Massive Data Analytics

Home Work 3

Ilia Hashemi Rad

99102456

Twitter Recommendation Systems

Collaborative filtering

Explanations:

Collaborative filtering is a technique that can filter out items that a user might like on the basis of reactions by similar users. It works by searching a large group of people and finding a smaller set of users with tastes similar to a particular user. It looks at the items they like and combines them to create a ranked list of suggestions.

A twitter recommendation system can use collaborative filtering to recommend tweets, users, or hashtags to a user based on their previous interactions (such as likes, retweets, follows, etc.) and the interactions of other similar users. For example, if a user A likes tweets from user B and user C, and user B also likes tweets from user D, then the system can recommend user D to user A.

There are two main types of collaborative filtering algorithms: memory-based and model-based:

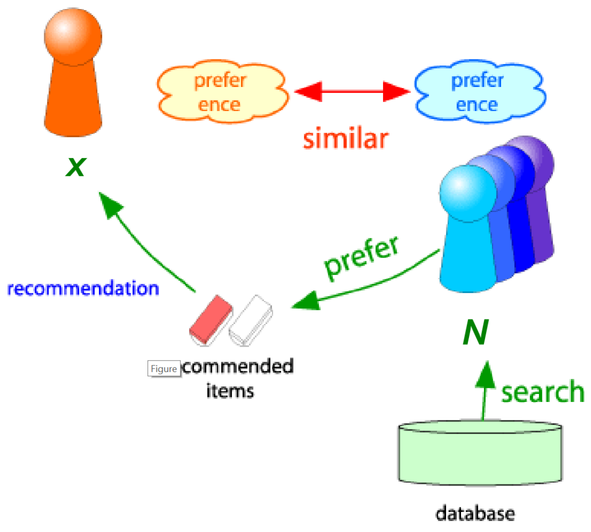
1. **Memory-based** : Memory-based algorithms use the entire feedback matrix to find similarities between users or items and make predictions based on the ratings of the most similar ones.
2. **Model-based** : Model-based algorithms use the feedback matrix to learn a lower-dimensional representation of users and items, called embeddings, and make predictions based on the embeddings.

Comparison:

- Memory-based algorithms can be further divided into user-based and item-based collaborative filtering. User-based collaborative filtering finds users who are similar to the target user and recommends items that they have rated highly. Item-based collaborative filtering finds items that are similar to the items that the target user has rated highly and recommends them.
- Model-based algorithms can use various techniques to learn the embeddings, such as matrix factorization, neural networks, or clustering. Matrix factorization is one of the most popular methods, as it can capture the latent factors that influence the user preferences and item characteristics. For example, a matrix factorization model can learn that some users prefer comedy movies and some movies are more comedic than others, and use these factors to make recommendations.

One of the challenges of collaborative filtering is dealing with sparse and imbalanced data. Since most users only interact with a small fraction of items, the feedback matrix is mostly empty and it is hard to find similar users or items. Moreover, some items or users may have much more interactions than others, which can bias the recommendations. To overcome these issues, some techniques that can be used are:

- Data augmentation: adding more data from other sources, such as user profiles, item descriptions, or social networks, to enrich the feedback matrix and provide more information for the algorithms.
- Data normalization: applying some transformations to the feedback matrix, such as mean-centering, standardization, or binarization, to reduce the impact of outliers and different rating scales.
- Data sampling: selecting a subset of the feedback matrix, such as the most recent or the most relevant interactions, to reduce the sparsity and noise of the data.
- Hybrid methods: combining collaborative filtering with other techniques, such as content-based filtering or demographic filtering, to leverage the strengths of different approaches and provide more diverse and accurate recommendations.



Content-based Recommendation

Explanations:

Content-based recommendation is a technique that can recommend items to a user based on the content or features of the items and the user's preferences or profile. It works by analyzing the textual or visual content of the items, such as tweets, hashtags, images, or videos, and extracting relevant keywords, topics, or concepts. It also analyzes the user's profile, such as their bio, tweets, likes, or follows, and infers their interests or preferences. It then matches the items that are most similar or relevant to the user's profile and recommends them to the user.

A twitter recommendation system can use content-based recommendation to recommend tweets, users, or hashtags to a user based on their content. For example, if a user likes tweets about sports, the system can recommend tweets that contain sports-related keywords, hashtags, or images. Similarly, if a user follows users who tweet about politics, the system can recommend users who have similar political views or opinions.

There are two main steps in content-based recommendation algorithms: content analysis and recommendation generation. Content analysis is the process of extracting features or representations from the items and the user profile. Recommendation generation is the process of computing the similarity or relevance between the items and the user profile and ranking the items accordingly.

Content analysis can use various techniques to extract features or representations from the items and the user profile, such as:

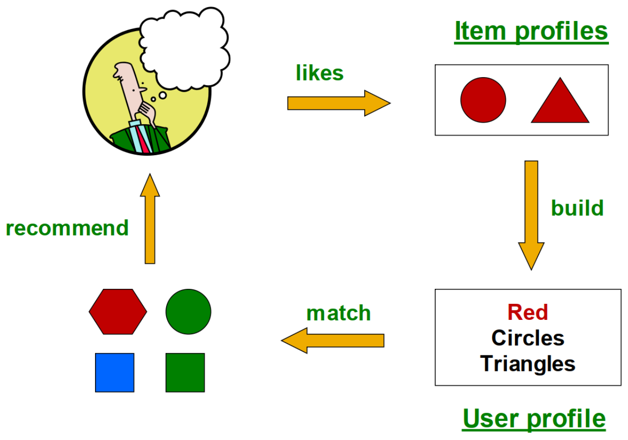
- **Text analysis**: applying natural language processing techniques, such as tokenization, stemming, stopword removal, term frequency-inverse document frequency (TF-IDF), topic modeling, or sentiment analysis, to extract keywords, topics, or sentiments from the text content of the items or the user profile.
- **Image analysis**: applying computer vision techniques, such as face detection, object recognition, scene classification, or image captioning, to extract entities, attributes, or concepts from the image content of the items or the user profile.
- **Video analysis**: applying video processing techniques, such as shot segmentation, keyframe extraction, video summarization, or video captioning, to extract features or representations from the video content of the items or the user profile.

Recommendation generation can use various techniques to compute the similarity or relevance between the items and the user profile, such as:

- Cosine similarity: measuring the angle between the feature vectors of the items and the user profile and selecting the items that have the smallest angle or the highest cosine value.
- Euclidean distance: measuring the distance between the feature vectors of the items and the user profile and selecting the items that have the smallest distance or the lowest euclidean value.

- Jaccard similarity: measuring the overlap between the feature sets of the items and the user profile and selecting the items that have the largest overlap or the highest jaccard value.
- Pearson correlation: measuring the linear relationship between the feature vectors of the items and the user profile and selecting the items that have the strongest positive or negative correlation or the highest pearson value.

One of the challenges of content-based recommendation is dealing with limited or noisy data. Since the recommendations are based on the content of the items and the user profile, the quality and quantity of the data can affect the accuracy and diversity of the recommendations. To overcome these issues, some techniques that can be used are like the collaborative's ones. So, I don't refer them again.



