Deep Learning Recommender.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

TPU RAM Disk

Import Libraries

```
# To store the data
import pandas as pd

# To do linear algebra
import numpy as np

# To create plots
import matplotlib.pyplot as plt

# To create interactive plots
from plotly.offline import init_notebook_mode, plot, iplot
import plotly.graph_objs as go
init_notebook_mode(connected=True)

# To shift lists
from collections import deque

# To compute similarities between vectors
from sklearn.metrics import mean_squared_error
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer

# To use recommender systems
import surprise as sp
from surprise.model_selection import cross_validate

# To create deep learning models
from keras.layers import Input, Embedding, Reshape, Dot, Concatenate, Dense, Dropout
from keras.models import Model

# To create sparse matrices
from scipy.sparse import coo_matrix

# To light fm
from lightfm import LightFM
from lightfm.evaluation import precision_at_k

# To stack sparse matrices
from scipy.sparse import vstack
```

Load Movie-Data

```
[4] # Load data for all movies
movie_titles = pd.read_csv('/content/drive/MyDrive/research/movies/netflix/movie_titles.csv', on_bad_lines='skip',
                           encoding = 'ISO-8859-1',
                           header = None,
                           names = ['Id', 'Year', 'Name']).set_index('Id')

print('Shape Movie-Titles:\t{}'.format(movie_titles.shape))
movie_titles.sample(5)
```

```
Shape Movie-Titles:     (17434, 2)
```

Id	Year	Name
13623	1966.0	Closely Watched Trains
299	2001.0	Bridget Jones's Diary
9361	2001.0	Second to Die
14266	2000.0	Cutey Honey
7137	2000.0	GateKeepers

```
[5] # Load a movie metadata dataset
movie_metadata = pd.read_csv('/content/drive/MyDrive/research/movies/The Movies Dataset/movies_metadata.csv', low_memory=False)[['original_title', 'overview', 'vote_c
# Remove the long tail of rarly rated moves
movie_metadata = movie_metadata[movie_metadata['vote_count']>10].drop('vote_count', axis=1)

print('Shape Movie-Metadata:\t{}'.format(movie_metadata.shape))
movie_metadata.sample(5)
```

```
Shape Movie-Metadata:   (21604, 1)
```

original_title	overview
Tremors	Hick handymen Val McKee and Earl Bassett can b...
La Totale!	To his family, François Voisin is nothing more...
The Juror	With his gangster boss on trial for murder, a ...
Wolf	A young boxer is making a name for himself ins...

The Bucket List	Corporate billionaire Edward Cole and working ...

```
[6] # Load a movie metadata dataset
    movie_metadata = pd.read_csv('/content/drive/MyDrive/research/movies/The Movies Dataset/movies_metadata.csv', low_memory=False)[['original_title', 'overview', 'vote_c
    # Remove the long tail of rarly rated moves
    movie_metadata = movie_metadata[movie_metadata['vote_count']>10].drop('vote_count', axis=1)

    print('Shape Movie-Metadata:\t{}'.format(movie_metadata.shape))
    movie_metadata.sample(5)
```

```
Shape Movie-Metadata:    (21604, 1)
```

original_title	overview
King Solomon's Mines	Ever in search of adventure, explorer Allan Qu...
Night of the Demon	American professor John Holden arrives in Lond...
Clerks - The Flying Car	Video short from Kevin Smith. Dante and Randa...
Girls Trip	Four girlfriends take a trip to New Orleans fo...
The Miracle Woman	A phony faith healer falls for a blind man and...

Load User-Data And Preprocess Data-Structure

```
[7] # Load single data-file
    df_raw = pd.read_csv('/content/drive/MyDrive/research/movies/netflix/combined_data_1.txt', header=None, names=['User', 'Rating', 'Date'], usecols=[0, 1, 2])


    # Find empty rows to slice dataframe for each movie
    tmp_movies = df_raw[df_raw['Rating'].isna()]['User'].reset_index()
    movie_indices = [[index, int(movie[:-1])] for index, movie in tmp_movies.values]

    # Shift the movie_indices by one to get start and endpoints of all movies
    shifted_movie_indices = deque(movie_indices)
    shifted_movie_indices.rotate(-1)


    # Gather all dataframes
    user_data = []

    # Iterate over all movies
    for [df_id_1, movie_id], [df_id_2, next_movie_id] in zip(movie_indices, shifted_movie_indices):

        # Check if it is the last movie in the file
        if df_id_1<df_id_2:
            tmp_df = df_raw.loc[df_id_1+1:df_id_2-1].copy()
        else:
            tmp_df = df_raw.loc[df_id_1+1:].copy()

        # Create movie_id column
        tmp_df['Movie'] = movie_id

        # Append dataframe to list
        user_data.append(tmp_df)

    # Combine all dataframes
    df = pd.concat(user_data)
    del user_data, df_raw, tmp_movies, tmp_df, shifted_movie_indices, movie_indices, df_id_1, movie_id, df_id_2, next_movie_id
    print('Shape User-Ratings:\t{}'.format(df.shape))
    df.sample(5)
```

```
Shape User-Ratings:      (24053764, 4)
```

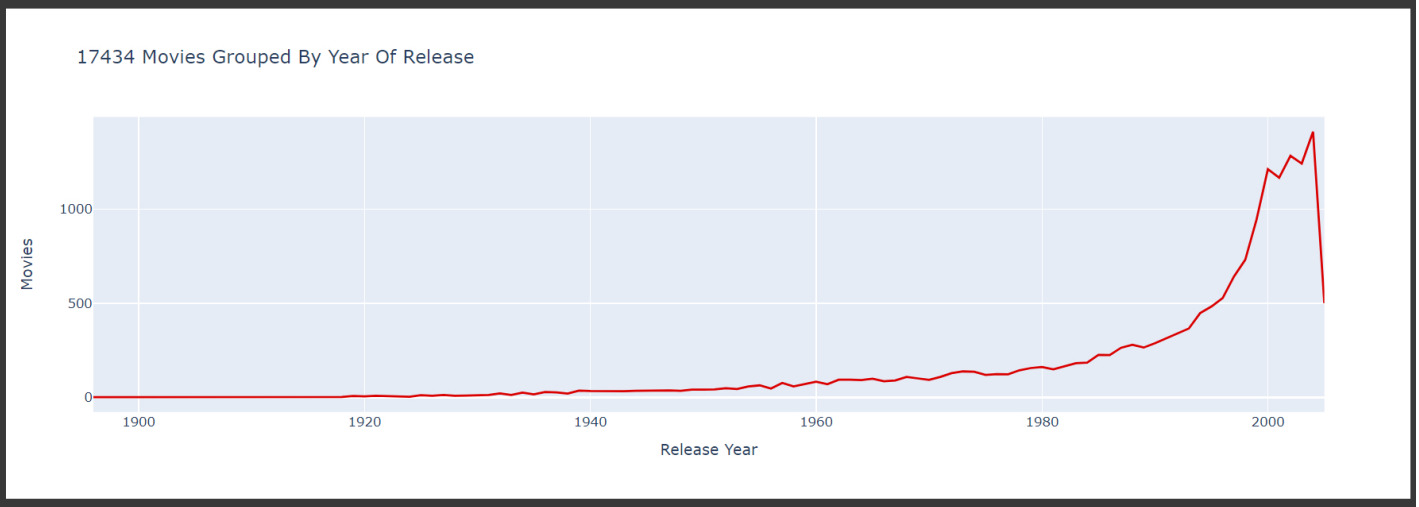
	User	Rating	Date	Movie
8051319	938477	5.0	2004-12-24	1615
12392287	437978	3.0	2001-01-06	2385
19723026	2618385	4.0	2003-08-07	3756
2332340	1661216	4.0	2003-12-05	443
20415055	2566541	4.0	2005-07-13	3875

