Assignment 2 - Document Similarity and Hashing Pranav Rajan

January 28, 2020

[1]:

```
# import some stuff
import numpy as np
from math import inf
import time as time
import hashlib
import sys
```

[2]:

```
# read and store files
def document_reader(file_name):
    f = open(file_name, "r")
    if (f.mode == "r"):
        contents = f.read()
        return contents
```

[3]:

```
# a set of functions that generates k grams based on different criteria and the␣
 ↪jaccardian similiary between documents

# function that generates 2 character k grams
def generate_2_character_gram(document):
    # set to ensure that there are no duplicates
    two_char_gram_set = set()

    for i in range(0, len(document) - 1):
        two_char_string = ""
        char_1 = document[i]
        char_2 = document[i + 1]
        two_char_string += char_1 + char_2
        two_char_gram_set.add(two_char_string)
    return two_char_gram_set

# function that generates 3 character k grams
def generate_3_character_gram(document):
    # set to ensure that there are no duplicates
    three_char_gram_set = set()

    # set to ensure that there are no duplicates
    three_char_gram_set = set()
    for j in range(0, len(document) - 2):
        three_char_string = ""
        char_1 = document[j]
        char_2 = document[j + 1]
        char_3 = document[j + 2]
        three_char_string += char_1 + char_2 + char_3
        three_char_gram_set.add(three_char_string)

    return three_char_gram_set

# function that generates 2 word k grams
def generate_2_word_gram(document):
    # split the document into tokens (words)
    document_words = document.split()

    # set to ensure that there are no duplicates
    two_word_gram_set = set()

    # use the same logic from generate_2_character_gram function to construct␣
 ↪the grams
    for h in range(0, len(document_words) - 1):
        two_word_string = ""
        word_1 = document_words[h]
        word_2 = document_words[h + 1]
        two_word_string += word_1 + " " + word_2
        two_word_gram_set.add(two_word_string)

    return two_word_gram_set

def jaccardian_similarity(a, b):
    # compute the intersection of a and b
    a_intersect_b = a.intersection(b)

    # compute the magnitude of the intersection of a intersect b
    a_intersect_b_magnitude = len(a_intersect_b)

    # compute the union of a and b
    a_union_b = a.union(b)

    # compute the magnitude of the union of a and b
    a_union_b_magnitude = len(a_union_b)

    similarity = a_intersect_b_magnitude / a_union_b_magnitude

    return similarity
```

[4]:

```python
# a set of functions for experimenting with min hashing

# function that generates a bunch of hashed k grams of different lengths
def min_hashing(k_grams, m, t):
    # generate a number of salt values to generate t number of hash functions
    salt_list = []
    salt_list.extend(range(t))

    # list that stores k number of lists containing t different hash functions
    hash_vector_list = []

    for k in k_grams:
        # list for storing t different hash functions
        hash_vector = []

        # salt and hash some stuff
        for p in range(len(salt_list)):
            salt_value = ""
            salt_value += str(salt_list[p])

            # k_gram + salt for hashing
            salted_string = k + salt_value

            # hash salted string
            hash_func = hashlib.md5()
            hash_func.update(salted_string.encode())
            hex_hash = hash_func.hexdigest()
            hash_value = int(hex_hash, 16) % m
            hash_vector.append(hash_value)

        hash_vector_list.append(hash_vector)

    return hash_vector_list

# function that takes a list of lists containing hash values and finds the␣
 ↪minimum values
def generate_minimum_hash_values(hash_list, t):
    min_hash_list = [inf for g in range(t)]

    for h in range(len(hash_list)):
        hash_value_list = hash_list[h]
        for g in range(t):
            if hash_value_list[g] < min_hash_list[g]:
                min_hash_list[g] = hash_value_list[g]
    return min_hash_list

# function that generates the jaccardian similarity for min hashing
```

```
def min_hashing_jaccardian_similarity(a, b, t):
    # keeping track of the sum
    sum = 0
    comparison_list = []
    for d in range(len(a)):
        if a[d] == b[d]:
            comparison_list.append(1)
        else:
            comparison_list.append(0)
    for p in range(len(comparison_list)):
        sum += comparison_list[p]
    return sum / t
```

