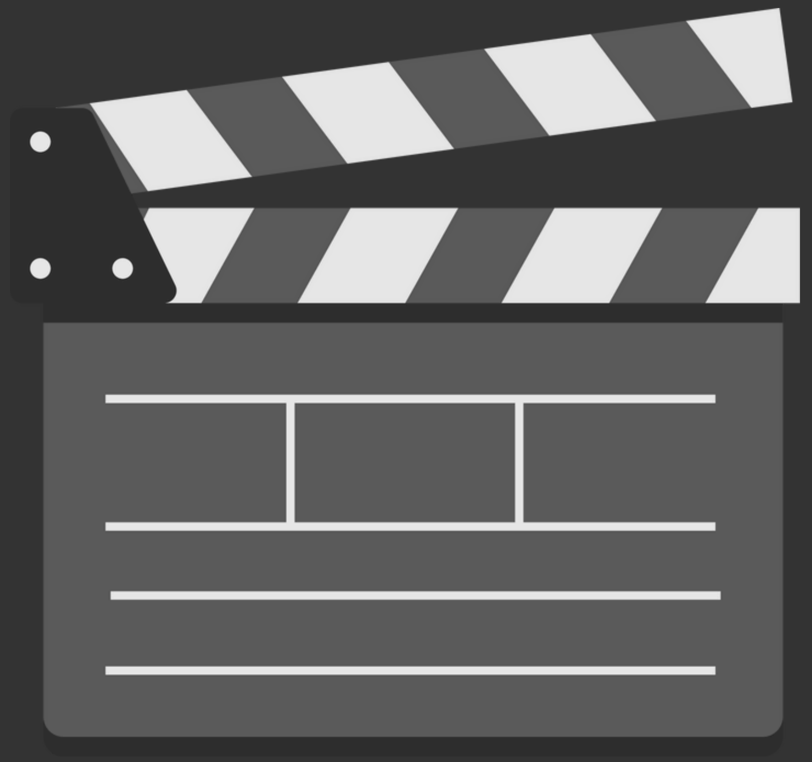
Machine Learning Term Project

Movie Recommendation System

GROUP A 김소정 김현지 김동하 김유현 이기태

INDEX

Purpose



Movie Recommendation System

- Helps users discover new movies more easily by recommending movies they are likely to like.

- We provide recommendations optimized for each individual based on features such as the movie's genre, plot, cast, and director, as well as users' movie review data.

Datasets

TMDB 5000 Movie dataset

TMDB_5000_movies.csv

Metatdata about movie
(MovieId, Title, Genre etc..)

TMDB_5000_credits.csv

MovieId, Cast & Crew Information

The Movies dataset

ratings_small.csv

UserId & MovieId,
User’s ratings of movie

Machine Learning Term Project

Model	#1	

Collaborative Filtering

Collaborative Filtering

Import nessary libaries

- Pandas, Numpy
- Normalizer
- KNN, SVD, NMF
- Metrics(RMSE, MAE)

```python
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.neighbors import NearestNeighbors
from sklearn.decomposition import TruncatedSVD, NMF as SklearnNMF
from sklearn.preprocessing import Normalizer
from surprise import SVD, Dataset, Reader
from surprise import KNNBasic, NMF as SurpriseNMF
from surprise.model_selection import cross_validate
```

Preprocessing Dataset

- Combining 'ratings' and 'movies' data, we create a new DataFrame called ratings_movies.

→ Refer to the title and rating of the movie at once.

```python
# 1. Load and preprocess data
drive.mount('/content/drive')
ratings = pd.read_csv('/content/drive/MyDrive/ratings_small.csv')
movies = pd.read_csv('/content/drive/MyDrive/tmdb_5000_movies.csv')
movies.rename(columns={'id': 'movieId'}, inplace=True)
ratings_movies = pd.merge(ratings, movies[['movieId', 'title']], on='movieId')
data = ratings_movies.pivot_table('rating', index='userId', columns='title').fillna(0)
```

SVD based Function

SVD Model

Based on 'User Past Ratings'

- The rating data is converted into a Surprise-style dataset and trained on the SVD model.

