Average Ratings Implementation

Mike Altschwager

Outlines a base scikit learn pipeline for using average ratings to determine recommendations for a user.
Lessons learned:

*scipy api is quite funky. Long gone are the days of linq queries to manipulate sets. Thought it was interesting that coo matrixes do not allow grabbing a single entry

*slicing the test user set into another 80/20 set of "inputs for the model" vs "things to test agaisnt" was quite tricky and I'm sure I overthought it

*big data, big pain! Took a while to figure out performance

**Things were much easier to figure out when I realized I should just process a subset of the dataset to figure out the logic and then circle back to figure out performance

In [3]:
```
import pandas as pd
from scipy.sparse import csr_matrix, csc_matrix
import numpy as np
```

```
C:\Users\miked\AppData\Local\Temp\ipykernel_62928\1377437179.py:1: DeprecationWarnin
g:
Pyarrow will become a required dependency of pandas in the next major release of pan
das (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better inte
roperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

Load Data

In [4]:
```
movies_df = pd.read_csv('data/movies.csv')
ratings_df = pd.read_csv('data/ratings.csv')
```

In [5]:
```
original_movie_ids = set(movies_df["movieId"])
movie_id_map = {original : new for new, original in enumerate(original_movie_ids) }
movies_df["movieId"] = movies_df["movieId"].map(movie_id_map)
movies_df.head()
```

Out[5]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [6]:

```
original_movie_ids = set(ratings_df["movieId"])
movie_id_map = {original : new for new, original in enumerate(original_movie_ids) }
ratings_df["movieId"] = ratings_df["movieId"].map(movie_id_map)
ratings_df.head()
```

Out[6]:

	userId	movieId	rating	timestamp
0	1	296	5.0	1147880044
1	1	306	3.5	1147868817
2	1	307	5.0	1147868828
3	1	665	5.0	1147878820
4	1	899	3.5	1147868510

Build Raiding Matrix

In [7]:

```
# Get the unique user IDs and movie IDs
user_ids = ratings_df['userId'].unique()
movie_ids = ratings_df['movieId'].unique()

# Create a dictionary to map movie IDs to column indices
movie_id_map = {movie_id: i for i, movie_id in enumerate(movie_ids)}
```

In [8]:

```
len(movie_ids)
```

Out[8]: 59047

In [9]:

```
# Initialize lists to store the row indices, column indices, and ratings
row_indices = []
col_indices = []
ratings = []

# Iterate over the ratings dataframe
for _, row in ratings_df.iterrows():
    row_indices.append(int(row['userId']))
    col_indices.append(int(row['movieId']))
    ratings.append(row['rating'])
```

In [10]:
```
# Create the sparse spatial matrix
sparse_matrix = csr_matrix((ratings, (row_indices, col_indices)))
sparse_matrix.shape
```

Out[10]:
```
(162542, 59047)
```

Define training function

In [12]:
```
def Average(lst):
    return sum(lst) / len(lst)
```

In [13]:
```
def getColumnAverage(matrix, column_index):
    col = matrix.getcol(column_index)
    non_zero_column = col[col.nonzero()]
    return Average(np.squeeze(np.asarray(non_zero_column)))
```

In [14]:
```
from sklearn.base import BaseEstimator

class AverageRatingModel(BaseEstimator):
    avg_ratings = None

    def fit(self, training_ratings_matrix: csc_matrix):
        # Calculate the column-wise average of the sparse matrix
        averages = training_ratings_matrix.sum(0) / training_ratings_matrix.getnnz(
        self.avg_ratings = averages[0]

    def predict(self, user_ratings_matrix):
        # Create a matrix of avg_ratings with duplicate rows
        avg_ratings_matrix = np.repeat(self.avg_ratings, user_ratings_matrix.shape[
        # Return the avg_ratings_matrix
        return avg_ratings_matrix
```

Run Training

In [16]:
```
from sklearn.model_selection import train_test_split

train_dataset, test_dataset = train_test_split(sparse_matrix, test_size=0.2, train_
```

In [17]:
```
avg_rat_model = AverageRatingModel()

avg_rat_model.fit(train_dataset)
```

```
C:\Users\miked\AppData\Local\Temp\ipykernel_62928\3076830561.py:8: RuntimeWarning: i
nvalid value encountered in divide
  averages = training_ratings_matrix.sum(0) / training_ratings_matrix.getnnz(0)
```

In [18]:
```
print(avg_rat_model.avg_ratings.shape)

print(train_dataset.shape)
```

```
print(test_dataset.shape)
```

```
(1, 59047)
(130033, 59047)
(32509, 59047)
```

Evaluate

In [20]:
```
import numpy as np