[178]:

```
# Question 1

# Read documents
doc_1 = document_reader("D1.txt")
doc_2 = document_reader("D2.txt")
doc_3 = document_reader("D3.txt")
doc_4 = document_reader("D4.txt")

# print statements for debugging the documents
# print(doc_1)
# print(doc_2)
# print(doc_3)
# print(doc_4)

# Generate the different 2 character grams for the documents
doc_1_2_char_gram = generate_2_character_gram(doc_1)
doc_2_2_char_gram = generate_2_character_gram(doc_2)
doc_3_2_char_gram = generate_2_character_gram(doc_3)
doc_4_2_char_gram = generate_2_character_gram(doc_4)

# Generate the different 3 character grams for the documents
doc_1_3_char_gram = generate_3_character_gram(doc_1)
doc_2_3_char_gram = generate_3_character_gram(doc_2)
doc_3_3_char_gram = generate_3_character_gram(doc_3)
doc_4_3_char_gram = generate_3_character_gram(doc_4)

# Generate the different 2 word grams for the documents
doc_1_2_word_gram = generate_2_word_gram(doc_1)
doc_2_2_word_gram = generate_2_word_gram(doc_2)
doc_3_2_word_gram = generate_2_word_gram(doc_3)
doc_4_2_word_gram = generate_2_word_gram(doc_4)

# Compute the jaccardian similarity between the documents
# 2 character gram similarities
doc1_doc2_2_char = jaccardian_similarity(doc_1_2_char_gram, doc_2_2_char_gram)
doc1_doc3_2_char = jaccardian_similarity(doc_1_2_char_gram, doc_3_2_char_gram)
doc1_doc4_2_char = jaccardian_similarity(doc_1_2_char_gram, doc_4_2_char_gram)
doc2_doc3_2_char = jaccardian_similarity(doc_2_2_char_gram, doc_3_2_char_gram)
doc2_doc4_2_char = jaccardian_similarity(doc_2_2_char_gram, doc_4_2_char_gram)
doc3_doc4_2_char = jaccardian_similarity(doc_3_2_char_gram, doc_4_2_char_gram)

# 3 character gram similarities
doc1_doc2_3_char = jaccardian_similarity(doc_1_3_char_gram, doc_2_3_char_gram)
doc1_doc3_3_char = jaccardian_similarity(doc_1_3_char_gram, doc_3_3_char_gram)
doc1_doc4_3_char = jaccardian_similarity(doc_1_3_char_gram, doc_4_3_char_gram)
doc2_doc3_3_char = jaccardian_similarity(doc_2_3_char_gram, doc_3_3_char_gram)
doc2_doc4_3_char = jaccardian_similarity(doc_2_3_char_gram, doc_4_3_char_gram)
doc3_doc4_3_char = jaccardian_similarity(doc_3_3_char_gram, doc_4_3_char_gram)

# 2 word gram similarities
doc1_doc2_2_word = jaccardian_similarity(doc_1_2_word_gram, doc_2_2_word_gram)
doc1_doc3_2_word = jaccardian_similarity(doc_1_2_word_gram, doc_3_2_word_gram)
doc1_doc4_2_word = jaccardian_similarity(doc_1_2_word_gram, doc_4_2_word_gram)
doc2_doc3_2_word = jaccardian_similarity(doc_2_2_word_gram, doc_3_2_word_gram)
doc2_doc4_2_word = jaccardian_similarity(doc_2_2_word_gram, doc_4_2_word_gram)
doc3_doc4_2_word = jaccardian_similarity(doc_3_2_word_gram, doc_4_2_word_gram)

# Generate the results
print(f"The number of distinct 2 character grams for document 1:␣
 ↪{len(doc_1_2_char_gram)}")
print(f"The number of distinct 2 character grams for document 2:␣
 ↪{len(doc_2_2_char_gram)}")
print(f"The number of distinct 2 character grams for document 3:␣
 ↪{len(doc_3_2_char_gram)}")
print(f"The number of distinct 2 character grams for document 4:␣
 ↪{len(doc_4_2_char_gram)}")
print()
print(f"The number of distinct 3 character grams for document 1:␣
 ↪{len(doc_1_3_char_gram)}")
print(f"The number of distinct 3 character grams for document 2:␣
 ↪{len(doc_2_3_char_gram)}")
print(f"The number of distinct 3 character grams for document 3:␣
 ↪{len(doc_3_3_char_gram)}")
print(f"The number of distinct 3 character grams for document 4:␣
 ↪{len(doc_4_3_char_gram)}")
print()
print(f"The number of distinct 2 word grams for document 1:␣
 ↪{len(doc_1_2_word_gram)}")
```

```python
print(f"The number of distinct 2 word grams for document 2: {len(doc_2_2_word_gram)}")
print(f"The number of distinct 2 word grams for document 3: {len(doc_3_2_word_gram)}")
print(f"The number of distinct 2 word grams for document 4: {len(doc_4_2_word_gram)}")
print()
print(f"The jaccardian similarity for 2 character grams for document 1 and document 2: {doc1_doc2_2_char}")
print(f"The jaccardian similarity for 2 character grams for document 1 and docuemnt 3: {doc1_doc3_2_char}")
print(f"The jaccardian similarity for 2 character grams for document 1 and document 4: {doc1_doc4_2_char}")
print(f"The jaccardian similarity for 2 character grams for document 2 and document 3: {doc2_doc3_2_char}")
print(f"The jaccardian similarity for 2 character grams for document 2 and document 4: {doc2_doc4_2_char}")
print(f"The jaccardian similarity for 2 character grams for document 3 and document 4: {doc3_doc4_2_char}")
print()
print(f"The jaccardian similarity for 3 character grams for document 1 and document 2: {doc1_doc2_3_char}")
print(f"The jaccardian similarity for 3 character grams for document 1 and document 3: {doc1_doc3_3_char}")
print(f"The jaccardian similarity for 3 character grams for document 1 and document 4: {doc1_doc4_3_char}")
print(f"The jaccardian similarity for 3 character grams for document 2 and document 3: {doc2_doc3_3_char}")
print(f"The jaccardian similarity for 3 character grams for document 2 and document 4: {doc2_doc4_3_char}")
print(f"The jaccardian similarity for 3 character grams for document 3 and document 4: {doc3_doc4_3_char}")
print()
print(f"The jaccardian similarity for 2 word grams for document 1 and document 2: {doc1_doc2_2_word}")
print(f"The jaccardian similarity for 2 word grams for document 1 and document 3: {doc1_doc3_2_word}")
print(f"The jaccardian similarity for 2 word grams for document 1 and document 4: {doc1_doc4_2_word}")
print(f"The jaccardian similarity for 2 word grams for document 2 and document 3: {doc2_doc3_2_word}")
print(f"The jaccardian similarity for 2 word grams for document 2 and document 4: {doc2_doc4_2_word}")
print(f"The jaccardian similarity for 2 word grams for document 3 and document 4: {doc3_doc4_2_word}")
```

```
The number of distinct 2 character grams for document 1: 266
The number of distinct 2 character grams for document 2: 264
The number of distinct 2 character grams for document 3: 296
The number of distinct 2 character grams for document 4: 249

The number of distinct 3 character grams for document 1: 770
The number of distinct 3 character grams for document 2: 759
The number of distinct 3 character grams for document 3: 978
The number of distinct 3 character grams for document 4: 770

The number of distinct 2 word grams for document 1: 289
The number of distinct 2 word grams for document 2: 297
The number of distinct 2 word grams for document 3: 390
The number of distinct 2 word grams for document 4: 364

The jaccardian similarity for 2 character grams for document 1 and document 2:
0.9924812030075187
The jaccardian similarity for 2 character grams for document 1 and docuemnt 3:
0.7841269841269841
The jaccardian similarity for 2 character grams for document 1 and document 4:
0.6666666666666666
The jaccardian similarity for 2 character grams for document 2 and document 3:
0.7834394904458599
The jaccardian similarity for 2 character grams for document 2 and document 4:
0.6601941747572816
The jaccardian similarity for 2 character grams for document 3 and document 4:
0.6717791411042945

The jaccardian similarity for 3 character grams for document 1 and document 2:
0.9552429667519181
The jaccardian similarity for 3 character grams for document 1 and document 3:
0.5030094582975064
The jaccardian similarity for 3 character grams for document 1 and document 4:
0.3061916878710772
The jaccardian similarity for 3 character grams for document 2 and document 3:
0.4987057808455565
The jaccardian similarity for 3 character grams for document 2 and document 4:
0.3034953111679454
The jaccardian similarity for 3 character grams for document 3 and document 4:
0.31329827197595794

The jaccardian similarity for 2 word grams for document 1 and document 2:
0.7920489296636085
The jaccardian similarity for 2 word grams for document 1 and document 3:
0.1954225352112676
The jaccardian similarity for 2 word grams for document 1 and document 4:
0.007716049382716049
The jaccardian similarity for 2 word grams for document 2 and document 3:
```

```
0.17636986301369864
The jaccardian similarity for 2 word grams for document 2 and document 4:
0.009160305343511450
The jaccardian similarity for 2 word grams for document 3 and document 4:
0.012080536912751677
```

[17]:

```
# Question 2
# Read documents
doc_1 = document_reader("D1.txt")
doc_2 = document_reader("D2.txt")

# generate the sets of character grams for document 1 and document 2
doc_1_3_char_gram = generate_3_character_gram(doc_1)
doc_2_3_char_gram = generate_3_character_gram(doc_2)

# the number of bins for the hash table
m = 10000

# list that contains t values
t_list = [20, 60, 150, 300, 600]

for w in range(len(t_list)):
    t_value = t_list[w]
    print(f"the current t value is: {t_value}")

    # compute the hashed strings
    document_1_hash_vector = min_hashing(doc_1_3_char_gram, m, t_value)
    document_2_hash_vector = min_hashing(doc_2_3_char_gram, m, t_value)

    # compute the minimum hash values
    document_1_min_hash_vector =␣
 ↪generate_minimum_hash_values(document_1_hash_vector, t_value)
    document_2_min_hash_vector =␣
 ↪generate_minimum_hash_values(document_2_hash_vector, t_value)

    # compute the jaccardian similarity for the minhashed documents
    jaccard_min_hash_similarity =␣
 ↪min_hashing_jaccardian_similarity(document_1_min_hash_vector,␣
 ↪document_2_min_hash_vector, t_value)

    # generate the result
    print(f"The jaccard similarity for min hashed document 1 and min hashed␣
 ↪document 2 for {t_value} hash functions is: {jaccard_min_hash_similarity}")
```

```
the current t value is: 20
The jaccard similarity for min hashed document 1 and min hashed document 2 for
20 hash functions is: 0.9
```

```
the current t value is: 60
The jaccard similarity for min hashed document 1 and min hashed document 2 for
60 hash functions is: 0.9166666666666666
the current t value is: 150
The jaccard similarity for min hashed document 1 and min hashed document 2 for
150 hash functions is: 0.9333333333333333
the current t value is: 300
The jaccard similarity for min hashed document 1 and min hashed document 2 for
300 hash functions is: 0.9333333333333333
the current t value is: 600
The jaccard similarity for min hashed document 1 and min hashed document 2 for
600 hash functions is: 0.95
```

[16]:

```
# Timing and Accuracy for min hashing experiments

# Question 2
# Read documents
doc_1 = document_reader("D1.txt")
doc_2 = document_reader("D2.txt")

# generate the sets of character grams for document 1 and document 2
doc_1_3_char_gram = generate_3_character_gram(doc_1)
doc_2_3_char_gram = generate_3_character_gram(doc_2)

# the number of bins for the hash table
m = 10000

# list that contains t values
t_list = [20, 60, 150, 300, 600, 800, 1000, 1300, 1500, 2000, 2500, 3000, 3500,␣
 ↪4000, 4500, 5000, 5500, 10000, 15000, 20000]

def time_min_hash(t_value):
    # start the clock
    start_time = time.time()
    # compute the hashed strings
    document_1_hash_vector = min_hashing(doc_1_3_char_gram, m, t_value)
    document_2_hash_vector = min_hashing(doc_2_3_char_gram, m, t_value)

    # compute the minimum hash values
    document_1_min_hash_vector =␣
 ↪generate_minimum_hash_values(document_1_hash_vector, t_value)
    document_2_min_hash_vector =␣
 ↪generate_minimum_hash_values(document_2_hash_vector, t_value)

    # compute the jaccardian similarity for the minhashed documents
    jaccard_min_hash_similarity =␣
↪min_hashing_jaccardian_similarity(document_1_min_hash_vector,␣
↪document_2_min_hash_vector, t_value)

    # compute the time delta
    delta_time = time.time() - start_time
    print(f"The jaccardian minhash similarity result:␣
↪{jaccard_min_hash_similarity}")

    return delta_time


for w in range(len(t_list)):
    t_value = t_list[w]
    print(f"the current t value is: {t_value}")
    time_val = time_min_hash(t_value)
    print(f"The time taken for {t_value} hash functions is: {time_val}")
```

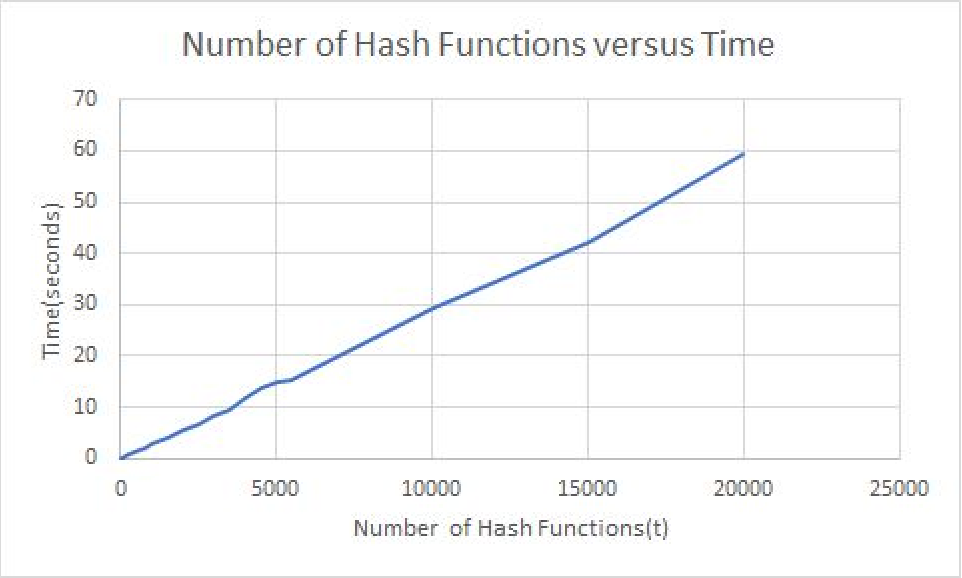
```
the current t value is: 20
The jaccardian minhash similarity result: 0.9
The time taken for 20 hash functions is: 0.06391119956970215
the current t value is: 60
The jaccardian minhash similarity result: 0.9166666666666666
The time taken for 60 hash functions is: 0.16402673721313477
the current t value is: 150
The jaccardian minhash similarity result: 0.9333333333333333
The time taken for 150 hash functions is: 0.42069482803344727
the current t value is: 300
The jaccardian minhash similarity result: 0.9333333333333333
The time taken for 300 hash functions is: 0.8694252967834473
the current t value is: 600
The jaccardian minhash similarity result: 0.95
The time taken for 600 hash functions is: 1.6506919860839844
the current t value is: 800
The jaccardian minhash similarity result: 0.93875
The time taken for 800 hash functions is: 2.196354866027832
the current t value is: 1000
The jaccardian minhash similarity result: 0.943
The time taken for 1000 hash functions is: 2.68377423286438
the current t value is: 1300
The jaccardian minhash similarity result: 0.9446153846153846
The time taken for 1300 hash functions is: 3.4903547763824463
the current t value is: 1500
The jaccardian minhash similarity result: 0.9446666666666667
The time taken for 1500 hash functions is: 4.036346673965454
the current t value is: 2000
The jaccardian minhash similarity result: 0.9465
The time taken for 2000 hash functions is: 5.479223728179932
the current t value is: 2500
The jaccardian minhash similarity result: 0.9468
The time taken for 2500 hash functions is: 6.769505977630615
the current t value is: 3000
The jaccardian minhash similarity result: 0.9433333333333334
The time taken for 3000 hash functions is: 8.373437643051147
the current t value is: 3500
The jaccardian minhash similarity result: 0.9465714285714286
The time taken for 3500 hash functions is: 9.542792320251465
the current t value is: 4000
The jaccardian minhash similarity result: 0.94775
The time taken for 4000 hash functions is: 11.681848049163818
the current t value is: 4500
The jaccardian minhash similarity result: 0.9462222222222222
The time taken for 4500 hash functions is: 13.831008911132812
the current t value is: 5000
The jaccardian minhash similarity result: 0.947
The time taken for 5000 hash functions is: 14.714429378509521
the current t value is: 5500
The jaccardian minhash similarity result: 0.9481818181818182
The time taken for 5500 hash functions is: 15.156139850616455
the current t value is: 10000
The jaccardian minhash similarity result: 0.9536
The time taken for 10000 hash functions is: 29.260077238082886
the current t value is: 15000
The jaccardian minhash similarity result: 0.9544
The time taken for 15000 hash functions is: 42.078519344329834
the current t value is: 20000
The jaccardian minhash similarity result: 0.954
The time taken for 20000 hash functions is: 59.27454710006714
```

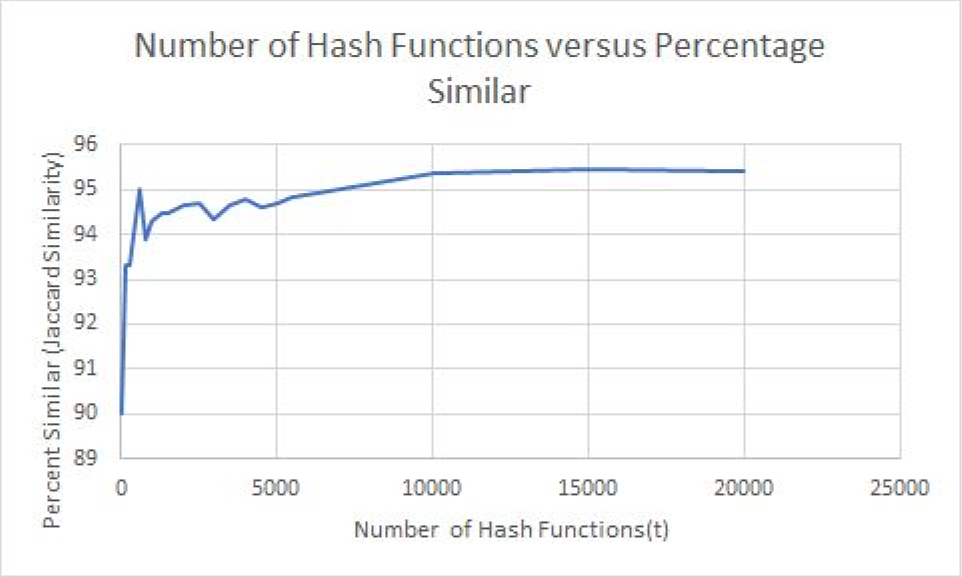
Pranav Rajan
January 28, 2020

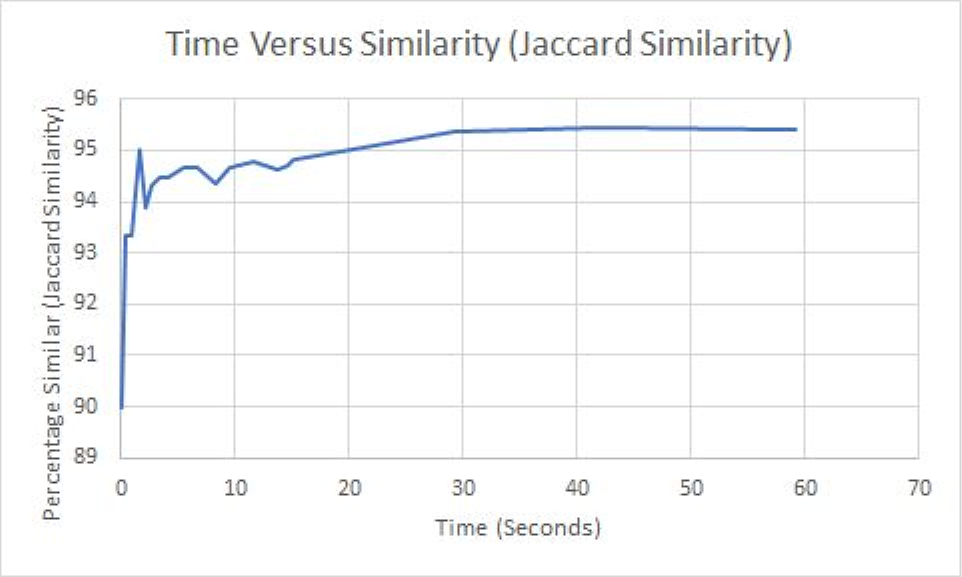
Document Similarity and Hashing

Question 2b

600 seems to be a good value for the number of hash functions. I conducted a few more experiments using the following list for t values: [20, 60, 150, 300, 600, 800, 1000, 1300, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 10000, 15000, 20000]. The following charts were generated with this domain for t.







Based on the experiments, the percentage similarity increased to a threshold of 95% and then plateaued out. Increasing the number of hash functions did not really seem to have an effect on getting a better percentage similarity.