Construction and Analysis of News Website Recommendation System

Group Name: RecomGenius

Group Member:

LIU JIAXIN 23433698 (leader)

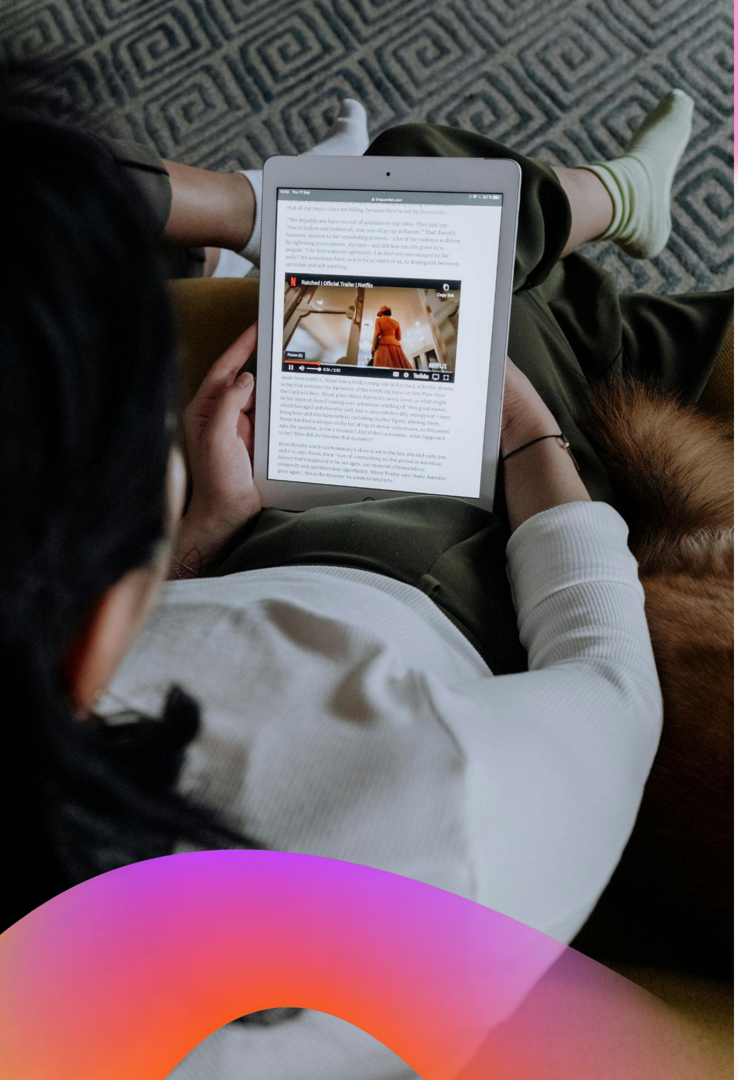
ZHOU YIFAN 23467312

LUO FAN 23435968

QIAN CHENGFENG 23467223

ZHANG ZHENGYI 23467266

Table of contents



01

Introduction

Introduction

News has been playing a multi-faceted role since its birth, and also assumes many social and entertainment attributes. Before the Internet era, the only choices for people to watch news were newspapers, radio and television, which was a passive acceptance. In the Internet era, audiences have more choices and they are free to choose the type of news they like. So in this case, how can we provide better services to the audience?

Introduction

To this end, our group, based on:

1. Improve user click-through rate
2. Increase user retention rate
3. Provide users with a better experience

The recommendation system is written for the above three purposes.

Introduction

We use the user-item basis matrix as the basis for weighting user behavior, supplemented by other algorithms and calculations such as collaborative filtering to build the recommendation system.

We hope to improve recommendation accuracy and user satisfaction with the algorithms we build. and then leave the next part to the team members