# Initialize empty lists to store the row indices, column indices, and values
row_indices_80 = []
col_indices_80 = []
values_80 = []

row_indices_20 = []
col_indices_20 = []
values_20 = []

# Iterate over each row in test_dataset
for row_idx in range(test_dataset.shape[0]):
    # Get the non-zero indices and values for the current row
    non_zero_indices = test_dataset[row_idx].nonzero()[1]
    non_zero_values = test_dataset[row_idx].data

    # Calculate the number of non-zero values to include in the 80% matrix
    num_values_80 = int(len(non_zero_values) * 0.8)

    # Split the non-zero indices and values into 80% and 20% portions
    indices_80 = np.random.choice(non_zero_indices, size=num_values_80, replace=Fal
    indices_20 = np.setdiff1d(non_zero_indices, indices_80)

    values_80.extend(non_zero_values[np.isin(non_zero_indices, indices_80)])
    values_20.extend(non_zero_values[np.isin(non_zero_indices, indices_20)])

    row_indices_80.extend([row_idx] * len(indices_80))
    row_indices_20.extend([row_idx] * len(indices_20))

    col_indices_80.extend(indices_80)
    col_indices_20.extend(indices_20)

# Create the 80% and 20% csc_matrix objects
input_ratings = csc_matrix((values_80, (row_indices_80, col_indices_80)))
test_ratings = csc_matrix((values_20, (row_indices_20, col_indices_20)))
```

In [21]:
```
#Need to resize the input_ratings and test_ratings to match the shape of the origin
input_ratings.resize(test_dataset.shape)
test_ratings.resize(test_dataset.shape)
```

In [22]:
```
#Load the ratings back into an array in order to compare them
def compare_ratings(predictions: csc_matrix, test_ratings: csc_matrix):
    test_ratings_array = []
    predictions_array = []
```

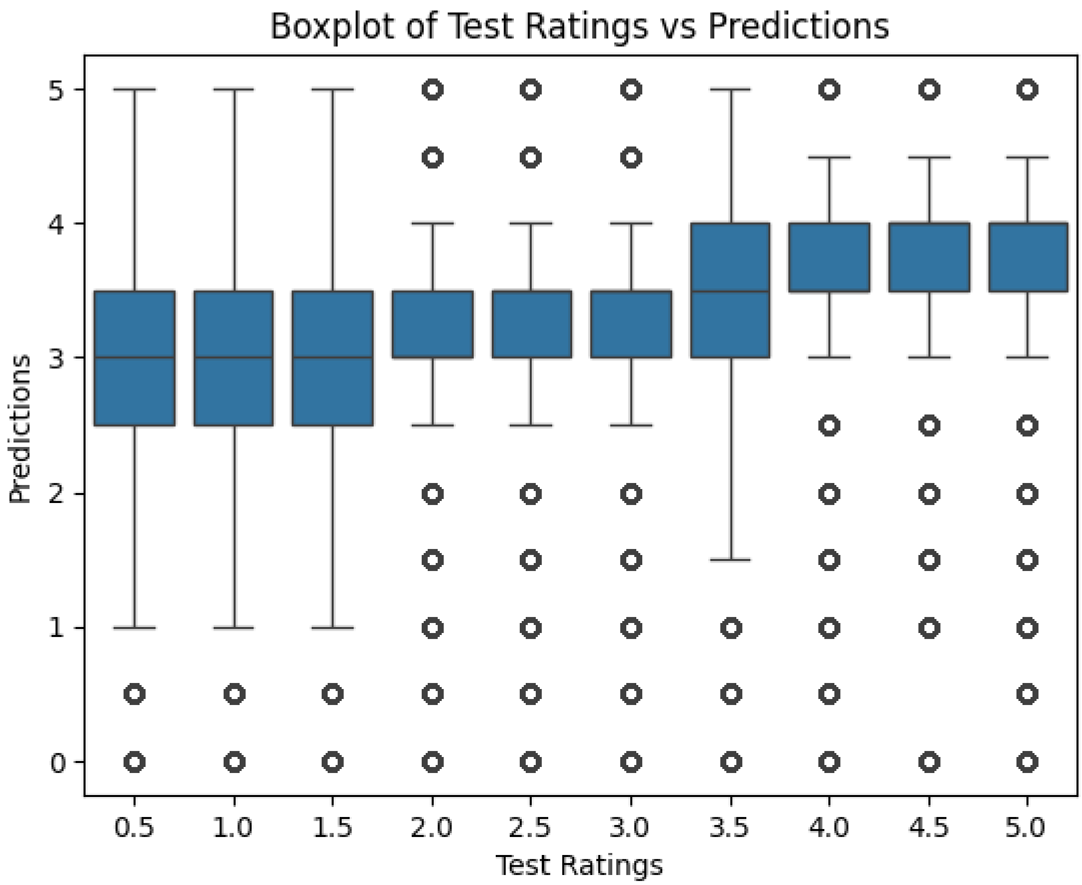
```
    non_zero_indices = test_ratings.nonzero()
    for i in non_zero_indices[0]:
        for j in non_zero_indices[1]:
            test_ratings_array.append(test_ratings[i,j])
            predictions_array.append(predictions[i,j])
    return test_ratings_array, predictions_array
```

In [24]:
```
test_ratings_array = []
predictions_array = []
batch_size = 1 #batch size of 1 ended up being significantly faster, scaling expone