Movies Release

```
[8] # Get data
    data = movie_titles['Year'].value_counts().sort_index()

    # Create trace
    trace = go.Scatter(x = data.index,
                       y = data.values,
                       marker = dict(color = '#db0000'))
    # Create layout
    layout = dict(title = '{} Movies Grouped By Year Of Release'.format(movie_titles.shape[0]),
                  xaxis = dict(title = 'Release Year'),
                  yaxis = dict(title = 'Movies'))

    # Create plot
    fig = go.Figure(data=[trace], layout=layout)
```

```
# Download the plot
fig.write_html("release.html")
```

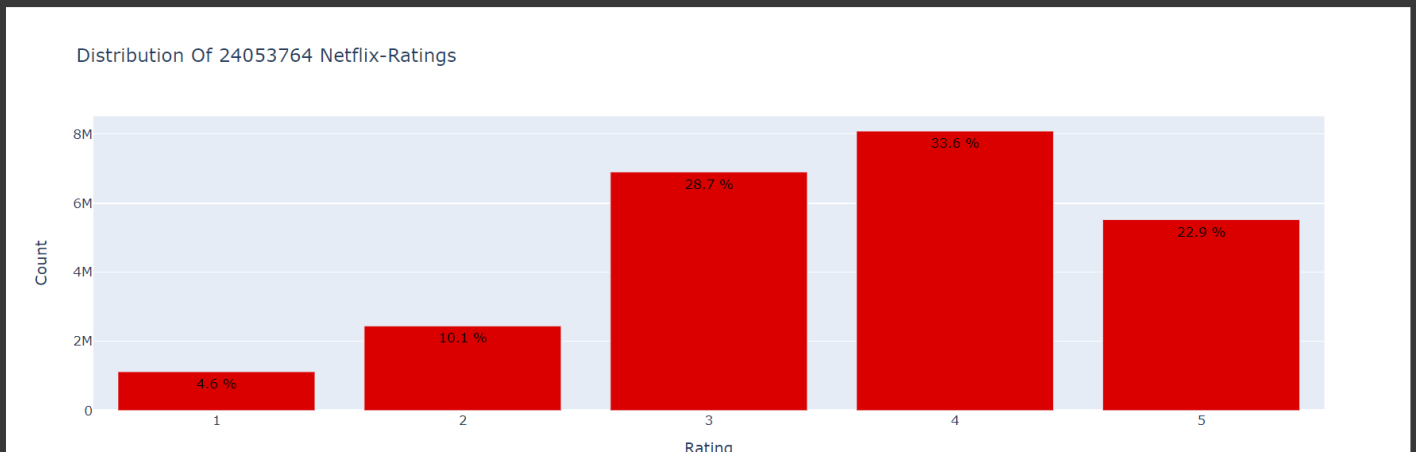
```
[9]  # it seems that plotly not directly show the plot data, so we need to make it into html file, and show it by calling using other functions

     from IPython.display import HTML

     # Read the HTML file
     with open("release.html", "r") as f:
         html_content = f.read()

     # Display the HTML content
     HTML(html_content)
```



Ratings Distributed

```
[10] # Get data
     data = df['Rating'].value_counts().sort_index(ascending=False)

     # Create trace
     trace = go.Bar(x = data.index,
                    text = ['{:.1f} %'.format(val) for val in (data.values / df.shape[0] * 100)],
                    textposition = 'auto',
                    textfont = dict(color = '#000000'),
                    y = data.values,
                    marker = dict(color = '#db0000'))
     # Create layout
     layout = dict(title = 'Distribution Of {} Netflix-Ratings'.format(df.shape[0]),
                   xaxis = dict(title = 'Rating'),
                   yaxis = dict(title = 'Count'))
     # Create plot
     fig = go.Figure(data=[trace], layout=layout)

     # Download the plot
     fig.write_html("rating.html")
```

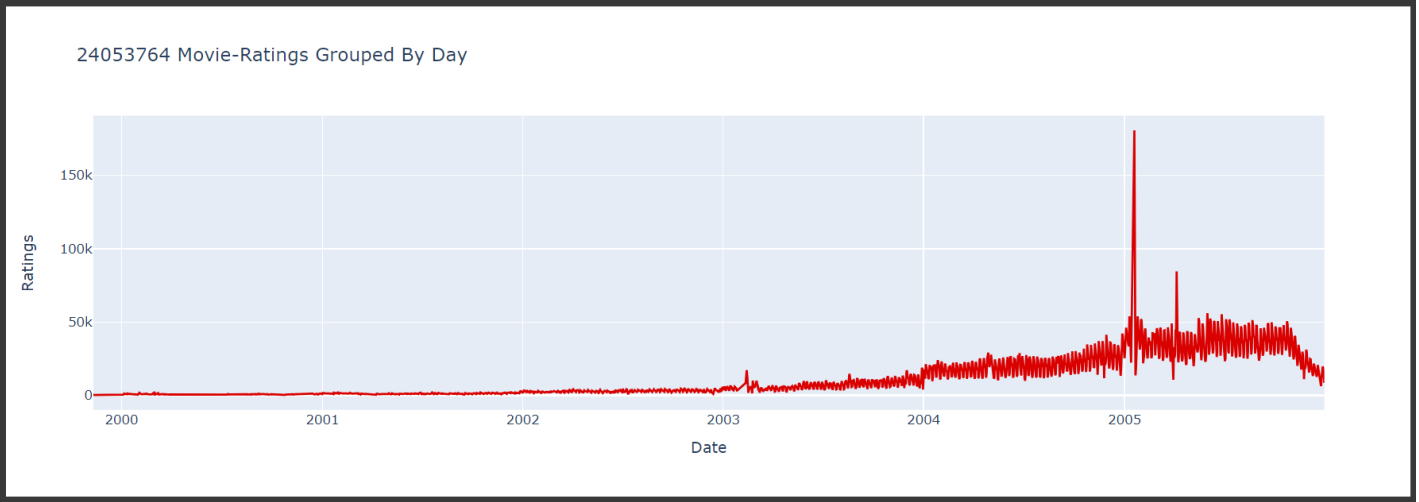
```
[11] # it seems that plotly not directly show the plot data, so we need to make it into html file, and show it by calling using other functions

     from IPython.display import HTML

     # Read the HTML file
     with open("rating.html", "r") as f:
         html_content = f.read()

     # Display the HTML content
     HTML(html_content)
```



Movies Rated Time

```
[12] # Get data
     data = df['Date'].value_counts()
     data.index = pd.to_datetime(data.index)
     data.sort_index(inplace=True)

     # Create trace
     trace = go.Scatter(x = data.index,
                        y = data.values,
                        marker = dict(color = '#db0000'))
     # Create layout
     layout = dict(title = '{} Movie-Ratings Grouped By Day'.format(df.shape[0]),
                   xaxis = dict(title = 'Date'),
                   yaxis = dict(title = 'Ratings'))

     # Create plot
     fig = go.Figure(data=[trace], layout=layout)

     # Download the plot
     fig.write_html("time.html")
```

```
[13] # it seems that plotly not directly show the plot data, so we need to make it into html file, and show it by calling using other functions

     from IPython.display import HTML

     # Read the HTML file
     with open("time.html", "r") as f:
         html_content = f.read()

     # Display the HTML content
     HTML(html_content)
```

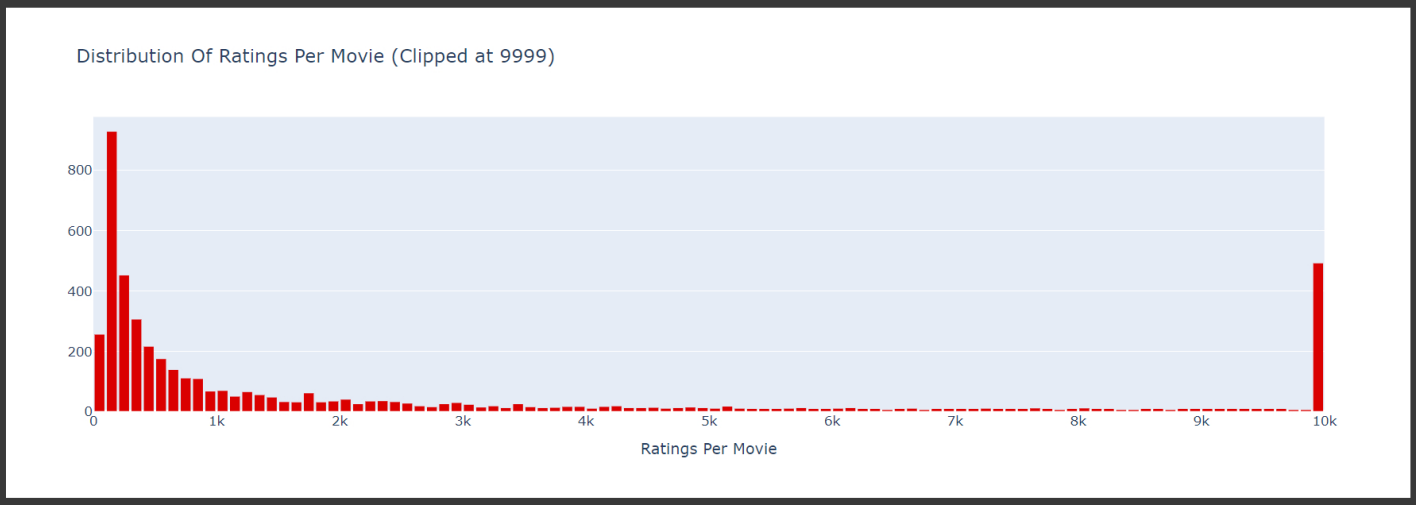


The Number Of Ratings Distributed For The Movies And The Users

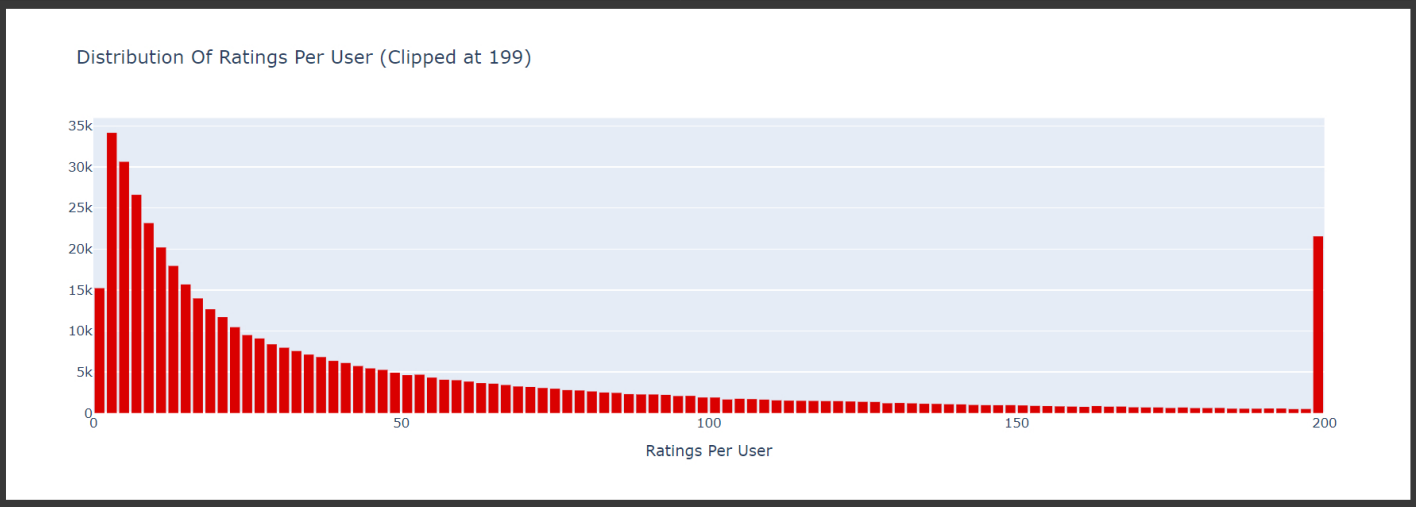
```
[14] ##### Ratings Per Movie #####
     # Get data
     data = df.groupby('Movie')['Rating'].count().clip(upper=9999)

     # Create trace
     trace = go.Histogram(x = data.values,
                          name = 'Ratings',
                          xbins = dict(start = 0,
                                       end = 10000,
                                       size = 100),
                          marker = dict(color = '#db0000'))
     # Create layout
     layout = go.Layout(title = 'Distribution Of Ratings Per Movie (Clipped at 9999)',
                        xaxis = dict(title = 'Ratings Per Movie'),
                        yaxis = dict(title = 'Count'),
                        bargap = 0.2)

     # Create plot
     fig = go.Figure(data=[trace], layout=layout)

     # Download the plot
     fig.write_html("rating_per_movie.html")



     ##### Ratings Per User #####
     # Get data
     data = df.groupby('User')['Rating'].count().clip(upper=199)

     # Create trace
```

```
    trace = go.Histogram(x = data.values,
                         name = 'Ratings',
                         xbins = dict(start = 0,
                                      end = 200,
                                      size = 2),
                         marker = dict(color = '#db0000'))
    # Create layout
    layout = go.Layout(title = 'Distribution Of Ratings Per User (Clipped at 199)',
                       xaxis = dict(title = 'Ratings Per User'),
                       yaxis = dict(title = 'Count'),
                       bargap = 0.2)

    # Create plot
    fig = go.Figure(data=[trace], layout=layout)

    # Download the plot
    fig.write_html("rating_per_user.html")
```

[15]
```
# it seems that plotly not directly show the plot data, so we need to make it into html file, and show it by calling using other functions

from IPython.display import HTML

# Read the HTML file
with open("rating_per_movie.html", "r") as f:
    html_content = f.read()

# Display the HTML content
HTML(html_content)
```



[16]
```
# it seems that plotly not directly show the plot data, so we need to make it into html file, and show it by calling using other functions

from IPython.display import HTML

# Read the HTML file
with open("rating_per_user.html", "r") as f:
    html_content = f.read()

# Display the HTML content
HTML(html_content)
```



Filter Sparse Movies And Users

[17]
```
# To reduce the dimensionality of the dataset I am filtering rarely rated movies and rarely rating users out.
```

```
# Filter sparse movies
min_movie_ratings = 10000
filter_movies = (df['Movie'].value_counts()>min_movie_ratings)
filter_movies = filter_movies[filter_movies].index.tolist()

# Filter sparse users
min_user_ratings = 200
filter_users = (df['User'].value_counts()>min_user_ratings)
filter_users = filter_users[filter_users].index.tolist()

# Actual filtering
df_filterd = df[(df['Movie'].isin(filter_movies)) & (df['User'].isin(filter_users))]
del filter_movies, filter_users, min_movie_ratings, min_user_ratings
print('Shape User-Ratings unfiltered:\t{}'.format(df.shape))
print('Shape User-Ratings filtered:\t{}'.format(df_filterd.shape))
```

```
Shape User-Ratings unfiltered:  (24053764, 4)
Shape User-Ratings filtered:    (4178032, 4)
```

Create Train- And Testset

[18] 1s

```
# Shuffle DataFrame
df_filterd = df_filterd.drop('Date', axis=1).sample(frac=1).reset_index(drop=True)

# Testingsize
n = 100000

# Split train- & testset
df_train = df_filterd[:-n]
df_test = df_filterd[-n:]
```

Transform The User-Ratings To User-Movie-Matrix

A large, sparse matrix will be created in this step. Each row will represent a user and its ratings and the columns are the movies. The interesting entries are the empty values in the matrix.

Empty values are unrated movies and could contain high values and therefore should be good recommendations for the respective user. The objective is to estimate the empty values to help our users.

