	4.5	3.5	4.0	0.0	3.5	0.0	0.0	0.0	3.0	0.0	0.0	3.0	3.0	0.0
2	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	3.0	3.0	3.5	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0
3	4.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0

3 rows × 364 columns

```
[ ] from sklearn.neighbors import NearestNeighbors
    from sklearn.model_selection import KFold

    model_knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)

    kf = KFold(n_splits=10)

    for train_index, test_index in kf.split(ratings_f2):
      model_knn.fit(ratings_f2.iloc[train_index])
      distances, indices = model_knn.kneighbors(ratings_f2.iloc[test_index], n_neighbors= 1)
      print('Average Distance : %lf'%(np.sum(distances)/test_index.shape[0]))
```

```
Average Distance : 0.422038
Average Distance : 0.307906
Average Distance : 0.271300
Average Distance : 0.193227
Average Distance : 0.156026
Average Distance : 0.118066
Average Distance : 0.184404
Average Distance : 0.160687
Average Distance : 0.113645
Average Distance : 0.071319
```

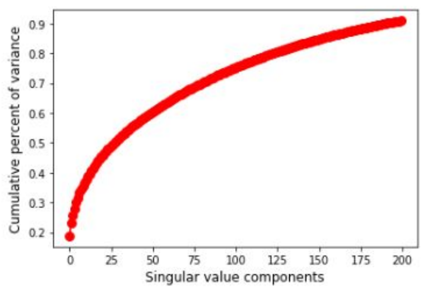
2.2 Apply K-nearest on user’s rating laten matrix (to reduce the dimension of dataset)
We compute the latent matrix by using TruncatedSVD. So the number of components that we selected is 200. Graph showed that 200 features is enough as shown in cumulative percentage of variance.

```
[ ] from sklearn.decomposition import TruncatedSVD
    svd = TruncatedSVD(n_components=200)
    latent_matrix_ratings = svd.fit_transform(ratings_f2)
    latent_matrix_ratings_df = pd.DataFrame(latent_matrix_ratings, index=Final.title.tolist())
    latent_matrix_ratings_df.head(3)
```

	0	1	2	3	4	5	6	7	8	9	
Toy Story (1995)	36.391240	-4.958788	14.331820	1.844157	-3.077921	-1.351171	8.341425	-0.561090	-1.898201	1.657860	2.4
Jumanji (1995)	20.447499	0.675231	11.354479	-7.325453	-3.505243	3.585611	4.393395	-5.564558	0.434475	0.351441	2.9
Grumpier Old Men (1995)	8.407816	-5.190801	4.378314	-6.189039	-0.333121	1.924148	-1.548363	0.795948	-3.196044	2.121786	0.7

3 rows × 200 columns

```
# plot variance expalined to see what latent dimensions to use
explained = svd.explained_variance_ratio_.cumsum()
plt.plot(explained, '.-', ms = 16, color='red')
plt.xlabel('Singular value components', fontsize= 12)
plt.ylabel('Cumulative percent of variance', fontsize=12)
plt.show()
```

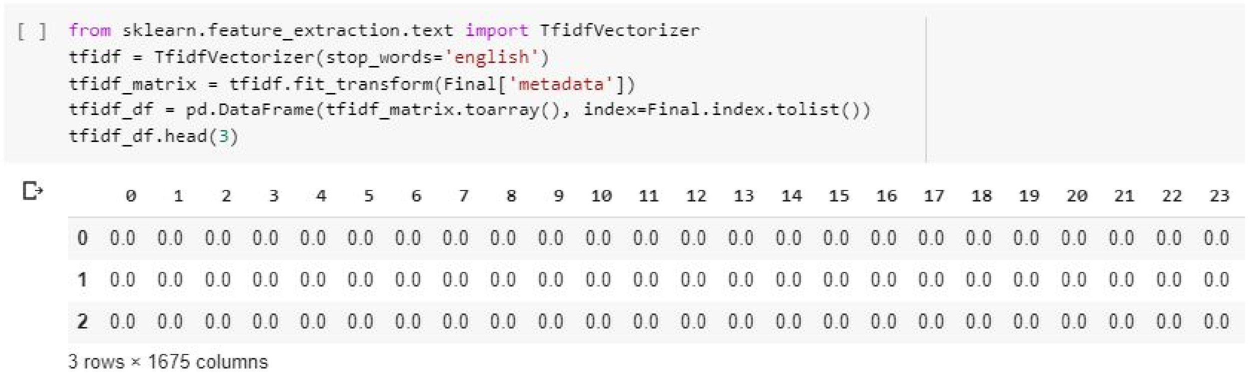


```
[ ] from sklearn.neighbors import NearestNeighbors
    from sklearn.model_selection import KFold

    model_knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)

    kf = KFold(n_splits=10)

    for train_index, test_index in kf.split(latent_matrix_ratings_df):
      model_knn.fit(latent_matrix_ratings_df.iloc[train_index])
      distances, indices = model_knn.kneighbors(latent_matrix_ratings_df.iloc[test_index], n_neighbors= 1)
      print('Average Distance : %lf'%(np.sum(distances)/test_index.shape[0]))
```

```
Average Distance : 0.371932
Average Distance : 0.281997
Average Distance : 0.249290
Average Distance : 0.176680
Average Distance : 0.144887
Average Distance : 0.108654
Average Distance : 0.171673
Average Distance : 0.148810
Average Distance : 0.104808
Average Distance : 0.062865
```