Other Types of Recommendation Systems

Explanations:

Besides collaborative filtering and content-based recommendation, there are other types of recommendation systems that can be used for twitter. Some of them are:

- **Demographic filtering**: This type of recommendation system uses the demographic information of the users, such as age, gender, location, language, etc., to segment them into different groups and recommend items that are popular or relevant for each group.
- **Social filtering**: This type of recommendation system uses the social network of the users, such as their friends, followers, or influencers, to recommend items that are endorsed or shared by their connections.
- **Context-aware filtering**: This type of recommendation system uses the contextual information of the users, such as their location, time, device, mood, etc., to recommend items that are suitable or personalized for each situation.
- **Knowledge-based filtering**: This type of recommendation system uses the domain knowledge of the items, such as their features, attributes, or categories, to recommend items that match the user's needs or preferences.
- **Alternating Least Squares (ALS)**: This type of recommendation system works by alternating between optimizing two matrices, X and Y, that approximate the ratings matrix R, such that $XY^T = R$. It tries to find the matrices X and Y that have low rank and minimize the error between the observed and predicted ratings.

These types of recommendation systems can be combined with each other or with collaborative filtering or content-based recommendation to create hybrid systems that can improve the accuracy, diversity, and coverage of the recommendations.

My Approach:

Chosen Algorithm

Explanations:

In our case, with the second version of twitter data, according to the content of the dataset, I chose **Memory-based Collaborative filtering** as the best of approaches explained above and in the slides. Also, to overcome its challenges that were explained, I used Pearson Correlation to find similar users, which is considered a kind of normalization for data. On the other hand, the data is crawled in a user-tracker way, which makes it more probable to find similar users and items based on collaborative filtering.

More Explanations

Explanations:

The collaborative filtering algorithm presented in my provided code aims to deliver personalized tweet recommendations for a target user within a large dataset. The algorithm processes a vast collection of tweets, filtering out irrelevant ones and extracting essential information such as user IDs, tweet types, and tweet IDs. Ratings are assigned to each tweet based on its type: 1 for replies, 2 for retweets, and 3 for quoted tweets. I substantiated these ratings by surveying a few people who used Twitter a lot. that a person's reply to a tweet shows the importance and interest of that person in that tweet. But if he retweets it, it means that it is more attractive and important to him, and if he quotes it, it means that the tweet is the most important and attractive for the person, and it should have more weight than others for recommending tweets to the person.

Subsequently, the algorithm constructs a user-tweet matrix, where each user is associated with a dictionary containing tweet IDs and their corresponding ratings. This matrix serves as the foundation for computing Pearson correlation coefficients between users. The correlation coefficients quantify the similarity between users based on their shared ratings for common tweets.

To identify similar users for a given target user, the algorithm calculates Pearson correlation coefficients with all other users and selects the top N similar users. Leveraging these similarities, the algorithm predicts missing ratings for the tweets that the target user has not rated. This prediction process is grounded in the ratings and similarities obtained from similar users.

The final step involves recommending the top tweets to the target user based on the predicted ratings. The algorithm outputs a list of these top recommended tweets, sorted by their predicted ratings in descending order. The result is a personalized recommendation list that reflects the target user's preferences, derived collaboratively from the behavior of similar users in the dataset.

Section 1: Preprocessing and Exploring data

```python
tweets_rdd = sc.textFile("twitter_data_v2.jsonl")

import json

def create_tree(tweet_data):
    tree = {}

    # Extracting top-level fields
    tree["id"] = tweet_data["id"]
    tree["user"] = {
        "id": tweet_data["user"]["id"],
        "screen_name": tweet_data["user"]["screen_name"],
        "name": tweet_data["user"]["name"],
        "description": tweet_data["user"]["description"]
    }
    tree["tweet_type"] = tweet_data["tweet_type"]
    tree["in_reply_to_status_id_str"] = tweet_data["in_reply_to_status_id_str"]
    tree["in_reply_to_user_id_str"] = tweet_data["in_reply_to_user_id_str"]

    # Quoted status
    quoted_status = tweet_data.get("quoted_status")
    if quoted_status:
        tree["quoted_status"] = {
            "id": quoted_status["id"],
        }

    # Retweeted status
    retweeted_status = tweet_data.get("retweeted_status")
    if retweeted_status:
        tree["retweeted_status"] = {
            "id": retweeted_status["id"],
            "created_at": retweeted_status["created_at"],
            "user": {
                "id": retweeted_status["user"]["id"],
                "screen_name": retweeted_status["user"]["screen_name"],
                "name": retweeted_status["user"]["name"],
                "description": retweeted_status["user"]["description"]
            }
        }

    return tree

# Example for retweeted data
sample_data = json.loads(tweets_rdd.take(10)[0])
tweet_tree = create_tree(sample_data)
print(json.dumps(tweet_tree, indent=2, ensure_ascii=False))
```

Explanations:

Here, in this code, I Preprocessed the data to filter the relevant parts of data which was indicated in the explanations of assignment, and showed that in a hierarchical or tree like shape. That was done for all the 4 types of tweet to get more familiar with them and know how to determine ratings of each user for an specific tweet. Here, is a sample output of that, for each type:

Retweeted Tweets

```
{
  "id": "1736536753815834746",
  "user": {
    "id": "2808957837",
    "screen_name": "maryaa_maryam",
    "name": "maryam jafari",
    "description": ""
  },
  "tweet_type": "retweeted",
  "in_reply_to_status_id_str": null,
  "in_reply_to_user_id_str": null,
  "retweeted_status": {
    "id": "1736379038376120479",
    "created_at": 1702819997,
    "user": {
      "id": "2960042892",
      "screen_name": "Ivar_lathbrug2",
      "name": "آیوار ۲",
      "description": "باورمند به نظام پادشاهی مشروطه با @PahlaviReza #جاویدشاه @PhoenixprjIran     https://t.co/kjMxf53cIQ"
    }
  }
}
```

Replied Tweets

```
{
  "id": "1736536759964512322",
  "user": {
    "id": "1406742149048287242",
    "screen_name": "sun_saeid",
    "name": "(تلخ) حق نامه",
    "description": "از زندگی بگیم ولی واقعیتشو"
  },
  "tweet_type": "replied",
  "in_reply_to_status_id_str": "1736356968162378084",
  "in_reply_to_user_id_str": "1643210821068046338"
}
```

Generated Tweets

```
{
  "id": "1736536756835512564",
  "user": {
    "id": "22495731",
    "screen_name": "AlArabiya_Fa",
    "name": "العربیه فارسی",
    "description": "«العربیه فارسی» به عنوان بخشی از شبکه «العربیه» در سال 2008 راه‌اندازی شد. \nالعربیه فارسی ایمیل تحریریه العربیه فارسی:\neditor.farsi@alarabiya.net\nhttp
  },
  "tweet_type": "generated",
  "in_reply_to_status_id_str": null,
  "in_reply_to_user_id_str": null
}
```

Quoted Tweets

```
{
  "id": "1736536759943450888",
  "user": {
    "id": "1655373597001523200",
    "screen_name": "ScarFace2582",
    "name": "SCARFACE ♔",
    "description": "♧\nایران_معبد_ماست #پاینده_ایران #جاویدشاه#\n♧"
  },
  "tweet_type": "quoted",
  "in_reply_to_status_id_str": null,
  "in_reply_to_user_id_str": null,
  "quoted_status": {
    "id": "1736113075911077948"
  },
  "retweeted_status": {
    "id": "1736536140063109215",
    "created_at": 1702857453,
    "user": {
      "id": "893757772025798656",
      "screen_name": "YaarDabestaani",
      "name": "ـ۞Yaar - یار دبستانی 🏳️‍🌈",
      "description": "Human rights activist, 🌈LGBTQ🏳️‍🌈, animal lover🐈🐕⬛🐒, Atheist, 🧕=👦💼=👱‍♀️=🧑🏿💼 #MahsaAmini,  My inner child: @bri30s"
    }
  }
}
```

