SHERPA_V2.0

March 18, 2024

[1]:
```
import pandas as pd
import numpy as np
from scipy.sparse import csc_matrix
from numpy.linalg import lstsq
from sklearn.metrics import mean_squared_error
from math import sqrt
```

0.1 Dataset Importing Function

First, let's include the function for importing the dataset, which parses the data from a given file and formats it into a pandas DataFrame.

[2]:
```
movies_df = pd.read_csv("movie_titles.csv", on_bad_lines='skip')
```

[3]:
```
# Parsing the data
def parse_data(file_path):
    data = []
    current_movie_id = None
    with open(file_path, 'r', encoding='utf-8-sig') as file:
        for line in file:
            try:
                if ':' in line:
                    current_movie_id = int(line.split(':')[0])
                else:
                    customer_id, rating, _ = line.strip().split(',')
                    data.append([current_movie_id, int(customer_id),␣
 ↪int(rating)])
            except ValueError:
                # This handles lines that don't have the expected format
                continue  # Skips to the next line
    return pd.DataFrame(data, columns=['MovieID', 'CustomerID', 'Rating'])

# Function to load data for training, test, and validation
def load_dataset(file_paths):
    data_frames = [parse_data(file_path) for file_path in file_paths]␣
 ↪#parse_data function to load and structure the data into a DataFrame
    combined_data = pd.concat(data_frames)
```

```
    return combined_data

# File paths setup
train_files = ['training_set_c1.txt','training_set_c2.txt']
test_file = ['test_set_c1.txt', 'test_set_c2.txt']
validation_file = ['validation_set_c1.txt','validation_set_c2.txt']

# Load datasets
train_movie_data = load_dataset(train_files)

train_movie_data.head()
```

[3]:

	MovieID	CustomerID	Rating
0	1	401047	4
1	1	14756	4
2	1	2566259	5
3	1	1398626	2
4	1	1294335	2

0.2 Preparing the Ratings Matrix

After importing the dataset, convert it to a sparse matrix format that the ALS algorithm can process.

[4]:
```
from scipy.sparse import csr_matrix

# Create a user-item matrix
train_ratings_df = train_movie_data.pivot(index='CustomerID',␣
↪columns='MovieID', values='Rating').fillna(0)

# Convert to CSR format
train_ratings_matrix = csr_matrix(train_ratings_df.values)
```

```
/tmp/ipykernel_12920/1423468332.py:4: PerformanceWarning: The following
operation may generate 4384890210 cells in the resulting pandas object.
  train_ratings_df = train_movie_data.pivot(index='CustomerID',
columns='MovieID', values='Rating').fillna(0)
```

