Massive Data Analytics

Home Work 2

Ilia Hashemi Rad

99102456

General Approach

Explanations:

The general approach which I made to find the similar, is explained briefly here: After processing the body of the news as I did in the previous home work, we got the body of news cleaned from unwanted characters and stop words. Now, doing some steps on the processed body, we can find the similar news by following these steps:

- **Shingling** :
  This step is to transform the text of each article into a set of overlapping sequences of k tokens, called shingles. Tokens can be characters, words, ... which here I chose words as explained further.
  This allows us to capture the similarity of the text based on the commonality of the shingles. For example, if we use a character shingle length of 3, the sentence "The sky is blue" would be shingled into "The", "he ", "e s", " sk", "sky", "ky ", "y i", " is", "is ", "s b", " bl", "blu", "lue". I will do this for each article in our processed RDD and store the shingles in a new RDD.

- **Min-hashing** :
  This step is to reduce the dimensionality of the shingles and create a signature vector for each article. We will need to define a number of hash functions, which are random linear functions that map the shingles to integers. For each article, we will apply each hash function to its shingles and take the minimum value as the signature for that hash function. We will end up with a signature matrix that has the number of hash functions as rows and the number of articles as columns (actually, in our case it is done in an RDD and each element of RDD is a signature vector, but because this result is known as signature matrix, I told so). This way, we can preserve the similarity of the articles based on the probability of having the same signature for the same hash function.

- **LSH** :
  This step is to group the articles into buckets based on their similarity. We will need to define a number of bands and rows, which are parameters that control the trade-off between precision and recall. We will divide the signature matrix into bands, each containing a number of rows. For each band, we will hash the columns (articles) into buckets using another hash function. We will consider two articles as candidates for similarity if they fall into the same bucket for at least one band. This way, we can reduce the number of comparisons we need to make and focus on the most likely candidates.

- **Clustering** :
  This step is to find the similar news based on the candidates generated by LSH. We will need to define a similarity threshold, which is the minimum Jaccard similarity between two articles to be considered as similar. The Jaccard similarity is the ratio of the size of the intersection of the shingles to the size of the union of the shingles. For example, if article A has shingles a, b, c, d and article B has shingles b, c, e, f, their Jaccard similarity is $2/6 = 0.33$.

  – I used Jaccard similarity because it we are in context of analyzing text and no other measure works well as Jaccard measure. The more shingles they share, the more likely they have similar text, even if the order of the words is different. The Jaccard similarity is also easy to compute and has some nice properties, such as being bounded between 0 and 1, and being equal to 1 if and only if the documents have the same shingles

  We will compare the shingles of each candidate pair and compute their Jaccard similarity. If it is above the threshold, we will consider them as similar and assign them to the same cluster. This way, we can group the news articles based on their textual similarity and identify the ones that are related to each other.

Section 1: Preprocessing and Shingling

```python
import json
import re

# Parse the JSON lines and extract the UID and body text from each line
parsed_news_rdd = news_rdd.map(lambda line: (json.loads(line)['uid'], json.loads(line)['body']))

# Define a function to remove useless characters including '\n', '\u200c', and '\\n'
def remove_useless_characters(text):
    # Remove special characters, digits, and unwanted unicode characters
    text = re.sub(r'\\|\\n|\\u200c|\n', ' ', text)

    # Define a regular expression pattern that matches one or more spaces
    pattern = re.compile(r" +")
    # Apply the pattern to the text and replace the matches with a single space
    text = pattern.sub(" ", text)
    return text

# Define a function to determine a list of Farsi stop words
def load_stop_words(file_paths):
    stop_words_set = set()
    for file_path in file_paths:
        with open(file_path, 'r', encoding='utf-8') as file:
            stop_words_set.update(file.read().splitlines())
    return stop_words_set

# Define a function to create shinglings of words.
def generate_ngrams(text, n):
    # Split the text into words
    words = text.split()
    # Generate n-grams using a sliding window of size n
    ngrams = [tuple(words[i:i+n]) for i in range(len(words) - n + 1)]
    return ngrams

# List of paths to your stop words files
stop_words_files = ['verbal.txt', 'persian.txt', 'short.txt', 'chars.txt', 'nonverbal.txt']

# Load stop words from files
stop_words = load_stop_words(stop_words_files)

# Remove useless characters from the body text in parsed_news_rdd
clean_news_rdd = parsed_news_rdd.map(lambda x: (x[0], remove_useless_characters(x[1])))

# Remove the stop words from clean_news_rdd without splitting the text
clean_news_rdd = clean_news_rdd.map(lambda x: (x[0], " ".join(filter(lambda w: w not in stop_words, x[1].split()))))

# Process the cleaned news RDD by generating bi-grams (2-grams)
processed_news_rdd = clean_news_rdd.map(lambda x: (x[0], generate_ngrams(x[1], 2)))

# Display the processed news RDD for one record
print(processed_news_rdd.take(1))
```

Choosing The Type Of Shingling

Explanations:

Here, in this code, I cleaned the news as explained in HomeWork 1, (so I don't explain it here) and then did shingling on the cleaned news. I just used shingling of words. Because, after testing many times for shingling of characters(for the three cases of 5, 7 and 9 continuous sequences of characters), I realized because of removing stop words, the character sequence shingles, have a lower similarity between two similar news and that will force the threshold to be much lower to capture the similarity and lower the False negatives. Also, running the Min-Hashing and LSH on 5000 number of news from the JSON data, the code executed much faster on shingling of words comparing to the characters; Actually 4 times faster that is so significant.

If you look at the matter logically, this is obviously true because in our particular case, the text that needs to be processed is a clean text with its own useful words and nothing extra, so naturally the best case is that The words themselves should be compared and shingled because if we take the characters, many shingles will be added in vain and reduce the speed and accuracy of our processing. Please note that we are not in a situation where, for example, the news text is combined with advertisements or has additional things that we want to consider other shingling that includes characters or stop words.