Section 2: Determining Users' Rating for Tweets

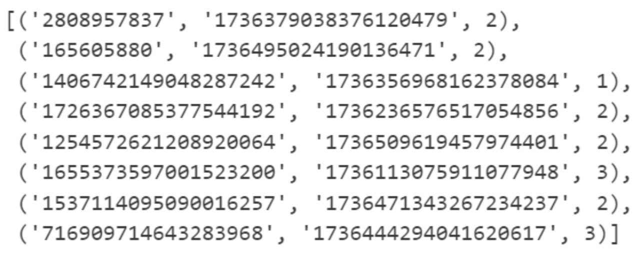
```python
    import json

tweets_rdd = sc.textFile("twitter_data_v2.jsonl")

# Parse JSON strings to Python dictionaries
tweets_json = tweets_rdd.map(json.loads)
del tweets_rdd

# Filter out generated tweets
relevant_tweets = tweets_json.filter(lambda x: x['tweet_type'] in ["retweeted", "replied", "quoted"])


# Extract tweet rates for each user
def extract_tweet_rates(tweet):
    user_id = tweet.get("user", {}).get("id", None)
    tweet_type = tweet.get("tweet_type", None)
    if tweet_type == 'replied':
        in_reply_to_status_id_str = tweet.get("in_reply_to_status_id_str", None)
        return (user_id, in_reply_to_status_id_str, 1)
    elif tweet_type == 'retweeted':
        retweeted_tweet_id = tweet.get("retweeted_status", {}).get("id", None)
        return (user_id, retweeted_tweet_id, 2)
    else:
        quoted_tweet_id = tweet.get("quoted_status", {}).get("id", None)
        return (user_id, quoted_tweet_id, 3)

user_rating = relevant_tweets.map(extract_tweet_rates)

user_rating.take(8)
```

Explanations:

Here is a brief explanation of what I did in the code:

- **Parse Twitter Data:**
  - I parsed the JSON strings to Python dictionaries using `map(json.loads)`.
- **Filter Out Generated Tweets:**
  - I filtered out tweets based on their types, keeping only "retweeted," "replied," and "quoted" tweets and filtering "Generated" ones, because we can't extract any rating from them.
- **Extract Tweet Rates for Each User:**
  - I defined a function `extract_tweet_rates` that extracts relevant information such as user ID, tweet type, and associated tweet ID.
  - Then, mapped each tweet in `relevant_tweets` using the `extract_tweet_rates` function.
  - The resulting RDD contains tuples with user ID, tweet ID, and a the rate of the user to the tweet (1 for replied, 2 for retweeted, 3 for quoted).

Here, is some samples of `user_rating` which expectedly, has the form of (user ID, tweet ID, rate):



```
[('2808957837', '1736379038376120479', 2),
 ('165605880', '1736495024190136471', 2),
 ('1406742149048287242', '1736356968162378084', 1),
 ('1726367085377544192', '1736236576517054856', 2),
 ('1254572621208920064', '1736509619457974401', 2),
 ('1655373597001523200', '1736113075911077948', 3),
 ('1537114095090016257', '1736471343267234237', 2),
 ('716909714643283968', '1736444294041620617', 3)]
```

Section 3: Creating User Tweet Matrix

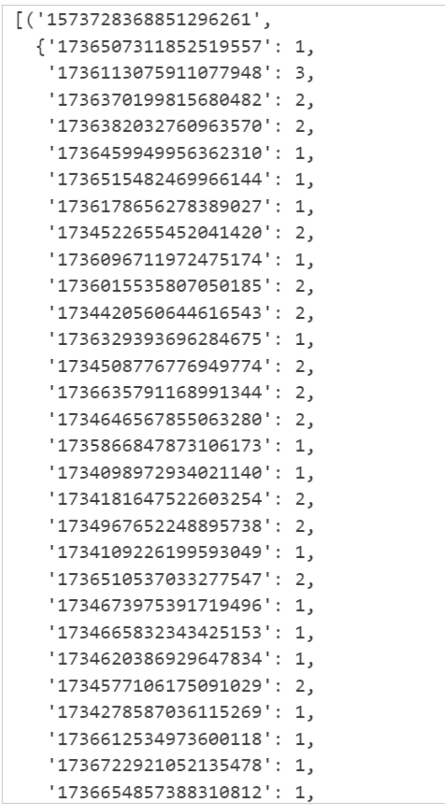
```python
# Function to create a matrix of users-tweets and their respective ratings
def create_matrix(rdd):
    user_tweet_matrix = rdd.map(lambda x: (x[0], [(x[1], x[2])])) \
        .reduceByKey(lambda x, y: x + y) \
        .map(lambda x: (x[0], dict(x[1])))
    return user_tweet_matrix

user_tweet_matrix = create_matrix(user_rating)

user_tweet_matrix.take(5)[2:4]
```

Explanations:

Here is a brief explanation of what I did in the code:

- **Create Matrix of Users-Tweets and Ratings:**
  - I defined a function `create_matrix` that transforms the RDD `user_rating` into a matrix-like structure.
  - Used `map` to transform each tuple in `user_rating` into a key-value pair with the user ID as the key and a list of tuples (tweet ID, rating) as the value.
  - Applied `reduceByKey` to group the lists of tuples by user ID.
  - Used `map` again to convert each user's list of tuples into a dictionary, where tweet IDs are keys and ratings are values.
  - The resulting `user_tweet_matrix` RDD contains tuples with user ID and a dictionary of associated tweet IDs and ratings.

Here, is one shot of some samples of `user_tweet_matrix` which expectedly, has the form of (user ID, {tweet ID 1: rate 1, tweet ID 2: rate 2, ...}): (complete output can be seen in the Jupyter Notebook)



```
[('1573728368851296261',
  {'1736507311852519557': 1,
   '1736113075911077948': 3,
   '1736370199815680482': 2,
   '1736382032760963570': 2,
   '1736459949956362310': 1,
   '1736515482469966144': 1,
   '1736178656278389027': 1,
   '1734522655452041420': 2,
   '1736096711972475174': 1,
   '1736015535807050185': 2,
   '1734420560644616543': 2,
   '1736329393696284675': 1,
   '1734508776776949774': 2,
   '1736635791168991344': 2,
   '1734646567855063280': 2,
   '1735866847873106173': 1,
   '1734098972934021140': 1,
   '1734181647522603254': 2,
   '1734967652248895738': 2,
   '1734109226199593049': 1,
   '1736510537033277547': 2,
   '1734673975391719496': 1,
   '1734665832343425153': 1,
   '1734620386929647834': 1,
   '1734577106175091029': 2,
   '1734278587036115269': 1,
   '1736612534973600118': 1,
   '1736722921052135478': 1,
   '1736654857388310812': 1,
```

