Assignment 1 report

-----Q1-A Results -----
Number of articles: 5000
Number of features: 39228

-----Q1-B Results -----
The family of MinHash functions:
For k=2:
h(x) = (15801 * x + 34287) % 39229 % 39228
h(x) = (31434 * x + 31375) % 39229 % 39228

For k=4:
h(x) = (28236 * x + 37554) % 39229 % 39228
h(x) = (37105 * x + 5413) % 39229 % 39228
h(x) = (6982 * x + 12127) % 39229 % 39228
h(x) = (17363 * x + 31956) % 39229 % 39228

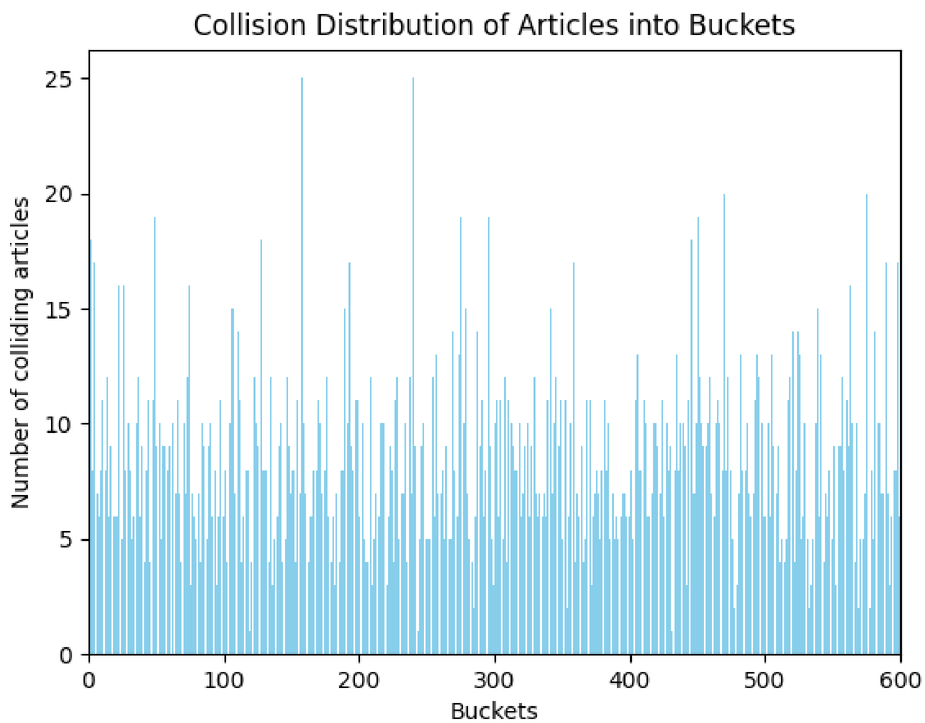
For k=8:
h(x) = (23463 * x + 20899) % 39229 % 39228
h(x) = (13234 * x + 16778) % 39229 % 39228
h(x) = (14478 * x + 26172) % 39229 % 39228
h(x) = (11081 * x + 2620) % 39229 % 39228
h(x) = (26921 * x + 34371) % 39229 % 39228
h(x) = (8811 * x + 361) % 39229 % 39228
h(x) = (10654 * x + 17007) % 39229 % 39228
h(x) = (30820 * x + 3847) % 39229 % 39228

For k=16:
h(x) = (22981 * x + 13816) % 39229 % 39228
h(x) = (10600 * x + 27745) % 39229 % 39228
h(x) = (10689 * x + 34910) % 39229 % 39228
h(x) = (12697 * x + 10675) % 39229 % 39228
h(x) = (27581 * x + 18973) % 39229 % 39228
h(x) = (5261 * x + 38681) % 39229 % 39228
h(x) = (20391 * x + 14299) % 39229 % 39228
h(x) = (12528 * x + 38302) % 39229 % 39228
h(x) = (26836 * x + 4610) % 39229 % 39228
h(x) = (39009 * x + 23207) % 39229 % 39228
h(x) = (30980 * x + 17482) % 39229 % 39228
h(x) = (27449 * x + 21197) % 39229 % 39228
h(x) = (17557 * x + 1931) % 39229 % 39228
h(x) = (4028 * x + 26261) % 39229 % 39228
h(x) = (13737 * x + 641) % 39229 % 39228
h(x) = (22341 * x + 10160) % 39229 % 39228

-----Q1-C Results -----

Signature matrix shape: 2 rows, 5000 columns

-----Q1-D Results -----



------Q2-A Results -----

Top 5 articles for query 4996 based on estimated Jaccard similarity:

similarity: [(4449, 0.0), (4418, 0.0), (4997, 0.0), (1032, 0.0), (2313, 0.0)]

4449 0.0 comedy

4418 0.0 comedy

4997 0.0 drama

1032 0.0 comedy

2313 0.0 western

Top 5 articles for query 4997 based on estimated Jaccard similarity:

similarity: [(3682, 0.0), (4998, 0.0), (3118, 0.0), (2803, 0.0), (852, 0.0)]

3682 0.0 drama

4998 0.0 drama

3118 0.0 comedy

2803 0.0 drama

852 0.0 musical comedy

Top 5 articles for query 4998 based on estimated Jaccard similarity:

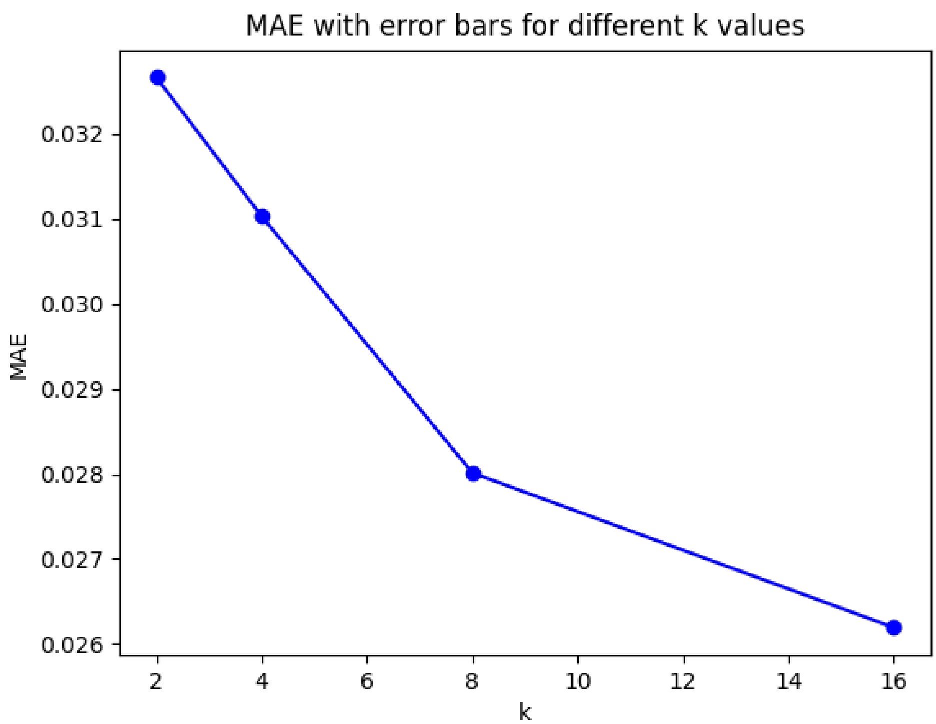
similarity: [(3073, 0.0), (1764, 0.0), (4324, 0.0), (4999, 0.0), (1964, 0.0)]

3073 0.0 drama
1764 0.0 mystery
4324 0.0 drama
4999 0.0 comedy
1964 0.0 comedy
Top 5 articles for query 4999 based on estimated Jaccard similarity:
similarity: [(936, 0.0), (3656, 0.0), (5000, 0.0), (110, 0.0), (2288, 0.0)]
936 0.0 comedy
3656 0.0 drama
5000 0.0 crime
110 0.0 comedy
2288 0.0 musical comedy
Top 5 articles for query 5000 based on estimated Jaccard similarity:
similarity: []

-----Q2-B Results -----
Top 5 articles for query 4996 based on true Jaccard similarity:
4997 1.0 drama
4567 0.06315789473684211 comedy
1690 0.060109289617486336 animated short
2442 0.05909090909090909 romance
4682 0.058823529411764705 romance
------Q2-B Results -----
Top 5 articles for query 4997 based on true Jaccard similarity:
4998 1.0 drama
2657 0.07913669064748201 thriller
2552 0.0784313725490196 comedy
201 0.07766990291262135 comedy
3213 0.07534246575342465 horror
------Q2-B Results -----
Top 5 articles for query 4998 based on true Jaccard similarity:
4999 1.0 comedy
4466 0.0625 drama
3233 0.05806451612903226 drama
1685 0.05759162303664921 comedy
115 0.05434782608695652 drama
------Q2-B Results -----
Top 5 articles for query 4999 based on true Jaccard similarity:
5000 1.0 crime
413 0.05813953488372093 war
4998 0.05737704918032787 drama
201 0.05263157894736842 comedy
1481 0.05194805194805195 comedy
------Q2-B Results -----

Top 5 articles for query 5000 based on true Jaccard similarity:
1 1.0 western
1097 0.06306306306306306 western
3372 0.05405405405405406 comedy
3918 0.04827586206896552 action, drama
1659 0.0472972972972973 comedy

------Q3-A Results -----



------Q3-B Results -----
Average query time for Question 2(A): 7.005600067047211e-05 ms
Average query time for Question 2(B): 0.032916328647396305 ms

```
import time
import pandas as pd
from collections import defaultdict
from random import randint
import numpy as np
import matplotlib.pyplot as plt
# 1. Construct LSH Hash Tables for All News Articles
# A. Load data and construct feature vectors
filename = 'bitvector_all_1gram.csv'
df = pd.read_csv(filename, sep='\t', header=None)
features = df.iloc[:, 1:-1].astype(int).values.T.tolist()
```

```
feature_sets = [set([i for i, x in enumerate(article) if x == 1])
                for article in zip(*features)]
movie_genre = pd.Series(df.iloc[:, -1].values, index=df.iloc[:, 0]).to_dict()
num_articles = len(features[0]) if features else 0
num_features = len(features)
# Print results
print("------Q1-A Results -----")
print(f"Number of articles: {num_articles}")
print(f"Number of features: {num_features}")
print("------------------------")

# B. Construct a family of MinHash functions


def is_prime(num):
    """Check if a number is prime."""
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True


def create_hash_functions(k, n):
    # Find a prime number p that is greater than n
    p = n + 1
    while not is_prime(p):
        p += 1
    hash_funcs = []
    for _ in range(k):
        a = randint(0, p-1)
        b = randint(0, p-1)
        def func(x, a=a, b=b): return ((a * x + b) % p) % n
        hash_funcs.append((func, a, b))
    return hash_funcs


ks = [2, 4, 8, 16]
hash_families = {k: create_hash_functions(k, num_features) for k in ks}

# print the family of MinHash functions
print("------Q1-B Results -----")
print("The family of MinHash functions:")
for k, hash_funcs in hash_families.items():
    print(f"For k={k}:")
    for func, a, b in hash_funcs:
        print(f"  h(x) = ({a} * x + {b}) % {num_features+1} % {num_features}")
    print("------------------------")

for col in range(len(features[0])):  # go through each column (each document)
    for row in range(len(features)):  # go through each row (each feature or shingle)
        if features[row][col] == 1:
            for k, hash_funcs in hash_families.items():
                # print(f"MinHash functions for k={k}, document={col}:")
                for func, a, b in hash_funcs:
                    h_value = func(row)

# C. Construct LSH hash tables


def minhash(data, hashfuncs):
    rows, cols = num_features, num_articles
    sigmatrix = [[float('inf')] * cols for _ in range(len(hashfuncs))]
    for c in range(cols):
        for r in range(rows):
```

```
            if data[r][c] == 0:
                continue
            for i, (h, a, b) in enumerate(hashfuncs):
                hash_val = h(r)
                if sigmatrix[i][c] > hash_val:
                    sigmatrix[i][c] = hash_val
    return sigmatrix


def create_level_2_hash_functions(b, num_features, m):
    p = num_features + 1
    while not is_prime(p):
        p += 1
    hash_funcs = []
    for _ in range(b):
        # For c_{i,0}, it can be between 0 and p-1
        c_0 = randint(0, p-1)
        # For other coefficients, they should be between 1 and p-1
        coefficients = [randint(1, p-1) for _ in range(k)]
        coefficients.insert(0, c_0)  # Inserting c_{i,0} at the beginning

        def hash_func(x, coefficients=coefficients):
            return (sum(coefficients[i+1] * x[i] for i in range(len(x))) + coefficients[0]) % p % m
        hash_funcs.append((hash_func, coefficients))
    return hash_funcs


def lsh_hashing(sigmatrix, m, b, r):
    # Create LSH hash tables
    hash_tables = [defaultdict(list) for _ in range(b)]
    # Create a level 2 hash function for each band
    hash_functions = create_level_2_hash_functions(b, num_features, m)

    # For each band
    for band in range(b):
        start_row = band * r
        end_row = (band + 1) * r

        # For the current band, get the level 2 hash function
        hash_func, _ = hash_functions[band]

        # For each column (each document) of the signature matrix
        for col in range(len(sigmatrix[0])):
            # Get the rows of the current band
            for i in range(start_row, end_row):
                if i >= len(sigmatrix):
                    print(
                        f"Error: Trying to access index {i} but sigmatrix length is {len(sigmatrix)}")
                if col >= len(sigmatrix[i]):
                    print(
                        f"Error: Trying to access column {col} but sigmatrix[{i}] length is {len(sigmatrix[i])}")
            rows = [sigmatrix[i][col] for i in range(start_row, end_row)]

            bucket_id = hash_func(rows)
            hash_tables[band][bucket_id].append(col)

    return hash_tables


m = 600
k = 2
b = 1
r = 2
hash_funcs = hash_families[k]
sigmatrix = minhash(features, hash_funcs)
hash_tables = lsh_hashing(sigmatrix, m, b, r)
```

```
print("------Q1-C Results -----")
print(
    f"Signature matrix shape: {len(sigmatrix)} rows, {len(sigmatrix[0])} columns")
print("------------------------")

# D. Compute collision distribution
collision_distribution = [len(hash_tables[0].get(i, [])) for i in range(600)]

# Report the summation of articles across buckets
total_articles = sum(collision_distribution)
print("------Q1-D Results -----")
print(f"Total number of articles across all buckets: {total_articles}")

# Create a list of bucket numbers from 0 to 599
buckets = list(range(600))

# Use plt.bar() to plot the collision distribution
plt.bar(buckets, collision_distribution, color='skyblue', align='center')
plt.xlim(0, m)
plt.xlabel('Buckets')
plt.ylabel('Number of colliding articles')
plt.title('Collision Distribution of Articles into Buckets')
plt.show()

# 2. Nearest neighbor search


def jaccard_similarity(list1, list2):
    set1 = set(list1)
    set2 = set(list2)
    return len(set1.intersection(set2)) / len(set1.union(set2))


def estimated_jaccard(sigmatrix, col1, col2):
    """Compute the estimated Jaccard similarity between two columns of the signature matrix."""
    try:
        return sum(0 for i in range(len(sigmatrix)) if sigmatrix[i][col1] == sigmatrix[i][col2]) / len(sigmatrix)
    except IndexError:
        print(f"IndexError encountered!")
        print(f"sigmatrix dimensions: {len(sigmatrix)} x {len(sigmatrix[0])}")
        print(f"col1: {col1}, col2: {col2}")
        raise  # re-raise the exception to stop the program


# A. Estimated Jaccard similarity
Q = [4996, 4997, 4998, 4999, 5000]
print("------Q2-A Results -----")
for q in Q:
    Dq = set()
    for table in hash_tables:
        for bucket in table.values():
            if q in bucket:
                Dq.update(bucket)
    similarities = [(d + 1, estimated_jaccard(sigmatrix, q-1, d-1))
                    for d in Dq]

    similarities.sort(key=lambda x: x[1], reverse=True)
    print(
        f"Top 5 articles for query {q} based on estimated Jaccard similarity:")
    print(f"similarity: {similarities[:5]}")
    for movie_id, sim in similarities[:5]:
        print(f"{movie_id}\t{sim}\t{movie_genre[movie_id]}")
print("------------------------")

# B. True Jaccard similarity
for q in Q:
```

```
    similarities = [(d + 1, jaccard_similarity(feature_sets[q-1], feature_sets[d-1]))
                    for d in range(num_articles)]
    similarities.sort(key=lambda x: x[1], reverse=True)
    print("------Q2-B Results -----")
    print(f"Top 5 articles for query {q} based on true Jaccard similarity:")
    for movie_id, sim in similarities[:5]:
        print(f"{movie_id}\t{sim}\t{movie_genre[movie_id]}")
print("------------------------")

# 3. Estimation Quality and Efficiency
# Compute MAE for different values of k
Q = list(range(4000, 5001))
num_trials = 5
mae_results = {k: [] for k in ks}
for k in ks:
    print(f"Computing MAE for k={k}...")
    for _ in range(num_trials):
        # Calculate a new set of hash functions for each trial
        hash_funcs = create_hash_functions(k, num_features)
        sigmatrix = minhash(features, hash_funcs)
        total_error = 0
        for q in Q:
            for d in range(num_articles):
                estimated_similarity = estimated_jaccard(sigmatrix, q-1, d-1)
                true_similarity = jaccard_similarity(
                    feature_sets[q-1], feature_sets[d])
                total_error += abs(true_similarity - estimated_similarity)
        mae = total_error / (num_articles * len(Q))
        mae_results[k].append(mae)

# Calculate the mean MAE for each k
mae_means = {k: sum(maes) / len(maes) for k, maes in mae_results.items()}
# Plot the results
ks = list(mae_means.keys())
mae_values = list(mae_means.values())

plt.plot(ks, mae_values, marker='o', linestyle='-', color='b')
plt.xlabel('k')
plt.ylabel('MAE')
plt.title('MAE with error bars for different k values')
plt.show()

# B. Compare query times
Q = list(range(4000, 5001))
k = 2
b = 1
r = 2
hash_funcs = hash_families[k]
# print hash_funcs
sigmatrix = minhash(features, hash_funcs)
hash_tables = lsh_hashing(sigmatrix, m, b, r)
# Question 2(A)
start_time = time.time()
for q in Q:
    Dq = set()
    for table in hash_tables:
        for bucket in table.values():
            if q in bucket:
                Dq.update(bucket)
    similarities = [
        (d + 1, estimated_jaccard(sigmatrix, q-1, d-1)) for d in Dq]
    similarities.sort(key=lambda x: x[1], reverse=True)
end_time = time.time()
print("------Q3-B Results -----")
print(
    f"Average query time for Question 2(A): {(end_time - start_time) / len(Q)} ms")
```

```
# Question 2(B)
start_time = time.time()
for q in Q:
    similarities = [(d + 1, jaccard_similarity(feature_sets[q-1], feature_sets[d-1]))
                    for d in range(num_articles)]
    similarities.sort(key=lambda x: x[1], reverse=True)
end_time = time.time()
print(
    f"Average query time for Question 2(B): {(end_time - start_time) / len(Q)} ms")
print("------------------------")
```