T3am Zoom3rs

Project Option A

Yu Chen – Associations between the databases

Aditya Jain – Collaborative filtering and recommendations

Thinh Nguyen – Evaluation methods and RMSE calculation

Problem Introduction

The problem is creating a recommender system that accurately suggests movies that a particular user would like. This requires a system that uses the user's past data and other users with similar preferences. Some of the challenges arise from finding the right dataset and how to deal with sparse matrices that are naturally a property of such recommender systems. Without a labeled dataset, this imposes the use of unsupervised techniques which has the downside of lower accuracy.

Solution

We employed collaborative filtering since the technique is specifically designed for recommender systems. There are two paradigms with such systems. First, there is user-based collaborative filtering in which the premise is finding nearby users for the user of interest. We assume that doing so will give a set of users with similar preferences. These nearby users are calculated using cosine similarity. Each user is represented as a geometric point in N-D dimensional Euclidean space. The N is the number of movies that exist where each axis represents a movie rating. For example, if there are five movies, there would be a 5-Dimensional Euclidean space, and a user with a coordinate of (1,1,1,1,1) means that such user has rated 1 for all movies. So, we populate this N-D dimensional space with all the user data. For a user of interest, we find their top-k nearest neighbors. To give them a list of recommended movies, we iterate over a list of their unrated movies and predict each one. The prediction is as follows. One, we iterated through their top-k neighbors that have rated the movie and calculated the weighted average. The weight in this average is the cosine similarity. This process is repeated for all unrated movies of the user. Finally, the list of predicted movies is sorted in descending order, and the first couple is selected as the movies to recommend.

Similarly, the above process is the same with item-based collaborative filtering. The only difference is that Euclidean space is determined by the number of users, and each point in the space is a movie. Each movie's coordinate in the space represents the ratings from each user. For example, a ten-user dataset will have a 10-dimensional Euclidean space. An item with a rating of (1,2,1,4,2,5,6,2,7,5) means that user1 has rated 1, user2 has rated 2, user3 has rated 1, and so on. The space is populated with all movies. And the process is the same in finding the top-k nearest items with similar ratings from a similar set of users. The similarity is based on cosine similarity. Again, the process is the same as user-based; except now we are predicting a movie's rating based on the average rating of its nearby neighbors from that user.

Function to create matrix for user-based and item-based filtering

```python
# read the ratings data from the CSV file
def read_ratings_user(ratings_df):

    # pivot the DataFrame to create a matrix of user ratings
    ratings_matrix = ratings_df.pivot_table(index='userId', columns='movieId', values='rating')
    # fill missing values with zeros
    ratings_matrix = ratings_matrix.fillna(0)

    return ratings_matrix

def read_ratings_item(ratings_df):

    # pivot the DataFrame to create a matrix of user ratings
    ratings_matrix = ratings_df.pivot_table(index='movieId', columns='userId', values='rating')
    # fill missing values with zeros
    ratings_matrix = ratings_matrix.fillna(0)

    return ratings_matrix
```

The ratings data is passed through the above functions to generate sparse matrix for user and item-based filtering. The output of the function is then passed through a recommendation function which performs collaborative filtering to calculate the weighted average for each user movie pair and predict the rating.

```python
def cf(user_id, k, ratings_df, movie_id):
    user_ratings = ratings_df.loc[user_id]

    # filter the ratings data to include only the users who have rated the movie
    movie_ratings = ratings_df[ratings_df[movie_id] != 0]

    # calculate the cosine similarity between user_id and the users who have rated the movie
    similarity_matrix = cosine_similarity(user_ratings.values.reshape(1, -1), movie_ratings.values)

    # create a dataframe of (user_id, similarity) pairs
    user_similarity_df = pd.DataFrame(index = movie_ratings.index, data = similarity_matrix[0], columns=['similarity'])

    # user_similarity_df

    # sort the users by their similarity to user_id
    sorted_users = user_similarity_df.sort_values(by='similarity', ascending=False)

    # # get the top-k most similar users
    top_k_users = sorted_users.iloc[1:k+1]

    num = 0
    den = 0
    numerator = 0
    denominator = 0
    for top_user_id in top_k_users.index:
        # print('a')
        # print(top_user_id)
        numerator = user_similarity_df.loc[user_similarity_df.index == top_user_id, 'similarity'] * ratings_df.loc[top_user_id, movie_id]
        num = num + numerator.values
        denominator = user_similarity_df.loc[user_similarity_df.index == top_user_id, 'similarity']
        den = den + denominator.values

    if den == 0 :
        den = 1

    predicted_rating = num/den

    return predicted_rating
```