Section 4: Choosing One target User And Showing his Rated Tweets

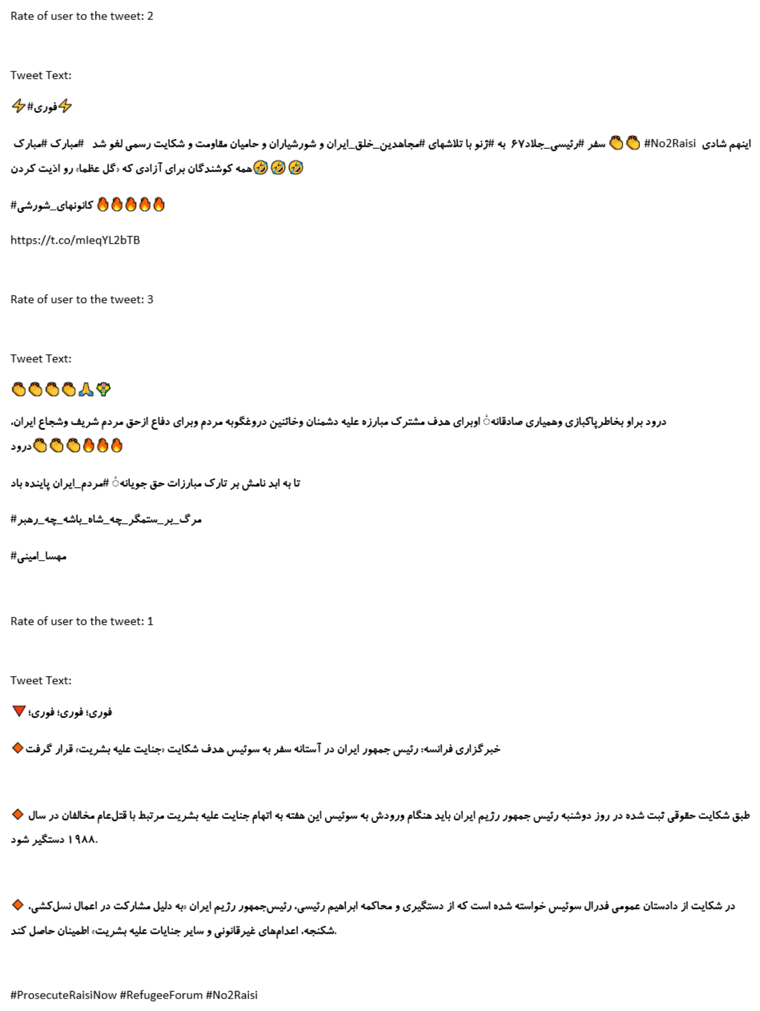
```
# Choose a target user to suggest tweets
target_user_id = '165605880'

# Determine ratings of the target user (corresponding column in the matrix for user)
target_user_ratings = user_tweet_matrix.lookup(target_user_id)[0]

# Create the list of tweets which the user rated as (Tweet ID, Tweet text)
user_rated_tweets = tweets_json.filter(lambda x: x['id'] in list(target_user_ratings.keys())).map(lambda x: (x['id'], x['text'])).c

# Show some samples of user's favorite tweets
for tweet in user_rated_tweets[:5] + user_rated_tweets[6:8]:
    print(f'Rate of user to the tweet: {target_user_ratings[tweet[0]]}\n')
    print('Tweet Text:')
    print(tweet[1])
    print('\n\n')
```

Explanations:

Here is a brief explanation of what I did in the code:

- **Choose a Target User:**
  - I specified a random target user ID for whom tweet suggestions will be generated.
- **Determine Ratings of the Target User:**
  - I used `lookup` on `user_tweet_matrix` to retrieve the ratings (a dictionary of tweet IDs and ratings) of the target user.
  - Assigned the result to `target_user_ratings`.
- **Create List of Rated Tweets for the Target User:**
  - I filtered `tweets_json` to keep only those tweets with IDs found in the target user's ratings.
  - Used `map` to create a list of tuples containing tweet ID and tweet text for each rated tweet.
  - Assigned the result to `user_rated_tweets`.

Here, is one shot of some samples of the target user's favorite tweets: (complete output can be seen in the Jupyter Notebook)



Rate of user to the tweet: 2

Tweet Text:

⚡#فوری⚡

اینهم شادی #No2Raisi 👏👏 سفر #رئیسی_جلاد۶۷ به #ژنو با تلاشهای #مجاهدین_خلق_ایران و شورشیاران و حامیان مقاومت و شکایت رسمی لغو شد #مبارک #مبارک 🤣🤣🤣همه کوشندگان برای آزادی که «گل عظما» رو اذیت کردن

🔥🔥🔥🔥🔥 کانونهای_شورشی#

https://t.co/mIeqYL2bTB

Rate of user to the tweet: 3

Tweet Text:

👏👏👏👏🙏💐

درود براو بخاطرپاکبازی وهمیاری صادقانه‌ٔ اوبرای هدف مشترک مبارزه علیه دشمنان وخائنین دروغگوبه مردم وبرای دفاع ازحق مردم شریف وشجاع ایران،
🔥🔥🔥👏👏👏درود

تا به ابد نامش بر تارک مبارزات حق جویانه‌ٔ #مردم_ایران پاینده باد

مرگ_بر_ستمگر_چه_شاه_باشه_چه_رهبر#

مهسا_امینی#

Rate of user to the tweet: 1

Tweet Text:

🔻فوری؛ فوری؛ فوری؛

🔸خبرگزاری فرانسه: رئیس جمهور ایران در آستانه سفر به سوئیس هدف شکایت «جنایت علیه بشریت» قرار گرفت

🔸 طبق شکایت حقوقی ثبت شده در روز دوشنبه رئیس جمهور رژیم ایران باید هنگام ورودش به سوئیس این هفته به اتهام جنایت علیه بشریت مرتبط با قتل‌عام مخالفان در سال ۱۹۸۸ دستگیر شود.

🔸 در شکایت از دادستان عمومی فدرال سوئیس خواسته شده است که از دستگیری و محاکمه ابراهیم رئیسی، رئیس‌جمهور رژیم ایران «به دلیل مشارکت در اعمال نسل‌کشی، شکنجه، اعدام‌های غیرقانونی و سایر جنایات علیه بشریت» اطمینان حاصل کند.

#ProsecuteRaisiNow #RefugeeForum #No2Raisi

Explanations:

As you can see, the user mostly likes the political tweets and has a specific politic side which can be seen from his attractive tweets. Keep that in mind, to consist the recommended tweets with these and see the similarity.

Section 5: Finding Similar Users

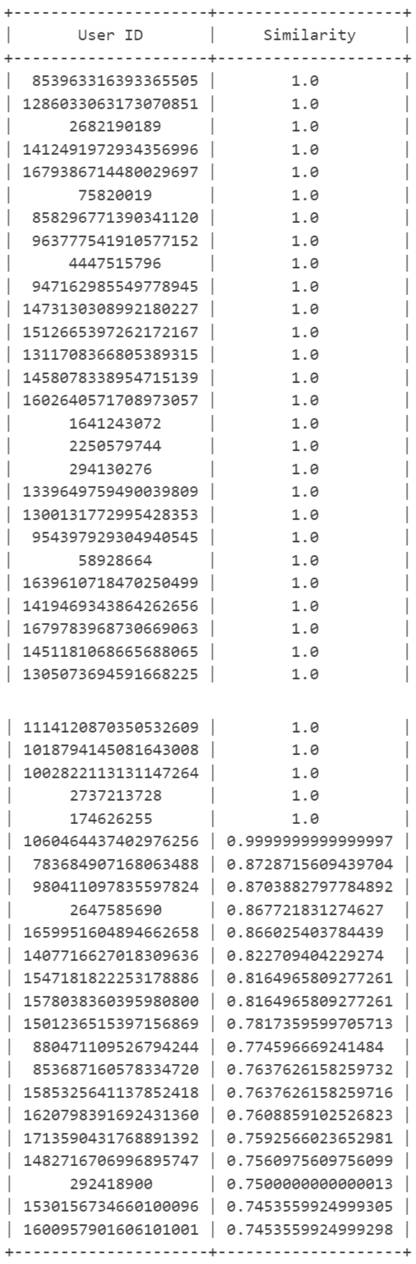
```python
from prettytable import PrettyTable

# Function to calculate Pearson correlation coefficient
def pearson_corr(user1_ratings, user2_ratings):
    common_tweets = set(user1_ratings.keys()) & set(user2_ratings.keys())
    if not common_tweets:
        return 0.0

    sum_xy = sum(user1_ratings[tweet] * user2_ratings[tweet] for tweet in common_tweets)
    sum_x = sum(user1_ratings[tweet] for tweet in common_tweets)
    sum_y = sum(user2_ratings[tweet] for tweet in common_tweets)

    sum_x_squared = sum(user1_ratings[tweet] ** 2 for tweet in common_tweets)
    sum_y_squared = sum(user2_ratings[tweet] ** 2 for tweet in common_tweets)

    n = len(common_tweets)

    numerator = sum_xy - (sum_x * sum_y / n)
    denominator = ((sum_x_squared - (sum_x ** 2 / n)) * (sum_y_squared - (sum_y ** 2 / n))) ** 0.5

    if denominator == 0:
        return 0.0

    return numerator / denominator

# Function to find similar users based on Pearson correlation
def find_similar_users(user_id, user_tweet_matrix, NumSim):
    target_user_ratings = user_tweet_matrix.lookup(user_id)[0]
    similarities = user_tweet_matrix.filter(lambda x: x[0] != user_id) \
        .map(lambda x: (x[0], pearson_corr(target_user_ratings, x[1])))

    # Get top k similar users
    similar_users = similarities.takeOrdered(NumSim, key=lambda x: -x[1])

    return similar_users

# The target user
target_user_id = '165605880'

# Number of top similar users which we're gonna find
NumberOfSimilarUsrs = 50

# Find similar users
similar_users = find_similar_users(target_user_id, user_tweet_matrix, NumberOfSimilarUsrs)

table = PrettyTable()
table.field_names = ["User ID", "Similarity"]

for user, similarity in similar_users:
    table.add_row([user, similarity])

print(table)
```

Explanations:

Here is a brief explanation of what I did in the code:

- **Pearson Correlation Coefficient Calculation:**
  - I defined the `pearson_corr` function to calculate the Pearson correlation coefficient between two users based on their tweet ratings.
  - Used common tweets for the calculation and handle cases where the denominator is zero.
- **Find Similar Users Function:**
  - I defined the `find_similar_users` function to find similar users to a target user based on Pearson correlation.
  - Looked up the target user's ratings from `user_tweet_matrix`.
  - Calculated Pearson correlation with other users and get the top $NumSim$ similar users.
- **Find Similar Users:**
  - I used `find_similar_users` to get a list of similar users.
  - Created a `PrettyTable` and populate it with the user IDs and their similarities.
  - Finally, printed the resulting table.

Here, is 50 top similar users to the target user:



```
+---------------------+--------------------+
|       User ID       |     Similarity     |
+---------------------+--------------------+
|  853963316393365505 |        1.0         |
| 1286033063173070851 |        1.0         |
|      2682190189     |        1.0         |
| 1412491972934356996 |        1.0         |
| 1679386714480029697 |        1.0         |
|       75820019      |        1.0         |
|  858296771390341120 |        1.0         |
|  963777541910577152 |        1.0         |
|      4447515796     |        1.0         |
|  947162985549778945 |        1.0         |
| 1473130308992180227 |        1.0         |
| 1512665397262172167 |        1.0         |
| 1311708366805389315 |        1.0         |
| 1458078338954715139 |        1.0         |
| 1602640571708973057 |        1.0         |
|      1641243072     |        1.0         |
|      2250579744     |        1.0         |
|      294130276      |        1.0         |
| 1339649759490039809 |        1.0         |
| 1300131772995428353 |        1.0         |
|  954397929304940545 |        1.0         |
|       58928664      |        1.0         |
| 1639610718470250499 |        1.0         |
| 1419469343864262656 |        1.0         |
| 1679783968730669063 |        1.0         |
| 1451181068665688065 |        1.0         |
| 1305073694591668225 |        1.0         |

| 1114120870350532609 |        1.0         |
| 1018794145081643008 |        1.0         |
| 1002822113131147264 |        1.0         |
|      2737213728     |        1.0         |
|      174626255      |        1.0         |
| 1060464437402976256 | 0.9999999999999997 |
|  783684907168063488 | 0.8728715609439704 |
|  980411097835597824 | 0.8703882797784892 |
|      2647585690     | 0.867721831274627  |
| 1659951604894662658 | 0.866025403784439  |
| 1407716627018309636 | 0.822709404229274  |
| 1547181822253178886 | 0.8164965809277261 |
| 1578038360395980800 | 0.8164965809277261 |
| 1501236515397156869 | 0.7817359599705713 |
|  880471109526794244 | 0.774596669241484  |
|  853687160578334720 | 0.7637626158259732 |
| 1585325641137852418 | 0.7637626158259716 |
| 1620798391692431360 | 0.7608859102526823 |
| 1713590431768891392 | 0.7592566023652981 |
| 1482716706996895747 | 0.7560975609756099 |
|      292418900      | 0.7500000000000013 |
| 1530156734660100096 | 0.7453559924999305 |
| 1600957901606101001 | 0.7453559924999298 |
+---------------------+--------------------+
```

Explanations:

As you can see, the results seem logical. Note that because of the data being crawled user-tracked, as I said before, there must be high similarity (like 1 which is caused by the same reaction of users to highly viral tweets) between users in this dataset.

Section 6: Recommend Tweets to target user based on his Similar User Preferences

```python
# Function to predict missing ratings for a given user
def predict_ratings(user_id, user_tweet_matrix, similar_users):
    target_user_ratings = user_tweet_matrix.lookup(user_id)

    if not target_user_ratings:
        return {}

    target_user_ratings = target_user_ratings[0]

    # Extract tweet IDs rated by similar users
    similar_users_ids = [similar_user for similar_user, _ in similar_users]
    similar_users_ids.append(user_id)

    # Filter the original RDD to get only the rows corresponding to similar users
    similar_tweet_matrix = user_tweet_matrix.filter(lambda x: x[0] in similar_users_ids).collect()

    # Create a dictionary for quick access to ratings of similar users
    similar_users_ratings_dict = dict(similar_tweet_matrix)

    # Find the set of all tweet IDs rated by similar users
    similar_users_tweets = set()
    for similar_user_id in similar_users_ids:
        if similar_user_id in similar_users_ratings_dict:
            similar_users_tweets.update(similar_users_ratings_dict[similar_user_id].keys())

    # Remove the tweets that the target user has already rated
    similar_users_tweets -= set(target_user_ratings.keys())

    predicted_ratings = {}

    for tweet_id in similar_users_tweets:
        numerator = 0.0
        denominator = 0.0

        for similar_user_id, similarity in similar_users:
            if tweet_id in similar_users_ratings_dict.get(similar_user_id, {}):
                numerator += similarity * similar_users_ratings_dict[similar_user_id][tweet_id]
                denominator += abs(similarity)

        if denominator != 0:
            predicted_ratings[tweet_id] = numerator / denominator
        else:
            predicted_ratings[tweet_id] = None

    return predicted_ratings


predicted_ratings = predict_ratings(target_user_id, user_tweet_matrix, similar_users)

# Print the top recommended tweets based on predicted ratings
top_recommended_tweets = sorted(predicted_ratings.items(), key=lambda x: x[1], reverse=True)

table = PrettyTable()
table.field_names = ["Tweet ID", "Predicted Rating"]

top_recom_ids = []
i = 1
for tweet_id, rating in top_recommended_tweets:
    if i in range(1,15) or i in range(115, 130) or i in range(1822, 1852):
        top_recom_ids.append(tweet_id)
    i = i+1
    table.add_row([tweet_id, rating])

print(table)
```

Explanations:

Here is a brief explanation of what I did in the code:

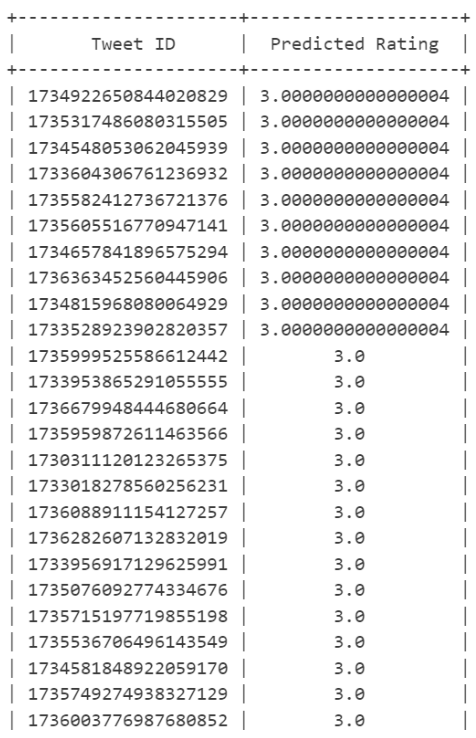
- **Predict Missing Ratings Function:**
  - I defined the `predict_ratings` function to predict missing ratings for a given user.
  - Extracted the target user's ratings from `user_tweet_matrix`.
  - Filtered the original matrix to get only similar users' ratings.
  - For each tweet not rated by the target user, calculated the predicted rating based on the ratings of similar users by this formula which was in the slides:

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} . r_{yi}}{\sum_{y \in N} s_{xy}}$$

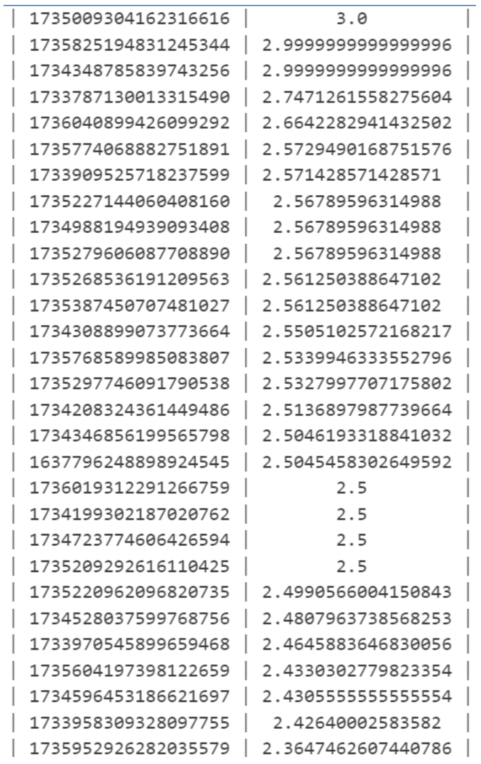
    in which, r_{xi} denotes the prediction for tweet i of user x, and s_{xy} denotes Pearson correlation coefficient between user x and y, and N is the set of k users most similar to x who have rated tweet i.
  - Created a dictionary of predicted ratings.

- **Predict Ratings for the Target User:**
  - I used `predict_ratings` to get a dictionary of predicted ratings for the target user.
- **Print Top Recommended Tweets:**
  - I sorted the predicted ratings and print the top recommended tweets based on their predicted ratings.
  - Finally, displayed some tweets with different predicted ratings.

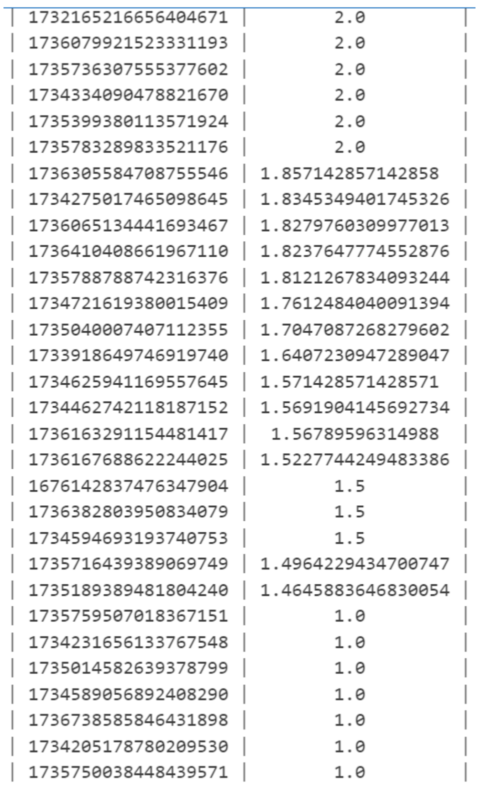
Here, is 3 shots of recommended tweets to user with the score of recommendation (actually, predicted rate of user to tweets): (complete output can be seen in the Jupyter Notebook)



```
+---------------------+--------------------+
|       Tweet ID      |  Predicted Rating  |
+---------------------+--------------------+
| 1734922650844020829 | 3.0000000000000004 |
| 1735317486080315505 | 3.0000000000000004 |
| 1734548053062045939 | 3.0000000000000004 |
| 1733604306761236932 | 3.0000000000000004 |
| 1735582412736721376 | 3.0000000000000004 |
| 1735605516770947141 | 3.0000000000000004 |
| 1734657841896575294 | 3.0000000000000004 |
| 1736363452560445906 | 3.0000000000000004 |
| 1734815968080064929 | 3.0000000000000004 |
| 1733528923902820357 | 3.0000000000000004 |
| 1735999525586612442 |        3.0         |
| 1733953865291055555 |        3.0         |
| 1736679948444680664 |        3.0         |
| 1735959872611463566 |        3.0         |
| 1730311120123265375 |        3.0         |
| 1733018278560256231 |        3.0         |
| 1736088911154127257 |        3.0         |
| 1736282607132832019 |        3.0         |
| 1733956917129625991 |        3.0         |
| 1735076092774334676 |        3.0         |
| 1735715197719855198 |        3.0         |
| 1735536706496143549 |        3.0         |
| 1734581848922059170 |        3.0         |
| 1735749274938327129 |        3.0         |
| 1736003776987680852 |        3.0         |
```



```
| 1735009304162316616 |        3.0         |
| 1735825194831245344 | 2.9999999999999996 |
| 1734348785839743256 | 2.9999999999999996 |
| 1733787130013315490 | 2.7471261558275604 |
| 1736040899426099292 | 2.6642282941432502 |
| 1735774068882751891 | 2.5729490168751576 |
| 1733909525718237599 | 2.571428571428571  |
| 1735227144060408160 |  2.56789596314988  |
| 1734988194939093408 |  2.56789596314988  |
| 1735279606087708890 |  2.56789596314988  |
| 1735268536191209563 | 2.561250388647102  |
| 1735387450707481027 | 2.561250388647102  |
| 1734308899073773664 | 2.5505102572168217 |
| 1735768589985083807 | 2.5339946333552796 |
| 1735297746091790538 | 2.5327997707175802 |
| 1734208324361449486 | 2.5136897987739664 |
| 1734346856199565798 | 2.5046193318841032 |
| 1637796248898924545 | 2.5045458302649592 |
| 1736019312291266759 |        2.5         |
| 1734199302187020762 |        2.5         |
| 1734723774606426594 |        2.5         |
| 1735209292616110425 |        2.5         |
| 1735220962096820735 | 2.4990566004150843 |
| 1734528037599768756 | 2.4807963738568253 |
| 1733970545899659468 | 2.4645883646830056 |
| 1735604197398122659 | 2.4330302779823354 |
| 1734596453186621697 | 2.4305555555555554 |
| 1733958309328097755 |  2.42640002583582  |
| 1735952926282035579 | 2.3647462607440786 |
```



```
| 1732165216656404671 |        2.0         |
| 1736079921523331193 |        2.0         |
| 1735736307555377602 |        2.0         |
| 1734334090478821670 |        2.0         |
| 1735399380113571924 |        2.0         |
| 1735783289833521176 |        2.0         |
| 1736305584708755546 | 1.857142857142858  |
| 1734275017465098645 | 1.8345349401745326 |
| 1736065134441693467 | 1.8279760309977013 |
| 1736410408661967110 | 1.8237647774552876 |
| 1735788788742316376 | 1.8121267834093244 |
| 1734721619380015409 | 1.7612484040091394 |
| 1735040007407112355 | 1.7047087268279602 |
| 1733918649746919740 | 1.6407230947289047 |
| 1734625941169557645 | 1.571428571428571  |
| 1734462742118187152 | 1.5691904145692734 |
| 1736163291154481417 |  1.56789596314988  |
| 1736167688622244025 | 1.5227744249483386 |
| 1676142837476347904 |        1.5         |
| 1736382803950834079 |        1.5         |
| 1734594693193740753 |        1.5         |
| 1735716439389069749 | 1.4964229434700747 |
| 1735189389481804240 | 1.4645883646830054 |
| 1735759507018367151 |        1.0         |
| 1734231656133767548 |        1.0         |
| 1735014582639378799 |        1.0         |
| 1734589056892408290 |        1.0         |
| 1736738585846431898 |        1.0         |
| 1734205178780209530 |        1.0         |
| 1735750038448439571 |        1.0         |
```

Explanations:

As you can see, the results seem logical. Note that because of the data being crawled user-tracked, as I said before, there must be high similarity (like 1 which is caused by the same reaction of users to highly viral tweets) between users in this dataset.

Section 7: Evaluation of Recommended Tweets

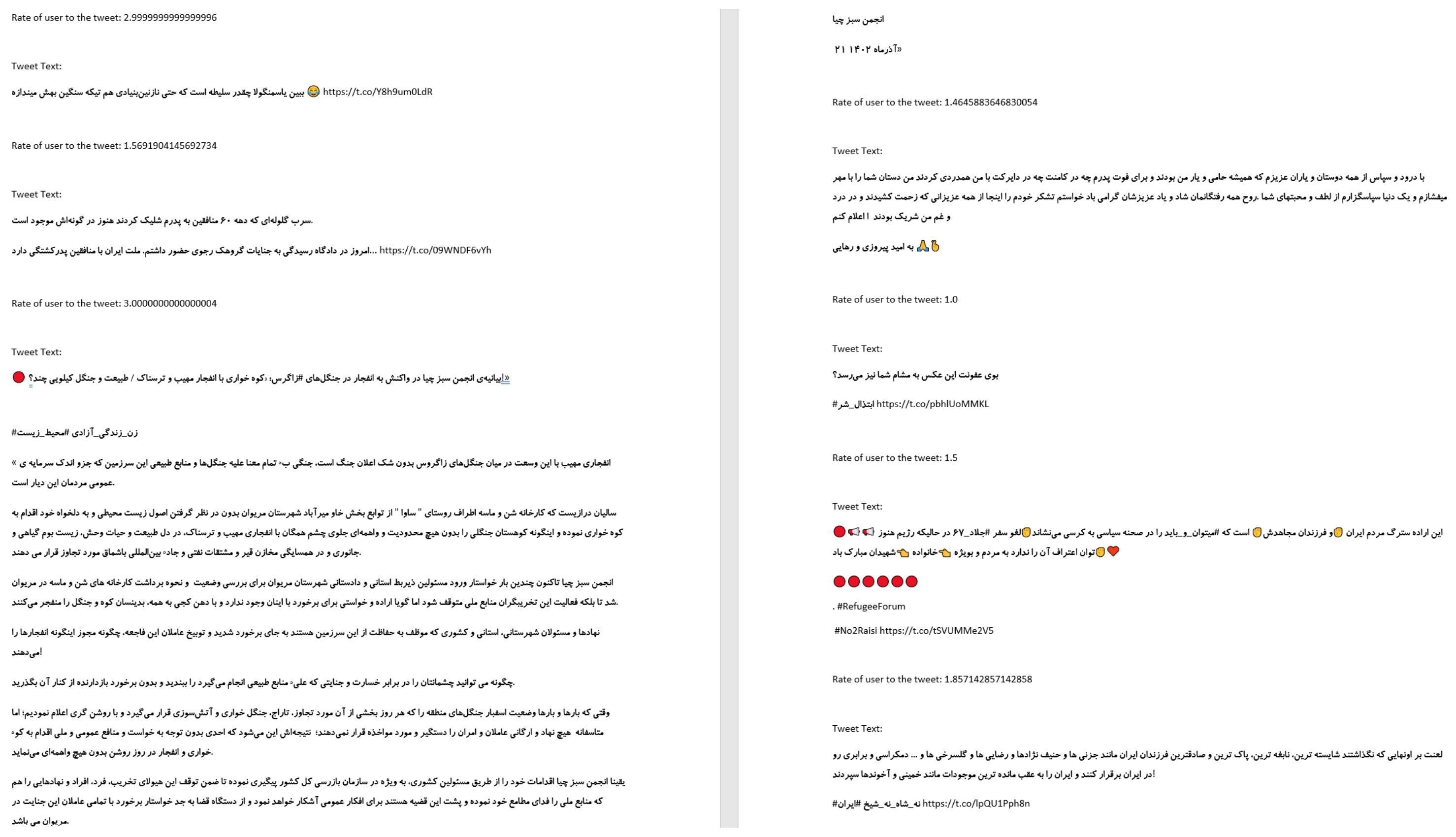
```python
# Create the list of tweets which is recommended to user (Tweet ID, Tweet text)
user_recommended_tweets = tweets_json.filter(lambda x: x['id'] in top_recom_ids).map(lambda x: (x['id'], x['text'])).collect()

top_recommended_tweets = dict(top_recommended_tweets)

# Show some recommended tweets to user and their corresponding recommendation score(predication rate)
for tweet in user_recommended_tweets:
    print(f'Rate of user to the tweet: {top_recommended_tweets[tweet[0]]}\n')
    print('Tweet Text:')
    print(tweet[1])
    print('\n\n')
```

Explanations:

Here is a brief explanation of what I did in the code:

- **Create List of Recommended Tweets:**
  - I used the `filter` function to create a list of recommended tweets (`user_recommended_tweets`) for the target user based on the top recommended tweet IDs.
  - Mapped each recommended tweet to a tuple containing the tweet ID and text.
- **Show Recommended Tweets:**
  - Displayed the recommended tweets and their corresponding recommendation scores (prediction rates) using the `top_recommended_tweets` dictionary.