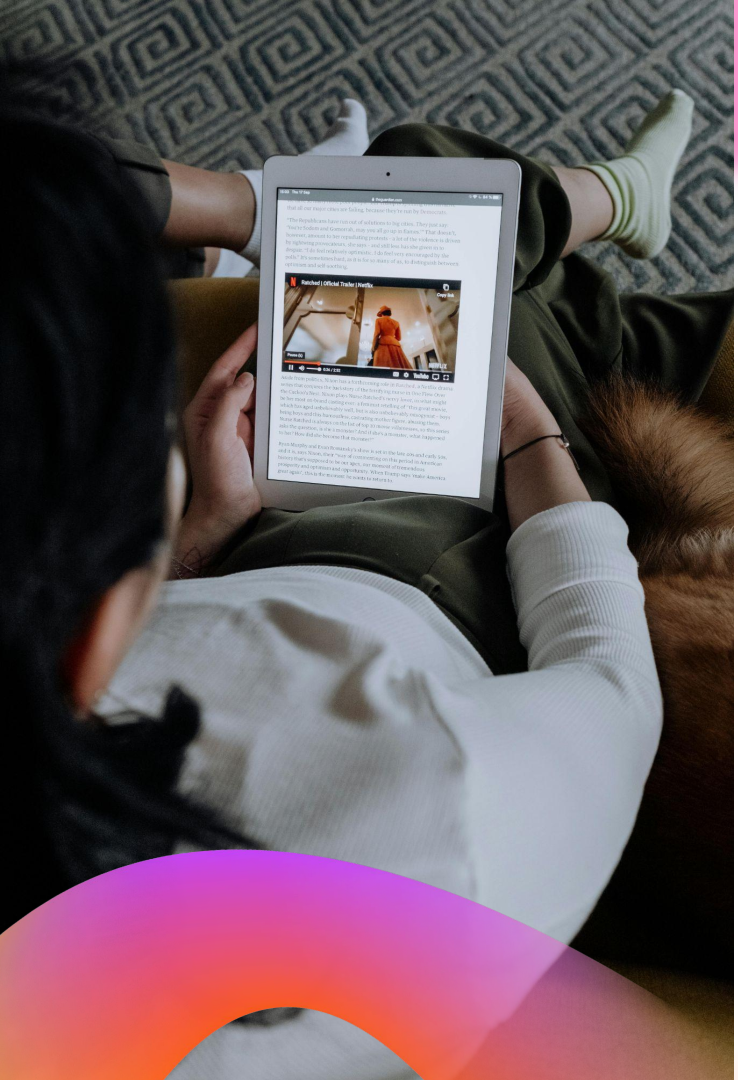
```
def itemCF_recommender(df,iiSimMatrix,currentUser,numItems):
    cuRatedItems = uiMatrixSelection.loc[currentUser].dropna().sort_values(ascending=Fals
    first_item_id = cuRatedItems.index[0]
    itemTocompare = first_item_id
    recommend_list = itemCF_precomputed(iiSimMatrix, itemTocompare, numItems)

    paired_results = []

    for item in recommend_list:
        result = itemCF_prediction(df, currentUser, item)
        paired_results.append((item, result))  # Append a tuple of (element, result)

    paired_results = sorted(paired_results, key=lambda x: x[1], reverse=True)
    return paired_results
```

```
def normalizeUiMatrix(uiMatrix):
    # Fill the NaN value with the minimum value fisrt
    uiMatrix_filled = uiMatrix.fillna(uiMatrix.min().min())

    # Calculate the minimum and maximum values in a matrix
    min_value = np.nanmin(uiMatrix)
    max_value = np.nanmax(uiMatrix)

    # Perform max-min normalisation
    # Subtract the minimum value to get a new matrix that makes the minimum value 0
    # Calculate the difference between the maximum and minimum values to get a normalised range
    # Divide by the normalised range to get the normalised matrix such that the maximum value becomes 1 and the minimum value
    # Multiply the normalised matrix by 10
    uiMatrix_filled_normalized = ((uiMatrix - min_value) / (max_value - min_value)) * 10

    # Reset the NaN value back
    uiMatrix_normalized = uiMatrix_filled_normalized.where(uiMatrix.notna())
    return uiMatrix_normalized
```

```
uiMatrix_normalized = normalizeUiMatrix(uiMatrix)
uiMatrix_normalized.head()
```

	1624	101	1984	801	222	987	381	762	1788	1869	...	1718	6	1072	957	415	900	769	1617
2786	3.571429	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2469	NaN	1.666667	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

ZHANG ZHENGYI



02

User-based CF

Creating a user-item matrix

To compute the user similarity, we first constructed the user-item matrix, where each row represents a user and each column represents a news.

```
[ ]  #Create a user-item binary matrix
     uiMatrix = pd.DataFrame(columns=content, index=users)
     uiMatrix.head(2)
```

	1624	101	1984	801	222	987	381	762	1788	1869	...	1718	6	1072	957	415	900	769	1617	534	442
2786	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2469	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

2 rows × 1973 columns

Some problems

However User-based CF has some problems of its own, such as cold start of users: for new users or new items, recommendations cannot be made accurately due to the lack of sufficient historical data;
Or data bias: users' ratings of items may be biased, e.g., some users tend to give high or low ratings, which affects the accuracy of similarity calculation and recommendation results.

Basic solutions

In order to solve several problems with UserDF, we chose to adopt the following solution：

1.Similarity is best calculated using the Pearson correlation coefficient

```
# Compute correlation using corrwith
corrDf = df.corrwith(cuDf, axis=1, method='pearson')
```

Basic solutions

2. Set the Behavioural Implicit Ratings

we give read a weight of 0, and read_more represents that users click the news or expand the news page for more detail, hence we give it a higher weight 20.Considering that the user is interested in the news before checking the category, Based on this, we gave the category a weight of 50 and gave the author a higher weight of 80.

```
eventWeights = {
        'read': 0,
        'read_more': 20,
        'author': 80,
        'category': 50,
        'read_comments': 100}
```

About the read_comments, comment section is always located at the bottom of the news webpage, if users take the operation of reading comments, which means he/she might has read whole page of news report, so we give it the highest weight of 100 among these 5 operations.

