ASSIGNMENT 2

Course: CSL7110
Assignment: Min-Hashing and LSH
Name: Shasank Dixit
Roll Number: P25CS0008
Github Link: https://github.com/Dixitji26/CSL7110-Minhash-LSH.git

1.Introduction

This assignment explores similarity detection using k-grams, Jaccard similarity, Min-Hashing, and Locality Sensitive Hashing (LSH). The goal is to approximate document similarity efficiently and apply these techniques to both small text documents and the MovieLens 100k dataset.

We compare exact Jaccard similarity with MinHash estimates and evaluate LSH performance in terms of false positives and false negatives.

PART1A:

Q)For each value of (t), report the approximate Jaccard similarity between the pair of documents D1 and D2, estimating the Jaccard similarity:
You should report 5 numbers.

Code:

```
t_values = [20, 60, 150, 300, 600]

print("Exact Jaccard (3-grams) D1-D2:",
      jaccard(char3["D1"], char3["D2"]))

print("\nApproximate Jaccard using MinHash:")

for t in t_values:
    hash_funcs = generate_hash_functions(t)

    sig1 = minhash_signature(char3["D1"], hash_funcs)
    sig2 = minhash_signature(char3["D2"], hash_funcs)

    approx_sim = signature_similarity(sig1, sig2)

    print(f"t = {t}: {approx_sim:.4f}")
```

Output:

```
Exact Jaccard (3-grams) D1-D2: 0.977979274611399

Approximate Jaccard using MinHash:
t = 20: 0.9500
t = 60: 0.9833
t = 150: 0.9867
t = 300: 0.9867
t = 600: 0.9817
```

PART1B:

Q)Compute the Jaccard similarity between all pairs of documents for each type of k-gram. You should report:

Code:

```
from itertools import combinations

pairs = list(combinations(docs.keys(), 2))

print("===== Character 2-grams =====")
for d1, d2 in pairs:
    print(f"{d1}-{d2}: {jaccard(char2[d1], char2[d2]):.4f}")

print("\n===== Character 3-grams =====")
for d1, d2 in pairs:
    print(f"{d1}-{d2}: {jaccard(char3[d1], char3[d2]):.4f}")

print("\n===== Word 2-grams =====")
for d1, d2 in pairs:
    print(f"{d1}-{d2}: {jaccard(word2[d1], word2[d2]):.4f}")
```

Output:

```
===== Character 2-grams =====
D1-D2: 0.9811
D1-D3: 0.8157
D1-D4: 0.6444
D2-D3: 0.8000
D2-D4: 0.6413
D3-D4: 0.6530

===== Character 3-grams =====
D1-D2: 0.9780
D1-D3: 0.5804
D1-D4: 0.3051
D2-D3: 0.5680
D2-D4: 0.3059
D3-D4: 0.3121

===== Word 2-grams =====
D1-D2: 0.9408
D1-D3: 0.1823
D1-D4: 0.0302
D2-D3: 0.1737
D2-D4: 0.0303
D3-D4: 0.0161
```

Smaller k gives higher similarity

Word 2-grams produce lower similarity

D1 and D2 show highest similarity

PART2A:

Q)Using 3-grams to build a min-hash signature for documents D1 and D2 using:
t = 20, 60, 150, 300, 600
hash functions.
For each value of (t), report the approximate Jaccard similarity between the pair of documents D1 and D2, estimating the Jaccard similarity.
You should report 5 numbers

```
Exact Jaccard (3-grams) D1-D2: 0.977979274611399

Approximate Jaccard using MinHash:
t = 20: 0.9500
t = 60: 0.9833
t = 150: 0.9867
t = 300: 0.9867
t = 600: 0.9817
```

As t increases, the MinHash estimate converges to the exact Jaccard similarity. Small t values produce unstable estimates, while larger t values produce more accurate results.

PART2B:

Q) What is good t?

- t = 150 or 300 is good tradeoff.
- 600 is more accurate but slower**.**

t = 300 provides a good balance between computational cost and accuracy.

PART3A:

Code:

r = 5
b = 32
candidates = lsh_candidates(signatures, r, b)

Given that the total number of hash functions is:

t = 160

We must choose values of r (rows per band) and b (number of bands) such that:

r × b = 160

To achieve good separation near the similarity threshold τ = 0.7, we use the threshold approximation formula:

s ≈ (1/b)^(1/r)

We test possible combinations of r and b that satisfy r × b = 160.

For the choice:

r = 5
b = 32

We compute the approximate threshold:

(1/32)^(1/5) ≈ 0.707

This value is very close to the desired threshold τ = 0.7. Therefore, the S-curve transitions sharply around 0.7. Document pairs with similarity above 0.7 will have a high probability of becoming candidate pairs, while pairs below 0.7 will have a low probability.

Output:

```
Candidate pairs from LSH:
{('D2', 'D3'), ('D1', 'D2'), ('D1', 'D3')}
```

Part 3B:

Q)Using your choice of (r) and (b) and (f(·)), what is the probability of each pair of the four documents (using 3-grams) being estimated to have a similarity greater than (τ)? Report 6 numbers.

Code:

```
pairs = list(combinations(docs.keys(), 2))

for d1, d2 in pairs:
    s = jaccard(char3[d1], char3[d2])
    prob = lsh_probability(s, r, b)
    print(f"{d1}-{d2}: {prob:.4f}")
```

Output:

```
Probability of being candidate (based on exact Jaccard):
D1-D2: 1.0000
D1-D3: 0.8869
D1-D4: 0.0812
D2-D3: 0.8579
D2-D4: 0.0823
D3-D4: 0.0906
```

Different configuration of rows and bands along with different similarity values:

Code:

```
def lsh_probability(s, r, b):
    return 1 - (1 - s**r)**b

configs = [
    (4, 40),
    (5, 32),
    (8, 20),
    (10, 16),
    (16, 10)
]

similarities = [0.5, 0.6, 0.7, 0.8, 0.9]

for r, b in configs:
    print(f"\nConfiguration: r={r}, b={b}")
    for s in similarities:
        prob = lsh_probability(s, r, b)
        print(f"s={s} → {prob:.4f}")
```

Output:

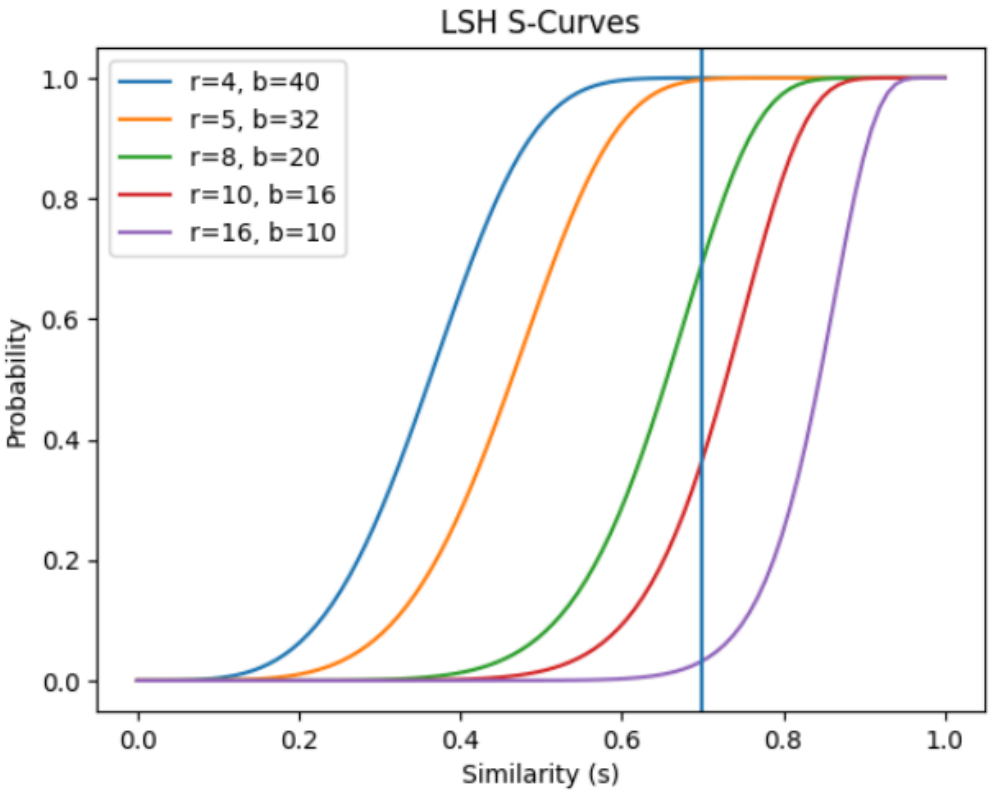
```
Configuration: r=4, b=40
s=0.5 → 0.9243
s=0.6 → 0.9961
s=0.7 → 1.0000
s=0.8 → 1.0000
s=0.9 → 1.0000

Configuration: r=5, b=32
s=0.5 → 0.6379
s=0.6 → 0.9250
s=0.7 → 0.9972
s=0.8 → 1.0000
s=0.9 → 1.0000

Configuration: r=8, b=20
s=0.5 → 0.0753
s=0.6 → 0.2874
s=0.7 → 0.6950
s=0.8 → 0.9746
s=0.9 → 1.0000

Configuration: r=10, b=16
s=0.5 → 0.0155
s=0.6 → 0.0925
s=0.7 → 0.3677
s=0.8 → 0.8376
s=0.9 → 0.9990

Configuration: r=16, b=10
s=0.5 → 0.0002
s=0.6 → 0.0028
s=0.7 → 0.0327
s=0.8 → 0.2484
s=0.9 → 0.8712
```



From the plotted S-curves, the configuration r=5 and b=32 provides the best separation near the threshold τ=0.7. The transition from low to high probability occurs close to 0.7, ensuring that document pairs with similarity above 0.7 are likely to be selected while minimizing false positives below the threshold. Other configurations either shift the transition too early (r=4, b=40) or too late (r≥8), leading to increased false positives or false negatives.

