CSCI4022_F21_HW2

September 15, 2021

1 CSCI4022 Homework 2; Minhashing

1.1 Due Monday, September 13 at 11:59 pm to Canvas and Gradescope

Submit this file as a .ipynb with *all cells compiled and run* to the associated dropbox.

Your solutions to computational questions should include any specified Python code and results as well as written commentary on your conclusions. Remember that you are encouraged to discuss the problems with your classmates, but **you must write all code and solutions on your own**.

NOTES:

- Any relevant data sets should be available on Canvas. To make life easier on the graders if they need to run your code, do not change the relative path names here. Instead, move the files around on your computer.
- If you're not familiar with typesetting math directly into Markdown then by all means, do your work on paper first and then typeset it later. Here is a reference guide linked on Canvas on writing math in Markdown. **All** of your written commentary, justifications and mathematical work should be in Markdown. I also recommend the wikibook for LaTex.
- Because you can technically evaluate notebook cells is a non-linear order, it's a good idea to do **Kernel** → **Restart & Run All** as a check before submitting your solutions. That way if we need to run your code you will know that it will work as expected.
- It is **bad form** to make your reader interpret numerical output from your code. If a question asks you to compute some value from the data you should show your code output **AND** write a summary of the results in Markdown directly below your code.
- 45 points of this assignment are in problems. The remaining 5 are for neatness, style, and overall exposition of both code and text.
- This probably goes without saying, but… For any question that asks you to calculate something, you **must show all work and justify your answers to receive credit**. Sparse or nonexistent work will receive sparse or nonexistent credit.
- There is *not a prescribed API* for these problems. You may answer coding questions with whatever syntax or object typing you deem fit. Your evaluation will primarily live in the clarity of how well you present your final results, so don't skip over any interpretations! Your code should still be commented and readable to ensure you followed the given course algorithm.

---

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import re
import string

# problem 1 sanity check
import itertools
```

Back to top # Problem 1 (Theory: minhashing; 15 pts)

Consider minhash values for a single column vector that contains 10 components/rows. Six of rows hold 0 and four hold 1. Consider taking all 10! = 3,628,800 possible distinct permutations of ten rows. When we choose a permutation of the rows and produce a minhash value for the column, we will use the number of the row, in the permuted order, that is the first with a 1. Use Markdown cells to demonstrate answers to the following.

a) For exactly how many of the 3,628,800 permutations is the minhash value for the column a 8? What proportion is this?

First things first, the column vector would look like this if all of the ones are in the last possible positions they can be in:

$$\begin{bmatrix} 0_0 \\ 0_1 \\ 0_2 \\ 0_3 \\ 0_4 \\ 0_5 \\ 1_6 \\ 1_7 \\ 1_8 \\ 1_9 \end{bmatrix}$$

Or this if we don't want to zero-index:

$$\begin{bmatrix} 0_1 \\ 0_2 \\ 0_3 \\ 0_4 \\ 0_5 \\ 0_6 \\ 1_7 \\ 1_8 \\ 1_9 \\ 1_{10} \end{bmatrix}$$

Moving forward, I'm going to assume one-indexing because part B wouldn't be an interesting problem to solve if we zero-indexed (it would be the same answer as part A)

Exactly 0. As can be seen by both of my column vectors above, regardless of how we index the column vector, the max minhash value we can get is 7 (or 6 for the zero-indexed column vector). Because of this, the proportion is obviously $\frac{0}{3628800} = \boxed{0 \text{ permutations}}$. In other words, exactly $\boxed{0\%}$ of permutations contain a minhash value of 7.

---

b) For exactly how many of the 3,628,800 permutations is the minhash value for the column a 7? What proportion is this?

When we one-index, our column vector has to look like the second one I provided above. We have 4! ways to permute the 4 ones and 6! ways to permute the 6 leading zeros. We multiply these to get all of the permutations where the ones are in the last four positions (giving us a minhash value of 7). $6! \cdot 4! = \boxed{17,200 \text{ permutations}}$. Thus, we'd have a proportion of $\frac{17200}{3628800} = 0.0047619048$. In other words, roughly $\boxed{0.5\%}$ of permutations contain a minhash value of 7.

---

c) For exactly how many of the 3,628,800 permutations is the minhash value for the column a 3?

A little refresher of permutations: ${}_nP_r = \frac{n!}{(n-r)!}$. Here, we need to figure out the number of ways we can choose the first two zeros from all six zeros. Thus, ${}_nP_r$ becomes ${}_6P_2 = \frac{6!}{(6-2)!} = 6 \cdot 5 = \boxed{30}$. So there are thirty ways to permute the first two zeros. Next, there are four ways to choose our one that sits in the third index. And finally, there are seven more values to permute, which comes out to 7!. We multiply all of these values together to get: $30 \cdot 4 \cdot 7! = 120 \cdot 5040 = \boxed{604,800 \text{ permutations}}$. This proportion comes out to $\frac{604800}{3628800} = 0.1666666667$. In other words, roughly $\boxed{16.67\%}$ of permutations contain a minhash value of 3.

As we can see from my little test below, we were correct in parts B and C.

---

[2]:
```
permutations = list(itertools.permutations([1, 1, 1, 1, 0, 0, 0, 0, 0, 0]))
total_permutations = len(permutations)