LIU JIAXIN

Basic solutions

This allows us to calculate the implied ratings for each user-item combination.
Populate the user-item matrix uiMatrix with IR values.

$$IR_(i, u) = (w_1 * \#event_1) + (w_2 * \#event_2) + \cdots + (w_n * \#event_n)$$

```
[ ] # Iterate the evidence
    for index, row in evidence.iterrows():
            # Select the user and items involved
            currentUser = row['UserID']
            currentContent = row['ContentID']

            # Extract the appropriate weight for the event
            w = eventWeights[row['Event']] # w is weight

            # Find the value eventually stored for the current user-item combination
            currentValue = uiMatrix.at[currentUser, currentContent]
            if np.isnan(currentValue):
                    currentValue = 0

            # Compute the new value and update the user-item matrix
            updatedValue = currentValue + w #+ (1 * w)
            uiMatrix.at[currentUser, currentContent] = updatedValue
```

Basic solutions

3. Normalise the matrix

We update the user-item matrix by normalizing the values between 0 and 10.

```
def normalizeUiMatrix(uiMatrix):
    # Fill the NaN value with the minimum value fisrt
    uiMatrix_filled = uiMatrix.fillna(uiMatrix.min().min())

    # Calculate the minimum and maximum values in a matrix
    min_value = np.nanmin(uiMatrix)
    max_value = np.nanmax(uiMatrix)

    # Perform max-min normalisation
    # Subtract the minimum value to get a new matrix that makes the minimum value 0
    # Calculate the difference between the maximum and minimum values to get a normalised range
    # Divide by the normalised range to get the normalised matrix such that the maximum value becomes 1 and the minimum value bec
    # Multiply the normalised matrix by 10
    uiMatrix_filled_normalized = ((uiMatrix - min_value) / (max_value - min_value)) * 10

    # Reset the NaN value back
    uiMatrix_normalized = uiMatrix_filled_normalized.where(uiMatrix.notna())
    return uiMatrix_normalized
```

```
[ ] uiMatrix_normalized = normalizeUiMatrix(uiMatrix)
    uiMatrix_normalized.head()
```

	1624	101	1984	801	222	987	381	762	1788	1869	...	1718	6	1072	957	415	900	769	1617	
2786	3.571429	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2469	NaN	1.666667	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

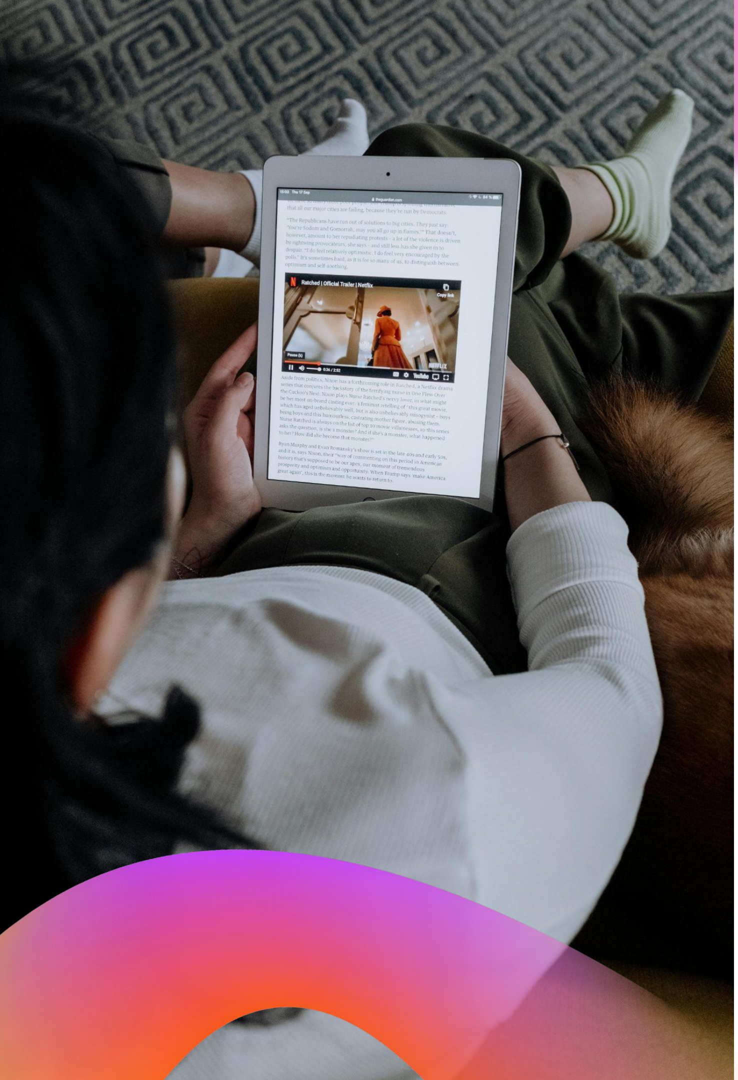
Recommended Demos

Now we can define good recommendation functions to perform recommendations for the user:

```
userCF_prediction(uiMatrix_normalized, 2786, 2, 5)
```

```
/usr/local/lib/python3.10/dist-packages/numpy/lib/function_base.py:2889: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
/usr/local/lib/python3.10/dist-packages/numpy/lib/function_base.py:2748: RuntimeWarning: divide by zero encountered in divide
  c *= np.true_divide(1, fact)
559     5.952381
1502    4.047619
1117    4.047619
1959    4.047619
1076    4.047619
dtype: float64
```

Here, we want to predict the reading taste of user who id is 2786. We set the number of similar users used to calculate the prediction to 2, and return 5 news as recommendations.