for i in range(0, input_ratings.shape[0], batch_size):
    batch_input_og = input_ratings[i:i+batch_size]
    batch_predictions = avg_rat_model.predict(batch_input_og)
    batch_test_ratings = test_ratings[i:i+batch_size]
    tr, pred = compare_ratings(batch_predictions, batch_test_ratings)
    test_ratings_array.extend(tr)
    predictions_array.extend(pred)
    #break #this is just to short circuit the loop for testing purposes
```

In [27]:
```
import numpy as np

# Replace NaNs that came from dividing by 0 in the fit with 0s, probabbly could do
predictions_array = np.nan_to_num(predictions_array)
contains_nan = np.isnan(predictions_array).any()
print(contains_nan)
```

```
False
```

In [31]:
```
from matplotlib import pyplot as plt
import seaborn as sns


# Create a new array to store the rounded predictions
rounded_predictions_array = np.round(predictions_array * 2) / 2

# Combine the test_ratings_array and rounded_predictions_array into a DataFrame
data = pd.DataFrame({'Test Ratings': test_ratings_array, 'Predictions': rounded_pre

# Create a figure and axis
fig, ax = plt.subplots()

# Create a boxplot using seaborn
sns.boxplot(data=data, x='Test Ratings', y='Predictions', ax=ax)

# Set the labels for x-axis and y-axis
ax.set_xlabel('Test Ratings')
ax.set_ylabel('Predictions')

# Set the title of the plot
ax.set_title('Boxplot of Test Ratings vs Predictions')

# Show the plot
plt.show()
```



In [32]:
```
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
#Root mean-squared error (RMSE)
rmse = mean_squared_error(test_ratings_array, predictions_array)
print('rmse: ', rmse)
#Pearson's Correlation Coefficient (R2)
r2 = r2_score(test_ratings_array, predictions_array)
print('r2: ', r2)
#Fraction of user-movie pairs with non-zero predicted ratings
print('Fraction of user-movie pairs with non-zero predicted ratings ', np.count_non
#Fraction of user-movie ratings with a predicted values (recall)
subtracted_array = np.subtract(predictions_array, test_ratings_array) #zeros mean p
subtracted_array_rounded = np.round(subtracted_array * 2) / 2 #round to nearest 0.5

print('Fraction of user-movie pairs with non-zero predicted ratings ', np.count_non
#RMSE is appropriate if we want to exactly the predict the ratings of the users.  R
```

```
rmse:  0.9499019275199527
r2:  0.17882501693724306
Fraction of user-movie pairs with non-zero predicted ratings  0.9990399113515667
Fraction of user-movie pairs with non-zero predicted ratings  0.7619437210368096
```

In [ ]: