RMIT UNIVERSITY

EFFECTIVE MISSING DATA PREDICTION FOR COLLABORATIVE FILTERING AND SOLUTION IMPLEMENTATION

I certify that this is all my own original work. If I took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in my submission. I will show I agree to this honor code by typing “Yes": Yes.

Joyal Joy Madeckal
S3860476

Contents

1. Brief intro to Collaborative filtering
2. Effective Missing Data Prediction
3. How?
4. Why?
5. Solution Implementation

Collaborative Filtering

- Collaborative filtering is a main technique used in recommendation systems. This algorithm helps to provide a suitable recommendation to a user in which the user may be interested in.
- Two types – Memory based and Model based
- Data sparsity is the main issue with the algorithm
- The missing values is assumed to have mean rating while implementation.
- There are other models which predict the missing ratings based on Pearson similarity, but they will assign a value to all the missing ratings which can have a negative influence.

Effective Missing Data Prediction

- Instead of considering the missing data as average ratings the solution gives an idea to predict the missing data (missing ratings)
- First step is Pearson correlation coefficient (PCC) algorithm enhancement
- Second step is to predict the missing data using the user by user and item by item Pearson similarity matrix

$$Sim(a,u) = \frac{\sum_{i \in I(a) \cap I(u)} (r_{a,i} - \overline{r}_a) \cdot (r_{u,i} - \overline{r}_u)}{\sqrt{\sum_{i \in I(a) \cap I(u)} (r_{a,i} - \overline{r}_a)^2} \cdot \sqrt{\sum_{i \in I(a) \cap I(u)} (r_{u,i} - \overline{r}_u)^2}},$$

PCC Algorithm

$$Sim'(a,u) = \frac{Min(|I_a \cap I_u|, \gamma)}{\gamma} \cdot Sim(a,u),$$

Significance Weighting

(Ma, King & Lyu, 2007)

How?

- In PCC algorithm an extra parameter η is introduced for filtering similar users and similar items.

$$S(u) = \{u_a | Sim'(u_a, u) > \eta, u_a \neq u\},$$

Filter criteria for similar users/items

- The parameter η helps to figure out the most similar users and items.
- If there are no users or items satisfying the criteria then that particular user or item is supposed to have no similar user or items.
- The missing data is considered to be zero in this case.

(Ma, King & Lyu, 2007)

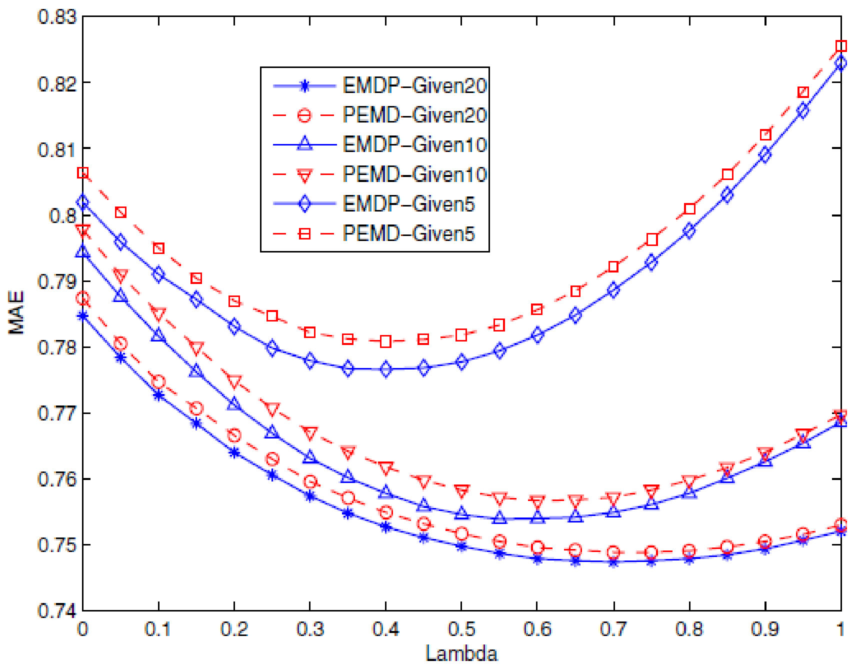
- Similar users and similar items should contribute when missing ratings are predicted.

$$P(r_{u,i}) = \lambda \times (\overline{u} + \frac{\sum_{u_a \in S(u)} Sim'(u_a, u) \cdot (r_{u_a,i} - \overline{u}_a)}{\sum_{u_a \in S(u)} Sim'(u_a, u)}) + (1-\lambda) \times (\overline{i} + \frac{\sum_{i_k \in S(i)} Sim'(i_k, i) \cdot (r_{u,i_k} - \overline{i}_k)}{\sum_{i_k \in S(i)} Sim'(i_k, i)}),$$

Equation for Missing data prediction

- Above equation is suggested by the paper, for determining the missing ratings which is almost similar to the real PCC equation for prediction
- The factor λ is introduced to control the contributions from items and users.
- On the test data set, if the updated user item matrix with predicted ratings is used it is found that it is giving more accurate results

(Ma, King & Lyu, 2007)



- The graph shows a comparison between predicting every missing data (PEMD) and current algorithm (EMDP) predicting missing data based on criteria under same conditions.
- All EMDP values are below the corresponding PEMD values showing the performance of EMDP is better.

(Ma, King & Lyu, 2007)

Why?

- Earlier algorithms' selects similar users/items based on user/item having the highest similarity values. Here, there is a criteria under which the similar users/items are selected. The criteria eliminates the chance for a dissimilar item/user to be present in similar users/items.
- Threshold factor η gives the confidence to decide the threshold value above which the similar users/items are considered.
- Earlier the missing data prediction algorithms predicted the values for all of the missing data whereas here missing data prediction is selective.
- Missing data prediction is done considering user-user and item-item similarities.
- Contribution factor λ gives us the liberality to change the contribution percentages for different recommendation systems as required.
- Data sparsity in the user-item matrix reduces.

Solution Implementation

- To implement the solution we require the user by user and item by item similarity matrix.
- The first block of code in the solution is to form the user by user Pearson similarity matrix.
- The second block of code in the solution is to form the item by item Pearson similarity matrix.
- The first two blocks of code don't have anything related to the solution suggested.
- The last block of code uses the Pearson similarity matrices for users and items to implement the solution as per the paper (Prediction of missing values).
- Solution developed is heavily dependent on the libraries Pandas and NumPy.

```python
# Looping the users against each other for forming the pearson similarity matrix.
for selected_user_data in imputed_train_ds.itertuples():
    for looping_user_data in imputed_train_ds.itertuples():

        # Forming the ratings vector for the pair of users.
        looping_user_vec = np.array(looping_user_data[1:])
        selected_user_vec = np.array(selected_user_data[1:])

        # Finding the indices of items commonly rated by the users.
        common_indices = np.intersect1d(np.where(selected_user_vec > 0), np.where(looping_user_vec>0))
        # If there are no commonly rated items then the user by user similarity is considered 0 and will proceed with the next p
        if not len(common_indices):
            continue

        # Finding the average ratings of the users.
        looping_user_vec_avg = np.sum(looping_user_vec)/(np.count_nonzero(looping_user_vec) + EPSILON)
        selected_user_vec_avg = np.sum(selected_user_vec)/(np.count_nonzero(selected_user_vec) + EPSILON)

        # Centering the ratings of the users by subtracting the average ratings from ratings.
        looping_user_vec_common_avgs= looping_user_vec[common_indices] - looping_user_vec_avg
        selected_user_vec_common_avgs = selected_user_vec[common_indices] - selected_user_vec_avg

        # Calculation of squares of the centered user ratings.
        looping_user_vec_common_squares = np.square(looping_user_vec_common_avgs)
        selected_user_vec_common_squares = np.square(selected_user_vec_common_avgs)

        # Similarity calculation between the users.
        similarity = (np.sum(looping_user_vec_common_avgs * selected_user_vec_common_avgs)/
                      (np.sqrt(np.sum(looping_user_vec_common_squares)) * np.sqrt(np.sum(selected_user_vec_common_squares)) +
                       EPSILON))

        # Applying the significance weighting to calculated similarities
        weighted_similarity = (min(len(common_indices), GAMMA)/ GAMMA) * similarity

        # Adding the weighted similarity to the pearson similarity matrix
        np_user_pearson_corr_user_by_user[selected_user_data[0]][looping_user_data[0]] = weighted_similarity
```

This is the block of the code which calculates the Pearson similarity matrix for the users.

```python
# Looping the items against each other for forming the pearson similarity matrix.
for selected_item_data in imputed_train_ds.transpose().itertuples():
    for looping_item_data in imputed_train_ds.transpose().itertuples():

        # Forming the ratings vector for the pair of items.
        looping_item_vec = np.array(looping_item_data[1:])
        selected_item_vec = np.array(selected_item_data[1:])

        # Finding the indices of users who have rated the pair of items.
        common_indices = np.intersect1d(np.where(selected_item_vec > 0), np.where(looping_item_vec>0))
        # If there are no common users then the item by item similarity is considered 0 and will proceed with the next pair of
        if not len(common_indices):
            continue

        # Finding the average ratings of the items.
        looping_item_vec_avg = np.sum(looping_item_vec)/(np.count_nonzero(looping_item_vec) + EPSILON)
        selected_item_vec_avg = np.sum(selected_item_vec)/(np.count_nonzero(selected_item_vec) + EPSILON)

        # Centering the ratings of the items by subtracting the average ratings from ratings.
        looping_item_vec_common_avgs= looping_item_vec[common_indices] - looping_item_vec_avg
        selected_item_vec_common_avgs = selected_item_vec[common_indices] - selected_item_vec_avg

        # Calculation of squares of the centered item ratings.
        looping_item_vec_common_squares = np.square(looping_item_vec_common_avgs)
        selected_item_vec_common_squares = np.square(selected_item_vec_common_avgs)

        # Similarity calculation between the items.
        similarity = (np.sum(looping_item_vec_common_avgs * selected_item_vec_common_avgs)/
                      (np.sqrt(np.sum(looping_item_vec_common_squares)) * np.sqrt(np.sum(selected_item_vec_common_squares)) +
                       EPSILON))

        # Applying the significance weighting to calculated similarities
        weighted_similarity = (min(len(common_indices), DELTA)/ DELTA) * similarity

        # Adding the weighted similarity to the pearson similarity matrix
        np_item_pearson_corr_item_by_item[selected_item_data[0]][looping_item_data[0]] = weighted_similarity
```

This is the block of the code which calculates the Pearson similarity matrix for the items.

```python
# Missing value predictions
# Looping thorugh each user, item , rating combination
for (current_user, current_item), rating in np.ndenumerate(imputed_train_ds.values):
    # Condition to check whether the rating is 0. (Those ratings need to be predicted)
    if not rating:

        # Finding similar user ids and item ids based on the condition mentioned in the paper.
        similar_users_ids_with_current_user_condition_based = np.argwhere(np_user_pearson_corr_user_by_user[current_user] > ITA).flatten()
        similar_items_ids_with_current_item_condition_based = np.argwhere(np_item_pearson_corr_item_by_item[current_item] > THETA).flatten()

        # removing the current user and current item from the array.
        similar_users_ids_with_current_user_condition_based = similar_users_ids_with_current_user_condition_based[similar_users_ids_with_current_user_condition_based != current_user]
        similar_items_ids_with_current_item_condition_based = similar_items_ids_with_current_item_condition_based[similar_items_ids_with_current_item_condition_based != current_item]

        # Skipping the calculation if there are no similar items and similar users.
        if not len(similar_users_ids_with_current_user_condition_based) and not len(similar_items_ids_with_current_item_condition_based):
            continue

        # Finding the pearson coefficients for similar users and items.
        pearson_coeff_similar_users = np_user_pearson_corr_user_by_user[current_user][similar_users_ids_with_current_user_condition_based]
        pearson_coeff_similar_items = np_item_pearson_corr_item_by_item[current_item][similar_items_ids_with_current_item_condition_based]

        # Finding the similar users and items.
        similar_users = imputed_train_ds.values[similar_users_ids_with_current_user_condition_based]
        similar_items = imputed_train_ds.transpose().values[similar_items_ids_with_current_item_condition_based]

        # Calculating the current user and item ratings mean
        current_user_mean = np.sum(imputed_train_ds.values[current_user]) / (np.count_nonzero(imputed_train_ds.values[current_user]) + EPSILON)
        current_item_mean = np.sum(imputed_train_ds.transpose().values[current_item]) / (np.count_nonzero(imputed_train_ds.transpose().values[current_item]) + EPSILON)

        # Calculating the means of all the similar users and items.
        similar_users_mean = np.sum(similar_users, axis=1)/ (np.count_nonzero(similar_users, axis=1) + EPSILON)
        similar_items_mean = np.sum(similar_items, axis=1)/ (np.count_nonzero(similar_items, axis=1) + EPSILON)

        # Condition for finding the users from similar users who has rated current item.
        mask_for_users = similar_users[:,current_item] > 0
        # Condition for finding items from similar items which has been rated by the current user.
        mask_for_items = similar_items[:,current_user] > 0

        # Calculation of the numerator values (pearson coeff * (similar user/item - similar user/item mean)) for both user and item based equations.
        equation_numerator_for_users = pearson_coeff_similar_users[mask_for_users] * (similar_users[mask_for_users, current_item] - similar_users_mean[mask_for_users])
        equation_numerator_for_items = pearson_coeff_similar_items[mask_for_items] * (similar_items[mask_for_items, current_user] - similar_items_mean[mask_for_items])

        # Calculation of the contribution by the similar users/items for determining the missing value for rating.
        similar_user_contribution_for_missing_value = LAMBDA * (current_user_mean + np.sum(equation_numerator_for_users)/(np.sum(pearson_coeff_similar_users[mask_for_users]) + EPSILON))
        similar_item_contribution_for_missing_value = (1 - LAMBDA) * (current_item_mean + np.sum(equation_numerator_for_items)/(np.sum(pearson_coeff_similar_items[mask_for_items]) + EPSILON))

        # Predicted missing value is fitted into the data set to use it for further calculations.
        imputed_train_ds.loc[current_user,current_item] = similar_item_contribution_for_missing_value + similar_user_contribution_for_missing_value
```

- The solution is done carefully in the third block of the code.
- From the user-user and item-item similarity matrices along with user-item rating matrix the required components for the solution equation is calculated

$$P(r_{u,i}) = \lambda \times (\overline{u} + \frac{\sum\limits_{u_a \in S(u)} Sim'(u_a, u) \cdot (r_{u_a,i} - \overline{u}_a)}{\sum\limits_{u_a \in S(u)} Sim'(u_a, u)}) +$$

$$(1 - \lambda) \times (\overline{i} + \frac{\sum\limits_{i_k \in S(i)} Sim'(i_k, i) \cdot (r_{u,i_k} - \overline{i}_k)}{\sum\limits_{i_k \in S(i)} Sim'(i_k, i)}),$$

- The predicted missing value is inserted to the training matrix and it is then used for predicting the ratings for the active users.

RMIT UNIVERSITY

References

- Yongli, R 2021, ‘COSC2670 -> Modules’, Week 9 & Week 10 Learning Materials, COSC2670, RMIT University, viewed from 4th June to 15th June 2021, < https://rmit.instructure.com/courses/79792/modules>

- Ma, H, King, I & Lyu, M 2007, ‘Effective Missing Data Prediction for Collaborative Filtering’, SIGIR 2007: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, Pages 39–46, <https://doi.org/10.1145/1277741.1277751>

- Cielen, D, Meysman, A & Ali, M 2020, Introducing Data Science, Dreamtech Press, Delhi.

- Portilla, J 2021, Python for Data Science and Machine Learning Bootcamp, streaming video, Udemy for Business, viewed 4th June to 15th June 2021, <https://hexagoncci.udemy.com/course/python-for-data-science-and-machine-learning-bootcamp/learn/lecture/5440650#overview>