03

Item-based CF

Basic Solution

First, we calculate the similarity between items.
Base on the user-item matrix that has been normalized and using decay, we precompute the rating to create the item-item similarity matrix.

Steps:

1.Convert the user-item matrix to boolean matrix.
2.Compute overlapping ratings by using "Dot" function.
3.Calculate the similarity between items by using cosine similarity.

	1624	101	1984	801	222	987	381	762	1788	1869	...	1718	6
1624	1.000000	0.043026	−0.000226	0.000000	0.006829	0.000000	0.000000	0.001138	0.000000	0.000000	...	0.0	−0.004195
101	0.043026	1.000000	0.002062	0.003174	−0.000405	−0.010632	0.000000	0.000000	0.000000	−0.014894	...	0.0	0.000000
1984	−0.000226	0.002062	1.000000	0.000000	0.052577	0.010712	−0.009645	−0.016291	0.000000	0.005880	...	0.0	0.000000
801	0.000000	0.003174	0.000000	1.000000	−0.042078	0.007757	0.011382	0.000000	0.000000	0.000000	...	0.0	0.000000
222	0.006829	−0.000405	0.052577	−0.042078	1.000000	−0.010793	0.000000	0.021848	−0.011771	−0.000595	...	0.0	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...
900	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.000000
769	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.000000
1617	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.000000
534	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.000000
442	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.000000

1973 rows × 1973 columns

Basic Solution

Second, we order item similarity for recommendation.

Steps:
1.We check the news items that showcase user has rated,and choose one with highest rating for recommendation.
2.Define the itemCF_precomputed function to get the neighbourhood.

```
def itemCF_precomputed(similarityMatrix, currentItem, numItems):
    #Select current item from the similarity matrix, remove not rated items, sort the values and select the top-k items
    recommendationList = similarityMatrix[currentItem].dropna().sort_values(ascending=False).head(numItems)
    # Exclude the currentItem from the recommendationList
    recommendationList = recommendationList[recommendationList.index != currentItem]
    return recommendationList.index.to_list()
```

Basic Solution

Third, We designed the recommendation function based on the previous parts.

Steps:

1.Define the function itemCF_prediction for calculating the predicted rating of news.

We calculate the predicted rating according to this formula:

$$Pred(u,i) = \bar{r}_u + \frac{\sum_{j \in S_i} (sim(i,j) \times r_{u,j})}{\sum_{j \in S_i} sim(i,j)}$$

where

- $\bar{r}_u$ is the average rating of the user u.
- $r_{u,j}$ is the active user's u rating of item j.
- S_i is the set of items in the neighborhood that user u has rated.
- $Pred(u,i)$ is the predicted rating for user u of item i.
- $sim(i,j)$ is the similarity between item i and item j.

LUO FAN

Basic Solution

Steps:

2.Combine with itemCF_prediction and itemCF_precomputed functions to define the itemCF_recommender function to presenting the recommendation.

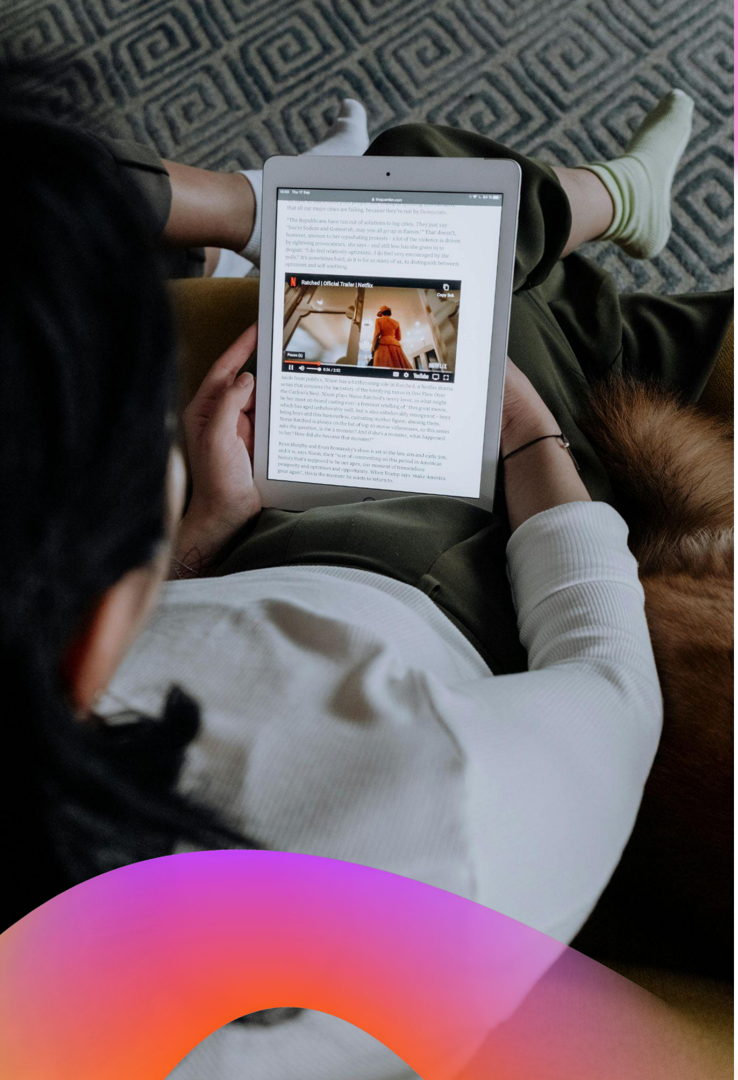
```python
def itemCF_recommender(df,iiSimMatrix,currentUser,numItems):
    cuRatedItems = uiMatrixSelection.loc[currentUser].dropna().sort_values(ascending=False
    first_item_id = cuRatedItems.index[0]
    itemTocompare = first_item_id
    recommend_list = itemCF_precomputed(iiSimMatrix, itemTocompare, numItems)

    paired_results = []

    for item in recommend_list:
        result = itemCF_prediction(df, currentUser, item)
        paired_results.append((item, result))  # Append a tuple of (element, result)

    paired_results = sorted(paired_results, key=lambda x: x[1], reverse=True)
    return paired_results
```