Now you may ask why I chose two-word sequences as shingles. Because with the single word sequence test, I realized that the number of candidates increases greatly, and the processing speed decreases, and it is possible that due to the presence of similar single words between several news items, they may be considered similar, which is wrong, so the number of false positives increases. Also, by choosing sequences of three words (or more), many special expressions that have similarity in themselves and are closer to two words on average, their similarity will not be discovered, and in general, we will see much less similarity between the shingles of even two similar news, and false negatives will increase.

Therefore, the best case after many tests was two-word sequences, which both increases the processing speed and accommodates the similarity between two news well, because similar news usually have a large number of similar two-word sequences and vice versa.

Code Explanations

Explanations:

Here's a brief explanation from what I did:

- **Generating Shinglings of Words:**
  - I defined a `generate_ngrams` function to create n-grams (shinglings) of words using a sliding window approach. This makes the shinglings of two continuous sequence of words.
- **Removing Useless Characters, Stop Words, and Generating Bi-grams:**
  - I loaded stop words from files and create a set of stop words.
  - Then, I used `map` to apply the `remove_useless_characters` function to clean the body text.
  - Used `map` again to remove stop words without splitting the text.
  - Finally, generated bi-grams using the `generate_ngrams` function on the cleaned text.

And the output (one sample of `processed_news_rdd`) is:

```
[('68feae4bbbedc2d54adbb2369', [('سرمربی', 'ملوان'), ('ملوان', 'تصاویر'), ('تصاویر', 'وایرالی'), ('وایرالی', 'دیدار'), ('دیدار', 'نساجی'), ('نساجی', 'پست'), ('پست', 'جالب'),
('جالب', 'منتشر'), ('منتشر', 'گزارش'), ('گزارش', 'ورزش'), ('ورزش', 'سه'), ('سه', 'مهدی'), ('مهدی', 'تارتار'), ('تارتار', 'خوشحالی'), ('خوشحالی', 'عجیب'), ('عجیب',
'غریب'), ('غریب', 'دقیقه'), ('دقیقه', 'چهرههای'), ('چهرههای', 'جالب'), ('جالب', 'هفته'), ('هفته', 'لیگ'), ('لیگ', 'برتر'), ('برتر', 'اختصاص'), ('اختصاص', 'موردتوجه'),
('موردتوجه', 'هواداران'), ('هواداران', 'فوتبال'), ('فوتبال', 'قرار'), ('قرار', 'تارتار'), ('تارتار', 'انتشار'), ('انتشار', 'پست'), ('پست', 'اینستاگرامی'), ('اینستاگرامی', 'تصاویر'), ('تصاویر',
'جشن'), ('جشن', 'خوشحالی'), ('خوشحالی', 'نوشته'), ('نوشته', 'جادوی'), ('جادوی', 'طرفداران'), ('طرفداران', 'انزلی'), ('انزلی', 'سرمربی'), ('سرمربی', 'ملوان'), ('ملوان',
'وعده'), ('وعده', 'تیمش'), ('تیمش', 'ادامه'), ('ادامه', 'فصل'), ('فصل', 'جنگید'), ('جنگید', 'رضایت'), ('رضایت', 'هواداران'), ('هواداران', 'جلب'), ('جلب', 'نوشته'), ('نوشته',
'ادامه'), ('ادامه', 'عکسها'), ('عکسها', 'متحیر'), ('متحیر', 'میشوم'), ('میشوم', 'یادم'), ('یادم', 'نمیآید'), ('نمیآید', 'شوری'), ('شوری', 'جادوی'), ('جادوی', 'سکو'), ('سکو',
'حس'), ('حس', 'انرژی'), ('انرژی', 'میدهید'), ('میدهید', 'لحظه'), ('لحظه', 'پیروزی'), ('پیروزی', 'وصفناشدنی'), ('وصفناشدنی', 'سکوها'), ('سکوها', 'خط'), ('خط', 'پای'),
('پای', 'تلویزیونها'), ('تلویزیونها', 'مستطیل'), ('مستطیل', 'سبز'), ('سبز', 'همدل'), ('همدل', 'یکصدا'), ('یکصدا', 'میرویم'), ('میرویم', 'باهم'), ('باهم', 'میجنگیم'), ('میجنگیم',
'باهم'), ('باهم', 'میخندیم'), ('میخندیم', 'هرروز'), ('هرروز', 'تلاش'), ('تلاش', 'خنده'), ('خنده', 'نگه'), ('نگه', 'پیروزی'), ('پیروزی', 'مردان'), ('مردان', 'نیک'), ('نیک', 'چمن'),
('چمن', 'مردمان'), ('مردمان', 'نیک'), ('نیک', 'سکو')])]
```

Explanations:

As you can see, the result has the uid of any news next to its shingles list, as expected.

Section 2: Min-Hashing and Signature Matrix

Choosing the value of r, b and the Similarity Threshold t

Explanations:

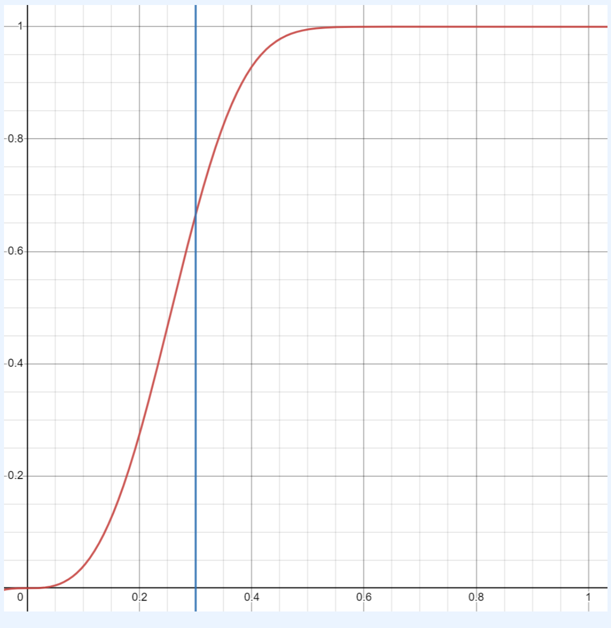
Since we have a significant number of different and special shingles, about 100 to 200 hash functions are needed to capture the similarities. Because by testing on lower values, I realized that some similarities may not be discovered and we get false negatives. Now, it is enough to first determine how much Jaccard similarity between news shingles guarantees the similarity of two news to a good extent. Then, as second, determine the best value of r (b can be determine from the relation of $b = number\ of\ bands = \frac{number\ of\ hash\ functions}{r}$).

