IST 707 – Applied Machine Learning

MOVIE RECOMMENDATION SYSTEM

Anusha Ramprasad

Mikhail Pinto

INTRODUCTION:

In this work-centric world, entertainment is a much-needed part for human beings to refresh their minds and restore their mental capacity by watching or doing something which helps them regain their energy. In general, people listen to music or watch movies or different shows of their choice to relax and have a good time. It is pretty time-consuming to look for movies you like as we don't exactly know what the movie will be about. For this reason, we need a movie recommendation system that is more reliable since it is very time efficient and there is a high chance of you enjoying the movie. We will be using Collaborative filtering, Content-based filtering, and a hybrid model using Cosine Similarity, Singular Value Decomposition (SVD, SVD++), and K-Nearest Neighbor (KNN). Hybrid models will help us eliminate the disadvantages of both collaborative and content-based methods and get better accuracy in predicting what a user might like.

OBJECTIVE:

The main objectives are as follows:

- To create a recommendation system to predict the rating or a preference the user gives to an item
- To provide an accurate recommendation based on recorded info on the user's preferences and behavior
- To improve the interaction time, boost profit and revenue, and enhance the user experience, thereby encouraging the users to use the services often.

This report presents an approach to showcase how to go about the process starting from the initial steps of the descriptive analysis, preprocessing the dataset, choosing the models based on the task, conducting the research, and using algorithms in proposed methodologies to improve the quality of recommender systems. The proposed approach shows that a movie recommendation system's accuracy, quality, and scalability are improved over a pure method.

EXPLORATORY DATA ANALYSIS:

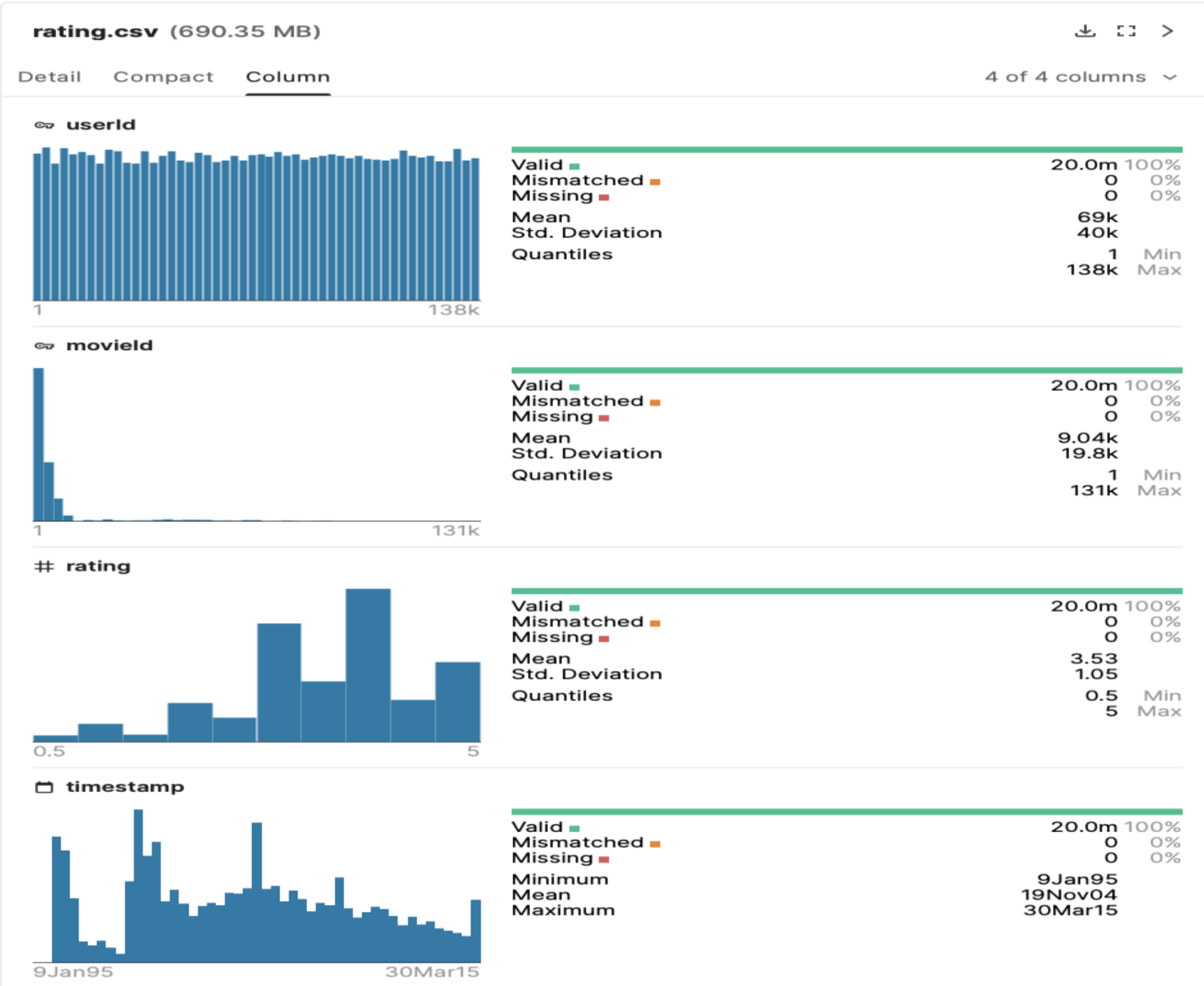
The dataset is from Kaggle (https://www.kaggle.com/datasets/grouplens/movielens-20m-dataset?select=movie.csv).

It contains ratings and free-text tagging activities from MovieLens. It has 20 million ratings and over 460000+ tag applications across 27K+ movies. This data was created by obtaining users' ratings who were selected at random, and every user had rated at least 20 movies.

No demographic information is included. An id represents each user, and no other information is provided.
The data are contained in six files.

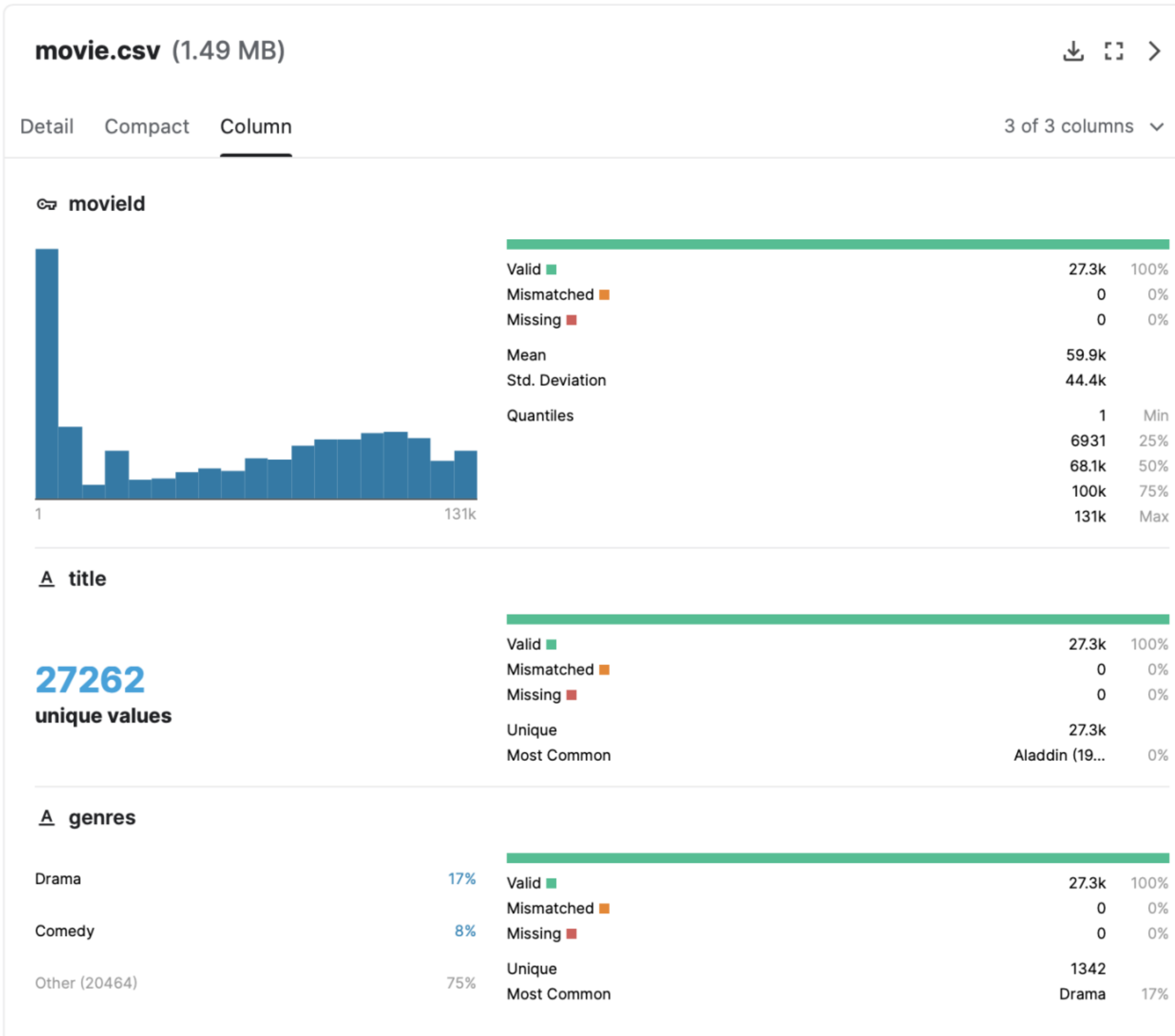
rating.csv that contains ratings of movies by users:

- userId
- movieId
- rating
- timestamp


rating.csv (690.35 MB)
Detail
Compact
Column
4 of 4 columns
userId
1
138k
Valid 20.0m 100%
Mismatched 0 0%
Missing 0 0%
Mean 69k
Std. Deviation 40k
Quantiles 1 Min
138k Max
movieId
1
131k
Valid 20.0m 100%
Mismatched 0 0%
Missing 0 0%
Mean 9.04k
Std. Deviation 19.8k
Quantiles 1 Min
131k Max
rating
0.5
5
Valid 20.0m 100%
Mismatched 0 0%
Missing 0 0%
Mean 3.53
Std. Deviation 1.05
Quantiles 0.5 Min
5 Max
timestamp
9Jan95
30Mar15
Valid 20.0m 100%
Mismatched 0 0%
Missing 0 0%
Minimum 9Jan95
Mean 19Nov04
Maximum 30Mar15

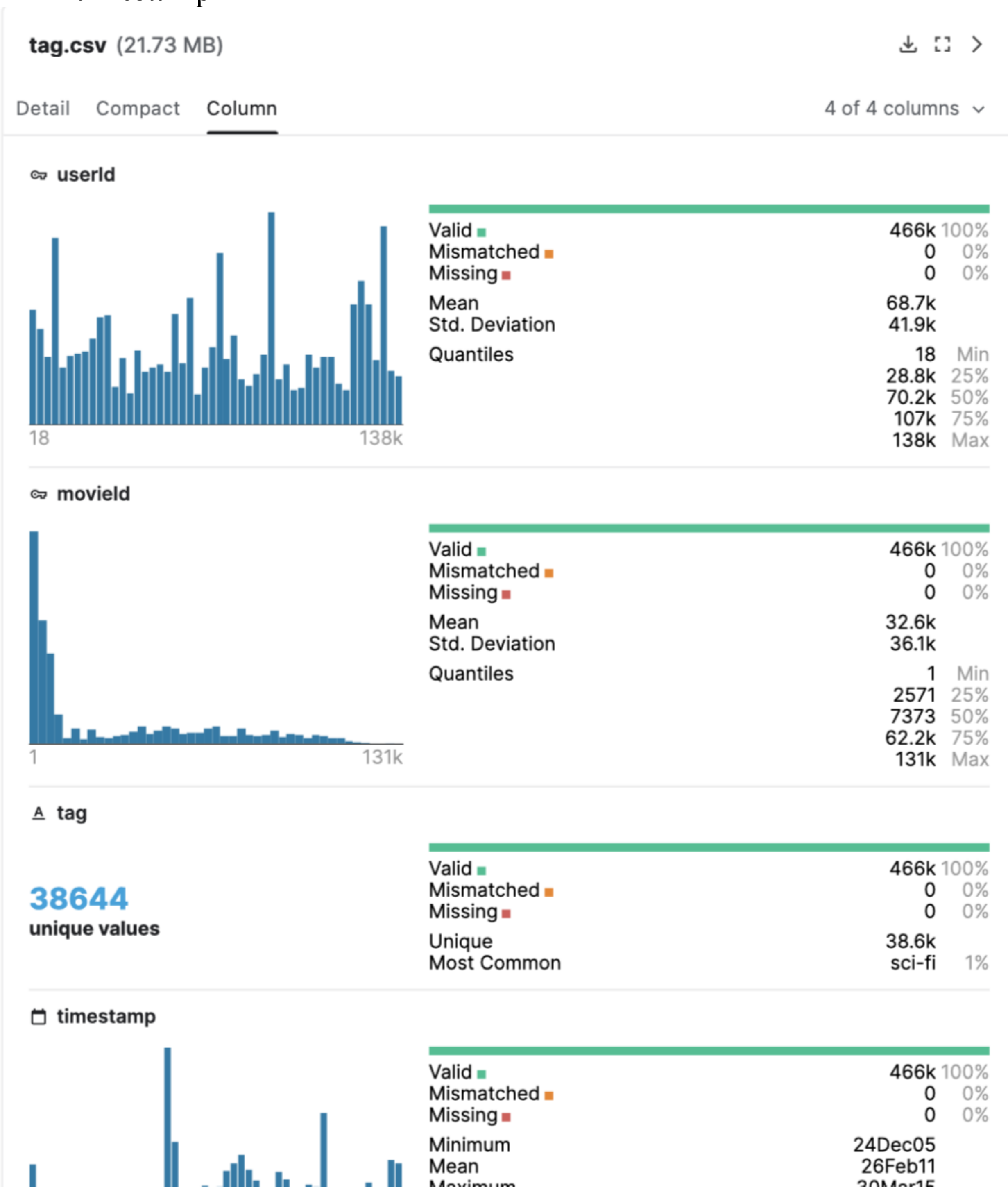
movie.csv that contains movie information:

- movieId
- title
- genres



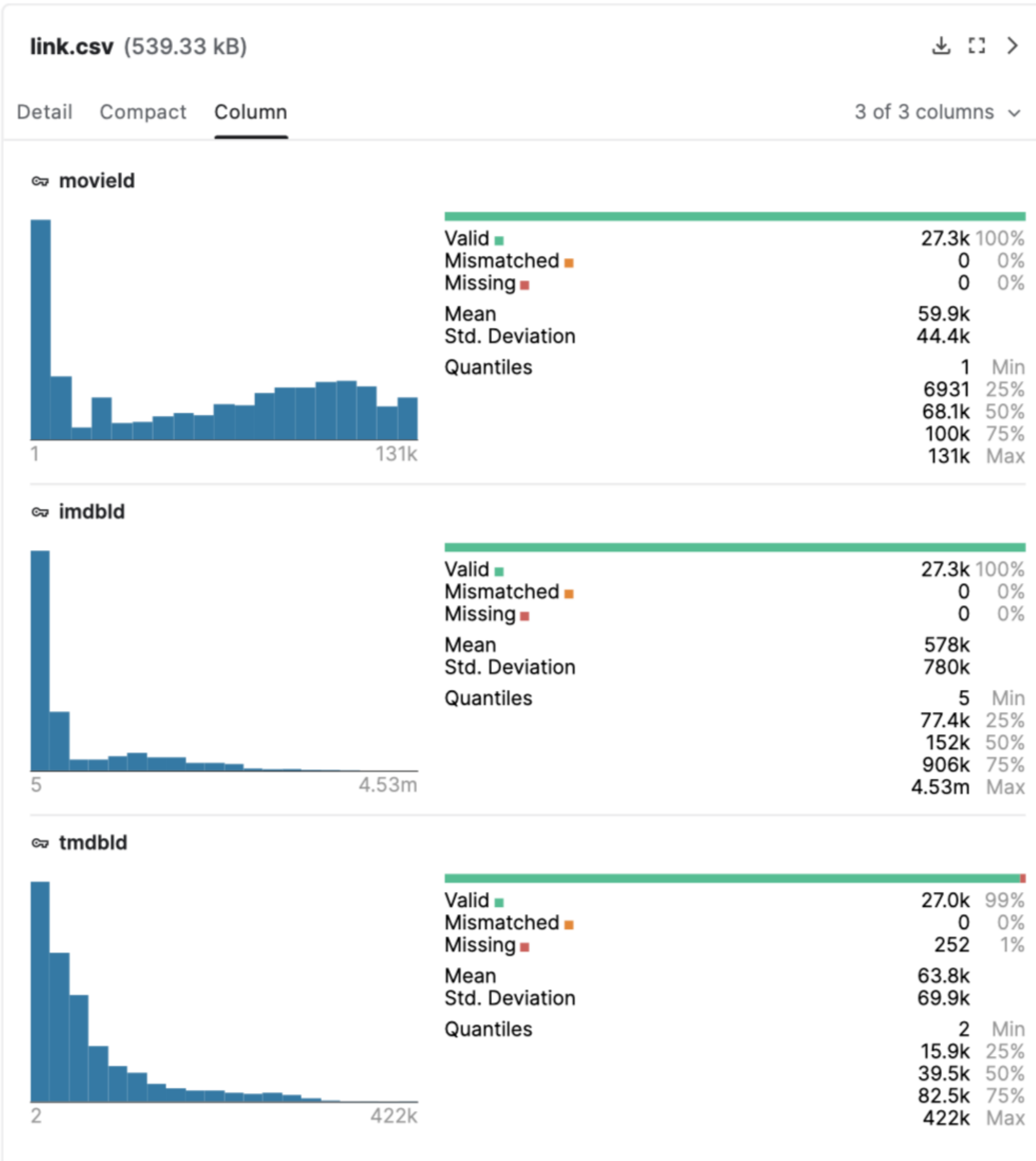
tag.csv that has tags applied to movies by users:

- userId
- movieId
- tag
- timestamp



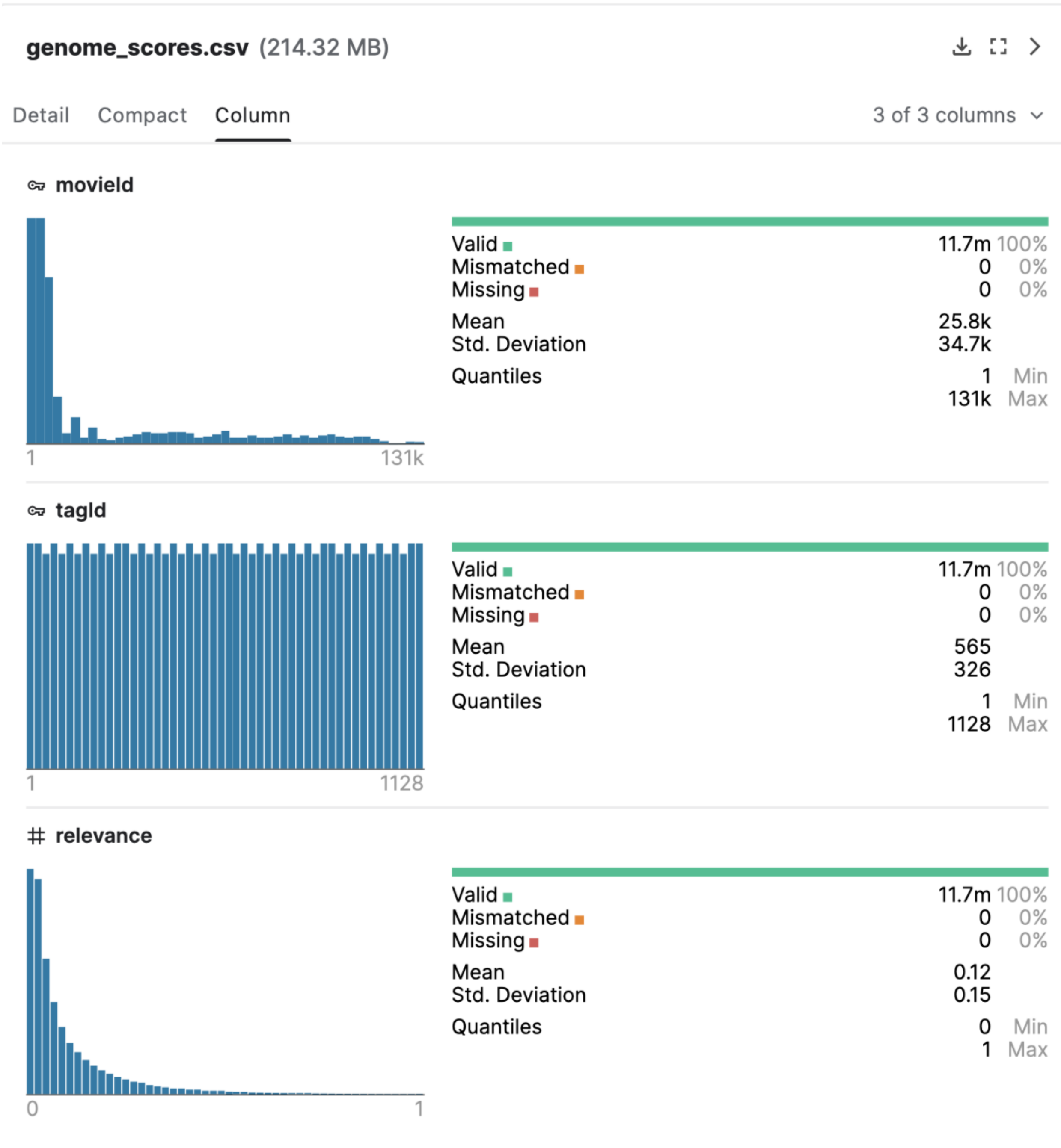
link.csv that contains identifiers that can be used to link to other sources:

- movieId
- imdbId
- tmbdId



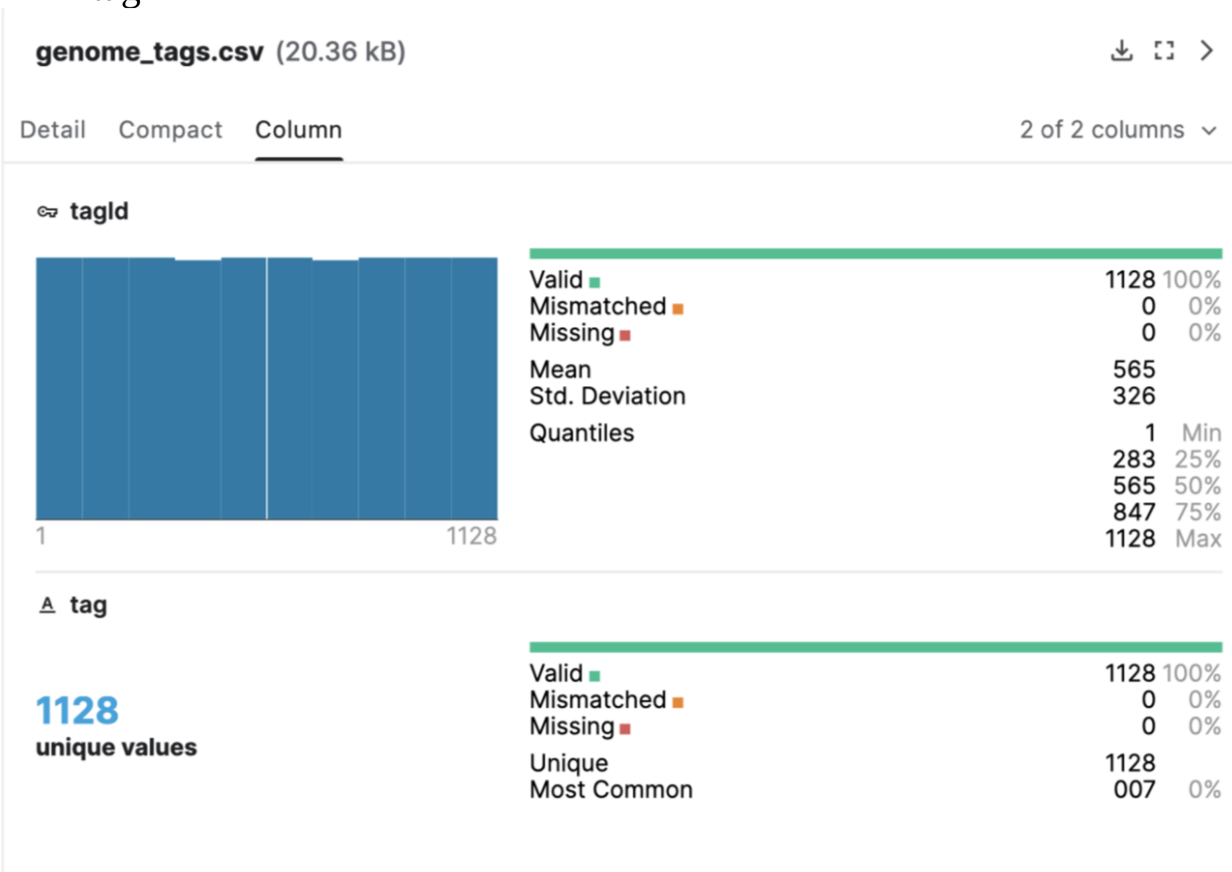
genome_scores.csv that contains movie-tag relevance data:

- movieId
- tagId
- relevance



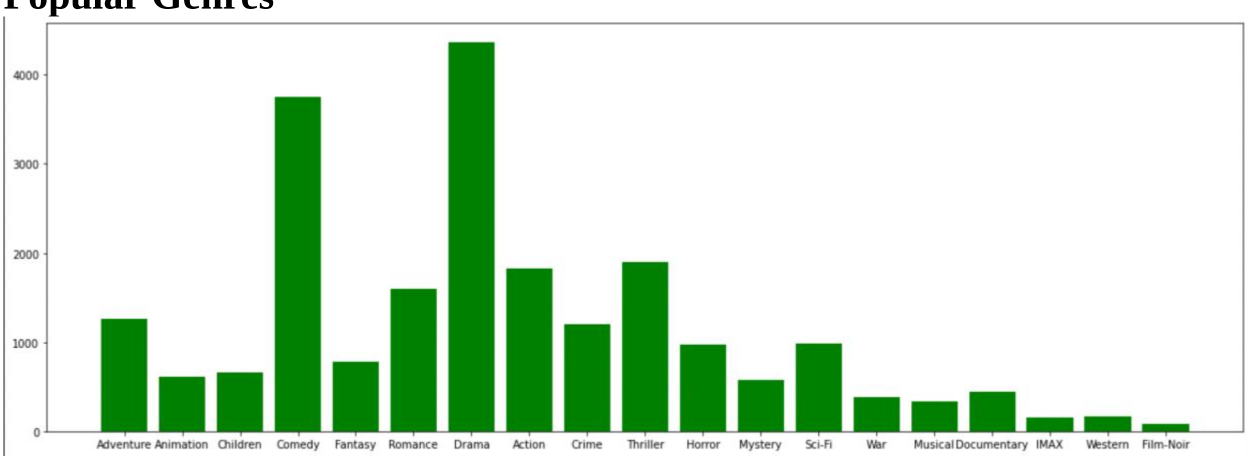
genome_tags.csv that contains tag descriptions:

- tagId
- tag

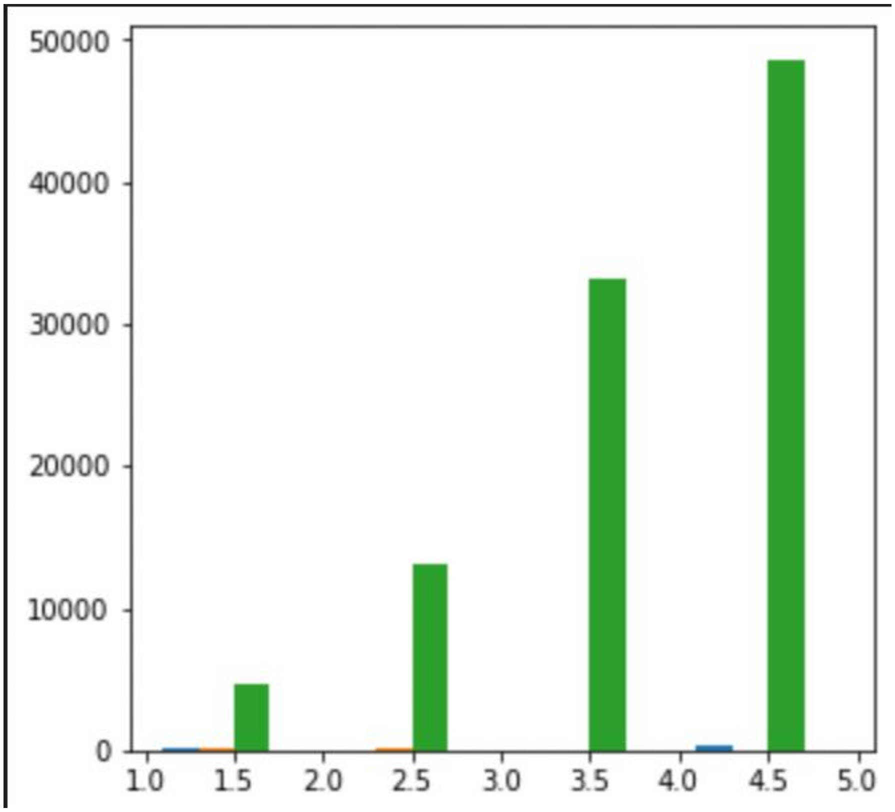


Users were selected at random for inclusion. All selected users had rated at least 20 movies.