Recommendation showcase

Here we choose user **2786** as our showcase, and recommender base on the news item of the user to recommend **3** item and predicted the rating of them.

```
result = itemCF_recommender(uiMatrixNorm,iiSimMatrix,2786,3)
result
```

```
[(992, 2.0879634577490864),
 (292, 1.714297800200254 7),
 (1881, 1.1132124199750673)]
```



04

Evaluate the performance

Evaluation: User-based CF

1.User Coverage

Iterate over all users and check if each user has at least one recommendation.

Since the matrix numbers are so large that using the original prediction function would make the computation time required to calculate user coverage too long, we chose to design a function specifically for calculating user coverage based on the following points.

```
[ ] import numpy as np

    def userCF_prediction_simple(df, currentUser, numUsers):
          # Select current user rating
          cuDf = df.loc[currentUser]

          # Check if the user similarity matrix has been pre-calculated
          if 'user_similarity' not in globals():
                global user_similarity
                user_similarity = df.T.corr(method='pearson')

          # Get the first numUsers of the most similar users.
          top_users = user_similarity[currentUser].drop(labels=[currentUser]).nlargest(numUsers)

          # Get items that have not been rated by the current user
          items_to_predict = cuDf[cuDf.isna()].index

          # Get ratings on these items from similar users
          ratings_of_similar_users = df.loc[top_users.index, items_to_predict]

          # Check for scoring of predicted items
          if ratings_of_similar_users.notna().any().any():
                # If there is a score, then at least one item can be predicted, which is all the i
                return True

          return False
```

Evaluation: User-based CF

1.User Coverage

Reprocessing the user-commodity matrix:

- There is no need to calculate exact predictive ratings. You stop the calculation as soon as you find a sufficient number of similar users who have already rated items that the user has not rated.
- Operations on a DataFrame can be slow when dealing with large-scale data. I chose to convert the data into NumPy arrays for the calculations.
- If there is a rating, then at least one item can be predicted, which means that the recommender system reaches this user, and can be returned directly.

```
[ ] def calculate_user_coverage(df, numUsers):
        # of users calculated to be able to recommend at least one item for them
        num_users_with_recommendations = 0

        # Iterate over each user
        for currentUser in df.index:
            # Check if at least one item can be recommended for this user
            if userCF_prediction_simple(df, currentUser, numUsers):
                num_users_with_recommendations += 1

        # Calculate user coverage
        user_coverage_rate = num_users_with_recommendations / len(df.index)

        return user_coverage_rate

    # Assume df is your user-item scoring matrix and numUsers is the number of similar users you wish
    # Call the function to calculate the user coverage
    coverage_rate = calculate_user_coverage(uiMatrix_normalized, 2)

    print(f"User coverage is: {coverage_rate:.2%}")

User coverage is: 100.00%
```

User coverage is : **100%**

Evaluation: User-based CF

2.Catalogue Coverage

Considering that the user-item matrix is large, we chose to redesign the function to calculate catalogue coverage.

In detail, the function iterates over all users, using the previously defined `userCF_prediction_simple02` function to generate a recommendation list for the current user, which includes the top `numItems` items that the user is likely to be interested in. The recommended items from all users are collected into a set named `recommended_items`.

Finally, the function calculates the ratio of the total number of unique recommended items to the total number of items, yielding the item coverage rate.