For the first step, after many tests, I realized that the best threshold value is 0.3, which has a very high accuracy, so that in all the cases I observed, if the similarity between two news was less than that, the two news were not really similar, and if the similarity was more, they were similar.

This is also logical. Because we have removed the unwanted characters and stop words, which have a high percentage of similarity between the two news (of course, removing them by accurately setting the best threshold value, as I did, also increases the processing speed. However, the overall amount of similarity between the shingles is reduced just by a DC scale, which is not important). Therefore, it is natural that the Jaccard similarity between the shingles of two similar news can be as low as 0.3. Because if you look carefully, this means that at least 30% of unique two-word phrases (because I make the list of shingles unique for comparison) are the same in two news items, which indicates the possible high similarity of the two news items.

Now, it's time to do second step and find the value of r. Our priority and goal is to have as few false negatives as possible so that we don't miss anything. Because in the next sections of the code, the additional candidates we have obtained can be deleted, but if we miss a really similar news, nothing can be done. For this, we are trying to increase the number of candidates as much as the processing power and code speed allow, which is achieved by keeping the number of rows in each band low. Therefore, $r = 3$ can be a very good choice.

According to the slides of the course and the relation of the Prob. Sharing a Bucket $(P = 1 - (1 - t^r)^b)$ and according to the similarity threshold, if we choose $r = 3$ and $b = 40$, we will reach the following graph, which is the best possible state after many tests. This also satisfies the first issue which I talked first, for the number of hash functions to be between 100 and 200; Now, it is $number\ of\ hash\ functions = k = 3 \times 40 = 120$.



According to this graph and existing relationship, it selects news with at least 30% similarity with a probability of 67% and news with at least 50% similarity with a probability of 99.5%. Therefore, we reached what we wanted, that false negatives are very low. But as mentioned, in order to reduce false positives, in the next steps after LSH and finding the candidates, all the shingles are compared if the candidates are similar, and if the similarity is more than desirable, they are considered similar.

```python
    import random
import sys

# Define the number of bands and rows for the Min-Hash algorithm
NUM_BAND = 40  # Number of bands
NUM_ROW = 3    # Number of rows per band

# Calculate the total number of hash functions
k = NUM_BAND * NUM_ROW

# Define a large prime number for the hash functions
p = 4294967311

# Generate k random coefficients for the hash functions
a = random.sample(range(1, p), k)
b = random.sample(range(0, p), k)

# Define a function to hash a shingle using the ith hash function
def hash_shingle(shingle, i):
    shingle = " ".join(shingle)
    return (a[i] * hash(shingle) + b[i]) % p

# Define a function to compute the min-hash signature of a set of shingles
def min_hash(shingles):
    # Initialize a signature vector of length k with positive infinity values
    signature = [sys.maxsize] * k
    # Loop over each shingle in the set
    for shingle in shingles:
        # Loop over each hash function
        for i in range(k):
            # Hash the shingle using the ith hash function
            hash_value = hash_shingle(shingle, i)
            # Update the ith value of the signature if the hash value is smaller to find the minimum signature
            if signature[i] > hash_value:
                signature[i] = hash_value
    # Return the signature vector
    return signature

# Min-Hash and Signature Matrix
# Apply the min_hash function to each set of shingles in processed_news_rdd
# to generate the min-hash signature matrix for each news article
signature_matrix = processed_news_rdd.map(lambda x: (x[0], min_hash(x[1])))
```

Code Explanations

Explanations:

- **Setting Up Min-Hash Parameters:**
  - I defined a large prime number (`p`) for the hash functions.
  - Them generated random coefficients (`a` and `b`) for the hash functions.
- **Hashing Shingles:**
  - I defined a function (`hash_shingle`) to hash a shingle using the ith hash function.
- **Computing Min-Hash Signature:**
  - I defined a function (`min_hash`) to compute the min-hash signature of a set of shingles.
  - Initialized a signature vector with positive infinity values.
  - Looped over each shingle in the set and hash it using multiple hash functions.
  - Updated the signature vector to find the minimum signature values.
- **Min-Hash and Signature Matrix:**
  - I applied the `min_hash` function to each set of shingles in `processed_news_rdd`.
  - Then generated the min-hash signature matrix for each news article, creating an RDD of tuples (`(UID, signature_vector)`).

Section 3: LSH and Candidate Pairs

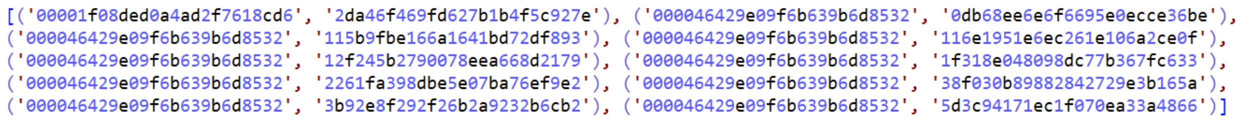
```python
 import itertools

# Define a function to hash a signature vector using the ith band
def hash_signature(signature, i):
    # Extract the band corresponding to the ith band index
    band = signature[i * NUM_ROW: (i + 1) * NUM_ROW]
    # Calculate the hash value of the band
    value = hash(tuple(band))
    return (a[i] * value + b[i]) % p

# Define a function to generate candidate pairs from a band
def generate_candidates(band):
    # Group the tuples by their second element using the groupby function
    grouped = itertools.groupby(sorted(band, key=lambda x: x[1]), key=lambda x: x[1])
    # Initialize an empty list of candidate pairs
    candidates = []
    # Loop over each group
    for _, group in grouped:
        # Extract the list of articles in the group
        articles = [x[0] for x in group]
        # If the group has more than one article, generate all possible pairs
        if len(articles) > 1:
            for i in range(len(articles)):
                for j in range(i + 1, len(articles)):
                    # Append the pair to the list of candidates
                    candidates.append((articles[i], articles[j]))
    # Return the list of candidate pairs
    return candidates

# LSH and generating candidate pairs

# Apply the hash_signature function to each signature vector in the RDD
hashed_matrix = signature_matrix.flatMap(lambda x: [(i, (x[0], hash_signature(x[1], i))) for i in range(NUM_BAND)])

# Group the hashed matrix by the band index
grouped_matrix = hashed_matrix.groupByKey().mapValues(list)

# Generate candidate pairs from each band
candidate_pairs = grouped_matrix.flatMap(lambda x: generate_candidates(x[1]))

# Remove duplicate pairs and sort them by the article ids
candidate_pairs = candidate_pairs.distinct().sortBy(lambda x: (x[0], x[1]))

# Display the first 10 candidate pairs
print(candidate_pairs.take(10))
```

Explanations:

Here, is a brief explanation of my code:

- **Hashing Signature Vectors:**
  - I defined a function (`hash_signature`) to hash a signature vector using the ith band.
  - Then, extracted the band corresponding to the ith band index and calculate the hash value of the band.
- **Generating Candidate Pairs from a Band:**
  - I defined a function (`generate_candidates`) to generate candidate pairs from a band.
  - Grouped the tuples by their second element using the `groupby` function.
  - For each group, if it has more than one article, generated all possible pairs and append them to the list of candidates.
- **LSH and Generating Candidate Pairs:**
  - I applied the `hash_signature` function to each signature vector in the RDD, creating a new RDD of hashed values.
  - Grouped the hashed matrix by the band index.
  - Generated candidate pairs from each band using the `generate_candidates` function.
  - Removed duplicate pairs and sort them by the article ids.
  - Finally, displayed the first 10 candidate pairs.

And the output of this code is: (10 samples of `candidate_pairs` RDD)



```
[('00001f08ded0a4ad2f7618cd6', '2da46f469fd627b1b4f5c927e'), ('000046429e09f6b639b6d8532', '0db68ee6e6f6695e0ecce36be'),
('000046429e09f6b639b6d8532', '115b9fbe166a1641bd72df893'), ('000046429e09f6b639b6d8532', '116e1951e6ec261e106a2ce0f'),
('000046429e09f6b639b6d8532', '12f245b2790078eea668d2179'), ('000046429e09f6b639b6d8532', '1f318e048098dc77b367fc633'),
('000046429e09f6b639b6d8532', '2261fa398dbe5e07ba76ef9e2'), ('000046429e09f6b639b6d8532', '38f030b89882842729e3b165a'),
('000046429e09f6b639b6d8532', '3b92e8f292f26b2a9232b6cb2'), ('000046429e09f6b639b6d8532', '5d3c94171ec1f070ea33a4866')]
```

Section 4: Clustering and Finding Similar News

```python
    # Define the query UID
query_uid = 'b272bcaeb94af1c6e9c6ed0aa'

# Filter candidate pairs for the given query UID
query_candidates = candidate_pairs.filter(lambda x: x[0] == query_uid or x[1] == query_uid).map(lambda x: x[1] if x[0] == query_uid

# Retrieve the set of shingles for the query UID
query_shingles = set(processed_news_rdd.lookup(query_uid)[0])

# Retrieve shingles and UIDs for the candidate pairs
candidates_uid_shingles = processed_news_rdd.filter(lambda x: x[0] in query_candidates).collect()

# Define the Jaccard similarity threshold
jaccard_threshold = 0.3

# Define a function to compute the Jaccard similarity
def compute_jaccard_similarity(set1, set2):
    intersection_size = len(set1.intersection(set2))
    union_size = len(set1.union(set2))
    return intersection_size / union_size if union_size > 0 else 0

# Find similar news based on Jaccard similarity
similar_news_uid = []
similar_news = []
for candidate_uid, candidate_shingles in candidates_uid_shingles:
    candidate_shingles = set(candidate_shingles)
    similarity = compute_jaccard_similarity(query_shingles, candidate_shingles)
    # Check if the Jaccard similarity is above the threshold
    if similarity > jaccard_threshold:
        similar_news.append((candidate_uid, similarity))
        similar_news_uid.append(candidate_uid)
```

Explanations:

This code gets one UID as a query article, then finds the similar news with that article by filtering candidates to be similar with that article from `candidate_pairs` RDD and then processing on them.

Please note that there are many samples of the dataset duplicated, and we have two of them in dataset. So, wherever I used `list(set(some_list))`, it was to get unique values from that list.