[19] 3s

```
# Create a user-movie matrix with empty values
df_p = df_train.pivot_table(index='User', columns='Movie', values='Rating')
print('Shape User-Movie-Matrix:\t{}'.format(df_p.shape))
df_p.sample(3)
```

```
Shape User-Movie-Matrix:        (20828, 491)
```

Movie	8	18	28	30	58	77	83	97	108	111	...	4392	4393	4402	4418	4420	4432	4472	4479	4488	4490
User																					
974823	3.0	3.0	NaN	4.0	NaN	NaN	3.0	3.0	NaN	NaN	...	NaN	NaN	4.0	NaN	NaN	2.0	NaN	4.0	3.0	NaN
164532	NaN	NaN	4.0	5.0	4.0	NaN	NaN	NaN	NaN	NaN	...	4.0	4.0	4.0	NaN	4.0	4.0	4.0	NaN	NaN	NaN
1476458	NaN	NaN	3.0	NaN	1.0	1.0	NaN	NaN	NaN	NaN	...	4.0	3.0	3.0	NaN	NaN	NaN	NaN	NaN	4.0	NaN

3 rows × 491 columns

Recommendation Engines

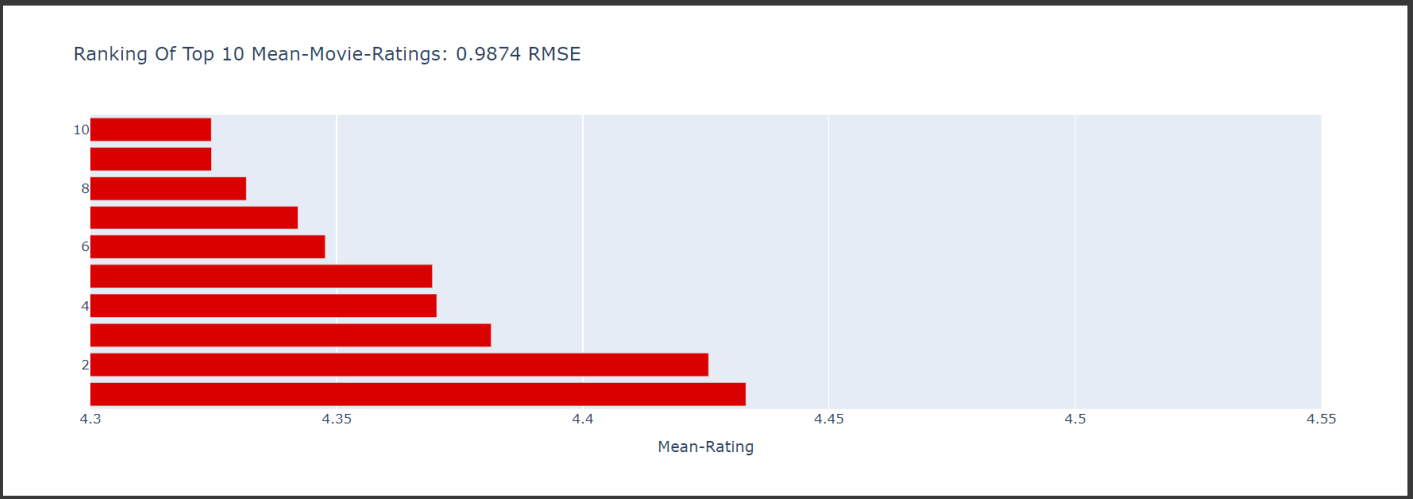
Mean Rating

[20] 1s

```
# Top n movies
n = 10

# Compute mean rating for all movies
ratings_mean = df_p.mean(axis=0).sort_values(ascending=False).rename('Rating-Mean').to_frame()

# Count ratings for all movies
ratings_count = df_p.count(axis=0).rename('Rating-Count').to_frame()

# Combine ratings_mean, ratings_count and movie_titles
ranking_mean_rating = ratings_mean.head(n).join(ratings_count).join(movie_titles.drop('Year', axis=1))


# Join labels and predictions
df_prediction = df_test.set_index('Movie').join(ratings_mean)[['Rating', 'Rating-Mean']]
y_true = df_prediction['Rating']
y_pred = df_prediction['Rating-Mean']

# Compute RMSE
rmse = np.sqrt(mean_squared_error(y_true=y_true, y_pred=y_pred))


# Create trace
trace = go.Bar(x = ranking_mean_rating['Rating-Mean'],
               text = ranking_mean_rating['Name'].astype(str) +': '+ ranking_mean_rating['Rating-Count'].astype(str) + ' Ratings',
               textposition = 'outside',
               textfont = dict(color = '#000000'),
               orientation = 'h',
               y = list(range(1, n+1)),
```

```
                marker = dict(color = '#db0000'))
    # Create layout
    layout = dict(title = 'Ranking Of Top {} Mean-Movie-Ratings: {:.4f} RMSE'.format(n, rmse),
                  xaxis = dict(title = 'Mean-Rating',
                               range = (4.3, 4.55)),
                  yaxis = dict(title = 'Movie'))
    # Create plot
    fig = go.Figure(data=[trace], layout=layout)

    # Download the plot
    fig.write_html("mean_rating.html")
```

```
[21] # it seems that plotly not directly show the plot data, so we need to make it into html file, and show it by calling using other functions

    from IPython.display import HTML

    # Read the HTML file
    with open("mean_rating.html", "r") as f:
        html_content = f.read()

    # Display the HTML content
    HTML(html_content)
```



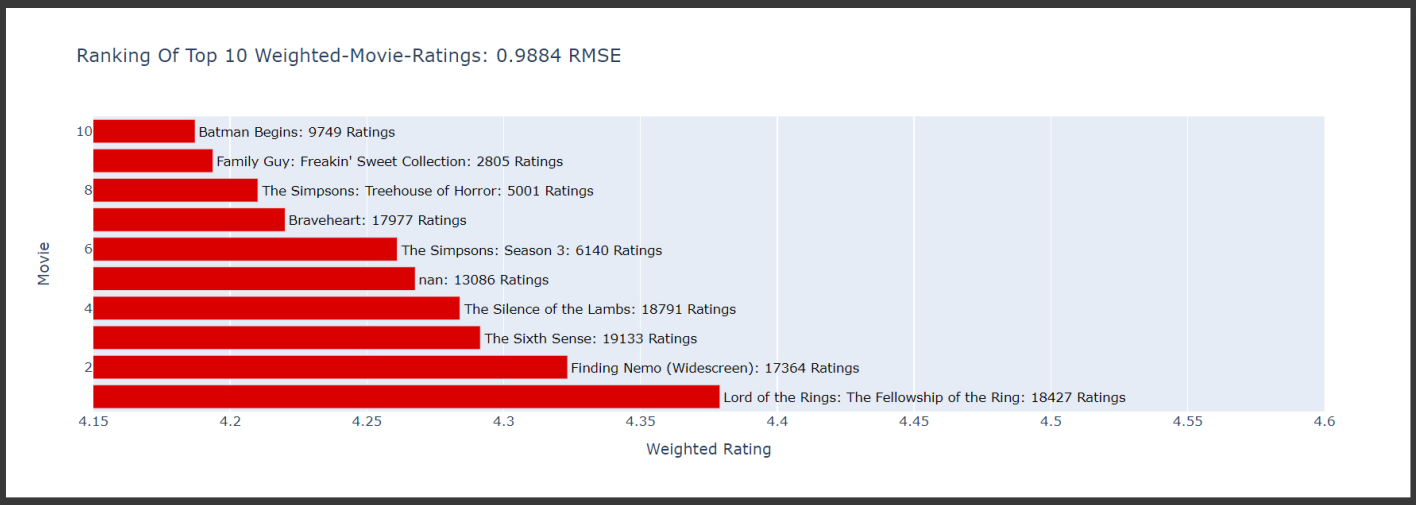
Weighted Mean Rating

```
[22] # Number of minimum votes to be considered
    m = 1000

    # Mean rating for all movies
    C = df_p.stack().mean()

    # Mean rating for all movies separatly
    R = df_p.mean(axis=0).values

    # Rating count for all movies separatly
    v = df_p.count().values


    # Weighted formula to compute the weighted rating
    weighted_score = (v/ (v+m) *R) + (m/ (v+m) *C)
    # Sort ids to ranking
    weighted_ranking = np.argsort(weighted_score)[::-1]
    # Sort scores to ranking
    weighted_score = np.sort(weighted_score)[::-1]
    # Get movie ids
    weighted_movie_ids = df_p.columns[weighted_ranking]


    # Join labels and predictions
    df_prediction = df_test.set_index('Movie').join(pd.DataFrame(weighted_score, index=weighted_movie_ids, columns=['Prediction']))[['Rating', 'Prediction']]
    y_true = df_prediction['Rating']
    y_pred = df_prediction['Prediction']

    # Compute RMSE
    rmse = np.sqrt(mean_squared_error(y_true=y_true, y_pred=y_pred))


    # Create DataFrame for plotting
    df_plot = pd.DataFrame(weighted_score[:n], columns=['Rating'])
    df_plot.index = weighted_movie_ids[:10]
    ranking_weighted_rating = df_plot.join(ratings_count).join(movie_titles)
    del df_plot


    # Create trace
    trace = go.Bar(x = ranking_weighted_rating['Rating'],
                   text = ranking_weighted_rating['Name'].astype(str) +': '+ ranking_weighted_rating['Rating-Count'].astype(str) + ' Ratings',
                   textposition = 'outside',
                   textfont = dict(color = '#000000'),
                   orientation = 'h',
```

```
                y = list(range(1, n+1)),
                marker = dict(color = '#db0000'))
    # Create layout
    layout = dict(title = 'Ranking Of Top {} Weighted-Movie-Ratings: {:.4f} RMSE'.format(n, rmse),
                  xaxis = dict(title = 'Weighted Rating',
                               range = (4.15, 4.6)),
                  yaxis = dict(title = 'Movie'))
    # Create plot
    fig = go.Figure(data=[trace], layout=layout)

    # Download the plot
    fig.write_html("weighted_mean_rating.html")
```