- Calculate the predicted ratings for each movie and return the films with the highest predicted ratings to the recommended list.

```python
# 2. SVD-based recommendation function (modified to output only movie titles)
def svd_recommendations(user_id, ratings_df, movies_df, top_n=10):
    reader = Reader(rating_scale=(0.5, 5))
    data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader)
    svd = SVD()
    trainset = data.build_full_trainset()
    svd.fit(trainset)

    # Generate predictions for all movies the user hasn't rated yet
    user_ratings = []
    for movie_id in ratings_df['movieId'].unique():
        pred = svd.predict(user_id, movie_id)
        user_ratings.append((movie_id, pred.est))

    # Sort movies by predicted rating in descending order and select the top_n recommendations
    user_ratings.sort(key=lambda x: x[1], reverse=True)
    top_movie_predictions = user_ratings[:top_n]

    # Extract only the movie titles
    recommended_titles = [
        movies_df[movies_df['movieId'] == movie_id]['title'].values[0]
        for movie_id, rating in top_movie_predictions
        if not movies_df[movies_df['movieId'] == movie_id].empty
    ]

    return recommended_titles
```

KNN based Function

KNN Model

Similar to a 'particular movie

- Using the ratings for each movie in 'user_movie_matrix', the KNN algorithm is used to find similar movies.

- If you enter your movie ID, recommend a movie similar to that movie

```python
# 3. KNN-based recommendation function
user_movie_matrix = ratings.pivot_table(index='userId', columns='movieId', values='rating').fillna(0
knn = NearestNeighbors(n_neighbors=6, algorithm='auto')
knn.fit(user_movie_matrix.T)

def knn_recommendations(movie_id, user_movie_matrix, movies_df, num_recommendations=5):
    knn = NearestNeighbors(n_neighbors=num_recommendations+1, algorithm='auto')
    knn.fit(user_movie_matrix.T)
    movie_idx = user_movie_matrix.columns.get_loc(movie_id)
    distances, indices = knn.kneighbors(user_movie_matrix.iloc[:, movie_idx].values.reshape(1, -1),
    recommended_movie_ids = user_movie_matrix.columns[indices.flatten()][1:]
    recommended_titles = movies_df[movies_df['movieId'].isin(recommended_movie_ids)]['title'].values

    return recommended_titles
```

NMF based Function

NMF Model

Decomposing the user's rating data

- Matrix decomposition technique

- Apply NMF to user_movie_matrix and compare the vectors of each film to recommend similar films

```
# 4. NMF-based recommendation function
sklearn_nmf = SklearnNMF(n_components=20, init='random', random_state=42)
nmf_matrix = sklearn_nmf.fit_transform(user_movie_matrix)
nmf_matrix_normalized = Normalizer().fit_transform(nmf_matrix)

def nmf_recommendations(movie_id, user_movie_matrix, movies_df, num_recommendations=5):
    # Normalize the NMF matrix
    nmf_matrix_normalized = Normalizer().fit_transform(sklearn_nmf.fit_transform(user_movie_matrix))

    # Get the index of the movie
    movie_idx = user_movie_matrix.columns.get_loc(movie_id)

    # Extract the movie vector from the NMF matrix
    movie_vector = nmf_matrix_normalized[movie_idx, :]

    # Compute similarity scores with all movies
    similarity_scores = np.dot(nmf_matrix_normalized, movie_vector)

    # Get the indices of the most similar movies
    recommended_idx = similarity_scores.argsort()[-num_recommendations-1:-1]

    # Get the recommended movie ids and titles
    recommended_movie_ids = user_movie_matrix.columns[recommended_idx]
    recommended_titles = movies_df[movies_df['movieId'].isin(recommended_movie_ids)]['title'].values

    return recommended_titles
```

Recommendation Results

SVD, KNN, NMF Recommendation Results

```python
# 6. Model evaluation and recommendation results
# SVD performance evaluation
reader = Reader(rating_scale=(0.5, 5))
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
svd = SVD()

print("\nSVD 추천 결과")
print(svd_recommendations(1, ratings, movies))

# KNN recommendation results
print("\nKNN 추천 결과")
print(knn_recommendations(1, user_movie_matrix, movies))

# NMF recommendation results
print("\nNMF 추천 결과")
print(nmf_recommendations(1, user_movie_matrix, movies))
```

```
SVD 추천 결과
["Pandora's Box", 'Galaxy Quest']

KNN 추천 결과
['Bridge to Terabithia' 'Reign Over Me']

NMF 추천 결과
['Blade Runner' 'Aliens' 'Pulp Fiction' 'D.E.B.S.']
```

Model Comparison & Results

Compare

RMSE, MAE(accuracy of the model), Fit time, and Test time

```python
# 7. Model evaluation across different algorithms
def evaluate_models(ratings_df):
    reader = Reader(rating_scale=(0.5, 5))
    data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader)

    # SVD model evaluation
    svd = SVD()
    svd_results = cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

    # KNN model evaluation
    knn = KNNBasic(sim_options={'name': 'cosine', 'user_based': True}, verbose=False)
    knn_results = cross_validate(knn, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

    # Surprise NMF model evaluation
    nmf = SurpriseNMF(n_factors=20, random_state=42)
    nmf_results = cross_validate(nmf, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

    # Results comparison
    results_df = pd.DataFrame({
        'Model': ['SVD', 'KNN', 'NMF'],
        'Mean RMSE': [np.mean(svd_results['test_rmse']),
                      np.mean(knn_results['test_rmse']),
                      np.mean(nmf_results['test_rmse'])],
        'Mean MAE': [np.mean(svd_results['test_mae']),
                     np.mean(knn_results['test_mae']),
                     np.mean(nmf_results['test_mae'])],
        'Fit Time': [np.mean(svd_results['fit_time']),
                     np.mean(knn_results['fit_time']),
                     np.mean(nmf_results['fit_time'])],
        'Test Time': [np.mean(svd_results['test_time']),
                      np.mean(knn_results['test_time']),
                      np.mean(nmf_results['test_time'])]
    })

    return results_df
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9069  0.8940  0.8834  0.9075  0.8917  0.8967  0.0093
MAE (testset)     0.6960  0.6877  0.6828  0.7000  0.6868  0.6907  0.0063
Fit time          1.53    1.55    1.82    2.38    1.64    1.78    0.31
Test time         0.12    0.27    0.19    0.22    0.11    0.18    0.06
Evaluating RMSE, MAE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9980  0.9917  0.9962  0.9945  0.9906  0.9942  0.0027
MAE (testset)     0.7712  0.7646  0.7698  0.7657  0.7672  0.7677  0.0024
Fit time          0.19    0.21    0.22    0.20    0.20    0.21    0.01
Test time         1.41    1.42    1.42    1.53    1.86    1.53    0.17
Evaluating RMSE, MAE of algorithm NMF on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9413  0.9451  0.9554  0.9422  0.9505  0.9469  0.0053
MAE (testset)     0.7161  0.7175  0.7277  0.7158  0.7233  0.7201  0.0047
Fit time          3.82    2.96    2.97    3.76    3.45    3.39    0.37
Test time         0.26    0.10    0.29    0.18    0.10    0.18    0.08
  Model  Mean RMSE  Mean MAE  Fit Time  Test Time
0   SVD   0.896701  0.690657  1.782597   0.182774
1   KNN   0.994208  0.767707  0.205861   1.525520
2   NMF   0.946883  0.720087  3.390776   0.184946
```

Machine Learning Term Project

Model | # 2

Content – Based Filtering

Content – Based Filtering

Import nessary libaries

- Pandas, Numpy
- TfidfVectorizer
- CountVectorizer
- Cosine_similarity

```
import pandas as pd
import numpy as np
import ast
import warnings; warnings.filterwarnings('ignore')

from ast import literal_eval
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

movies = pd.read_csv("/content/tmdb_5000_movies.csv")
credits = pd.read_csv("/content/tmdb_5000_credits.csv")
```

Join 'movies' &'credits' data frame.

- Merge the same data such as movie Id, Title, etc.

- Our System: 'Genre Based'
→ Remove less relevant characteristics and extract necessary data items.

```
#movie 데이터셋과 credit 데이터셋의 병합
movie_credits = pd.merge(movies, credits, left_on='id', right_on='movie_id', how='left') # 같은 id 기준으로 데이터 읽기
movie_credits = pd.merge(movies, credits, left_on='original_title', right_on='title', how='left')
movie_credits = movie_credits.drop(columns=['homepage', 'status', 'production_companies', 'production_countries',
                                            'cast', 'crew', 'original_language', 'tagline', 'revenue',
                                            'budget', 'spoken_languages','runtime', 'release_date'])

#병합된 데이터셋에서 필요한 항목만 추출
movie_credits_df = movie_credits[['id', 'genres', 'original_title', 'vote_average', 'vote_count', 'popularity', 'keywords']]
movie_credits.info()
# movie_credits.describe()
```

Data Parsing & Key Extraction

- Converting string data in genres and keywords columns of the data frame into list form.

```
movie_credits_df['genres'] = movie_credits_df['genres'].apply(literal_eval)
movie_credits_df['keywords'] = movie_credits_df['keywords'].apply(literal_eval)
```