3. Content Filtering

We convert the “metadata” that we created earlier using Term frequency-inverse document frequency (Tf-idf).

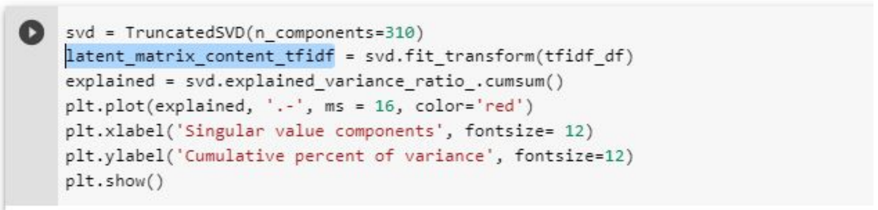


```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer
    tfidf = TfidfVectorizer(stop_words='english')
    tfidf_matrix = tfidf.fit_transform(Final['metadata'])
    tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), index=Final.index.tolist())
    tfidf_df.head(3)
```

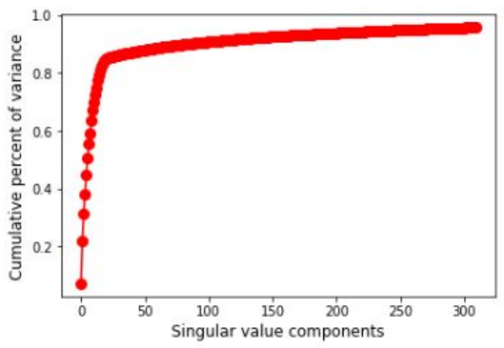
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

3 rows × 1675 columns

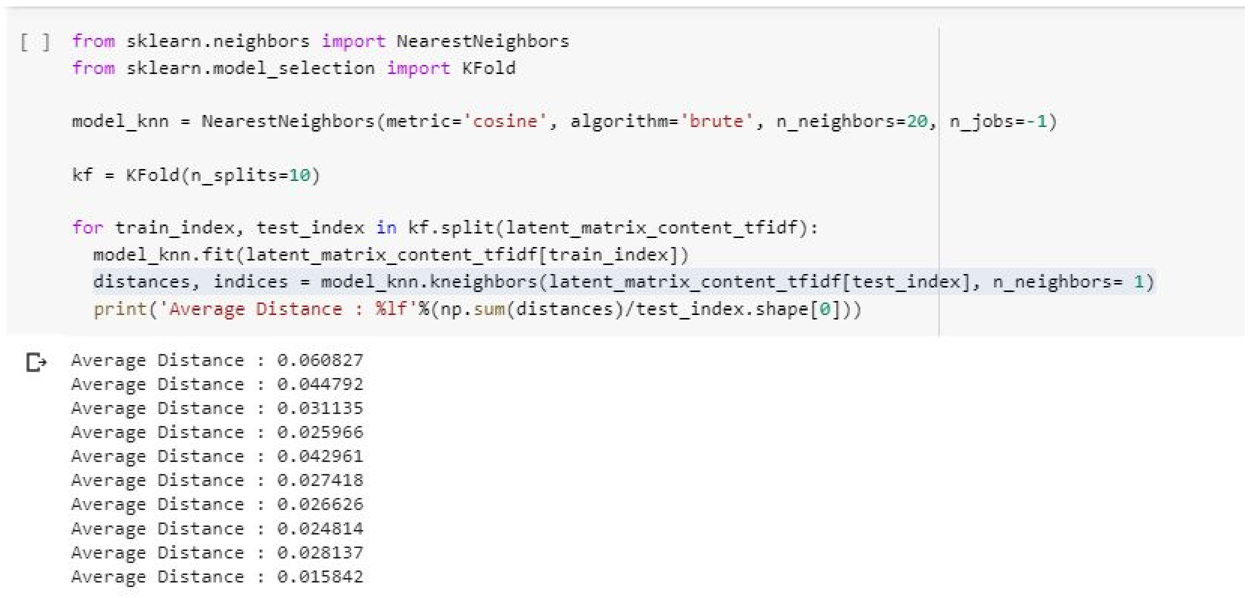
The result shown that the size of Tf-idf vector is 1675 column which is impractical to apply K-nearest Neighbor, so that we have to apply latent metrix to reduce the number of dimensions.



```
svd = TruncatedSVD(n_components=310)
latent_matrix_content_tfidf = svd.fit_transform(tfidf_df)
explained = svd.explained_variance_ratio_.cumsum()
plt.plot(explained, '.-', ms = 16, color='red')
plt.xlabel('Singular value components', fontsize= 12)
plt.ylabel('Cumulative percent of variance', fontsize=12)
plt.show()
```