0s

```
[23] # it seems that plotly not directly show the plot data, so we need to make it into html file, and show it by calling using other functions

    from IPython.display import HTML

    # Read the HTML file
    with open("weighted_mean_rating.html", "r") as f:
        html_content = f.read()

    # Display the HTML content
    HTML(html_content)
```



Cosine User-User Similarity

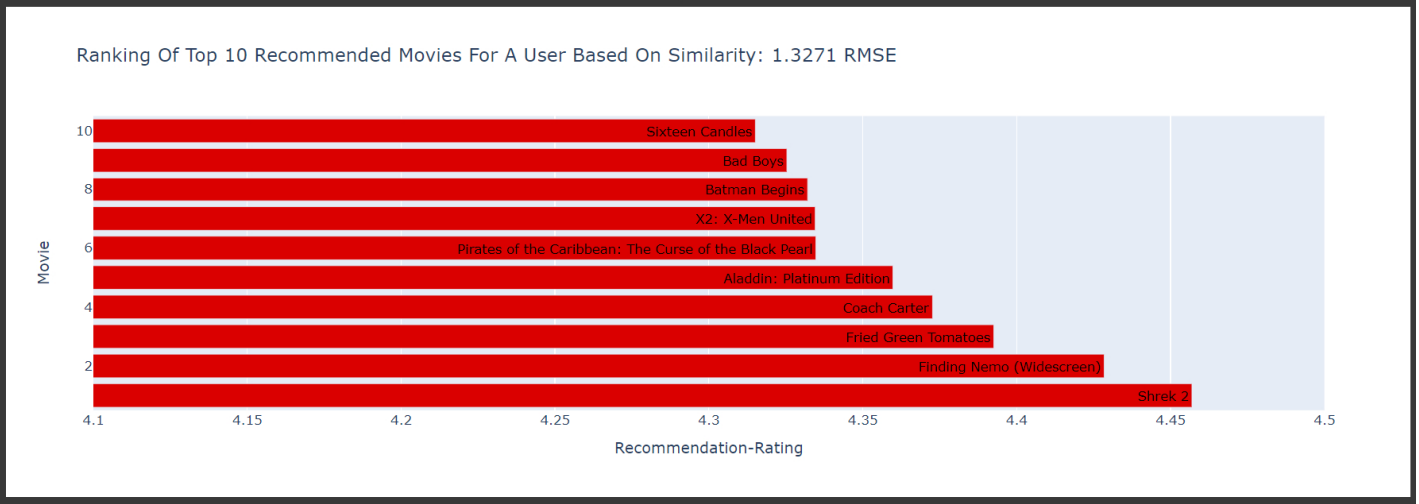
5m

```
[24] # User index for recommendation
    user_index = 0

    # Number of similar users for recommendation
    n_recommendation = 100

    # Plot top n recommendations
    n_plot = 10


    # Fill in missing values
    df_p_imputed = df_p.T.fillna(df_p.mean(axis=1)).T

    # Compute similarity between all users
    similarity = cosine_similarity(df_p_imputed.values)

    # Remove self-similarity from similarity-matrix
    similarity -= np.eye(similarity.shape[0])


    # Sort similar users by index
    similar_user_index = np.argsort(similarity[user_index])[::-1]
    # Sort similar users by score
    similar_user_score = np.sort(similarity[user_index])[::-1]


    # Get unrated movies
    unrated_movies = df_p.iloc[user_index][df_p.iloc[user_index].isna()].index

    # Weight ratings of the top n most similar users with their rating and compute the mean for each movie
    mean_movie_recommendations = (df_p_imputed.iloc[similar_user_index[:n_recommendation]].T * similar_user_score[:n_recommendation]).T.mean(axis=0)

    # Filter for unrated movies and sort results
    best_movie_recommendations = mean_movie_recommendations[unrated_movies].sort_values(ascending=False).to_frame().join(movie_titles)


    # Create user-id mapping
    user_id_mapping = {id:i for i, id in enumerate(df_p_imputed.index)}

    prediction = []
    # Iterate over all testset items
    for user_id in df_test['User'].unique():

        # Sort similar users by index
```

```
        similar_user_index = np.argsort(similarity[user_id_mapping[user_id]])[::-1]
        # Sort similar users by score
        similar_user_score = np.sort(similarity[user_id_mapping[user_id]])[::-1]

        for movie_id in df_test[df_test['User']==user_id]['Movie'].values:

            # Compute predicted score
            score = (df_p_imputed.iloc[similar_user_index[:n_recommendation]][movie_id] * similar_user_score[:n_recommendation]).values.sum() / similar_user_score[:n_reco
            prediction.append([user_id, movie_id, score])


    # Create prediction DataFrame
    df_pred = pd.DataFrame(prediction, columns=['User', 'Movie', 'Prediction']).set_index(['User', 'Movie'])
    df_pred = df_test.set_index(['User', 'Movie']).join(df_pred)


    # Get labels and predictions
    y_true = df_pred['Rating'].values
    y_pred = df_pred['Prediction'].values

    # Compute RMSE
    rmse = np.sqrt(mean_squared_error(y_true=y_true, y_pred=y_pred))


    # Create trace
    trace = go.Bar(x = best_movie_recommendations.iloc[:n_plot, 0],
                   text = best_movie_recommendations['Name'],
                   textposition = 'inside',
                   textfont = dict(color = '#000000'),
                   orientation = 'h',
                   y = list(range(1, n_plot+1)),
                   marker = dict(color = '#db0000'))
    # Create layout
    layout = dict(title = 'Ranking Of Top {} Recommended Movies For A User Based On Similarity: {:.4f} RMSE'.format(n_plot, rmse),
                  xaxis = dict(title = 'Recommendation-Rating',
                               range = (4.1, 4.5)),
                  yaxis = dict(title = 'Movie'))
    # Create plot
    fig = go.Figure(data=[trace], layout=layout)

    # Download the plot
    fig.write_html("cosine_user_similarity.html")
```