- 'Name' extraction

Use 'apply lambda' to extract 'name key in genres, keywords column.

```
# apply lambda -> genre, keyword column에서 name 값 추출

movie_credits_df['genres'] = movie_credits_df['genres'].apply(lambda x : [ y['name'] for y in x])
movie_credits_df['keywords'] = movie_credits_df['keywords'].apply(lambda x : [y['name'] for y in x])
movie_credits_df[['genres', 'keywords']][:5]
```

	genres	keywords
0	[Action, Adventure, Fantasy, Science Fiction]	[culture clash, future, space war, space colony, society, space travel, futuristic, romance, spa...
1	[Adventure, Fantasy, Action]	[ocean, drug abuse, exotic island, east india trading company, love of one's life, traitor, ship...
2	[Action, Adventure, Crime]	[spy, based on novel, secret agent, sequel, mi6, british secret service, united kingdom]
3	[Action, Crime, Drama, Thriller]	[dc comics, crime fighter, terrorist, secret identity, burglar, hostage drama, time bomb, gotham...
4	[Action, Adventure, Science Fiction]	[based on novel, mars, medallion, space travel, princess, alien, steampunk, martian, escape, edg...

Vectorization & Get Similarity

- **TfidVectorizer**

To convert the genre into vector and transfrom genre list for each movie as a vector.

- **Cosine_similarity**

To calculate the similarity between the genre vectors of different movies.

- **Stopword Removal**

Delete words that can't be used ('the', 'and').

```
# TF-IDF & 코사인 유사도
movie_credits_df['genres_literal'] = movie_credits_df['genres'].apply(lambda x: (' ').join(x))
tfidf = TfidfVectorizer(stop_words='english')
genre_mat = tfidf.fit_transform(movie_credits_df['genres_literal'])
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
```
✓ 0.3s

```
# 코사인 유사도 출력
print(cosine_sim.shape)
print(cosine_sim[:2])
```
✓ 0.0s

```
(4803, 4803)
[[1.         0.82354628 0.54669475 ... 0.11470537 0.         0.06005049]
 [0.82354628 1.         0.64396703 ... 0.10974101 0.         0.05745155]]
```

```
# cosine_sim -> movies_df의 genre_mat의 데이터 별 유사도 정보
# genre_sim_sorted_ind -> 각 레코드의 장르 cosine 유사도가 가장 높은 순의 index 값
# argsort() 함수 -> 유사도가 높은 순으로 정렬

genre_sim_sorted_ind = cosine_sim.argsort()[:, ::-1]
print(genre_sim_sorted_ind[:1])
```
✓ 0.8s

```
[[3494  813  232 ... 4714 4801 4716]]
```

Genre similarity-based Function

- Recommend movies based on high genre similarity with higher ragins.

- Using this function, we recommended '10' movies with similar genres to 'Avatar'.

```
# find_sim_movie(): 장르 유사도에 따라 영화 추천
def find_sim_movie(df, sorted_ind, title_name, top_n=10):
    # title column이 입력된 title_name의 값 출력
    title_movie = df[df['original_title'] == title_name]
    # title named를 가진 index를 ndarray로 변환 후
    # genre_sim_sorted_ind에서 유사도 순으로 n개의 index 추출
    title_index = title_movie.index.values
    similar_indexes = sorted_ind[title_index, :(top_n)]

    # 추출된 top_n_index 출력
    # 2차원 데이터 형태임으로 movies에서 새롭게 사용하기 위해 1차원 array로 변환
    print(similar_indexes)
    similar_indexes = similar_indexes.reshape(-1)

    return df.iloc[similar_indexes]
```

```
# find_sim_movie()를 통해 영화 10개 추천

similar_movies = find_sim_movie(movie_credits_df, genre_sim_sorted_ind, 'Avatar', 10)
similar_movies[['original_title', 'vote_average', 'genres']]
```

```
[[   0   14  870  813   46 3496 1297 1654  419  420]]
```

Initial Recommendation Result

- Occurrence of an unusually high rating reflection of an obscure movie.
(i.e., Low 'vote count' but High 'vote average')