Popular Genres

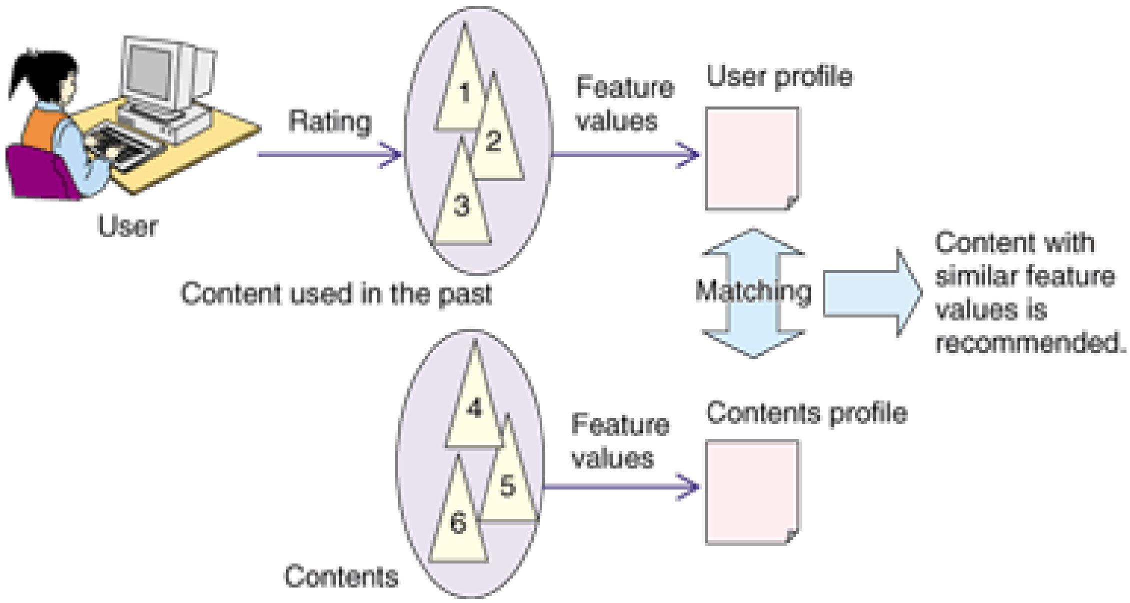


Rating Histogram



CONTENT-BASED FILTERING:

The Content-Based Filtering method uses attributes of the content to recommend similar content. It doesn’t have a cold-start problem because it works through details or tags of the content, such as actors, genres, or directors so that new movies can be recommended right away. It uses the concept of Term Frequency (TF) and Inverse Document Frequency (IDF). They are used to determine the relative importance of a document/article/news item/movie.



COSINE SIMILARITY:

Cosine similarity is a measure of similarity that quantifies the cosine of the angle between two non-zero vectors in an inner product space. The cosine similarity, cos(θ), is represented using a dot product and magnitude as Inline given two vectors of attributes, A and B. We will use cosine distance since we are interested in similarity here. That is, the higher the value, the closer they are. However, because the function provides the space, we will subtract it from 1.

```python
# Define a TF-IDF Vectorizer Object.
tfidf_movies_genres = TfidfVectorizer(token_pattern = '[a-zA-Z0-9\-]+')

#Replace NaN with an empty string
df_movies['genres'] = df_movies['genres'].replace(to_replace="(no genres listed)", value="")

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_movies_genres_matrix = tfidf_movies_genres.fit_transform(df_movies['genres'])
print(tfidf_movies_genres.get_feature_names())
#Compute the cosine similarity matrix
print(tfidf_movies_genres_matrix.shape)
print(tfidf_movies_genres_matrix.dtype)
cosine_sim_movies = linear_kernel(tfidf_movies_genres_matrix, tfidf_movies_genres_matrix)
print(cosine_sim_movies)
```

Python

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)

['action', 'adventure', 'animation', 'children', 'comedy', 'crime', 'documentary', 'drama', 'fantasy', 'film-noir', 'horror', 'imax', 'musical', 'mystery', 'romance', 'sci-fi', 'thriller', 'war', 'western']
```

```python
# Define a TF-IDF Vectorizer Object.
tfidf_movies_genres = TfidfVectorizer(token_pattern = '[a-zA-Z0-9\-]+')

#Replace NaN with an empty string
df_movies['genres'] = df_movies['genres'].replace(to_replace="(no genres listed)", value="")

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_movies_genres_matrix = tfidf_movies_genres.fit_transform(df_movies['genres'])
print(tfidf_movies_genres.get_feature_names())
#Compute the cosine similarity matrix
print(tfidf_movies_genres_matrix.shape)
print(tfidf_movies_genres_matrix.dtype)
cosine_sim_movies = linear_kernel(tfidf_movies_genres_matrix, tfidf_movies_genres_matrix)
print(cosine_sim_movies)
```

Python

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
['action', 'adventure', 'animation', 'children', 'comedy', 'crime', 'documentary', 'drama', 'fantasy', 'film-noir', 'horror', 'imax', 'musical', 'mystery', 'romance', 'sci-fi', 'thriller', 'war', 'western']
```

```python
def get_recommendation_content_model(userId):
    """
    Calculates top movies to be recommended to user based on movie user has watched.
    """
    recommended_movie_list = []
    movie_list = []
    df_rating_filtered = df_ratings[df_ratings["userId"]== userId]
    for key, row in df_rating_filtered.iterrows():
        movie_list.append((df_movies["title"][row["movieId"]==df_movies["movieId"]]).values)
    for index, movie in enumerate(movie_list):
        for key, movie_recommended in get_recommendations_based_on_movies(movie[0]).iteritems():
            recommended_movie_list.append(movie_recommended)

    # removing already watched movie from recommended list
    for movie_title in recommended_movie_list:
        if movie_title in movie_list:
            recommended_movie_list.remove(movie_title)

    return set(recommended_movie_list)
get_recommendation_content_model(1)
```

[30] ✓ 2.5s Python

```
Output exceeds the size limit. Open the full output data in a text editor
{'101 Dalmatians (One Hundred and One Dalmatians) (1961)',
 '39 Steps, The (1935)',
 'Ace Ventura: When Nature Calls (1995)',
 'Adventures in Babysitting (1987)',
 'Adventures of Baron Munchausen, The (1988)',
 'Adventures of Rocky and Bullwinkle, The (2000)',
 'Agent Cody Banks (2003)',
 'Alamo, The (1960)',
 'Alien Nation (1988)',
```

k NEAREST NEIGHBOR (kNN):

An unsupervised algorithm called K-Nearest Neighbor is applied to the movie lens dataset to produce the best-optimized outcome. In this method, the dataset is reshaped into a format that can be used as a parameter since the kNN model calculates the distance between the points. Data is distributed in the present technique, resulting in many clusters, whereas data is gathered in the suggested method, resulting in a small number of clusters. The proposed approach optimizes the process of movie suggestion. The proposed recommender system predicts the user's choice for a movie based on many characteristics. The recommender system assumes that people have similar preferences or options. These users will have an impact on each other's viewpoints.

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
def get_movie_label(movie_id):

    classifier = KNeighborsClassifier(n_neighbors=5)
    x= tfidf_movies_genres_matrix
    y = df_movies.iloc[:,-1]
    classifier.fit(x, y)
    y_pred = classifier.predict(tfidf_movies_genres_matrix[movie_id])
    #cf_matrix = confusion_matrix(x, y_pred)
    #print(cf_matrix)
    return y_pred
```
✓ 0.1s Python

MODEL EVALUATION WITH kNN:

Model Evaluation with kNN is done based on if there is a match of movies with the movies already watched by the user.

```python
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
true_count = 0
false_count = 0
def evaluate_content_based_model():
    """
    Evaluate content based model.
    """
    for key, colums in df_movies.iterrows():
        movies_recommended_by_model = get_recommendations_based_on_movies(colums["title"])
        predicted_genres  = get_movie_label(movies_recommended_by_model.index)
        for predicted_genre in predicted_genres:
            global true_count, false_count
            if predicted_genre == colums["genres"]:
                true_count = true_count+1
            else:
#                   print(colums["genres"])
#                   print(predicted_genre)
                false_count = false_count +1
evaluate_content_based_model()
total = true_count + false_count
print("Hit:"+ str(true_count/total))
print("Fault:" + str(false_count/total))
print("Precision: " + str(true_count/total))
print(true_count, false_count)
```
✓ 2m 45.2s