[25] 2s

```
# it seems that plotly not directly show the plot data, so we need to make it into html file, and show it by calling using other functions

from IPython.display import HTML

# Read the HTML file
with open("cosine_user_similarity.html", "r") as f:
    html_content = f.read()

# Display the HTML content
HTML(html_content)
```



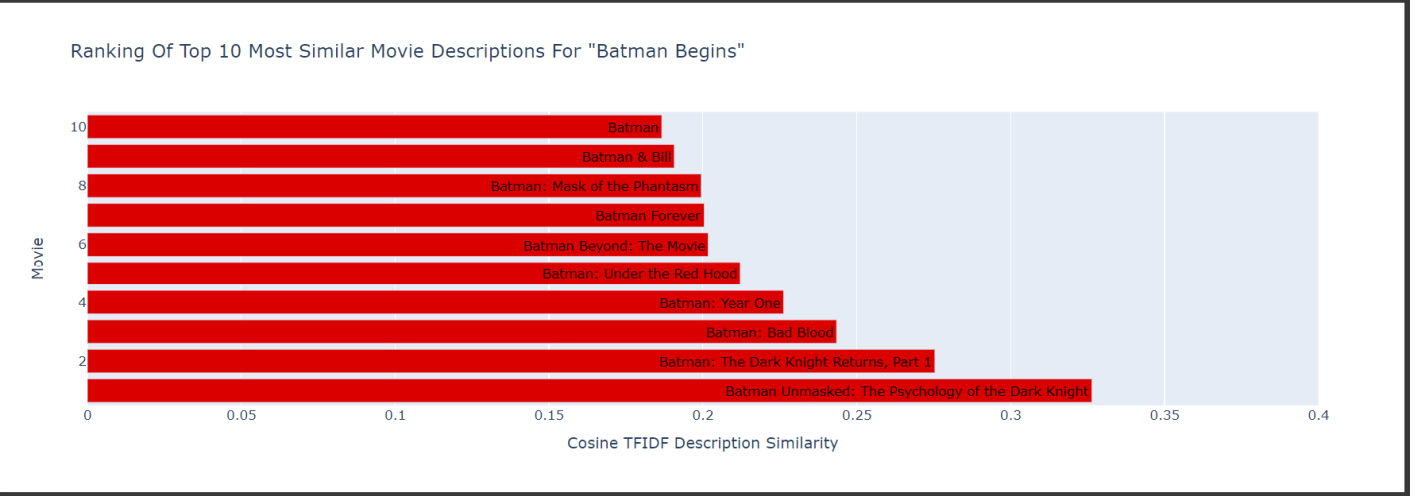
Cosine TFIDF

[26] 6s

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Create tf-idf matrix for text comparison
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(movie_metadata['overview'].dropna())


# Compute cosine similarity between all movie-descriptions
similarity = cosine_similarity(tfidf_matrix)
# Remove self-similarity from matrix
similarity -= np.eye(similarity.shape[0])
```

```
# Get index of movie to find similar movies
movie = 'Batman Begins'
n_plot = 10
index = movie_metadata.reset_index(drop=True)[movie_metadata.index==movie].index[0]

# Get indices and scores of similar movies
similar_movies_index = np.argsort(similarity[index])[::-1][:n_plot]
similar_movies_score = np.sort(similarity[index])[::-1][:n_plot]

# Get titles of similar movies
similar_movie_titles = movie_metadata.iloc[similar_movies_index].index


# Create trace
trace = go.Bar(x = similar_movies_score,
               text = similar_movie_titles,
               textposition = 'inside',
               textfont = dict(color = '#000000'),
               orientation = 'h',
               y = list(range(1, n_plot+1)),
               marker = dict(color = '#db0000'))
# Create layout
layout = dict(title = 'Ranking Of Top {} Most Similar Movie Descriptions For "{}"'.format(n_plot, movie),
              xaxis = dict(title = 'Cosine TFIDF Description Similarity',
                           range = (0, 0.4)),
              yaxis = dict(title = 'Movie'))
# Create plot
fig = go.Figure(data=[trace], layout=layout)

# Download the plot
fig.write_html("cosine_tfidf.html")
```

[27] 4s
```
# it seems that plotly not directly show the plot data, so we need to make it into html file, and show it by calling using other functions

from IPython.display import HTML

# Read the HTML file
with open("cosine_tfidf.html", "r") as f:
    html_content = f.read()

# Display the HTML content
HTML(html_content)
```



[28] 57s
```
# Create user- & movie-id mapping
user_id_mapping = {id:i for i, id in enumerate(df_filterd['User'].unique())}
movie_id_mapping = {id:i for i, id in enumerate(df_filterd['Movie'].unique())}


# Create correctly mapped train- & testset
train_user_data = df_train['User'].map(user_id_mapping)
train_movie_data = df_train['Movie'].map(movie_id_mapping)

test_user_data = df_test['User'].map(user_id_mapping)
test_movie_data = df_test['Movie'].map(movie_id_mapping)


# Get input variable-sizes
users = len(user_id_mapping)
movies = len(movie_id_mapping)
embedding_size = 10


##### Create model
# Set input layers
user_id_input = Input(shape=[1], name='user')
movie_id_input = Input(shape=[1], name='movie')

# Create embedding layers for users and movies
user_embedding = Embedding(output_dim=embedding_size,
                           input_dim=users,
                           input_length=1,
```

```
                            name='user_embedding')(user_id_input)
movie_embedding = Embedding(output_dim=embedding_size,
                            input_dim=movies,
                            input_length=1,
                            name='item_embedding')(movie_id_input)

# Reshape the embedding layers
user_vector = Reshape([embedding_size])(user_embedding)
movie_vector = Reshape([embedding_size])(movie_embedding)

# Compute dot-product of reshaped embedding layers as prediction
y = Dot(1, normalize=False)([user_vector, movie_vector])

# Setup model
model = Model(inputs=[user_id_input, movie_id_input], outputs=y)
model.compile(loss='mse', optimizer='adam')


# Fit model
model.fit([train_user_data, train_movie_data],
          df_train['Rating'],
          batch_size=256,
          epochs=1,
          validation_split=0.1,
          shuffle=True)

# Test model
y_pred = model.predict([test_user_data, test_movie_data])
y_true = df_test['Rating'].values

#  Compute RMSE
rmse = np.sqrt(mean_squared_error(y_pred=y_pred, y_true=y_true))
print('\n\nTesting Result With Keras Matrix-Factorization: {:.4f} RMSE'.format(rmse))
```

```
14337/14337 [==============================] - 43s 3ms/step - loss: 2.1631 - val_loss: 0.8570
3125/3125 [==============================] - 4s 1ms/step


Testing Result With Keras Matrix-Factorization: 0.9243 RMSE
```

1m

```
# deep learning - keras

# Setup variables
user_embedding_size = 20
movie_embedding_size = 10


##### Create model
# Set input layers
user_id_input = Input(shape=[1], name='user')
movie_id_input = Input(shape=[1], name='movie')

# Create embedding layers for users and movies
user_embedding = Embedding(output_dim=user_embedding_size,
                           input_dim=users,
                           input_length=1,
                           name='user_embedding')(user_id_input)
movie_embedding = Embedding(output_dim=movie_embedding_size,
                            input_dim=movies,
                            input_length=1,
                            name='item_embedding')(movie_id_input)

# Reshape the embedding layers
user_vector = Reshape([user_embedding_size])(user_embedding)
movie_vector = Reshape([movie_embedding_size])(movie_embedding)

# Concatenate the reshaped embedding layers
concat = Concatenate()([user_vector, movie_vector])

# Combine with dense layers
dense = Dense(256)(concat)
y = Dense(1)(dense)

# Setup model
model = Model(inputs=[user_id_input, movie_id_input], outputs=y)
model.compile(loss='mse', optimizer='adam')


# Fit model
model.fit([train_user_data, train_movie_data],
          df_train['Rating'],
          batch_size=256,
          epochs=1,
          validation_split=0.1,
          shuffle=True)