0.3 Optimized ALS with Sparse Matrices

[5]:
```
import numpy as np
from scipy.sparse import csr_matrix
from numpy.linalg import lstsq

def update_U(M, U, lambda_reg, ratings):
    """
    Update user features matrix U.
```

```python
    :param M: Movie features matrix.
    :param U: User features matrix to update.
    :param lambda_reg: Regularization parameter.
    :param ratings: Ratings CSR matrix.
    :return: Updated user features matrix.
    """
    num_factors = U.shape[1]  # Number of latent factors
    lambda_I = lambda_reg * np.eye(num_factors)
    updated_U = np.zeros(U.shape)

    for i in range(U.shape[0]):
        user_rated_indices = ratings[i].nonzero()[1]
        M_sub = M[user_rated_indices, :]
        ratings_sub = ratings[i, user_rated_indices].toarray().flatten()

        # Constructing the system to solve
        A = M_sub.T @ M_sub + lambda_I * len(user_rated_indices)
        b = M_sub.T @ ratings_sub
        updated_U[i, :] = lstsq(A, b, rcond=None)[0]

    return updated_U

def update_M(M, U, lambda_reg, ratings):
    """
    Update movie features matrix M.

    :param M: Movie features matrix to update.
    :param U: User features matrix.
    :param lambda_reg: Regularization parameter.
    :param ratings: Ratings CSR matrix.
    :return: Updated movie features matrix.
    """
    num_factors = M.shape[1]  # Number of latent factors
    lambda_I = lambda_reg * np.eye(num_factors)
    updated_M = np.zeros(M.shape)

    for i in range(M.shape[0]):
        movie_rated_by_indices = ratings[:, i].nonzero()[0]
        U_sub = U[movie_rated_by_indices, :]
        ratings_sub = ratings[movie_rated_by_indices, i].toarray().flatten()

        # Constructing the system to solve
        A = U_sub.T @ U_sub + lambda_I * len(movie_rated_by_indices)
        b = U_sub.T @ ratings_sub
        updated_M[i, :] = lstsq(A, b, rcond=None)[0]
```

```
    return updated_M

def ALS(ratings, num_factors=50, lambda_reg=0.1, iterations=10):
    """
    Alternating Least Squares algorithm for matrix factorization.

    :param ratings: Original ratings matrix as CSR.
    :param num_factors: Number of latent factors.
    :param lambda_reg: Regularization parameter.
    :param iterations: Number of iterations to run ALS.
    :return: User and Movie latent features matrices.
    """
    num_users, num_movies = ratings.shape

    # Initialize user and movie matrices with small random values
    U = np.random.rand(num_users, num_factors) * 0.01
    M = np.random.rand(num_movies, num_factors) * 0.01

    for iteration in range(iterations):
        U = update_U(M, U, lambda_reg, ratings)
        M = update_M(M, U, lambda_reg, ratings)

    return U, M
```

0.4 Running the ALS Algorithm

Now, you can use the previously defined ALS algorithm, ensuring it's ready to process the CSR matrix created from your dataset.

[6]:
```
# Assuming the ALS function and update functions are defined as in the previous␣
↪example

num_factors = 11  # Adjust based on your preference
lambda_reg = 0.1  # Regularization parameter
iterations = 5  # Number of iterations for ALS

# Running ALS on the ratings matrix
U, M = ALS(train_ratings_matrix, num_factors=num_factors,␣
↪lambda_reg=lambda_reg, iterations=iterations)

# U and M are the user and item (movie) latent factors matrices
```

[21]:
```
# Generate predictions
train_predictions = U.dot(M.T)

train_preds_df = pd.DataFrame(train_predictions, columns=train_ratings_df.
↪columns, index=train_ratings_df.index)
```

[22]:
```
#Content-Based Filtering
```

[23]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import sigmoid_kernel
from sklearn.metrics import ndcg_score

movies_df = movies_df.fillna('')
# Function to create weighted text
def create_weighted_text(row):
    return (row['Overview'] + ' ') * 45 + (row['Genre'] + ' ') * 25 + \
           (row['Director'] + ' ') * 15 + (row['Cast'] + ' ') * 15
movies_df['weighted_text'] = movies_df.apply(create_weighted_text, axis=1)

# Initialize TF-IDF Vectorizer
# from sklearn.feature_extraction.text import TfidfVectorizer
tfv = TfidfVectorizer(min_df=3, max_features=None, strip_accents='unicode',
                      analyzer='word', token_pattern=r'\w{1,}',
                      ngram_range=(1,3), stop_words='english')
# Fit the TF-IDF on the 'weighted_text'
tfv_matrix = tfv.fit_transform(movies_df['weighted_text'])
sig = sigmoid_kernel(tfv_matrix, tfv_matrix)
# Reverse mapping of indices and movie titles
indices = pd.Series(movies_df.index, index=movies_df['Movie_Name']).
 ↪drop_duplicates()
```

[24]:
```
#Hybrid Recommendation System
```

[41]:
```
def hybrid_recommendations(user_id=None, movie_name=None, preds_df=None,␣
 ↪movies_df=movies_df, sig=sig, indices=indices, top_n=10):
    if preds_df is None:
        raise ValueError("The predictions dataframe (preds_df) is required.")