From the graph, 310 features is enough according to cumulative percentage variance. Then, we apply K-Nearest Neighbor

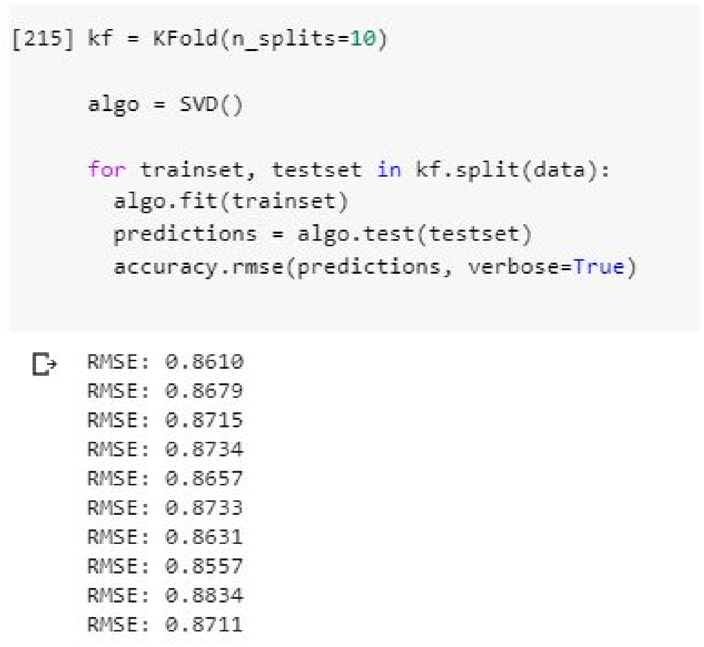


```
[ ] from sklearn.neighbors import NearestNeighbors
    from sklearn.model_selection import KFold

    model_knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)

    kf = KFold(n_splits=10)

    for train_index, test_index in kf.split(latent_matrix_content_tfidf):
      model_knn.fit(latent_matrix_content_tfidf[train_index])
      distances, indices = model_knn.kneighbors(latent_matrix_content_tfidf[test_index], n_neighbors= 1)
      print('Average Distance : %lf'%(np.sum(distances)/test_index.shape[0]))
```

```
Average Distance : 0.060827
Average Distance : 0.044792
Average Distance : 0.031135
Average Distance : 0.025966
Average Distance : 0.042961
Average Distance : 0.027418
Average Distance : 0.026626
Average Distance : 0.024814
Average Distance : 0.028137
Average Distance : 0.015842
```

4. Other methods (Collaborative Filtering using SVD and NMF)
SVD and NMF are matrix factorization mode. Both techniques attempt to predict the rating that user will rate the movies. Then, the system will recommend the movie that obtained the rating higher than threshold to the specific user. We use Surprise library, which is an easy-to-use Python scikit for recommender systems, to implement SVD and NMF based recommendation system as shown below.