```
Hit:0.919805994662287
Fault:0.080194005337713
Precision: 0.919805994662287
```

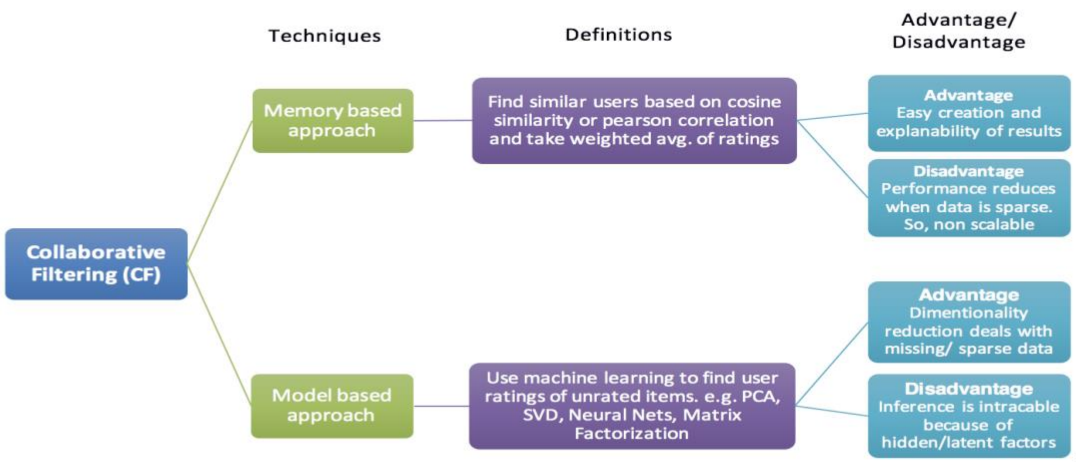
COLLABORATIVE FILTERING:

In collaborative filtering, movies are recommended based on how similar one user’s profile is to the other users, find the most similar users, and recommend items they have shown a preference for. The advantage of this technique is that it is easy to create and explain the results. However, it suffers when there is sparse or missing data, which reduces the performance, resulting in a cold start problem.

Collaborative Filtering follows two techniques:

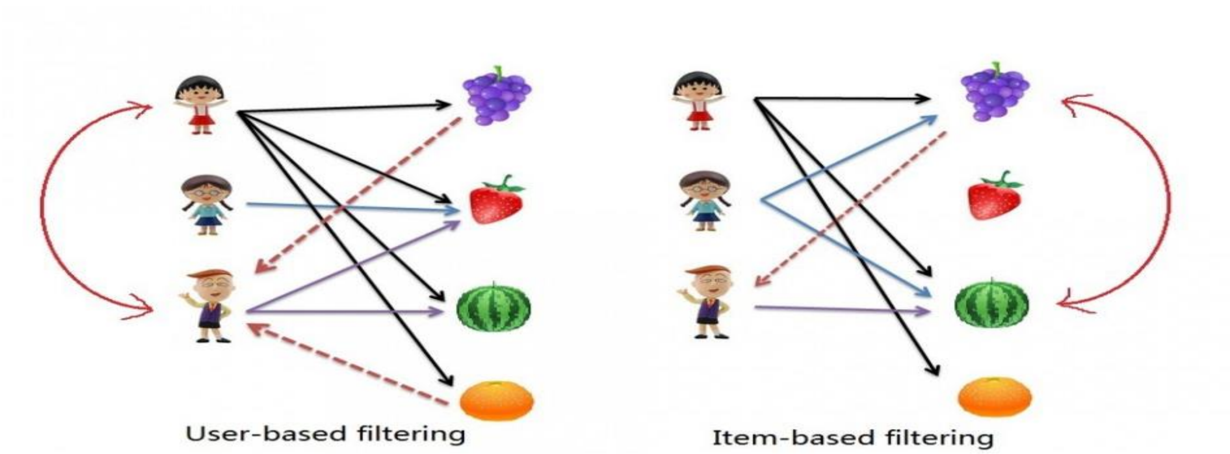
1. Memory-based
   User-Item Filtering
   Item-Item Filtering

2. Model-based
   Matrix Factorization
   Clustering



MEMORY-BASED APPROACH:

In either case, a similarity matrix is created. The user-similarity matrix will contain some distance metrics that assess the similarity between any two pairs of users for user-user collaborative filtering. Similarly, the item-similarity matrix will determine how similar any pair of items are.



ITEM-BASED FILTERING:

Item-based filtering is a type of collaborative filtering for recommender systems based on the similarity between items determined by people's ratings. This algorithm searches for items comparable to the articles that the user has already rated and recommends the most similar articles. Instead, similarity refers to how people react to two products regarding likes and dislikes. We try to find similar movies. We can easily recommend similar movies to users who have rated any movie in the dataset once we have the movie look-alike matrix. This algorithm uses a lot fewer resources than collaborative filtering amongst users. As a result, the algorithm takes considerably less time for a new user than user-user collaboration because we don't need all users' similarity ratings. And with a fixed number of movies, the movie look-alike matrix is fixed over time.

```
movie_similarity = 1 - pairwise_distances( ratings_matrix_item.to_numpy(), metric="cosine" )
np.fill_diagonal( movie_similarity, 0 )
ratings_matrix_items = pd.DataFrame( movie_similarity )
ratings_matrix_items
```

✓ 1.2s

	0	1	2	3	4	5	6	7	8	9
0	0.000000	0.410562	0.296917	0.035573	0.308762	0.376316	0.277491	0.131629	0.232586	0.395573
1	0.410562	0.000000	0.282438	0.106415	0.287795	0.297009	0.228576	0.172498	0.044835	0.417693
2	0.296917	0.282438	0.000000	0.092406	0.417802	0.284257	0.402831	0.313434	0.304840	0.242954
3	0.035573	0.106415	0.092406	0.000000	0.188376	0.089685	0.275035	0.158022	0.000000	0.095598
4	0.308762	0.287795	0.417802	0.188376	0.000000	0.298969	0.474002	0.283523	0.335058	0.218061
...	...	...	...	...	...	...	...	...	...	...
9719	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
9720	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
9721	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
9722	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
9723	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.072542

USER-BASED FILTERING:

The user-based collaborative filtering strategy is based on the notion that if two people have the same opinion on an issue, A is more likely to share B's opinion on a different topic than a randomly chosen individual. We discover similar users based on similarities and recommend movies that the initial user's look-alike has previously selected. This technique is quite effective, but it takes a long time and many resources to implement. It takes time to compute the information for each user pair. As a result, this technique is challenging to implement without a highly parallelizable system on large base platforms. The challenge with computing user similarity is that the client needs to have earlier purchases and should have rated them. This procedure doesn't work for new clients since the framework needs to hold on until the client makes a few buys and rates them, only after which similar users can be found, and recommendations can be made.

```python
def item_similarity(movieName):

    try:
        user_inp=movieName
        inp=df_movies[df_movies['title']==user_inp].index.tolist()
        inp=inp[0]

        df_movies['similarity'] = ratings_matrix_items.iloc[inp]
        df_movies.columns = ['movie_id', 'title', 'release_date','similarity']
    except:
        print("Sorry, the movie is not in the database!")
```
✓ 0.9s

```python
def recommendedMoviesAsperItemSimilarity(user_id):
    user_movie= df_movies_ratings[(df_movies_ratings.userId==user_id) & df_movies_ratings.rating.isin([5,4.5])][['title']]
    user_movie=user_movie.iloc[0,0]
    item_similarity(user_movie)
    sorted_movies_as_per_userChoice=df_movies.sort_values( ["similarity"], ascending = False )
    sorted_movies_as_per_userChoice=sorted_movies_as_per_userChoice[sorted_movies_as_per_userChoice['similarity'] >=0.45]['movie_id']
    recommended_movies=list()
    df_recommended_item=pd.DataFrame()
    user2Movies= df_ratings[df_ratings['userId']== user_id]['movieId']
    for movieId in sorted_movies_as_per_userChoice:
            if movieId not in user2Movies:
                df_new= df_ratings[(df_ratings.movieId==movieId)]
                df_recommended_item=pd.concat([df_recommended_item,df_new])
            best10=df_recommended_item.sort_values(["rating"], ascending = False )[1:10]
    return best10['movieId']
```
✓ 0.1s

```python
def movieIdToTitle(listMovieIDs):
    """
     Converting movieId to titles
    :param user_id: List of movies
    :return: movie titles
    """
    movie_titles= list()
    for id in listMovieIDs:
        movie_titles.append(df_movies[df_movies['movie_id']==id]['title'])
    return movie_titles
```
✓ 0.7s

```python
user_id=49
print("Recommended movies,:\n",movieIdToTitle(recommendedMoviesAsperItemSimilarity(user_id)))
```
✓ Pytho

```
Recommended movies,:
 [510    Silence of the Lambs, The (1991)
Name: title, dtype: object, 659    Godfather, The (1972)
Name: title, dtype: object, 510    Silence of the Lambs, The (1991)
Name: title, dtype: object, 659    Godfather, The (1972)
Name: title, dtype: object, 510    Silence of the Lambs, The (1991)
Name: title, dtype: object, 510    Silence of the Lambs, The (1991)
Name: title, dtype: object, 510    Silence of the Lambs, The (1991)
Name: title, dtype: object, 224    Star Wars: Episode IV - A New Hope (1977)
Name: title, dtype: object, 224    Star Wars: Episode IV - A New Hope (1977)
Name: title, dtype: object]
```

MODEL-BASED APPROACH:

The memory-based collaborative filtering approach of computing distance relationships between objects or users has two main problems:

- Large datasets, especially real-time recommendations based on user behavioral similarities, do not scale well.
- The rating matrix may not fit the user's tastes and representations of their liking.
- When using a distance-based "neighborhood" approach to raw data, match sparse, low-level details that are assumed to represent the user's preferred vector rather than the vector itself.

Hence, a model-based approach is used over a memory-based.

MATRIX FACTORIZATION:

Model-based collaborative filtering primarily focuses on matrix factorization (MF), attracting attention as an unsupervised learning method for latent variable decomposition and dimensionality reduction. Matrix factorization is often used in recommender systems that can better deal with scalability and economy than memory-based CF. The goal of Matrix Factorization is to learn potential user preferences and attributes of items from available ratings (then predict unknown ratings by the product of possible features of users and items.

Suppose you have a very sparse matrix with many dimensions. In that case, matrix factorization is used to reconstruct the user-item matrix into a low-level structure and multiply the two low-level matrices by the rows containing the latent vectors. This matrix is then fit to approximate the original matrix by multiplying the low-rank matrices and filling in the missing entries in the original matrix.

```
R = Ratings.values
user_ratings_mean = np.mean(R, axis = 1)
print(user_ratings_mean.size)
Ratings_demeaned = R - user_ratings_mean.reshape(-1, 1)
```
[137] ✓ 0.3s Python

```
610
```

```
sparsity = round(1.0 - len(ratings) / float(n_users * n_movies), 3)
print('The sparsity level of MovieLens dataset is ' +  str(sparsity * 100) + '%')
```
[156] ✓ Python

```
The sparsity level of MovieLens dataset is 98.3%
```

```
from scipy.sparse.linalg import svds
```
[139] ✓ 0.2s Python

```
U, sigma, Vt = svds(Ratings_demeaned, k = 50)
```
[140] ✓ 1.6s Python

```
print('Size of sigma: ' , sigma.size)
```
[141] ✓ 0.8s Python

```
Size of sigma:  50
```

```
sigma = np.diag(sigma)
```
[142] ✓ 0.5s Python

```
print('Shape of sigma: ', sigma.shape)
print(sigma)
```
[143] ✓ 0.1s Python

```
Shape of sigma:  (50, 50)
[[ 67.86628347   0.           0.         ...   0.           0.
    0.        ]
 [  0.          68.1967072    0.         ...   0.           0.
    0.        ]
 [  0.           0.          69.02678246 ...   0.           0.
    0.        ]
 ...
 [  0.           0.           0.         ... 184.86187801   0.
    0.        ]
 [  0.           0.           0.         ...   0.         231.22453421
    0.        ]
 [  0.           0.           0.         ...   0.           0.
  474.20606204]]
```

```python
def recommend_movies(predictions, userID, movies, original_ratings, num_recommendations):

    # sort the user's predictions
    user_row_number = userID - 1
    sorted_user_predictions = predictions.iloc[user_row_number].sort_values(ascending=False)

    user_data = original_ratings[original_ratings.userId == (userID)]
    user_full = (user_data.merge(movies, how = 'left', left_on = 'movieId', right_on = 'movieId').
                     sort_values(['rating'], ascending=False)
                 )

    print('User {0} has already rated {1} movies.'.format(userID, user_full.shape[0]))
    print('Recommending highest {0} predicted ratings movies not already rated.'.format(num_recommendations))


    recommendations = (movies[~movies['movieId'].isin(user_full['movieId'])].
         merge(pd.DataFrame(sorted_user_predictions).reset_index(), how = 'left',
               left_on = 'movieId',
               right_on = 'movieId').
         rename(columns = {user_row_number: 'Predictions'}).
         sort_values('Predictions', ascending = False).
                       iloc[:num_recommendations, :-1]
                      )

    return user_full, recommendations
```

[149] ✓ 0.3s

```python
already_rated, predictions = recommend_movies(preds, 150, movies, ratings, 20)
```

[150] ✓ 0.3s

```
User 150 has already rated 26 movies.
```

```python
# Top 20 movies that User might enjoy
predictions
```

[152] ✓ 0.3s

	movieId	title	genres
574	736	Twister (1996)	Action|Adventure|Romance|Thriller
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
211	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi
607	802	Phenomenon (1996)	Drama|Romance
12	17	Sense and Sensibility (1995)	Drama|Romance
87	112	Rumble in the Bronx (Hont faan kui) (1995)	Action|Adventure|Comedy|Crime
558	708	Truth About Cats & Dogs, The (1996)	Comedy|Romance
599	788	Nutty Professor, The (1996)	Comedy|Fantasy|Romance|Sci-Fi
886	1210	Star Wars: Episode VI - Return of the Jedi (1983)	Action|Adventure|Sci-Fi
634	852	Tin Cup (1996)	Comedy|Drama|Romance
565	719	Multiplicity (1996)	Comedy
1047	1393	Jerry Maguire (1996)	Drama|Romance
80	104	Happy Gilmore (1996)	Comedy
9	14	Nixon (1995)	Drama
532	661	James and the Giant Peach (1996)	Adventure|Animation|Children|Fantasy|Musical
587	762	Striptease (1996)	Comedy|Crime
4	9	Sudden Death (1995)	Action
621	832	Ransom (1996)	Crime|Thriller
523	637	Sgt. Bilko (1996)	Comedy
103	140	Up Close and Personal (1996)	Drama|Romance

SINGLE-VALUE DECOMPOSITION:

A well-known matrix factorization method is Singular value decomposition (SVD). SVD is an algorithm that decomposes a matrix into the best lower rank (i.e., smaller/simpler) approximation of the original matrix at a high level. Both SciPy and NumPy have functions that perform singular value decomposition. We use the SciPy function svds because we can choose the number of latent factors to use to approximate (rather than truncate) the original score matrix.