    # Initialize recommendation lists
    final_recs = []

    # Fetch Content-Based Recommendations
    content_based_recs = []
    if movie_name in indices:
        idx = indices[movie_name]
        sig_scores = list(enumerate(sig[idx]))
        sig_scores = sorted(sig_scores, key=lambda x: x[1], reverse=True)
        movie_indices = [i[0] for i in sig_scores[1:top_n+1]]
        content_based_recs = movies_df.iloc[movie_indices]['Movie_Name'].
 ↪tolist()

    # For existing users with a search query, combine collaborative and␣
 ↪content-based recommendations
```

```
    if user_id and movie_name:
        # Fetch collaborative filtering recommendations based on historical␣
 ↪ratings
        collaborative_recs_ids = preds_df.loc[user_id].
 ↪sort_values(ascending=False).head(top_n * 2).index.tolist()
        collaborative_recs_names = movies_df[movies_df['MovieID'].
 ↪isin(collaborative_recs_ids)]['Movie_Name'].tolist()

        # Combine lists with simple deduplication, prioritizing content-based␣
 ↪recommendations
        seen = set(content_based_recs)
        combined_recs = content_based_recs + [rec for rec in␣
 ↪collaborative_recs_names if rec not in seen]

        # Limit to top_n recommendations after combining
        final_recs = combined_recs[:top_n]
    elif user_id:
        # Only collaborative recommendations for existing users without search␣
 ↪query
        collaborative_recs_ids = preds_df.loc[user_id].
 ↪sort_values(ascending=False).head(top_n).index.tolist()
        final_recs = movies_df[movies_df['MovieID'].
 ↪isin(collaborative_recs_ids)]['Movie_Name'].tolist()
    else:
        # Only content-based recommendations for new users with a search query
        final_recs = content_based_recs

    return final_recs

# Testing the function with your scenarios
user_id = 401047  # Example user ID
movie_name = "The Company"  # Example movie name

print("Collaborative Recommendations for Existing User (No Search):")
collab_recs = hybrid_recommendations(user_id=user_id, preds_df=train_preds_df,␣
 ↪top_n=10)
for movie in collab_recs:
    print(movie)

print("\nHybrid Recommendations for Existing User (With Search):")
hybrid_recs = hybrid_recommendations(user_id=user_id, movie_name=movie_name,␣
 ↪preds_df=train_preds_df, top_n=10)
for movie in hybrid_recs:
    print(movie)

print("\nContent-Based Recommendations for New User (With Search):")
```

```
content_recs = hybrid_recommendations(movie_name=movie_name,␣
↪preds_df=train_preds_df, top_n=10)
for movie in content_recs:
    print(movie)
```

```
Collaborative Recommendations for Existing User (No Search):
ABC Primetime: Mel Gibson's The Passion of the Christ
Armageddon
Coach Carter
Secondhand Lions
The Winds of War
Braveheart
In the Face of Evil: Reagan's War in Word and Deed
Pretty Woman
24: Season 1
CSI: Season 3

Hybrid Recommendations for Existing User (With Search):
Center Stage
Ballet Favorites
Expo: Magic of the White City
A Raisin in the Sun
Robin and the 7 Hoods
Swan Lake: Tchaikovsky (Matthew Bourne)
Out of Sync
Orchestra Rehearsal
Category 6: Day of Destruction
What Have I Done to Deserve This?