# Test model
y_pred = model.predict([test_user_data, test_movie_data])
y_true = df_test['Rating'].values

#  Compute RMSE
rmse = np.sqrt(mean_squared_error(y_pred=y_pred, y_true=y_true))
print('\n\nTesting Result With Keras Deep Learning: {:.4f} RMSE'.format(rmse))
```

```
14337/14337 [==============================] - 57s 4ms/step - loss: 0.8677 - val_loss: 0.8225
3125/3125 [==============================] - 4s 1ms/step
```

Testing Result With Keras Deep Learning: 0.9048 RMSE

```
# Deep Hybrid System With Metadata And Keras
# Create user- & movie-id mapping
user_id_mapping = {id:i for i, id in enumerate(df['User'].unique())}
movie_id_mapping = {id:i for i, id in enumerate(df['Movie'].unique())}

# Use mapping to get better ids
df['User'] = df['User'].map(user_id_mapping)
df['Movie'] = df['Movie'].map(movie_id_mapping)


##### Combine both datasets to get movies with metadata
# Preprocess metadata
tmp_metadata = movie_metadata.copy()
tmp_metadata.index = tmp_metadata.index.str.lower()

# Preprocess titles
tmp_titles = movie_titles.drop('Year', axis=1).copy()
tmp_titles = tmp_titles.reset_index().set_index('Name')
tmp_titles.index = tmp_titles.index.str.lower()

# Combine titles and metadata
df_id_descriptions = tmp_titles.join(tmp_metadata).dropna().set_index('Id')
df_id_descriptions['overview'] = df_id_descriptions['overview'].str.lower()
del tmp_metadata,tmp_titles


# Filter all ratings with metadata
df_hybrid = df.drop('Date', axis=1).set_index('Movie').join(df_id_descriptions).dropna().drop('overview', axis=1).reset_index().rename({'index':'Movie'}, axis=1)


# Split train- & testset
n = 100000
df_hybrid = df_hybrid.sample(frac=1).reset_index(drop=True)
df_hybrid_train = df_hybrid[:1500000]
df_hybrid_test = df_hybrid[-n:]


# Create tf-idf matrix for text comparison
tfidf = TfidfVectorizer(stop_words='english')
tfidf_hybrid = tfidf.fit_transform(df_id_descriptions['overview'])


# Get mapping from movie-ids to indices in tfidf-matrix
mapping = {id:i for i, id in enumerate(df_id_descriptions.index)}

train_tfidf = []
# Iterate over all movie-ids and save the tfidf-vector
for id in df_hybrid_train['Movie'].values:
    index = mapping[id]
    train_tfidf.append(tfidf_hybrid[index])

test_tfidf = []
# Iterate over all movie-ids and save the tfidf-vector
for id in df_hybrid_test['Movie'].values:
    index = mapping[id]
    test_tfidf.append(tfidf_hybrid[index])


# Stack the sparse matrices
train_tfidf = vstack(train_tfidf)
test_tfidf = vstack(test_tfidf)


##### Setup the network
# Network variables
user_embed = 10
movie_embed = 10


# Create two input layers
user_id_input = Input(shape=[1], name='user')
movie_id_input = Input(shape=[1], name='movie')
tfidf_input = Input(shape=[24144], name='tfidf', sparse=True)

# Create separate embeddings for users and movies
user_embedding = Embedding(output_dim=user_embed,
                           input_dim=len(user_id_mapping),
                           input_length=1,
                           name='user_embedding')(user_id_input)
movie_embedding = Embedding(output_dim=movie_embed,
                            input_dim=len(movie_id_mapping),
                            input_length=1,
                            name='movie_embedding')(movie_id_input)

# Dimensionality reduction with Dense layers
tfidf_vectors = Dense(128, activation='relu')(tfidf_input)
tfidf_vectors = Dense(32, activation='relu')(tfidf_vectors)

# Reshape both embedding layers
user_vectors = Reshape([user_embed])(user_embedding)
movie_vectors = Reshape([movie_embed])(movie_embedding)

# Concatenate all layers into one vector
```

```
both = Concatenate()([user_vectors, movie_vectors, tfidf_vectors])

# Add dense layers for combinations and scalar output
dense = Dense(512, activation='relu')(both)
dense = Dropout(0.2)(dense)
output = Dense(1)(dense)


# Create and compile model
model = Model(inputs=[user_id_input, movie_id_input, tfidf_input], outputs=output)
model.compile(loss='mse', optimizer='adam')


# Train and test the network
model.fit([df_hybrid_train['User'], df_hybrid_train['Movie'], train_tfidf],
          df_hybrid_train['Rating'],
          batch_size=1024,
          epochs=2,
          validation_split=0.1,
          shuffle=True)

y_pred = model.predict([df_hybrid_test['User'], df_hybrid_test['Movie'], test_tfidf])
y_true = df_hybrid_test['Rating'].values

rmse = np.sqrt(mean_squared_error(y_pred=y_pred, y_true=y_true))
print('\n\nTesting Result With Keras Hybrid Deep Learning: {:.4f} RMSE'.format(rmse))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-30-b4f85fcda93c> in <cell line: 108>()
    106
    107 # Train and test the network
--> 108 model.fit([df_hybrid_train['User'], df_hybrid_train['Movie'], train_tfidf],
    109           df_hybrid_train['Rating'],
    110           batch_size=1024,

                                 1 frames
/usr/local/lib/python3.10/dist-packages/keras/src/engine/data_adapter.py in train_validation_split(arrays, validation_split)
   1774     unsplitable = [type(t) for t in flat_arrays if not _can_split(t)]
   1775     if unsplitable:
-> 1776         raise ValueError(
   1777             "`validation_split` is only supported for Tensors or NumPy "
   1778             "arrays, found following types in the input: {}".format(unsplitable)

ValueError: `validation_split` is only supported for Tensors or NumPy arrays, found following types in the input: [<class 'scipy.sparse._csr.csr_matrix'>]
```

[33]
```
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import vstack
import pandas as pd

# Combine movie overviews from both training and validation sets
combined_movies = pd.concat([train_movie, val_movie])
combined_overviews = df_id_descriptions.loc[combined_movies]['overview']

# Create and fit TF-IDF vectorizer on combined overviews
tfidf = TfidfVectorizer(stop_words='english')
tfidf_hybrid = tfidf.fit_transform(combined_overviews)

# Get mapping from movie-ids to indices in tfidf-matrix
mapping = {id:i for i, id in enumerate(df_id_descriptions.index)}

# Generate train and validation TF-IDF matrices
train_tfidf = []
for id in train_movie.values:
    index = mapping[id]
    train_tfidf.append(tfidf_hybrid[index])
train_tfidf = vstack(train_tfidf)

val_tfidf = []
for id in val_movie.values:
    index = mapping[id]
    val_tfidf.append(tfidf_hybrid[index])
val_tfidf = vstack(val_tfidf)

# Train the model
model.fit([train_user, train_movie, train_tfidf],
          train_rating,
          batch_size=1024,
          epochs=2,
          validation_data=([val_user, val_movie, val_tfidf], val_rating),
          shuffle=True)

# Evaluate on test data
y_pred = model.predict([df_hybrid_test['User'], df_hybrid_test['Movie'], test_tfidf])
y_true = df_hybrid_test['Rating'].values

rmse = np.sqrt(mean_squared_error(y_pred=y_pred, y_true=y_true))
print('\n\nTesting Result With Keras Hybrid Deep Learning: {:.4f} RMSE'.format(rmse))
```

```
WARNING:tensorflow:Keras is training/fitting/evaluating on array-like data. Keras may not be optimized for this format, so if your input data format is supported by Te
Epoch 1/2
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-33-5560e8cfccc3> in <cell line: 30>()
     28
     29 # Train the model
```

```
---> 30 model.fit([train_user, train_movie, train_tfidf],
     31           train_rating,
     32           batch_size=1024,