Here, is a brief explanation of what I did:

- **Query and Candidate Pairs:**
  - I filtered candidate pairs (`candidate_pairs`) to include pairs involving the query UID.
  - Then, i mapped the filtered pairs to extract the UID that is not the query UID.
  - Finally, I collected the UIDs of candidate news related to the query UID.
- **Retrieve Shingles for the Query UID:**
  - I retrieved the set of shingles for the query UID (`query_shingles`) from the processed news RDD.
- **Retrieve Shingles and UIDs for Candidate Pairs:**
  - I filtered the processed news RDD to include only the candidate UIDs found in the previous step.
  - Then I collected the UIDs and corresponding shingles for the candidate news.
- **Compute Jaccard Similarity:**
  - I defined a function (`compute_jaccard_similarity`) to compute the Jaccard similarity between two sets. Here, I used sets and converted the inputs of this function to sets instead of lists everywhere needed, because I wanted to process on unique shingles for more reliability and accuracy.
- **Find Similar News Based on Jaccard Similarity:**
  - I iterated over candidate UIDs and their corresponding shingles.
  - Computed the Jaccard similarity between the query shingles and candidate shingles.
  - If the similarity is above the threshold, added the UID and similarity to the list of similar news.

```python

# Extract the title of the query article
query_title = list(set(news_rdd.filter(lambda x: json.loads(x)['uid'] in [query_uid]).map(lambda x: json.loads(x)['title']).collect

# Extract the title and UID of similar news articles
similar_news_list = list(set(news_rdd.filter(lambda x: json.loads(x)['uid'] in similar_news_uid).map(lambda x: (json.loads(x)['uid'

# Create dictionaries for efficient lookup
dict1 = dict(similar_news)
dict2 = dict(similar_news_list)

# Join the dictionaries to get a list of tuples with common keys(uid)... final result is a list of tuples with uis, similarity and
joined_list = [(key, dict1[key], dict2[key]) for key in set(dict1) & set(dict2)]

# Display the query article title
print('The given article title: ')
print(query_title[0])
print('\n\n')

# Display information about similar news articles
print(f'For the given article, {len(joined_list)} unique similar news founded:\n')
for uid, similarity, title in joined_list:
    print(f"Title: {title}\n")
    print(f"UID: {uid}, Similarity: {similarity}\n\n")
```

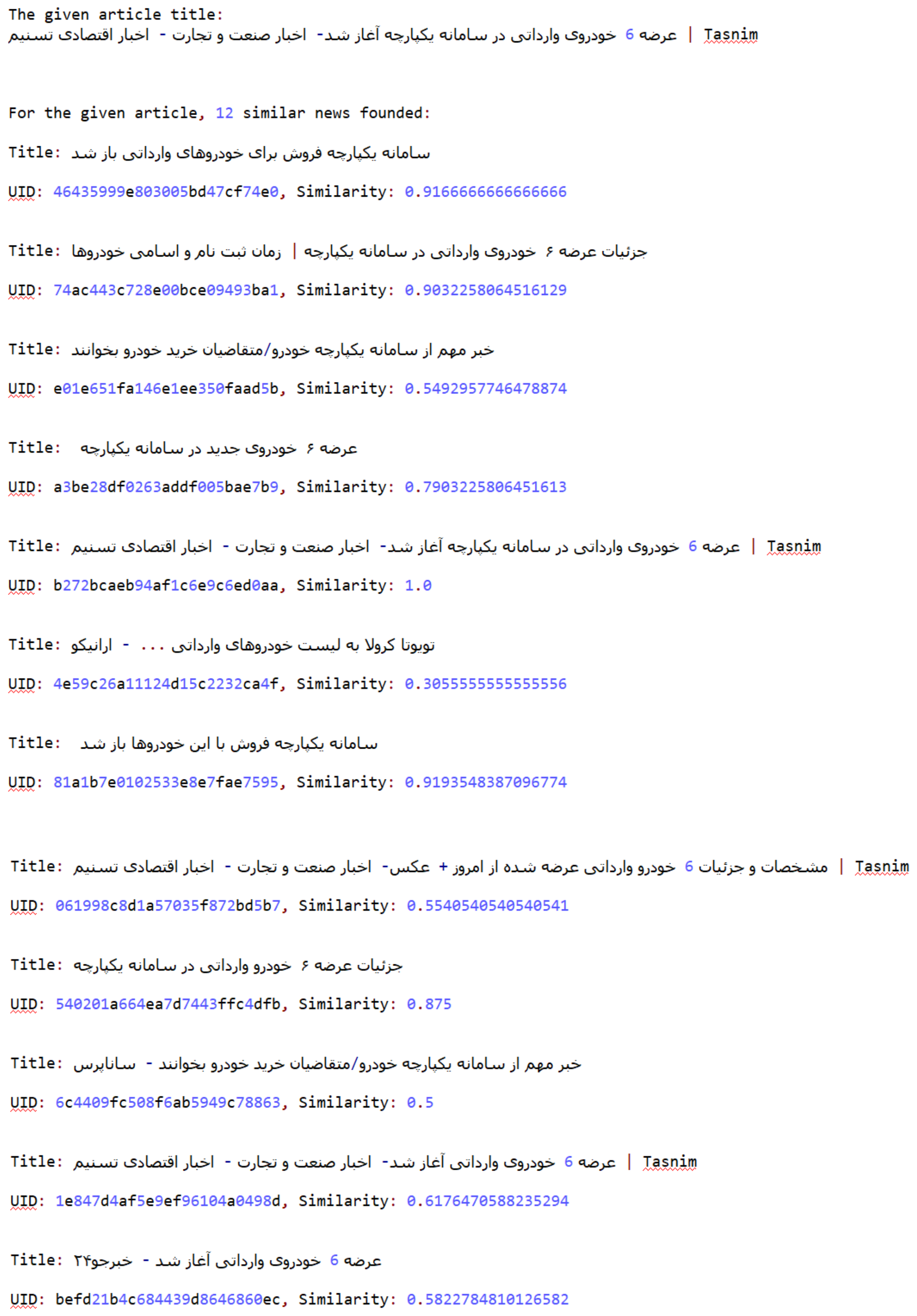
Explanations:

This code just shows the results(title and similarity of similar news) together to validate the correctness of whole of the code. Here, is a brief explanation of what I did:

- **Extract Title of Query Article:**
  - I filtered the news RDD to include only the query UID.
  - Extracted the title of the query article (`query_title`) by collecting and converting to a set.
- **Extract Title and UID of Similar News Articles:**
  - I filtered the news RDD to include only UIDs found in the list of similar news UIDs.
  - Extracted the UID and title of similar news articles (`similar_news_list`) by collecting and converting to a set.
- **Create Dictionaries for Efficient Lookup:**
  - I created dictionaries (`dict1` and `dict2`) for efficient lookup of UID, similarity, and title.
- **Join Dictionaries to Get a List of Tuples:**
  - I joined the dictionaries to get a list of tuples (`joined_list`) with common keys (UIDs). The final result is a list of tuples with uis, similarity and title of each similar news
- **Display Information About Similar News Articles:**
  - Finally, printed the reslusts about each similar news article, including UID, similarity, and title.