Content-Based Recommendations for New User (With Search):
Center Stage
Ballet Favorites
Expo: Magic of the White City
A Raisin in the Sun
Robin and the 7 Hoods
Swan Lake: Tchaikovsky (Matthew Bourne)
Out of Sync
Orchestra Rehearsal
Category 6: Day of Destruction
What Have I Done to Deserve This?
```

[39]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt

def calculate_rmse(actual, predictions):
    mask = actual.nonzero()  # Only consider non-zero entries
    actual = actual[mask]
```

```
    predictions = predictions[mask]
    return sqrt(mean_squared_error(actual, predictions))

# Convert the predictions matrix to a dense format since ratings_matrix is␣
 ↪sparse
#train_ratings_matrix = train_ratings_matrix.toarray()

# Calculate RMSE
rmse = calculate_rmse(train_ratings_matrix.toarray(), train_predictions)
print('Training RMSE:', rmse)
```

```
Training RMSE: 0.8771686076210009
```

[40]:
```
# Load datasets
test_movie_data = load_dataset(test_file)

# Create a Train user-item matrix
test_ratings_df= test_movie_data.pivot(index='CustomerID', columns='MovieID',␣
 ↪values='Rating').reindex(index=train_ratings_df.index,␣
 ↪columns=train_ratings_df.columns).fillna(0)

# Convert to CSR format
test_ratings_matrix = csr_matrix(test_ratings_df.values)
```

```
/tmp/ipykernel_12920/2213329239.py:5: PerformanceWarning: The following
operation may generate 4087377814 cells in the resulting pandas object.
  test_ratings_df= test_movie_data.pivot(index='CustomerID', columns='MovieID',
values='Rating').reindex(index=train_ratings_df.index,
columns=train_ratings_df.columns).fillna(0)
```

[43]:
```
# Mapping test user and movie indices to training set indices
test_user_indices = [np.where(train_ratings_df.index == uid)[0][0] for uid in␣
 ↪test_ratings_df.index if uid in train_ratings_df.index]
test_movie_indices = [np.where(train_ratings_df.columns == mid)[0][0] for mid␣
 ↪in test_ratings_df.columns if mid in train_ratings_df.columns]

# Generate predictions for test set
test_predictions = U[test_user_indices, :] @ M.T[:, test_movie_indices]

# Generate predictions
test_preds_df = pd.DataFrame(test_predictions, columns=test_ratings_df.columns,␣
 ↪index=test_ratings_df.index)
```

[55]:
```
# Testing the function with your scenarios
user_id = 401047   # Example user ID
movie_name = "The Company"   # Example movie name
```

```python
print("Collaborative Recommendations for Existing User (No Search):")
collab_recs = hybrid_recommendations(user_id=user_id, preds_df=test_preds_df,
    top_n=10)
for movie in collab_recs:
    print(movie)

print("\nHybrid Recommendations for Existing User (With Search):")
hybrid_recs = hybrid_recommendations(user_id=user_id, movie_name=movie_name,
    preds_df=test_preds_df, top_n=10)
for movie in hybrid_recs:
    print(movie)

print("\nContent-Based Recommendations for New User (With Search):")
content_recs = hybrid_recommendations(movie_name=movie_name,
    preds_df=test_preds_df, top_n=10)
for movie in content_recs:
    print(movie)
```

```
Collaborative Recommendations for Existing User (No Search):
ABC Primetime: Mel Gibson's The Passion of the Christ
Armageddon
Coach Carter
Secondhand Lions
The Winds of War
Braveheart
In the Face of Evil: Reagan’s War in Word and Deed
Pretty Woman
24: Season 1
CSI: Season 3

Hybrid Recommendations for Existing User (With Search):
Center Stage
Ballet Favorites
Expo: Magic of the White City
A Raisin in the Sun
Robin and the 7 Hoods
Swan Lake: Tchaikovsky (Matthew Bourne)
Out of Sync
Orchestra Rehearsal
Category 6: Day of Destruction
What Have I Done to Deserve This?

Content-Based Recommendations for New User (With Search):
Center Stage
Ballet Favorites
Expo: Magic of the White City
A Raisin in the Sun
```

```
Robin and the 7 Hoods
Swan Lake: Tchaikovsky (Matthew Bourne)
Out of Sync
Orchestra Rehearsal
Category 6: Day of Destruction
What Have I Done to Deserve This?
```

