Homework 1: Weighted Locality Sensitive Hashing

Name Jinsu Lim
Division School of Computing
Student number 20223558

1 Weighted Jaccard Similarity [40 Points]

1.1 Computing the weighted Jaccard similarity [10 Points]

$$M = \begin{bmatrix} 3 & 0 & 1 & 5 & 0 \\ 2 & 1 & 0 & 5 & 1 \\ 5 & 0 & 4 & 1 & 0 \end{bmatrix} \tag{1}$$

1.1.1 Calculate by hand the weighted Jaccard similarity between the three pair

$< 0,1 >$:

$$\frac{\sum_i min(M[0][i], M[1][i])}{\sum_i max(M[0][i], M[1][i])} = \frac{2+0+0+5+0}{3+1+1+5+1} = \frac{7}{11} = 0.6364$$

$< 0,2 >$:

$$\frac{\sum_i min(M[0][i], M[2][i])}{\sum_i max(M[0][i], M[2][i])} = \frac{3+0+1+1+0}{5+0+4+5+0} = \frac{5}{14} = 0.3571$$

$< 1,2 >$:

$$\frac{\sum_i min(M[1][i], M[2][i])}{\sum_i max(M[1][i], M[2][i])} = \frac{2+0+0+1+0}{5+1+4+5+1} = \frac{3}{16} = 0.1875$$

1.1.2 Which of the second and third items is more similar to the first item?

The second item is more similar to the first because $0.6364 > 0.3571$.

1.1.3 What if we do not consider the weights (i.e., ratings)? Is the second (or third) item still more similar to the first item?

The jaccard similarity of the first and second items is,
{user1, user4}/ {user1, user2, user3, user4, user5} = 2/5 = 0.4
The jaccard similarity of the first and third items is,
{user1, user3, user4}/{user1, user3, user4} = 3/3 = 1.0

The third item is more similar to the first item.

1.2 Connection to the Jaccard similarity [10 Points]

1.2.1 Transform matrix into a binary matrix $M_B \in \{0,1\}^{3\times s}$

each user's ratings transform into (user, rating) and, to store the rating semantics while calculating jaccard similarity, under score also add it.

$$M_B = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

When converting to the normal jaccard calculation, min should be replaced by intersection and max by union. To preserve the minimum and maximum results when they are changed, the shingle below the user's rating score is also checked to 1.

e.g. If the user rates the movie A as a 3. Check 1 together for the shingle corresponding to A_1, A_2, and A_3.

If the Jaccard similarity is calculated for each pair in the above matrix, you can see that the results are the same as shown below.

first, second $\Rightarrow intersection(first, second)/union(first, second) = 7/11 = 0.6364$

$first, third \Rightarrow intersection(first, third)/union(first, third) = 5/14 = 0.3571$

$second, third \Rightarrow intersection(second, third)/union(second, third) = 3/16 = 0.1875$

1.3 Computing MinHash Signatures [20 Points]

1.3.1 Calculate the minhash signatures of each row of M_B using the following 3 hash functions, where x is the index of each column (from 1 to 25) :

$$h_1(x) = 7x + 1 \mod 25;\ h_2(x) = 5x + 2 \mod 25;\ h_3(x) = 3x + 5 \mod 25 \tag{3}$$

The permutation π for each hash function is as follows

$$\begin{bmatrix} \pi_1(x) \\ \pi_2(x) \\ \pi_3(x) \end{bmatrix} =$$

$$\begin{bmatrix} 8 & 15 & 22 & 4 & 11 & 18 & 0 & 7 & 14 & 21 & 3 & 10 & 17 & 24 & 6 & 13 & 20 & 2 & 9 & 16 & 23 & 5 & 12 & 19 & 1 \\ 7 & 12 & 17 & 22 & 2 & 7 & 12 & 17 & 22 & 2 & 7 & 12 & 17 & 22 & 2 & 7 & 12 & 17 & 22 & 2 & 7 & 12 & 17 & 22 & 2 \\ 8 & 11 & 14 & 17 & 20 & 23 & 1 & 4 & 7 & 10 & 13 & 16 & 19 & 22 & 0 & 3 & 6 & 9 & 12 & 15 & 18 & 21 & 24 & 2 & 5 \end{bmatrix} \tag{4}$$

Signature matrix $M_S^{(Item \times Signature)}$ is

$$\begin{bmatrix} 2 & 2 & 3 \\ 2 & 2 & 3 \\ 3 & 2 & 3 \end{bmatrix} \tag{5}$$

1.3.2 Calculate the estimated Jaccard similarities of the three pairs of the rows and compare them with the true Jaccard similarities.