Evaluation: User-based CF

2.Catalogue Coverage

```
[ ] def calculate_item_coverage(df, numUsers, numItems):
        # Stores a collection of all recommended products
        recommended_items = set()

        # Iterate over all users
        for currentUser in df.index:
            # Generate recommendations for the current user
            topK = userCF_prediction_simple02(df, currentUser, numUsers, numItems)

            # Add recommended products to the collection
            recommended_items.update(topK.index)

        # Calculate product recommendation coverage
        item_coverage_rate = len(recommended_items) / df.shape[1]

        return item_coverage_rate


    # Call the function to calculate the product recommendation coverage
    coverage_rate = calculate_item_coverage(uiMatrix_normalized, 2, 5)

    print(f"Catalogue recommendation coverage is: {coverage_rate:.2%}")
```

```
/usr/local/lib/python3.10/dist-packages/numpy/lib/function_base.py:2889: RuntimeWarning:
  c = cov(x, y, rowvar, dtype=dtype)
/usr/local/lib/python3.10/dist-packages/numpy/lib/function_base.py:... RuntimeWarning:
  c *= np.true_divide(1, fact)
Catalogue recommendation coverage is: 95.34%
```

Catalogue coverage is : **95.34%**

ZHOU YIFAN

Evaluation: Item-based CF

1.User Coverage

- Count the number of users who received recommendations.
- Iterate over all users and generate recommendations for the current user.
- If the user received at least one recommendation, increment the counter.
- Calculate user coverage as the ratio of users who received recommendations to the total number of users and call the function to calculate the user coverage.

```
def calculate_user_coverage(df, iiSimMatrix, numItems):
    # Count the number of users who received recommendations
    users_with_recommendations = 0

    # Iterate over all users
    for currentUser in df.index:
        # Generate recommendations for the current user
        topK = itemCF_recommender(df, iiSimMatrix, currentUser, numItems)

        # If the user received at least one recommendation, increment the counter
        if topK:  # Assuming topK is a non-empty list of recommendations
            users_with_recommendations += 1

    # Calculate user coverage as the ratio of users who received recommendations to the total number of users
    user_coverage_rate = users_with_recommendations / len(df.index)

    return user_coverage_rate

# Call the function to calculate the user coverage
user_coverage_rate = calculate_user_coverage(uiMatrix, iiSimMatrix, 3)

print(f"User recommendation coverage is: {user_coverage_rate:.2%}")
```

```
User recommendation coverage is: 100.00%
```

User coverage is : **100%**

ZHOU YIFAN

Evaluation: Item-based CF

2.Catalogue Coverage

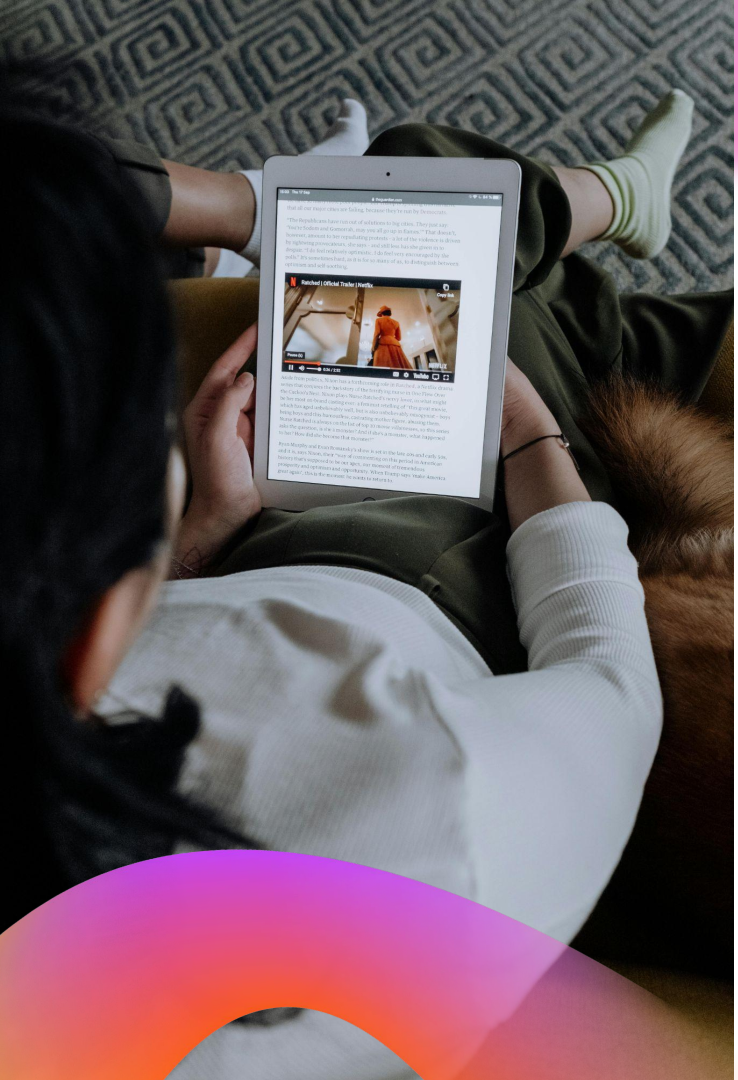
For the catalogue coverage, 74.86% means that not all the news items are enrolled in the recommendation process. It might be the reason that there are some users that have unpopular reading preferences, and these item will not be recommended to other users.

```
[ ] def calculate_catalogue_coverage(df, iiSimMatrix, numItems):
        # Stores a collection of all recommended products
        recommended_items = set()

        # Iterate over all users
        for currentUser in df.index:
            # Generate recommendations for the current user
            topK = itemCF_recommender(df, iiSimMatrix, currentUser, numItems)

            # Add recommended products to the collection if they are not already in the set
            for item, _ in topK:   # Assuming topK returns a list of (item, score) tuples
                recommended_items.add(item)

        # Calculate product recommendation coverage
        total_unique_items = len(iiSimMatrix.columns)   # Assumes iiSimMatrix has all items as columns
        item_coverage_rate = len(recommended_items) / total_unique_items

        return item_coverage_rate

    # Call the function to calculate the product recommendation coverage
    coverage_rate = calculate_catalogue_coverage(uiMatrixNorm, iiSimMatrix, 3)

    print(f"Catalogue recommendation coverage is: {coverage_rate:.2%}")
```

```
Catalogue recommendation coverage is: 74.86%
```