Result

To evaluate our implementation of the two paradigms of collaborative filtering. We calculated their accuracies by using the concept of a root mean square error (RMSE). We used this method on our test/train split data and compared the accuracy. RMSE were calculated by filtering the dataset such that items or users with N/A are removed. For instance. in user-based collaborative filtering, we iterate through each column while removing zero values from the data. After, we have a matrix with a variable length for each column. This, we mask out the values and use collaborative filtering to predict, and the RMSE of each column is calculated by working with the predicted values and the actual values from the data. Finally, each column has a sum of the squared differences, and we take the average of these sums and square root it for the final RMSE calculation.

```python
# calculate average rmse
def accuracy(df):

    # squared diffs of each column
    total_sd = 0
    count = 0

    for col_index in df.columns:

        # filter out zero cells
        filtered_series = df[col_index][df[col_index] != 0]

        # sum the squared differences of the column
        squared_diff = 0
        for row_index, actual in filtered_series.iteritems():
            predicted = cf(row_index, 10, df, col_index)
            squared_diff += (predicted - actual)**2
            count += 1

        total_sd += squared_diff

    # return the rmse
    return np.sqrt(total_sd / count)
```

```python
acc_df_user_train = accuracy(ratings_df_user_train)
acc_df_item_train = accuracy(ratings_df_item_train)
acc_df_user_test = accuracy(ratings_df_user_test)
acc_df_item_test = accuracy(ratings_df_item_test)
```

```python
print(acc_df_user_train,acc_df_item_train,acc_df_user_test,acc_df_item_test)
```

✓ 0.4s

```
[1.17638217] [0.85908481] [1.43586912] [0.95552221]
```

The first output is the RMSE of train data when user-based collaborative filtering is applied, while the second is also on train data but with item-based filtering. Similarly, for the third and fourth output RMSE is calculated for test dataset. The item-based filtering performed better with test accuracy of 0.955 RMSE, compared to 1.435 RMSE for user-based filtering.

Testing performance using surprise package for item-based recommendations

```
    ratings = ratings.drop(columns='timestamp')
    reader = Reader()
    #dataset creation
    data = Dataset.load_from_df(ratings, reader)
    #model
    knn = KNNBasic()
    #Evaluating the performance in terms of RMSE
    cross_validate(knn, data, measures=['RMSE', 'mae'], cv = 3)
```

```
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.

{'test_rmse': array([0.95804097, 0.95393686, 0.95986445]),
 'test_mae': array([0.73680026, 0.73299765, 0.735571  ]),
 'fit_time': (0.22442865371704102, 0.20461082458496094, 0.2236475944519043),
 'test_time': (3.532680034637451, 2.9211630821228027, 3.161729335784912)}
```

```
    #Define the SVD algorithm object
    svd = SVD()
    #Evaluate the performance in terms of RMSE
    cross_validate(svd, data, measures=['RMSE'], cv = 3)
```

```
{'test_rmse': array([0.88334761, 0.87721976, 0.87618597]),
 'fit_time': (1.868929386138916, 1.3211796283721924, 1.223247766494751),
 'test_time': (0.30908727645874023, 0.5051188468933105, 0.3965482711791992)}
```

Applying KNN gave comparable results with our collaborative filtering model, however by applying dimension reduction techniques like singular value decomposition (SVD) we can observe that the accuracy of our model increased on test dataset.

Functions to recommend Top-K movies for a particular user

```
# best recommendation for a user

def recommendation_user(user_id, ratings_df,movies, k):
    zero_indexes = np.where(ratings_df.loc[user_id] == 0)[0]
    column_names = ratings_df.iloc[:, zero_indexes].columns
    # prediction_user = []

    col_names = ['movieId', 'prediction']

    # Create an empty DataFrame with the defined column names
    prediction_df = pd.DataFrame(columns=col_names)

    # zero_indexes
    for movie in column_names:
        # print(movie)
        pred = cf(user_id,k,ratings_df, movie)
        # print(pred)
        row_dict = {'movieId': movie, 'prediction': pred}
        prediction_df = pd.concat([prediction_df, pd.DataFrame([row_dict])])
        # pd.concat(prediction_df,row_dict, axis= 0 ,ignore_index=True)

    prediction_df = shuffle(prediction_df)
    prediction_df = prediction_df.sort_values("prediction", ascending=False).head(k)

    movie_recommended = pd.merge(prediction_df, movies, on='movieId', how='left').drop(columns=['movieId', 'prediction'])

    return (movie_recommended)
```

```python
def recommendation_item(user_id, ratings_df,movies, k):
    zero_indexes = np.where(ratings_df[user_id] == 0)[0]
    # zero_indexes
    movie_names = ratings_df.iloc[zero_indexes,]
    movie_index = movie_names.index
    # movie_index
    col_names = ['movieId', 'prediction']

    # Create an empty DataFrame with the defined column names
    prediction_df = pd.DataFrame(columns=col_names)

    for movie in movie_index:
        # print(movie)
        pred = cf(movie,k,ratings_df,user_id)
        # print(pred)
        row_dict = {'movieId': movie, 'prediction': pred}
        prediction_df = pd.concat([prediction_df, pd.DataFrame([row_dict])])
    prediction_df = shuffle(prediction_df)
    prediction_df = prediction_df.sort_values("prediction", ascending=False).head(k)
    movie_recommended = pd.merge(prediction_df, movies, on='movieId', how='left').drop(columns=['movieId', 'prediction'])

    return(movie_recommended)
```