Therefore, r=5 and b=32 is selected as the optimal configuration.

Q) For each value, output the pairs that have an estimated similarity of at least 0.5 and report the number of false positives and false negatives that you obtain

→Code:

```
from itertools import combinations


def jaccard(set1, set2):
    return len(set1 & set2) / len(set1 | set2)
```

```
exact_similar_pairs = []

users = list(user_movies.keys())

for u1, u2 in combinations(users, 2):
    sim = jaccard(user_movies[u1], user_movies[u2])
    if sim >= 0.5:
        exact_similar_pairs.append((u1, u2))

print("Number of exact similar pairs (>=0.5):", len(exact_similar_pairs))
```

Output:

Number of exact similar pairs (>=0.5): 10

Q) For the false positives and negatives, report the averages for 5 different runs.

Code:

```
def evaluate_minhash(t, runs=5):
    fp_total = 0
    fn_total = 0

    for _ in range(runs):
        signatures = compute_signatures(t)
        estimated_pairs = []
```

```
        for u1, u2 in combinations(users, 2):
            sim = signature_similarity(signatures[u1], signatures[u2])
            if sim >= 0.5:
                estimated_pairs.append((u1, u2))

        estimated_set = set(estimated_pairs)
        exact_set = set(exact_similar_pairs)

        false_positives = len(estimated_set - exact_set)
        false_negatives = len(exact_set - estimated_set)

        fp_total += false_positives
        fn_total += false_negatives

    print(f"\nt = {t}")
    print("Average False Positives:", fp_total / runs)
    print("Average False Negatives:", fn_total / runs)
```

Output:

```
t = 50
Average False Positives: 78.8
Average False Negatives: 1.8

t = 100
Average False Positives: 20.2
Average False Negatives: 2.4

t = 200
Average False Positives: 6.8
Average False Negatives: 1.8
```

***Increasing the number of hash functions improves estimation accuracy. With t=200, both false positives and false negatives significantly decrease compared to t=50.**

Q) Experiment with:
(r = 5, b = 10) for the table with the 50 hash functions
(r = 5, b = 20) for the table with the 100 hash functions
(r = 5, b = 40) and (r = 10, b = 20) for the table with the 200 hash functionsReport the number of false positives and false negatives, taking the average over 5 runs. How do these numbers change if we want a similarity of at least 0.8?

Code:

```
Configuration: t=50, r=5, b=10, threshold=0.6
Avg False Positives: 599.8
Avg False Negatives: 0.2

Configuration: t=100, r=5, b=20, threshold=0.6
Avg False Positives: 1327.4
Avg False Negatives: 0.2

Configuration: t=200, r=5, b=40, threshold=0.6
Avg False Positives: 2451.8
Avg False Negatives: 0.0

Configuration: t=200, r=10, b=20, threshold=0.6
Avg False Positives: 2.6
Avg False Negatives: 1.6

Configuration: t=50, r=5, b=10, threshold=0.8
Avg False Positives: 503.6
Avg False Negatives: 0.0

Configuration: t=100, r=5, b=20, threshold=0.8
Avg False Positives: 1044.4
Avg False Negatives: 0.0

Configuration: t=200, r=5, b=40, threshold=0.8
Avg False Positives: 2149.8
Avg False Negatives: 0.0

Configuration: t=200, r=10, b=20, threshold=0.8
Avg False Positives: 6.0
Avg False Negatives: 0.0
```

```
Configuration: t=50, r=5, b=10, threshold=0.6
Avg False Positives: 599.8
Avg False Negatives: 0.2

Configuration: t=100, r=5, b=20, threshold=0.6
Avg False Positives: 1327.4
Avg False Negatives: 0.2

Configuration: t=200, r=5, b=40, threshold=0.6
Avg False Positives: 2451.8
Avg False Negatives: 0.0

Configuration: t=200, r=10, b=20, threshold=0.6
Avg False Positives: 2.6
Avg False Negatives: 1.6

Configuration: t=50, r=5, b=10, threshold=0.8
Avg False Positives: 503.6
Avg False Negatives: 0.0

Configuration: t=100, r=5, b=20, threshold=0.8
Avg False Positives: 1044.4
Avg False Negatives: 0.0

Configuration: t=200, r=5, b=40, threshold=0.8
Avg False Positives: 2149.8
Avg False Negatives: 0.0

Configuration: t=200, r=10, b=20, threshold=0.8
Avg False Positives: 6.0
Avg False Negatives: 0.0
```

Observations

- **Increasing t improves accuracy.**
- **Larger r reduces false positives but increases false negatives.**
- **Higher similarity threshold (0.8) increases false negatives.**
- **LSH significantly reduces comparisons compared to brute force.**

Conclusion:

This assignment demonstrates how MinHash effectively approximates Jaccard similarity and how LSH enables scalable similarity detection by reducing comparisons. Increasing the number of hash functions improves accuracy, while LSH parameters control the tradeoff between false positives and false negatives.

The experiments confirm that careful parameter selection is crucial for achieving good separation at a desired similarity threshold.