b = np.sum([1 if perm[6] == 1 and perm[7] == 1 and perm[8] == 1 and perm[9] ==␣
 ↪1 else 0 for perm in permutations])
```

```
print("There are {:,} permutations where the minhash value is 7. This␣
 ↪proportion is {:.4}\n".format(b, b/total_permutations)) # sanity check

c = np.sum([1 if perm[0] == 0 and perm[1] == 0 and perm[2] == 1 else 0 for perm␣
 ↪in permutations])
print("There are {:,} permutations where the minhash value is 3. This␣
 ↪proportion is {:.4}".format(c, c/total_permutations)) # sanity check
```

```
There are 17,280 permutations where the minhash value is 7. This proportion is
0.004762

There are 604,800 permutations where the minhash value is 3. This proportion is
0.1667
```

Back to top # Problem 2 (Applied Minhashing; 30 pts)

In this problem we compare similarities of 6 documents available on http://www.gutenberg.org

1) The first approximately 10000 characters of Alexander Dumas' *The Count of Monte Cristo*, written in French, in the file `countmc.txt`
2) The first approximately 10000 characters of Victor Hugo *Les Miserables*, written in French, in the file `lesmis.txt`
3) The first approximately 10000 characters of Jules Verne's *20,000 Leagues Under the Sea*, written in French and translated into English by Frederick Paul Walter, in the file `leagues.txt`
4) The first approximately 10000 characters of Kate Chopin's *The Awakening* in the file `awaken.txt`
5) The entirety of around 12000 characters of Kate Chopin's *Beyond the Bayou* in the file `BB.txt`
6) The first approximately 10000 characters of Homer's *The Odyssey*, translated into English by Samuel Butler, in the file `odyssey.txt`

1.1.1 a) Clean the 6 documents, scrubbing all punctuation, changes cases to lower case, and removing accent marks as appropriate.

For this problem, you may import any text-based packages you desire to help wrangle the data. I recommend looking at some functions within `string` or the RegEx `re` packages.

You can and probably should use functions in the string package such as `string.lower`, `string.replace`, etc.

All 6 documents have been saved in UTF-8 encoding.

After processing, you should have (at most) 27 unique characters in each book/section after cleaning, corresponding to white spaces and the 26 letters. Print out the set of unique characters to ensure this.

[3]:
```
def clean(filename):
    with open(filename, 'r', encoding='utf-8') as file:
        # Reads the whole file into cleaned_text
        cleaned_text = file.read()
        print(sorted(set(cleaned_text)), '\n', len(set(cleaned_text)))

        # Changes cleaned_text to all lowercase characters
        cleaned_text = cleaned_text.lower()

        # https://stackoverflow.com/questions/265960/
↪best-way-to-strip-punctuation-from-a-string
        # Although I didn't run my own test (maybe later???), this appears to␣
↪be the quickest way to
        # strip punctuation in newer versions of python.
        # Also, took me awhile to get the regex expressions correct...