                                    1 frames
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py in tf__train_function(iterator)
     13                 try:
     14                     do_return = True
---> 15                     retval_ = ag__.converted_call(ag__.ld(step_function), (ag__.ld(self), ag__.ld(iterator)), None, fscope)
     16                 except:
     17                     do_return = False

ValueError: in user code:

    File "/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py", line 1401, in train_function  *
        return step_function(self, iterator)
    File "/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py", line 1384, in step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py", line 1373, in run_step  **
        outputs = model.train_step(data)
    File "/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py", line 1150, in train_step
        y_pred = self(x, training=True)
    File "/usr/local/lib/python3.10/dist-packages/keras/src/utils/traceback_utils.py", line 70, in error_handler
        raise e.with_traceback(filtered_tb) from None
    File "/usr/local/lib/python3.10/dist-packages/keras/src/engine/input_spec.py", line 298, in assert_input_compatibility
        raise ValueError(

    ValueError: Input 2 of layer "model_2" is incompatible with the layer: expected shape=(None, 24144), found shape=(None, 12345)
```

Crossvalidated Comparison Of Surprise Algorithms

39s

```
# Load dataset into surprise specific data-structure
data = sp.Dataset.load_from_df(df_filterd[['User', 'Movie', 'Rating']].sample(20000), sp.Reader())

benchmark = []
# Iterate over all algorithms
for algorithm in [sp.SVD(), sp.SVDpp(), sp.SlopeOne(), sp.NMF(), sp.NormalPredictor(), sp.KNNBaseline(), sp.KNNBasic(), sp.KNNWithMeans(), sp.KNNWithZScore(), sp.Base
    # Perform cross validation
    results = cross_validate(algorithm, data, measures=['RMSE', 'MAE'], cv=3, verbose=False)

    # Get results & append algorithm name
    tmp = pd.DataFrame.from_dict(results).mean(axis=0)
    tmp['Algorithm'] = str(algorithm).split(' ')[0].split('.')[-1]

    # Store data
    benchmark.append(tmp)

# Convert the list of dictionaries to a DataFrame
benchmark_df = pd.DataFrame(benchmark)
```

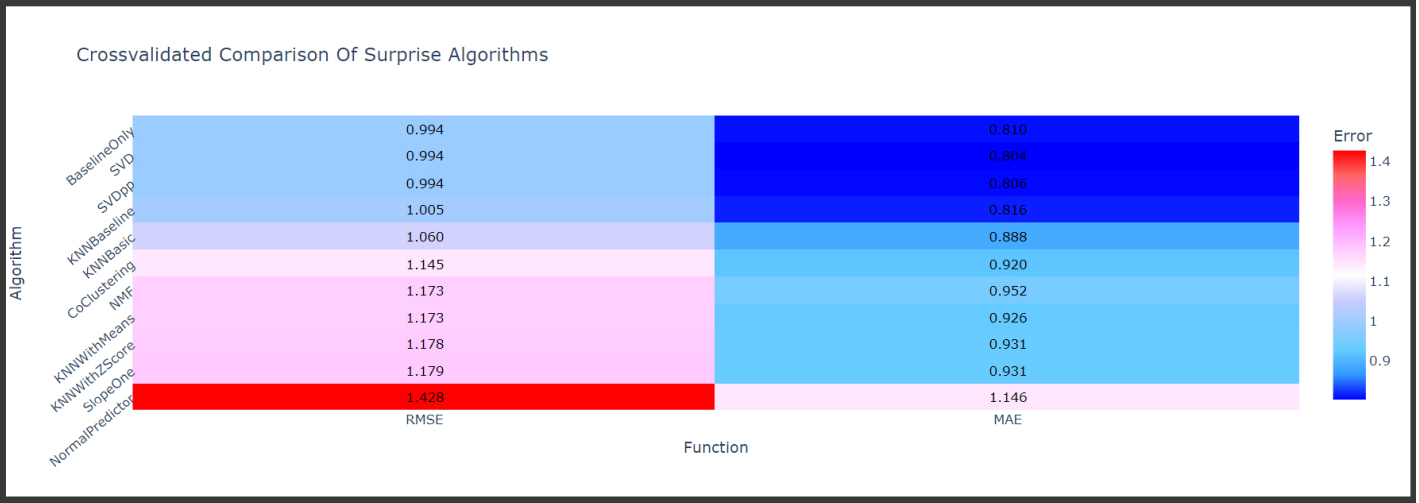
```
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
```

[36] 0s

```
# Store results
surprise_results = pd.DataFrame(benchmark).set_index('Algorithm').sort_values('test_rmse', ascending=False)

# Get data
data = surprise_results[['test_rmse', 'test_mae']]
grid = data.values

# Create axis labels
x_axis = [label.split('_')[1].upper() for label in data.columns.tolist()]
y_axis = data.index.tolist()

x_label = 'Function'
y_label = 'Algorithm'
```

```
# Get annotations and hovertext
hovertexts = []
annotations = []
for i, y_value in enumerate(y_axis):
    row = []
    for j, x_value in enumerate(x_axis):
        annotation = grid[i, j]
        row.append('Error: {:.3f}<br>{}: {}<br>{}: {}<br>Fit Time: {:.3f}s<br>Test Time: {:.3f}s'.format(annotation, y_label, y_value ,x_label, x_value, surprise_resu
        annotations.append(dict(x=x_value, y=y_value, text='{:.3f}'.format(annotation), ax=0, ay=0, font=dict(color='#000000')))
    hovertexts.append(row)

# Create trace
trace = go.Heatmap(x = x_axis,
                   y = y_axis,
                   z = data.values,
                   text = hovertexts,
                   hoverinfo = 'text',
                   colorscale = 'Picnic',
                   colorbar = dict(title = 'Error'))

# Create layout
layout = go.Layout(title = 'Crossvalidated Comparison Of Surprise Algorithms',
                   xaxis = dict(title = x_label),
                   yaxis = dict(title = y_label,
                                tickangle = -40),
                   annotations = annotations)

# Create plot
fig = go.Figure(data=[trace], layout=layout)

# Download plot
fig.write_html("benchmark.html")
```

```
[37] # it seems that plotly not directly show the plot data, so we need to make it into html file, and show it by calling using other functions

from IPython.display import HTML

# Read the HTML file
with open("benchmark.html", "r") as f:
    html_content = f.read()

# Display the HTML content
HTML(html_content)
```



lightfm library

```
# Create user- & movie-id mapping
user_id_mapping = {id:i for i, id in enumerate(df_filterd['User'].unique())}
movie_id_mapping = {id:i for i, id in enumerate(df_filterd['Movie'].unique())}


# Create correctly mapped train- & testset
train_user_data = df_train['User'].map(user_id_mapping)
train_movie_data = df_train['Movie'].map(movie_id_mapping)

test_user_data = df_test['User'].map(user_id_mapping)
test_movie_data = df_test['Movie'].map(movie_id_mapping)


# Create sparse matrix from ratings
shape = (len(user_id_mapping), len(movie_id_mapping))
train_matrix = coo_matrix((df_train['Rating'].values, (train_user_data.astype(int), train_movie_data.astype(int))), shape=shape)
test_matrix = coo_matrix((df_test['Rating'].values, (test_user_data.astype(int), test_movie_data.astype(int))), shape=shape)


# Instantiate and train the model
model = LightFM(loss='warp', no_components=20)
model.fit(train_matrix, epochs=20, num_threads=2)
```

```
# Evaluate the trained model
k = 20
print('Train precision at k={}:\t{:.4f}'.format(k, precision_at_k(model, train_matrix, k=k).mean()))
print('Test precision at k={}:\t\t{:.4f}'.format(k, precision_at_k(model, test_matrix, k=k).mean()))
```

```
Train precision at k=20:        0.9568
Test precision at k=20:         0.0154
```

the code is inspired from kaggle : https://www.kaggle.com/code/morrisb/how-to-recommend-anything-deep-recommender

```
[ ]
```