[48]:

```
# Calculate RMSE for test set
test_rmse = calculate_rmse(test_ratings_matrix.toarray(), test_predictions)
print('Test RMSE:', test_rmse)
```

```
Test RMSE: 0.9181483695344229
```

[49]:

```
# Load datasets
validation_movie_data = load_dataset(validation_file)

# Create a user-item matrix
validation_ratings_df= validation_movie_data.pivot(index='CustomerID',␣
↪columns='MovieID', values='Rating').reindex(index=train_ratings_df.index,␣
↪columns=train_ratings_df.columns).fillna(0)



# Convert to CSR format
validation_ratings_matrix = csr_matrix(validation_ratings_df.values)
```

```
/tmp/ipykernel_12920/1611650772.py:5: PerformanceWarning: The following
operation may generate 4087812450 cells in the resulting pandas object.
  validation_ratings_df= validation_movie_data.pivot(index='CustomerID',
columns='MovieID', values='Rating').reindex(index=train_ratings_df.index,
columns=train_ratings_df.columns).fillna(0)
```

[51]:

```
# Mapping validation user and movie indices to training set indices
validation_user_indices = [np.where(train_ratings_df.index == uid)[0][0] for␣
↪uid in validation_ratings_df.index if uid in train_ratings_df.index]
validation_movie_indices = [np.where(train_ratings_df.columns == mid)[0][0] for␣
↪mid in validation_ratings_df.columns if mid in train_ratings_df.columns]

# Generate predictions for validation set
validation_predictions = U[validation_user_indices, :] @ M.T[:,␣
↪validation_movie_indices]

validation_preds_df = pd.DataFrame(validation_predictions,␣
↪columns=validation_ratings_df.columns, index=validation_ratings_df.index)
```

[56]:

```
# Testing the function with your scenarios
user_id = 401047  # Example user ID
movie_name = "The Company"  # Example movie name
```

```python
print("Collaborative Recommendations for Existing User (No Search):")
collab_recs = hybrid_recommendations(user_id=user_id, preds_df=validation_preds_df, top_n=10)
for movie in collab_recs:
    print(movie)

print("\nHybrid Recommendations for Existing User (With Search):")
hybrid_recs = hybrid_recommendations(user_id=user_id, movie_name=movie_name, preds_df=validation_preds_df, top_n=10)
for movie in hybrid_recs:
    print(movie)

print("\nContent-Based Recommendations for New User (With Search):")
content_recs = hybrid_recommendations(movie_name=movie_name, preds_df=validation_preds_df, top_n=10)
for movie in content_recs:
    print(movie)
```

```
Collaborative Recommendations for Existing User (No Search):
ABC Primetime: Mel Gibson's The Passion of the Christ
Armageddon
Coach Carter
Secondhand Lions
The Winds of War
Braveheart
In the Face of Evil: Reagan’s War in Word and Deed
Pretty Woman
24: Season 1
CSI: Season 3

Hybrid Recommendations for Existing User (With Search):
Center Stage
Ballet Favorites
Expo: Magic of the White City
A Raisin in the Sun
Robin and the 7 Hoods
Swan Lake: Tchaikovsky (Matthew Bourne)
Out of Sync
Orchestra Rehearsal
Category 6: Day of Destruction
What Have I Done to Deserve This?

Content-Based Recommendations for New User (With Search):
Center Stage
Ballet Favorites
Expo: Magic of the White City
```

```
A Raisin in the Sun
Robin and the 7 Hoods
Swan Lake: Tchaikovsky (Matthew Bourne)
Out of Sync
Orchestra Rehearsal
Category 6: Day of Destruction
What Have I Done to Deserve This?
```

[53]:
```python
# Calculate RMSE for validation set
validation_rmse = calculate_rmse(validation_ratings_matrix.toarray(),␣
↪validation_predictions)
print('validation RMSE:', validation_rmse)
```

```
validation RMSE: 0.9183960844224338
```

[ ]:
```python
# 0.9525 Cinematch Netflix score
```

[ ]:
```python

```

[ ]:
```python

```