Here, is one shot of recommended tweets' text with their respective score of recommendation (actually, predicted rate of user to tweets): (complete output can be seen in the Jupyter Notebook)



```
Rate of user to the tweet: 2.9999999999999996

Tweet Text:
ببین یاسمنگولا چقدر سلیطه است که حتی نازنین‌بنیادی هم تیکه سنگین بهش میندازه 😂 https://t.co/Y8h9um0LdR

Rate of user to the tweet: 1.5691904145692734

Tweet Text:
سرب گلوله‌ای که دهه ۶۰ منافقین به پدرم شلیک کردند هنوز در گونه‌اش موجود است.
امروز در دادگاه رسیدگی به جنایات گروهک رجوی حضور داشتم. ملت ایران با منافقین پدرکشتگی دارد… https://t.co/09WNDF6vYh

Rate of user to the tweet: 3.0000000000000004

Tweet Text:
«!بیانیه‌ی انجمن سبز چیا در واکنش به انفجار در جنگل‌های #زاگرس: ‹کوه خواری با انفجار مهیب و ترسناک / طبیعت و جنگل کیلویی چند؟ 🔴

#زن_زندگی_آزادی #محیط_زیست

انفجاری مهیب با این وسعت در میان جنگل‌های زاگروس بدون شک اعلان جنگ است. جنگی ب‌ه تمام معنا علیه جنگل‌ها و منابع طبیعی این سرزمین که جزو اندک سرمایه ی »
عمومی مردمان این دیار است.

سالیان درازیست که کارخانه شن و ماسه اطراف روستای " ساوا " از توابع بخش خاو میرآباد شهرستان مریوان بدون در نظر گرفتن اصول زیست محیطی و به دلخواه خود اقدام به
کوه خواری نموده و اینگونه کوهستان جنگلی را بدون هیچ محدودیت و واهمه‌ای جلوی چشم همگان با انفجاری مهیب و ترسناک، در دل طبیعت و حیات وحش، زیست بوم گیاهی و
جانوری و در همسایگی مخازن قیر و مشتقات نفتی و جاده‌ی بین‌المللی باشماق مورد تجاوز قرار می دهند.

انجمن سبز چیا تاکنون چندین بار خواستار ورود مسئولین ذیربط استانی و دادستانی شهرستان مریوان برای بررسی وضعیت  و نحوه برداشت کارخانه های شن و ماسه در مریوان
شد تا بلکه فعالیت این تخریبگران منابع ملی متوقف شود اما گویا اراده و خواستی برای برخورد با اینان وجود ندارد و با دهن کجی به همه، بدینسان کوه و جنگل را منفجر می‌کنند.

نهادها و مسئولان شهرستانی، استانی و کشوری که موظف به حفاظت از این سرزمین هستند به جای برخورد شدید و توبیخ عاملان این فاجعه، چگونه مجوز اینگونه انفجارها را
!می‌دهند

چگونه می توانید چشمانتان را در برابر خسارت و جنایتی که علی‌ه منابع طبیعی انجام می‌گیرد را ببندید و بدون برخورد بازدارنده از کنار آن بگذرید.

وقتی که بارها و بارها وضعیت اسفبار جنگل‌های منطقه را که هر روز بخشی از آن مورد تجاوز، تاراج، جنگل خواری و آتش‌سوزی قرار می‌گیرد و با روشن گری اعلام نمودیم؛ اما
متاسفانه  هیچ نهاد و ارگانی عاملان و امران را دستگیر و مورد مواخذه قرار نمی‌دهند؛  نتیجه‌اش این می‌شود که احدی بدون توجه به خواست و منافع عمومی و ملی اقدام به کو‌ه
خواری و انفجار در روز روشن بدون هیچ واهمه‌ای می‌نماید.

یقینا انجمن سبز چیا اقدامات خود را از طریق مسئولین کشوری، به ویژه در سازمان بازرسی کل کشور پیگیری نموده تا ضمن توقف این هیولای تخریب، فرد، افراد و نهادهایی را هم
که منابع ملی را فدای مطامع خود نموده و پشت این قضیه هستند برای افکار عمومی آشکار خواهد نمود و از دستگاه قضا به جد خواستار برخورد با تمامی عاملان این جنایت در
مریوان می باشد.

انجمن سبز چیا

«آذرماه ۱۴۰۲ ۲۱

Rate of user to the tweet: 1.4645883646830054

Tweet Text:
با درود و سپاس از همه دوستان و یاران عزیزم که همیشه حامی و یار من بودند و برای فوت پدرم چه در کامنت چه در دایرکت با من همدردی کردند من دستان شما را با مهر
میفشازم و یک دنیا سپاسگزارم از لطف و محبتهای شما .روح همه رفتگانمان شاد و یاد عزیزشان گرامی باد خواستم تشکر خودم را اینجا از همه عزیزانی که زحمت کشیدند و در درد
و غم من شریک بودند  ا اعلام کنم

🤞🙏 به امید پیروزی و رهایی

Rate of user to the tweet: 1.0

Tweet Text:
بوی عفونت این عکس به مشام شما نیز می‌رسد؟

#ابتذال_شر https://t.co/pbhlUoMMKL

Rate of user to the tweet: 1.5

Tweet Text:
این اراده سترگ مردم ایران ✊و فرزندان مجاهدش✊ است که #میتوان_و_باید را در صحنه سیاسی به کرسی می‌نشاند✊لغو سفر #جلاد_۶۷ در حالیکه رژیم هنوز 📢📢🔴
❤️✊توان اعتراف آن را ندارد به مردم و بویژه 👈خانواده 👈شهیدان مبارک باد

🔴🔴🔴🔴🔴🔴

. #RefugeeForum

 #No2Raisi https://t.co/tSVUMMe2V5

Rate of user to the tweet: 1.857142857142858

Tweet Text:
لعنت بر اونهایی که نگذاشتند شایسته ترین، نابغه ترین، پاک ترین و صادقترین فرزندان ایران مانند جزنی ها و حنیف نژادها و رضایی ها و گلسرخی ها و ... دمکراسی و برابری رو
!در ایران برقرار کنند و ایران را به عقب مانده ترین موجودات مانند خمینی و آخوندها سپردند

#نه_شاه_نه_شیخ #ایران https://t.co/lpQU1Pph8n
```

Explanations:

Obviously, all the results are consistent with the taste of the user, and the ratings are also true. Because, the more rate the recommended tweet has, the more it is closer to the specific politic side of the user and more important in mind of user(according to his favorite ones shown earlier). Therefore, looking at recommended tweets meticulously, we can conclude the recommender system has a high accuracy and low bias and variance. Or in other words, low false Negatives and false positives.

Is the Algorithm Online or Offline?

Explanations:

The proposed collaborative filtering algorithm is a completely **online** model. Because, if you add a user, whenever he replies, retweets, or quotes a tweet, we can find easily the tweets which he had engagement to, and then add the column of the given user (and if some of the tweets which the user had rated, didn't exist, add the row of the tweet), to the user rating matrix (RDD), with the corresponding ratings in the entries. This way, we are done!

So, the model doesn't need to be determined from the scratch; And using some little processes, we can reach the desired result and find similar users or items easily though.