→ Introduce a new evaluation method to reflect 'vote count' in ratings.

```
# sort_values() -> vote_average 내림차순 10개 추출

movie_credits_df[['original_title', 'vote_average', 'vote_count', 'genres']].sort_values
```

	original_title	vote_average	vote_count	genres
4251	Me You and Five Bucks	10.0	2	[Romance, Comedy, Drama]
4049	Dancer, Texas Pop. 81	10.0	1	[Comedy, Drama, Family]
3521	Stiff Upper Lips	10.0	1	[Comedy]
4667	Little Big Top	10.0	1	[Comedy]
3996	Sardaarji	9.5	2	[]
2388	One Man's Hero	9.3	2	[Western, Action, Drama, History]
2972	There Goes My Baby	8.5	2	[Drama, Comedy]
1883	The Shawshank Redemption	8.5	8205	[Drama, Crime]
3339	The Godfather	8.4	5893	[Drama, Crime]
2798	The Prisoner of Zenda	8.4	11	[Adventure, Drama, Romance]

New Weighted Rating Formula

- **Weighted Rating**

(V/(V+m)) * R + (m/(V+m)) * C

V: vote_count
m: vote_average
R: average rating of each movie
C: average rating of total movie
M: Number of times the top x during voting

→ Apply a new rating method
'weights the number of vote'.

```
# m: 전체 중 상위 70%에 해당하는 횟수

C = movie_credits_df['vote_average'].mean()
m = movie_credits_df['vote_count'].quantile(0.7)
print('C: ', round(C, 3), 'm:', round(m, 3))
```

```
C:  6.092 m: 581.0
```

```
# 새로운 평점 정보: vote_weighted & 함수 명: weighted_vote_average()
# vote_count, vote_average, m, C 값을 바탕으로 record별 평점 반환
# V: vote_count, R: vote_average

def weighted_vote_average(record):
    V = record['vote_count']
    R = record['vote_average']

    return ((V/(V+m)) * R) + ((m/(m+V)) * C)

movie_credits_df['weighted_vote'] = movie_credits_df.apply(weighted_vote_average, axis=1)
```

Complete Recommendation System

Content-Based Filtering

- Adopting the method of extracting movies with high genre similarity in the order of 'weighted_vote'.

- Using the changed recommendation function, We recommend 10 movies similar to 'Avatar'

```python
# 새롭게 부여된 weighted_vote 평점 순으로 10개 추출
# weighted_vote를 기반으로 'Avatar'와 유사한 영화들을 높은 순서로 추천
def find_sim_movie_by_weighted_vote(df, sorted_ind, title_name, top_n=10):
    title_movie = df[df['original_title'] == title_name]

    # 해당 영화의 인덱스 추출
    title_index = title_movie.index.values
    similar_indexes = sorted_ind[title_index, :(top_n * 2)]
    similar_indexes = similar_indexes.reshape(-1)

    # 본인 제외하고 top_n개 유사 영화 추천
    similar_movies = df.iloc[similar_indexes].drop(title_index)

    # weighted_vote 기준으로 상위 top_n개 반환
    return similar_movies.sort_values('weighted_vote', ascending=False).head(top_n)

# 'Avatar'와 유사한 영화 추천
similar_movies_by_weighted_vote = find_sim_movie_by_weighted_vote(movie_credits_df, genre_sim_sorted_ind, 'Avatar', 10)
similar_movies_by_weighted_vote[['original_title', 'vote_average', 'weighted_vote', 'genres']]

# movie_credits_df[['original_title', 'vote_average', 'weighted_vote', 'vote_count', 'genres']].sort_values('weighted_vote', ascending = False)[:10]
```

	original_title	vote_average	weighted_vote	genres
46	X-Men: Days of Future Past	7.5	7.376331	[Action, Adventure, Fantasy, Science Fiction]
158	Star Trek	7.4	7.251006	[Science Fiction, Action, Adventure]
813	Superman	6.9	6.607285	[Action, Adventure, Fantasy, Science Fiction]
14	Man of Steel	6.5	6.465876	[Action, Adventure, Fantasy, Science Fiction]
420	Hellboy II: The Golden Army	6.5	6.387655	[Adventure, Fantasy, Science Fiction]
870	Superman II	6.5	6.304279	[Action, Adventure, Fantasy, Science Fiction]
232	The Wolverine	6.3	6.273970	[Action, Science Fiction, Adventure, Fantasy]
3210	Star Wars: Clone Wars (Volume 1)	8.0	6.177101	[Action, Adventure, Animation, Fantasy, Science Fiction]
1192	Small Soldiers	6.2	6.142745	[Comedy, Adventure, Fantasy, Science Fiction, Action]
1934	Sheena	5.0	6.052533	[Action, Adventure, Comedy, Fantasy, Science Fiction]

E N D

Thank You