Catalogue coverage is : **74.86%**

ZHOU YIFAN



05

Tuning the RS

Tuning 1: User-based CF

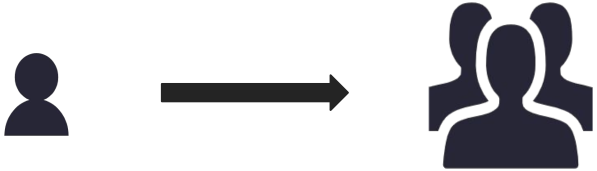
Method: using <u>Average Similarity of the results</u> to evaluate the recommend quality.

```
1 result = userCF_prediction(uiMatrix, 2786, 2, 5)
2 print(result)
3 print('Avg:',result.mean())
```

Original K-neighbors and K-items

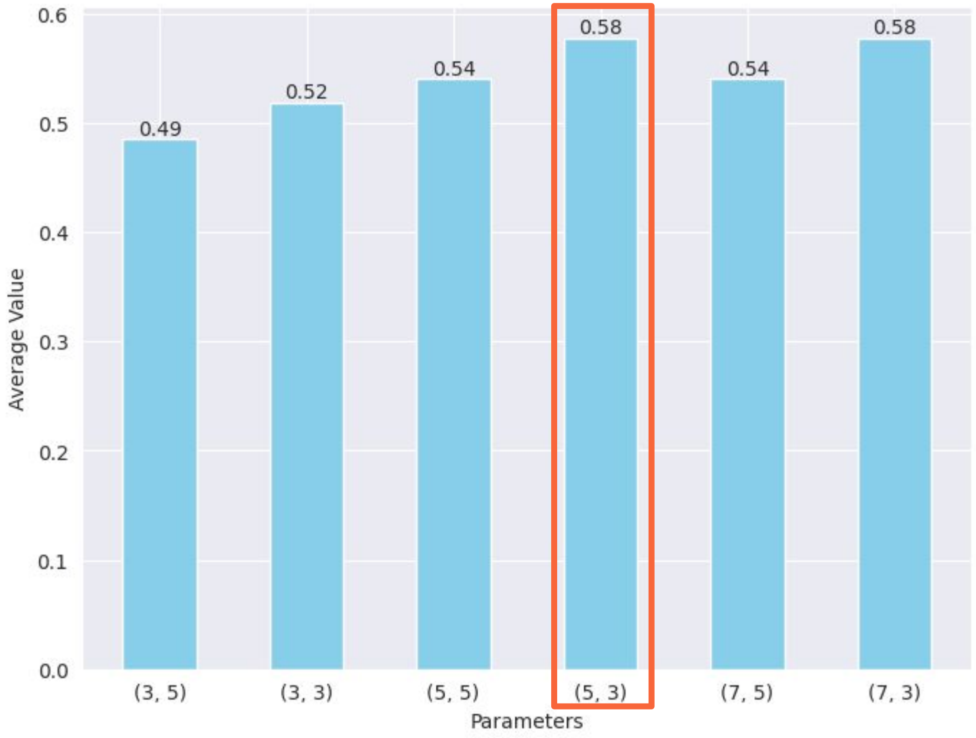
```
1562     0.589744
1351     0.456853
1925     0.434783
268      0.432570
207      0.423940
dtype: float64
Avg:  0.46757782289961136
```

Average Similarity of the results



Take more similar users into account

Tuning 1: User-based CF

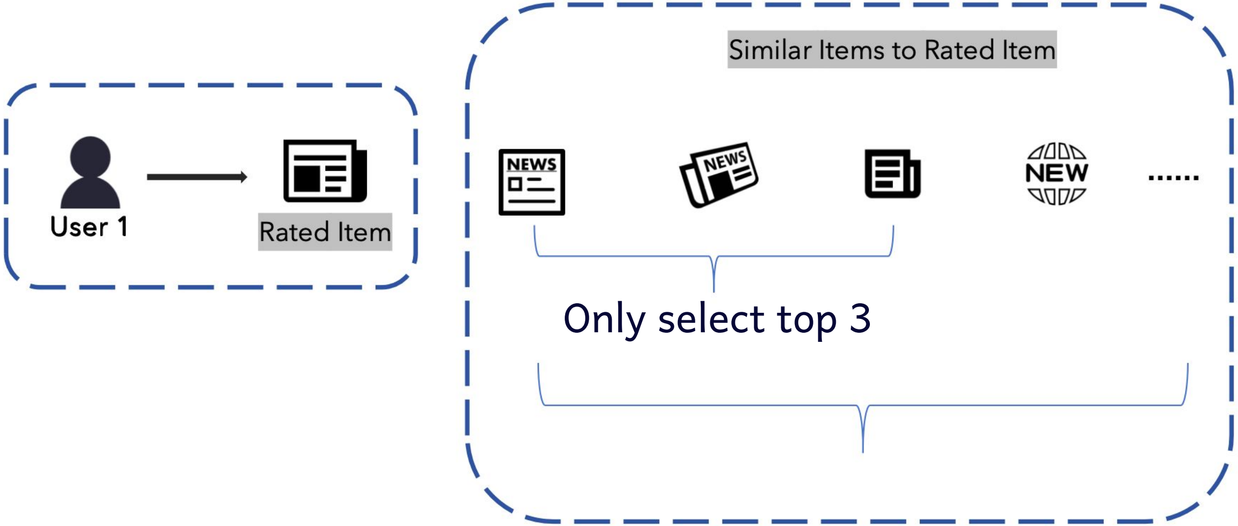


Set different Parameters:
K-neighbours= 3/ 5/ 7
K-items=3/ 5

Eventually, choosing **K-neighbors= 5** and **K-items=3**, and the Average Similarity increased from **0.47** to **0.58** .

QIAN CHENGFENG

Tuning 2: Item-based CF



Enlarge the number of similar contents
Narrow down the number of Recommendations

Tuning 2: Item-based CF

Method: using Average Similarity of the results to evaluate the recommend quality.

```
# Sort the items in the row by similarity and select the top 3
    top_3_items = item_row[rated_items].nlargest(3)
    user_ratings = uiMatrixCurrentUser.loc[:, top_3_items.index]
    weighted_ratings = top_3_items * user_ratings.loc[currentUser]
    sum_weighted_ratings = weighted_ratings.sum()
    sum_weighted_ratings
```

Only select top 3 items

```
result = itemCF_recommender(uiMatrixNorm,iiSimMatrix,2786,3)
```

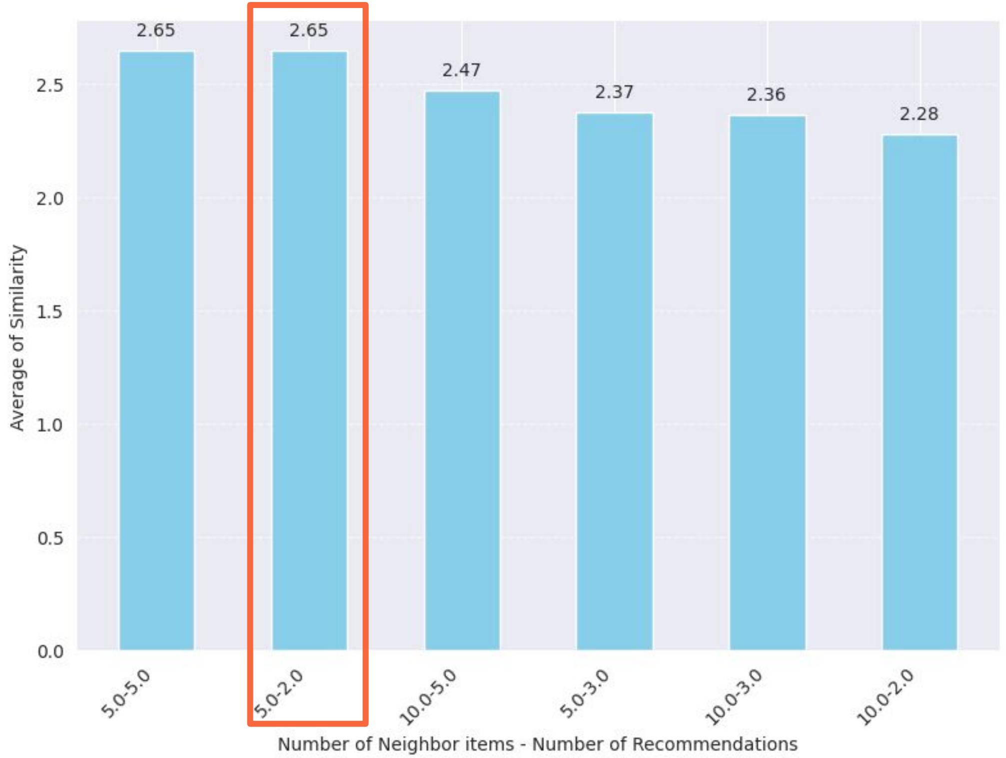
Return 3 results

```
[(992, 2.0879634577490864),
 (292, 1.7142978002002547),
 (1881, 1.1132124199750673)]
 Avg: 1.6384912259748028
```

Average Similarity of the results

QIAN CHENGFENG

Tuning 2: Item-based CF



Set different Parameters:

The number of similar contents = 5/ 10
The number of results = 2/ 3/ 5

The best combination is when we select top 5 similar items and return only 2 most relevant contents, the Average Similarity increased from **1.64** to 2.65.

QIAN CHENGFENG

Workload

	Introduction	User-based CF	Item-based CF	Evaluation	Tuning
LIU JIAXING		√			
ZHOU YIFAN				√	
LUO FAN			√		
QIAN CHENGFENG					√
ZHANG ZHENGYI	√				