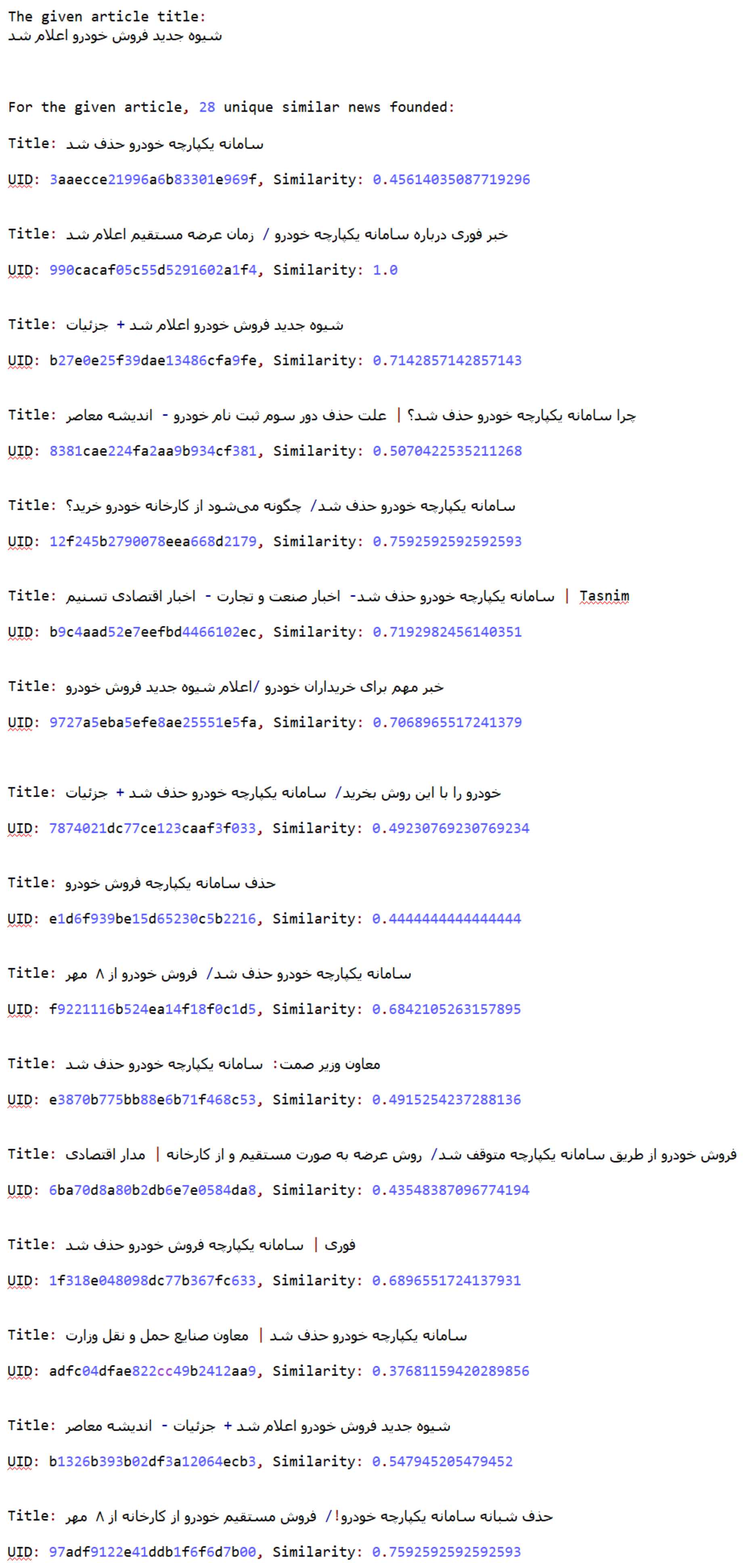
I tested the code for two different query articles. Here is the outputs:

Sample 1

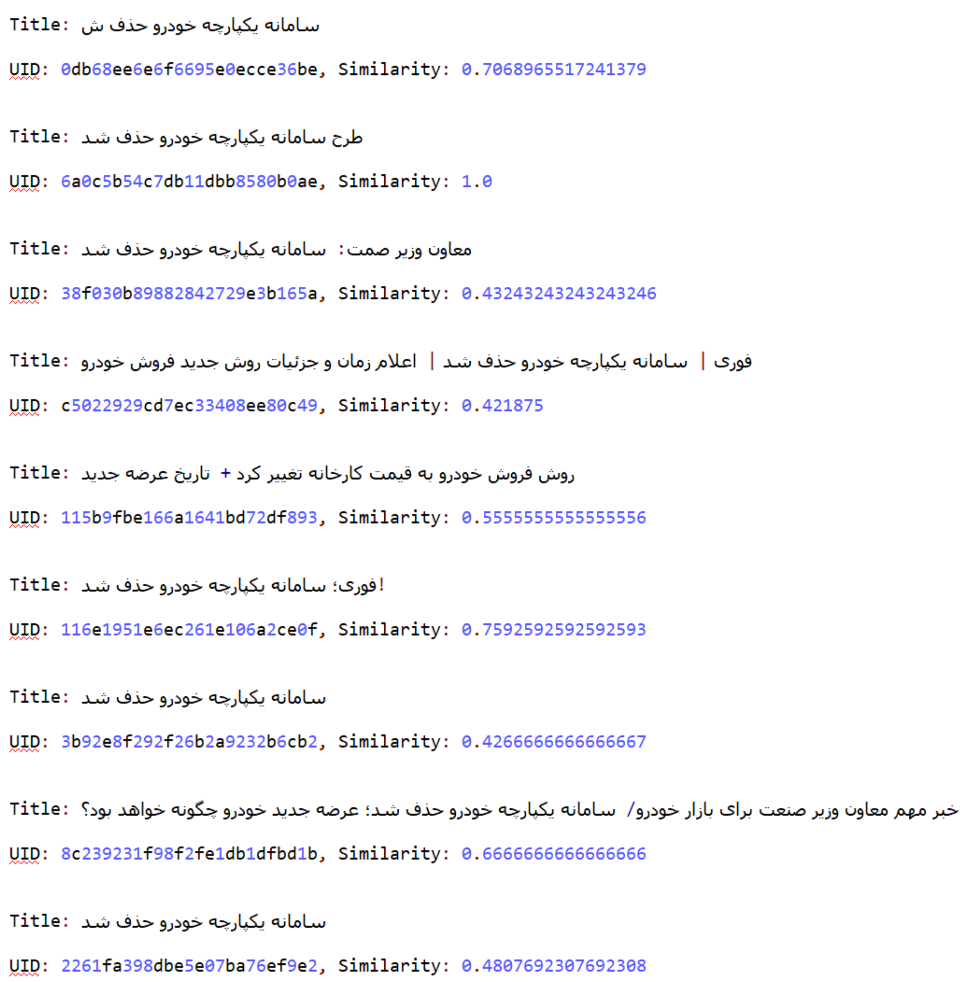


```
The given article title:
عرضه 6 خودروی وارداتی در سامانه یکپارچه آغاز شد- اخبار صنعت و تجارت - اخبار اقتصادی تسنیم | Tasnim


For the given article, 12 similar news founded:

Title: سامانه یکپارچه فروش برای خودروهای وارداتی باز شد

UID: 46435999e803005bd47cf74e0, Similarity: 0.9166666666666666


Title: جزئیات عرضه ۶ خودروی وارداتی در سامانه یکپارچه | زمان ثبت نام و اسامی خودروها

UID: 74ac443c728e00bce09493ba1, Similarity: 0.9032258064516129


Title: خبر مهم از سامانه یکپارچه خودرو/متقاضیان خرید خودرو بخوانند

UID: e01e651fa146e1ee350faad5b, Similarity: 0.5492957746478874


Title:  عرضه ۶ خودروی جدید در سامانه یکپارچه

UID: a3be28df0263addf005bae7b9, Similarity: 0.7903225806451613


Title: عرضه 6 خودروی وارداتی در سامانه یکپارچه آغاز شد- اخبار صنعت و تجارت - اخبار اقتصادی تسنیم | Tasnim

UID: b272bcaeb94af1c6e9c6ed0aa, Similarity: 1.0


Title: تویوتا کرولا به لیست خودروهای وارداتی ... - ارانیکو

UID: 4e59c26a11124d15c2232ca4f, Similarity: 0.3055555555555556


Title:  سامانه یکپارچه فروش با این خودروها باز شد

UID: 81a1b7e0102533e8e7fae7595, Similarity: 0.9193548387096774



Title: مشخصات و جزئیات 6 خودرو وارداتی عرضه شده از امروز + عکس- اخبار صنعت و تجارت - اخبار اقتصادی تسنیم | Tasnim

UID: 061998c8d1a57035f872bd5b7, Similarity: 0.5540540540540541


Title: جزئیات عرضه ۶ خودرو وارداتی در سامانه یکپارچه

UID: 540201a664ea7d7443ffc4dfb, Similarity: 0.875


Title: خبر مهم از سامانه یکپارچه خودرو/متقاضیان خرید خودرو بخوانند - ساناپرس

UID: 6c4409fc508f6ab5949c78863, Similarity: 0.5


Title: عرضه 6 خودروی وارداتی آغاز شد- اخبار صنعت و تجارت - اخبار اقتصادی تسنیم | Tasnim

UID: 1e847d4af5e9ef96104a0498d, Similarity: 0.6176470588235294


Title: عرضه 6 خودروی وارداتی آغاز شد - خبرجو۲۴

UID: befd21b4c684439d8646860ec, Similarity: 0.5822784810126582
```