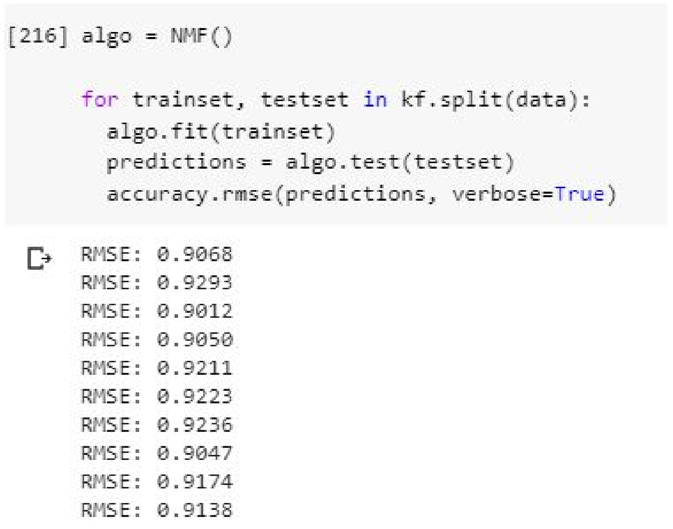
4.1 Singular value decomposition (SVD)



```
[215] kf = KFold(n_splits=10)

      algo = SVD()

      for trainset, testset in kf.split(data):
        algo.fit(trainset)
        predictions = algo.test(testset)
        accuracy.rmse(predictions, verbose=True)
```

```
RMSE: 0.8610
RMSE: 0.8679
RMSE: 0.8715
RMSE: 0.8734
RMSE: 0.8657
RMSE: 0.8733
RMSE: 0.8631
RMSE: 0.8557
RMSE: 0.8834
RMSE: 0.8711
```

4.2 Non-negative matrix factorization (NMF)



```
[216] algo = NMF()

      for trainset, testset in kf.split(data):
        algo.fit(trainset)
        predictions = algo.test(testset)
        accuracy.rmse(predictions, verbose=True)
```

```
RMSE: 0.9068
RMSE: 0.9293
RMSE: 0.9012
RMSE: 0.9050
RMSE: 0.9211
RMSE: 0.9223
RMSE: 0.9236
RMSE: 0.9047
RMSE: 0.9174
RMSE: 0.9138
```

5. Discussion and conclusion

We successfully develop recommendation system by applying K-nearest neighbors on movielens dataset.

Collaborative Filtering

We applied KNN on raw user-rating data. Cosine similarity was used as a distance function since user-rating matrix is a sparse matrix, many fields in the matrix contained 0 value. The algorithm successfully outputs the movie that the Cosine similarity distance close to the selected movie. However, KNN is suffering from the dimensionality problem, since user-rating matrix contain 364 features. We reduced the number of dimensions using latent semantic analysis (LSA). The number of dimensions is reduced to 200. Then, we applied KNN on the transformed matrix and obtained the result similar to the previous one.

Content-based Filtering

We combined Tags and Genres to make movie's metadata. Then, TF-IDF is applied on movie's metadata information. The output of TF-IDF is the set vectors of size 1675 dimensions. LSA also applied to TF-IDF matrix to reduce the number of dimensions. The experiment showed that 310 features of transformation matrix is enough to maintain all information of the original TF-IDF matrix. After that KNN is applied on the transformation matrix.

Discussion on the results of both methods.

After we closely inspected to the result of both methods, we found that the set of recommended movies is quite different. The reason is Collaborative filtering will search for the movie that has similar rating but Content-based filtering will search for the movie that has similar metadata (contained the similar tags and genre). We suggest to combine these recommended systems together and search for duplicated recommended movies to increase the performance of the system.

Other methods

We applied two matrix factorization methods, SVD and NMF, for developing the recommendation system. Matrix factorization algorithm takes user-rating matrix as input. Then, it will factor into two matrices. The first matrix represents the user's characteristic and the second matrix represent the movie's characteristic. The output of matrix factorization method is the predicted rating on the unknown field. The result shows that SVD get lower RMSE than NMF. The reason is SVD is a more insightful factorization technique. NMF on the other hand tries to get best fit decomposed matrices, which are trained on some training data and then done an evaluation on test data.