Item based and User based Recommendations

```python
a = recommendation_item(6,ratings_df_item_train,movies,10)
a
```

	title	genres
0	Taxi 3 (2003)	Action|Comedy
1	Lee Daniels' The Butler (2013)	Drama
2	Tale of Princess Kaguya, The (Kaguyahime no mo...	Animation|Drama|Fantasy
3	Breathe (2014)	Drama
4	Private Lives of Pippa Lee, The (2009)	Drama
5	Tooth Fairy 2 (2012)	Children|Comedy
6	Play Time (a.k.a. Playtime) (1967)	Comedy
7	Boy A (2007)	Crime|Drama
8	Race (2016)	Drama
9	Still Walking (Aruitemo aruitemo) (2008)	Drama

```python
b = recommendation_user(6,ratings_df_user_train,movies,10)
b
```

	title	genres
0	Trial, The (Procès, Le) (1962)	Drama
1	Hush... Hush, Sweet Charlotte (1964)	Horror|Thriller
2	With a Friend Like Harry... (Harry, un ami qui...	Drama|Thriller
3	They Call Me Trinity (1971)	Comedy|Western
4	Written on the Wind (1956)	Drama
5	Dolls (2002)	Drama|Romance
6	Tree of Life, The (2011)	Drama
7	Awful Truth, The (1937)	Comedy|Romance
8	Police Story (Ging chaat goo si) (1985)	Action|Comedy|Crime|Thriller
9	Hard Way, The (1991)	Action|Comedy

Lessons learned

In the progress of this project, we learned the use of cosine similarity with N-D dimensional Euclidean space to predict the specific values and the data frame with multiple unknown values to find the k nearest neighbors and calculate the accuracy by comparing to the actual non-NA data by using root mean square error method. The other thing we improved for accuracy is iterating the data, which is NA and predicting the value, then repeating this step for other NA-values with the result of the previous calculation instead of calculating the result all together at the same time.

The main lesson we learned is the implementation of collaborative filtering for item-based and user-based. This involved reading over the Pandas library to handle sparse matrices and NumPy for the calculation of RMSE to evaluate user-based and item-based collaborative filtering frameworks. Furthermore, we learned the downsides of replacing NA values with zeroes. The problem is the tacit assumption, by putting a zero, implies that the item has the lowest rating. To counteract this, we learned a normalization technique that subtracts the dataset from the mean of the dataset. Now, replacing NA values with zero no longer implies the lowest rating, it implies a neutral rating since the dataset has been centered around zero.

By implementing dimensional reduction techniques, we learned that the algorithm achieved better accuracy on test set. Thus, reducing unwanted noise and redundancy in data is crucial for better performance of algorithms, in terms of accuracy and speed.

Further discussion of the result

The dataset contains items with very few ratings from the user, one way to address the problem of sparsity, or the lack of data for some items, is by using algorithms that are less reliant on having many ratings for each item.

For example, instead of using user-based or item-based collaborative filtering, which requires a considerable number of ratings to make accurate predictions, we could use a content-based filtering approach, which makes recommendations based on the characteristics of the items themselves. This can help provide recommendations for items that have not been rated by many users, but still have similar characteristics to other items that the user has rated.

The recommendation algorithm thus can be improved by incorporating more information like movie genre, actors, and directors who were involved with the movie. Furthermore, we can increase our accuracy by adding movie release dates, which gives a sense of time to our algorithm.

Another approach is to use matrix factorization algorithms, which can effectively model the underlying relationships between users and items even with sparse data. These algorithms decompose the rating matrix into a product of lower-dimensional matrices (as tested during SVD), which can capture the latent preferences of users and the characteristics of items. This can help make accurate predictions for items that have not been rated by many users, and can also help identify related items that the user may be interested in.