Pair	1-2	1-3	2-3
Col/Col	0.6364	0.3571	0.1875
Sig/Sig	1.0	0.67	0.67

Table 1: compare signature jaccard similarities with true jaccard similarities

2 Locality Sensitive Hashing (LSH) [60 Points]

2.1 Dataset: MOVIELENS 10M

2.2 Preliminaries [20 Points]

2.2.1 Implement the `weighted_jaccard_similarity()`

```python
## Task 1. Implement the weighted_jaccard_similarity.

def weighted_jaccard_similarity(s1, s2):
    ans = 0
    #### Implement here ####

    sum_of_intersection = 0
    sum_of_union = 0
    for x_i, y_i in zip(s1, s2):
        sum_of_intersection += min(x_i, y_i)
        sum_of_union += max(x_i, y_i)

    ans = sum_of_intersection / sum_of_union

    ##########################

    return ans
```

```python
print(weighted_jaccard_similarity(item_vec[0], item_vec[1]))
print(weighted_jaccard_similarity(item_vec[1], item_vec[2]))
print(weighted_jaccard_similarity(item_vec[0], item_vec[2]))
```

```
0.6363636363636364
0.1875
0.35714285714285715
```

Figure 1: 2.2.1 code & result

2.2.2 Implement the `build_items_to_shingle_dictionary()`

CODE:

```python
# Sub-function that will be used in the next tasks.
# Please implement this function for future use.

def get_shingles(items):
    shingles = []

    #### Implement here ####

    users = set()
    ratings = set()
    for item in items.values():
        for feedback in item:
            users.add(feedback[0])
            ratings.add(feedback[1])

    shingles = list(itertools.product(users, ratings))

    ##########################

    return set(shingles)
```

```python
def build_items_to_shingle_dictionary(items, shingles):
    item_to_shingles = {}

    #### Implement here ####

    shingle_to_idx = {}
    for i, shingle in enumerate(sorted(shingles)):
        shingle_to_idx[shingle] = i


    for item_no, item in items.items():
        buf = []
        for feedback in item:
            user_no, rate = feedback
            for i in range(1, rate+1):
                buf.append((user_no, i))

        item_to_shingles[item_no] = [shingle_to_idx[i] for i in buf]

    ##########################


    return item_to_shingles
```

Figure 2: 2.2.2 code

```python
shingles = get_shingles(items)
item_to_shingles = build_items_to_shingle_dictionary(items, shingles)
```

```python
len(shingles)
```

```
25
```

```python
item_to_shingles
```

```
{0: [0, 1, 2, 10, 15, 16, 17, 18, 19],
 1: [0, 1, 5, 15, 16, 17, 18, 19, 20],
 2: [0, 1, 2, 3, 4, 10, 11, 12, 13, 15]}
```

Figure 3: 2.2.2 result

2.2.3 Implement the `jaccard_similarity()`

```
In [310]: # Task 3. Implement jaccard_similarity.
          def jaccard_similarity(s1, s2):
              ans = 0
              #### implement here ####

              ans = len(s1.intersection(s2)) / len(s1.union(s2))


              ##########################

              return ans

In [311]: print(jaccard_similarity(set(item_to_shingles[0]), set(item_to_shingles[1])))
          print(jaccard_similarity(set(item_to_shingles[1]), set(item_to_shingles[2])))
          print(jaccard_similarity(set(item_to_shingles[0]), set(item_to_shingles[2])))

          0.6363636363636364
          0.1875
          0.35714285714285715
```

Figure 4: 2.2.3 code &result

2.3 Locality Sensitive Hashing [15 Points]

implement the LSH algorithm. It receives signature matrix and b(the number of bands), r(the number of rows in each band) return the candidate paris of the similar items

2.3.1 Implement the `lsh` function

It was implemented by dividing the matrix of signatures by the number of bands, and putting the vector of each sub-signature into a dictionary as the key.

3.1 Min-Hash based LSH

```
In [322]: def lsh(signatures, b, r):
              """
                  b bands
                  r rows per band

                  각 band는 k개의 버킷을 가지고 있고, 밴드별로 할당된 signarue의 열(r 길이)을 k개의 버킷에 해
                  해싱을 할때 하나의 밴드라도 동일한 버킷에 들어간다면 두 컬럼은 비슷하다고 본다.

                  여기서 b가 커지면 similarity threshold가 낮다. (higher false positive)

                  즉, b가 커지면 r이 작아져서 문서들이 비슷하다고 판단될 확률이 높아진다고 볼 수 있음.

              """
              candidatePairs = set()

              M = signatures.shape[0]  # The number of min-hash functions
              C = signatures.shape[1]  # The number of movies

              assert M == b * r

              #### implement here ####

              # band 수로 나누기
              for band in np.split(signatures, b):

                  # band 별로 bucket 초기화
                  bucket = defaultdict(list)

                  # bucket에 매칭이 됨.
          #         print(len(band.T))
                  for idx, item in enumerate(band.T):
                      key = tuple(item) # hash를 조금 생각 해봐야할지도..
                      bucket[key].append(idx)

          #       print(len(bucket.keys()))
                  # add to candidatePairs
                  for _,matched in bucket.items():

                      for pair in itertools.combinations(matched, 2):
                          candidatePairs.add(pair)

              ##########################


              return candidatePairs
```

Figure 5: 2.3 lsh code

2.4 Analysis [15 Points]

Modified Point from TAs' pre-implemented code:
In the computeConditionPositives part of the code, I added the following code to get the ground-truth item pair that exceeds threshold s.

```
            true_sim = jaccard_similarity(shingle1, shingle2)
            if true_sim >= s:
                numConditionPositives += 1

                # add positive pairs (i, j), 추가 포인트
                positivePairs.add((i, j))

print(f"The number of condition positives: {numConditionPositives} when s={s}")
```

Figure 6: 2.4 modified code

2.4.1 Implement the `query_analysis` function to compute the query time, precision, recall, and F1 score as b and r change.

```
# Return query_time, precision, recall, F1 score
def query_analysis(signatures, b, s, numConditionPositives):

    #### Implement here ####

    start = time.time()
    candidatePairs = lsh(signatures, b, r=len(signatures)//b)
    query_time = time.time() - start


    recall = len(positivePairs.intersection(candidatePairs))/len(positivePairs)
    precision = len(positivePairs.intersection(candidatePairs)) /len(candidatePairs)

    f1 = 2 * (precision * recall) / (precision + recall)


    ##########################


    return query_time, precision, recall, f1
```

Figure 7: 2.4.1 query analysis code

2.4.2 Attach your query time graph using TAs' pre-implemented code & Explain the results in terms of the computational complexity.

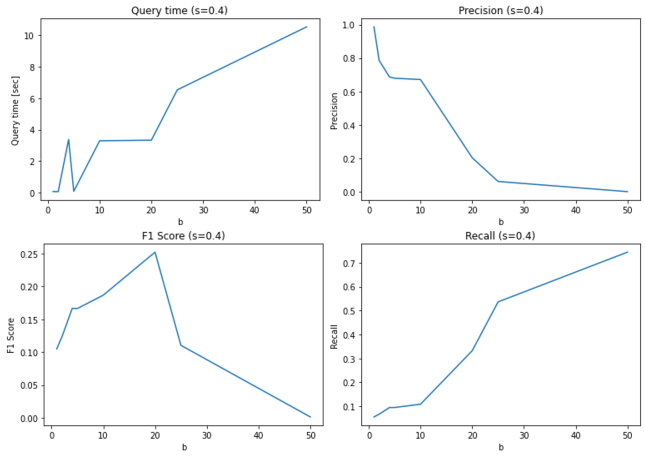

Figure 8: 2.4.2 graph results

In the first graph in Fig. 8, you can see that the query_time increases as B increases.
The computational complexity of the lsh function is O(BN).
Hash matching is O(1), And since it puts item N into the bucket for B times, it takes a total O(BN) time. In other words, the query speed increases as B increases.

2.4.3 Attach your precision, recall, and f1-score graphs. & Explain the results in detail.

In the graphs in Fig. 8, As b increases, precision decreases and recall increases. f1-score increases, showing the best score at 20, and then decreases again.

In the terms of the precision measure, as b gets smaller, r gets larger, which means that vectors are less likely to match and items in a bucket are less likely to conflict.
Therefore, as b gets smaller, precision performs better because only pairs of items with very similar vector will be matched. So, precision scores best when b is 1.