```
from surprise.model_selection import cross_validate

# Load Reader library
reader = Reader()

# Load ratings dataset with Dataset library
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
```

✓ 0.2s

```
# Use the SVD algorithm.
svd = SVD()

# Compute the RMSE of the SVD algorithm.
cross_validate(svd, data, measures=['RMSE'], cv=5)
```

✓ 20.3s

```
{'test_rmse': array([0.86884371, 0.87968017, 0.87936554, 0.87739145, 0.86824023]),
 'fit_time': (3.610891103744507,
  3.6862521171569824,
  3.604750871658325,
  3.6243929862976074,
  3.661891222000122),
 'test_time': (0.21634793281555176,
  0.20599794387817383,
  0.20424699783325195,
  0.2981231212615967,
  0.20337605476379395)}
```

MODEL EVALUATION OF SVD:

```python
def evaluation_collaborative_svd_model(userId,userOrItem):

    movieIdsList= list()
    global movieIdRating,predict
    movieRatingList=list()
    #movieIdRatingaa= df_movies_ratings(columns=['movieId','rating'])
    movieIdRating= pd.DataFrame(columns=['movieId','rating'])
    if userOrItem== True:
        movieIdsList=getRecommendedMoviesAsperUserSimilarity(userId)
    else:
        movieIdsList=recommendedMoviesAsperItemSimilarity(user_id)
    for movieId in movieIdsList:
        predict = svd.predict(userId, movieId)
        movieRatingList.append([movieId,predict.est])
        movieIdRating = pd.DataFrame(np.array(movieRatingList), columns=['movieId','rating'])
        count=movieIdRating[(movieIdRating['rating'])>=3]['movieId'].count()
        total=movieIdRating.shape[0]
        hit_ratio= count/total
    return hit_ratio
```
✓ 0.2s Python

```python
print("Hit ratio of User-user collaborative filtering")
print(evaluation_collaborative_svd_model(user_id,True))
print("Hit ratio of Item-Item collaborative filtering")
print(evaluation_collaborative_svd_model(user_id,False))
#print(classification_report(movieIdRating, predict))
```
✓ 0.6s Python

```
Hit ratio of User-user collaborative filtering
0.5555555555555556
Hit ratio of Item-Item collaborative filtering
0.8888888888888888
```

SVD++

To build a robust recommender system, you need to develop a model that considers explicit and implicit user feedback. There are less obvious types of implicit data in the MovieLens dataset. The dataset shows the ratings and which movies users rate, regardless of how they rated those movies. In other words, users implicitly tell their tastes by giving their opinions and giving them a (high or low) rating. This reduces the evaluation matrix to a binary matrix. Here, "1" means "evaluated" and "0" means "unevaluated". Indeed, this binary data is not as broad and independent as other sources of implicit feedback. Nevertheless, we have found that including this type of implicit data (specific to rating-based recommender systems) can significantly improve the accuracy of predictions. SVD ++ takes this implicit feedback into account and provides higher accuracy.

```
trainset, testset = train_test_split(data, test_size=.15)
#print(data)
type(data)
#print(data)
```

✓ 0.7s

```
surprise.dataset.DatasetAutoFolds
```

```
from sklearn.metrics import classification_report
algo_svdpp = SVDpp(n_factors=160, n_epochs=10, lr_all=0.005, reg_all=0.1)
algo_svdpp.fit(trainset)
test_pred = algo_svdpp.test(testset)
print("SVDpp : Test Set")
accuracy.rmse(test_pred, verbose=True)
```

✓ 3m 49.2s

```
SVDpp : Test Set
RMSE: 0.9511

0.9511003773997109
```

HYBRID MODEL (Content-Based + SVD)

Now that we've developed the individual models described above, we'll stack them for better results.

This model performs content-based filtering to determine which movies to recommend to users. It then filters and sorts the CF recommendations based on the SVD's predicted score.

```
df_movies=movies
def hybrid_content_svd_model(userId):

    recommended_movies_by_content_model = get_recommendation_content_model(userId)
    recommended_movies_by_content_model = df_movies[df_movies.apply(lambda movie: movie["title"] in recommended_movies_by_content_model, axis=1)]
    for key, columns in recommended_movies_by_content_model.iterrows():
        predict = svd.predict(userId, columns["movieId"])
        recommended_movies_by_content_model.loc[key, "svd_rating"] = predict.est
#          if(predict.est < 2):
#              recommended_movies_by_content_model = recommended_movies_by_content_model.drop([key])
    return recommended_movies_by_content_model.sort_values("svd_rating", ascending=False).iloc[0:11]

hybrid_content_svd_model(user_id)
```

✓ 4.7s

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  recommended_movies_by_content_model.loc[key, "svd_rating"] = predict.est
```

	movieId	title	genres	svd_rating
3638	4993	Lord of the Rings: The Fellowship of the Ring,...	Adventure|Fantasy	3.567286
224	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi	3.485764
828	1089	Reservoir Dogs (1992)	Crime|Mystery|Thriller	3.472068
826	1086	Dial M for Murder (1954)	Crime|Mystery|Thriller	3.464328
898	1196	Star Wars: Episode V - The Empire Strikes Back...	Action|Adventure|Sci-Fi	3.436822
949	1250	Bridge on the River Kwai, The (1957)	Adventure|Drama|War	3.432738
686	904	Rear Window (1954)	Mystery|Thriller	3.401315
254	293	LÃ©on: The Professional (a.k.a. The Profession...	Action|Crime|Drama|Thriller	3.338270
933	1233	Boot, Das (Boat, The) (1981)	Action|Drama|War	3.303858
599	745	Wallace & Gromit: A Close Shave (1995)	Animation|Children|Comedy	3.290091
1730	2324	Life Is Beautiful (La Vita Ã¨ bella) (1997)	Comedy|Drama|Romance|War	3.289404

Conclusion

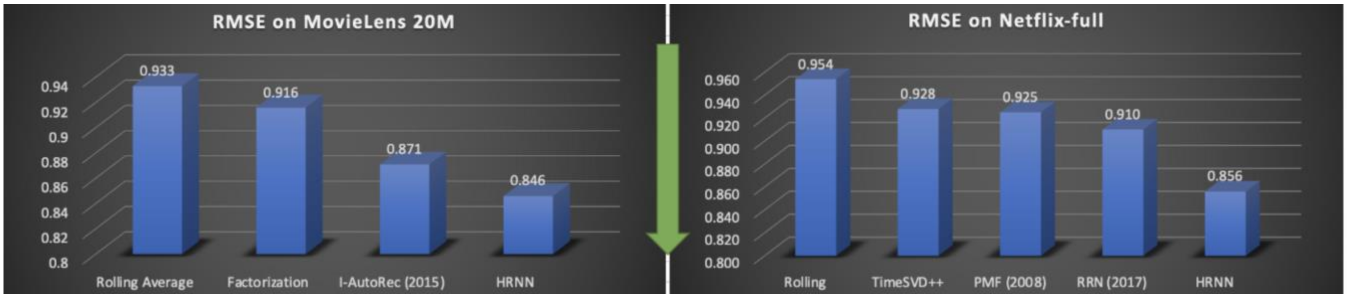
From the models, we can conclude the following: -

Parameters	**Collaborative**	**Content Based**	**Hybrid**
Precision	Low	Average	High
Scalability	Less	Average	High
Computing Time	Average	High	Average
Memory usage	Average	Low	High

We rate recommendation models based on RMSE (Root Mean Square Error) values.

RMSE for Hybrid (SVD + Content) is 0.92 and has a precision of 71.5%, whereas SVD++ has an RMSE of 0.95 and an accuracy of 55%, which is considerably higher for the hybrid model. The advantages of the hybrid model would be partially eliminating the disadvantages of both Collaborative and Content-Based filtering.

The state-of-the-art models built by Netflix and MovieLens have RMSE values, as mentioned below.



The Hybrid model fared well with the existing models but still has room for improvement.
The future scope of this is to enhance the hybrid model to give us better precision and even lower RMSE values. One more step would be incorporating NLP concepts into the synopsis of each movie and finding similarities between them, and recommending accordingly.

References

- https://www.datacamp.com/community/tutorials/recommender-systems-python
- https://medium.com/recombee-blog/machine-learning-for-recommender-systems-part-1-algorithms-evaluation-and-cold-start-6f696683d0ed
- https://www.quora.com/Whats-the-difference-between-SVD-and-SVD++
- https://github.com/gpfvic/IRR/blob/master/Factorization%20meets%20the%20neighborhood-%20a%20multifaceted%20collaborative%20filtering%20model.pdf
- https://blog.statsbot.co/singular-value-decomposition-tutorial-52c695315254
- https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

- https://medium.com/@james_aka_yale/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223
- http://www.awesomestats.in/python-recommending-movies/
- https://en.wikipedia.org/wiki/Singular_value_decomposition
- https://surprise.readthedocs.io/en/stable/index.html