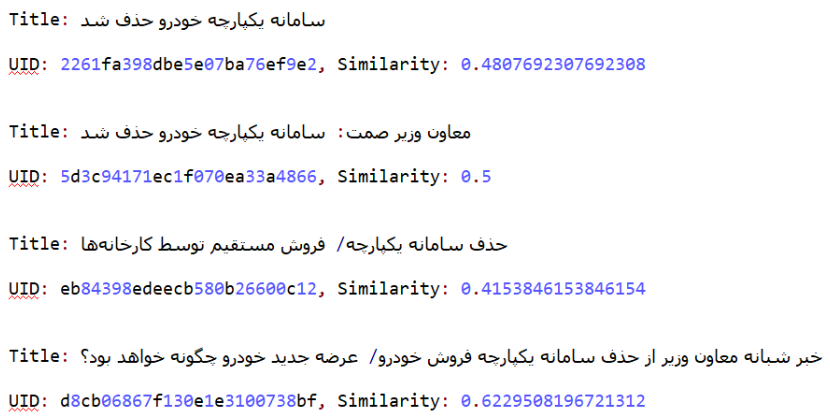
Sample 2



```
The given article title:
شیوه جدید فروش خودرو اعلام شد


For the given article, 28 unique similar news founded:

Title: سامانه یکپارچه خودرو حذف شد

UID: 3aaecce21996a6b83301e969f, Similarity: 0.45614035087719296


Title: خبر فوری درباره سامانه یکپارچه خودرو / زمان عرضه مستقیم اعلام شد

UID: 990cacaf05c55d5291602a1f4, Similarity: 1.0


Title: شیوه جدید فروش خودرو اعلام شد + جزئیات

UID: b27e0e25f39dae13486cfa9fe, Similarity: 0.7142857142857143


Title: چرا سامانه یکپارچه خودرو حذف شد؟ | علت حذف دور سوم ثبت نام خودرو - اندیشه معاصر

UID: 8381cae224fa2aa9b934cf381, Similarity: 0.5070422535211268


Title: سامانه یکپارچه خودرو حذف شد/ چگونه می‌شود از کارخانه خودرو خرید؟

UID: 12f245b2790078eea668d2179, Similarity: 0.7592592592592593


Title: سامانه یکپارچه خودرو حذف شد- اخبار صنعت و تجارت - اخبار اقتصادی تسنیم | Tasnim

UID: b9c4aad52e7eefbd4466102ec, Similarity: 0.7192982456140351


Title: خبر مهم برای خریداران خودرو /اعلام شیوه جدید فروش خودرو

UID: 9727a5eba5efe8ae25551e5fa, Similarity: 0.7068965517241379



Title: خودرو را با این روش بخرید/ سامانه یکپارچه خودرو حذف شد + جزئیات

UID: 7874021dc77ce123caaf3f033, Similarity: 0.49230769230769234


Title: حذف سامانه یکپارچه فروش خودرو

UID: e1d6f939be15d65230c5b2216, Similarity: 0.4444444444444444


Title: سامانه یکپارچه خودرو حذف شد/ فروش خودرو از ۸ مهر

UID: f9221116b524ea14f18f0c1d5, Similarity: 0.6842105263157895


Title: معاون وزیر صمت: سامانه یکپارچه خودرو حذف شد

UID: e3870b775bb88e6b71f468c53, Similarity: 0.4915254237288136


Title: فروش خودرو از طریق سامانه یکپارچه متوقف شد/ روش عرضه به صورت مستقیم و از کارخانه | مدار اقتصادی

UID: 6ba70d8a80b2db6e7e0584da8, Similarity: 0.43548387096774194


Title: فوری | سامانه یکپارچه فروش خودرو حذف شد

UID: 1f318e048098dc77b367fc633, Similarity: 0.6896551724137931


Title: سامانه یکپارچه خودرو حذف شد | معاون صنایع حمل و نقل وزارت

UID: adfc04dfae822cc49b2412aa9, Similarity: 0.37681159420289856


Title: شیوه جدید فروش خودرو اعلام شد + جزئیات - اندیشه معاصر

UID: b1326b393b02df3a12064ecb3, Similarity: 0.547945205479452


Title: حذف شبانه سامانه یکپارچه خودرو!/ فروش مستقیم خودرو از کارخانه از ۸ مهر

UID: 97adf9122e41ddb1f6f6d7b00, Similarity: 0.7592592592592593
```



```
Title: سامانه یکپارچه خودرو حذف ش

UID: 0db68ee6e6f6695e0ecce36be, Similarity: 0.7068965517241379

Title: طرح سامانه یکپارچه خودرو حذف شد

UID: 6a0c5b54c7db11dbb8580b0ae, Similarity: 1.0

Title: معاون وزیر صمت: سامانه یکپارچه خودرو حذف شد

UID: 38f030b89882842729e3b165a, Similarity: 0.43243243243243246

Title: فوری | سامانه یکپارچه خودرو حذف شد | اعلام زمان و جزئیات روش جدید فروش خودرو

UID: c5022929cd7ec33408ee80c49, Similarity: 0.421875

Title: روش فروش خودرو به قیمت کارخانه تغییر کرد + تاریخ عرضه جدید

UID: 115b9fbe166a1641bd72df893, Similarity: 0.5555555555555556

Title: فوری؛ سامانه یکپارچه خودرو حذف شد!

UID: 116e1951e6ec261e106a2ce0f, Similarity: 0.7592592592592593

Title: سامانه یکپارچه خودرو حذف شد

UID: 3b92e8f292f26b2a9232b6cb2, Similarity: 0.4266666666666667

Title: خبر مهم معاون وزیر صنعت برای بازار خودرو/ سامانه یکپارچه خودرو حذف شد؛ عرضه جدید خودرو چگونه خواهد بود؟

UID: 8c239231f98f2fe1db1dfbd1b, Similarity: 0.6666666666666666

Title: سامانه یکپارچه خودرو حذف شد

UID: 2261fa398dbe5e07ba76ef9e2, Similarity: 0.4807692307692308
```



```
Title: سامانه یکپارچه خودرو حذف شد

UID: 2261fa398dbe5e07ba76ef9e2, Similarity: 0.4807692307692308

Title: معاون وزیر صمت: سامانه یکپارچه خودرو حذف شد

UID: 5d3c94171ec1f070ea33a4866, Similarity: 0.5

Title: حذف سامانه یکپارچه/ فروش مستقیم توسط کارخانه‌ها

UID: eb84398edeecb580b26600c12, Similarity: 0.4153846153846154

Title: خبر شبانه معاون وزیر از حذف سامانه یکپارچه فروش خودرو/ عرضه جدید خودرو چگونه خواهد بود؟

UID: d8cb06867f130e1e3100738bf, Similarity: 0.6229508196721312
```

Explanations:

As you can see, the results are all correct and we have no False Positive.

On the other hand, when I checked the body of those news which were below the threshold in similarity, there was some similarity between them and the query article, but they were not similar and their news actually where different. However, they had some common sentences and information, they were for different purposes and had some different information. This means, according to previous explanations of what I did to reduce the False negatives as possible, I had also correctly filter them and have a very few False negatives(which can't be even seen). After probing many query articles like these, I founded that the code misses almost no similar news while capturing no wrong similar news (a dissimilar one) and has a high accuracy with a high speed.