In recall's terms, as b gets larger, r gets smaller, so the likelihood of items colliding in the bucket increases. Therefore, as b increases, the probability of items being grouped into similar groups increases, and the probability of items in the true similarity list being retrieved increases, which increases recall. Therefore, recall shows the best results when b is 50.

The f1-score is the harmonic mean of precision and recall, which increases as b increases up to 20 and then decreases thereafter. So from an f1-score perspective, the best result is when b is 20.

2.5 Case Study [10 Points]

search for the 10 movies that are estimated to be the most similar to it.
setting :
s = 0.3, b = 25
python code:
To get the Ground-truth list with weighted jaccard simmiarity (it is equivalant with jaccard sim with shingle)

```
sorted_list = sorted([(i, jaccard_similarity(set(item_to_shingles[query])
    , set(item_to_shingles[i]))) for i in range(numItems) if i != query ], key=lambda x : x[1]
    , reverse=True)
```

In the jaccard similarity above, items with a score of s or higher were treated as GT.

2.5.1 Star Wars: Episode VI - Return of the Jedi (1983) - index 37

```
----------------------------------------
idx:  37 title:  Star Wars: Episode VI - Return of the Jedi (1983) : precision@%d : 1.0

similar item list
1 : 0.5930796012403741, Star Wars: Episode V - The Empire Strikes Back (1980)
2 : 0.5865687451451169, Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
3 : 0.4483934754004366, Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
4 : 0.4245875644192358, Back to the Future (1985)
5 : 0.40740770976537627,  The (1984)
6 : 0.39834999689069767,  The (1999)
7 : 0.3722360108303249, Men in Black (1997)
8 : 0.3536450367165516, Aliens (1986)
9 : 0.35148015837815605, Alien (1979)
10 : 0.3494585374008557, Terminator 2: Judgment Day (1991)
```

The item had many exact similar matches, returned a sufficient number in the LSH results, and had a precision score of 1.0 for precision@10.

2.5.2 Wallace & Gromit: A Grand Day Out (1989) - index 1,459

```
----------------------------------------
idx:  1459 title:  Wallace & Gromit: A Grand Day Out (1989) : precision@%d : 0.3333333333333333

similar item list
1 : 0.44302151725931 22, Wallace & Gromit: A Close Shave (1995)
2 : 0.14377402141870965,  A (1988)
3 : 0.08985641500694766,  The (1988)
```

Based on the 0.3 threshold, there were 2 GTs, of which 1 item was retrieved. The LSH algorithm returned 3 candidates, with a precision of 0.3.

2.5.3 Toy Story (1995) - index 96

```
----------------------------------------
idx:  96 title:  Toy Story (1995) : precision@%d : 0.9

similar item list
1 : 0.3739034164069363, Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
2 : 0.3454862907397827, Star Wars: Episode VI - Return of the Jedi (1983)
3 : 0.3267358304473893, Mission: Impossible (1996)
4 : 0.3195196095805445, Aladdin (1992)
5 : 0.31898654286386685, Forrest Gump (1994)
6 : 0.31500530709053504, Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
7 : 0.3131560733648633, Jurassic Park (1993)
8 : 0.3095433571996818, Fargo (1996)
9 : 0.30241065139845047, Groundhog Day (1993)
10 : 0.2981312631977248, Apollo 13 (1995)
```

There were 15 GTs and 9 out of the top 10 were matched with a GT, resulting in a precision score of 0.9.

2.5.4 Harry Potter and the Goblet of Fire (2005) - index 1,239

```
----------------------------------------
idx:  1239 title:  Harry Potter and the Goblet of Fire (2005) : precision@%d : 0.3

similar item list
1 : 0.45226533800616475, Harry Potter and the Prisoner of Azkaban (2004)
2 : 0.34902324916967253, Harry Potter and the Chamber of Secrets (2002)
3 : 0.30341815991567533, Pirates of the Caribbean: Dead Man's Chest (2006)
4 : 0.26438507261474353, Batman Begins (2005)
5 : 0.15067119701442655,  The (2004)
6 : 0.13406593406593406, Jarhead (2005)
7 : 0.10991012873451543, Mulan (1998)
8 : 0.09234468416948272, Cloverfield (2008)
9 : 0.08926624978486604,  The (2004)
10 : 0.08580984477124183, My Cousin Vinny (1992)
```

6 GTs exist, of which three were retrieved. From a list of 10 search results, 3 were matched, giving a precision of 0.3.