        # Replace all puncuation, numbers and other miscellaneous symbols with␣
↪a whitespace.
        cleaned_text = re.sub(r'[\°\"\"\«\»\.\,\~\`\"\'\’\!
↪\¡\@\#\$\%\^\&\*\(\)\_\-\-\+\=\<\>\?\¿\/\:\;\{\}\[\]0-9]', r'', cleaned_text)
        # Replace all newline characters with a whitespace.
        cleaned_text = re.sub(r'\n', r' ', cleaned_text)
        # Googled this; this is the BOM (byte order mark) which indicates␣
↪endianness, replace with empty char.
        cleaned_text = re.sub(r'\ufeff', r'', cleaned_text)
        # Removes extraneous spaces and replaces with a single whitespace␣
↪character.
        cleaned_text = re.sub(r' +', r' ', cleaned_text)

        # https://stackoverflow.com/questions/2400504/
↪easiest-way-to-replace-a-string-using-a-dictionary-of-replacements
        # I saw this solution floating around and figured I'd use it to replace␣
↪the accented characters because
        # this is pretty modular. If I need to run my program on another text␣
↪with other accented characters I can
        # add them as I see fit.
        '''unidecode.unidecode(file_to_decode) from Piazza, although it would␣
↪be nice to do in one line, what I did works too...'''
        accented_chars = {
            'á': 'a',
            'à': 'a',
            'â': 'a',
            'é': 'e',
            'è': 'e',
            'ê': 'e',
            'í': 'i',
```

```
            'î': 'i',
            'ó': 'o',
            'ô': 'o',
            'ú': 'u',
            'ù': 'u',
            'ü': 'u',
            'û': 'u',
            'ñ': 'n',
            'ç': 'c',
        }
        pattern = re.compile('|'.join(accented_chars.keys()))
        cleaned_text = pattern.sub(lambda x: accented_chars[x.group()],␣
 ↪cleaned_text)
        print(sorted(set(cleaned_text)), '\n', len(set(cleaned_text)), '\n')



        return cleaned_text
```

[4]:
```
documents = {'countmc.txt': None, 'lesmis.txt': None, 'leagues.txt': None,␣
 ↪'odyssey.txt': None, 'awaken.txt': None, 'BB.txt': None}
keys = list(documents.keys())
for key in keys:
    print(key)
    documents[key] = clean(key)

for key in keys:
    if key == "BB.txt":
        documents[key] = documents[key][:12000]
    else:
        documents[key] = documents[key][:10000]
```

```
countmc.txt
['\n', ' ', '!', "'", ',', '-', '.', '0', '1', '2', '4', '5', '8', ':', ';',
'?', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'L', 'M', 'N', 'O', 'P',
'Q', 'R', 'S', 'T', 'U', 'V', '_', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'x', 'y', 'z', '«',
'»', 'À', 'à', 'â', 'ç', 'è', 'é', 'ê', 'î', 'ô', 'ù', 'û', '\ufeff']
 76
[' ', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'x', 'y', 'z']
 25

lesmis.txt
['\n', ' ', '!', "'", ',', '-', '.', '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', ':', ';', '?', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'L',
'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', '_', 'a', 'b', 'c', 'd', 'e',
```

```
'f', 'g', 'h', 'i', 'j', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'x', 'y', 'z', '«', '»', 'À', 'É', 'à', 'â', 'ç', 'è', 'é', 'ê', 'î', 'ô', 'ù',
'û', '\ufeff']
 81
[' ', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'x', 'y', 'z']
 25
```

leagues.txt

```
['\n', ' ', '!', '"', "'", '(', ')', ',', '-', '.', '/', '0', '1', '2', '3',
'4', '5', '6', '7', '8', ';', '?', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'R', 'S', 'T', 'U', 'W', '[', ']', 'a', 'b',
'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '°', '\ufeff']
 73
[' ', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
 27
```

odyssey.txt

```
['\n', ' ', ',', '-', '.', '1', '2', '3', '4', '5', '6', ':', ';', '?', 'A',
'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'R',
'S', 'T', 'U', 'V', 'W', 'Y', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
'-', '’', '"', '"', '\ufeff']
 68
[' ', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
 27
```

awaken.txt

```
['\n', ' ', '!', "'", ',', '-', '.', ':', ';', '?', 'A', 'B', 'C', 'E', 'F',
'G', 'H', 'I', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'W', 'Y', 'Z',
'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '"', '"', '\ufeff']
 60
[' ', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
 27
```

BB.txt

```
['\n', ' ', '!', "'", ',', '-', '.', ':', ';', '?', 'A', 'B', 'C', 'D', 'E',
'F', 'H', 'I', 'J', 'L', 'M', 'N', 'O', 'P', 'Q', 'S', 'T', 'U', 'V', 'W', 'Y',
'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '"', '"', '\ufeff']
 60
[' ', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

27

I print the number of *uncleaned* characters and the actual characters, as well as the number of *cleaned* characters along with the actual characters for each book.

And this makes sense after a quick google search, because it appears french doesn't contain a whole lot of **k**'s or **w**'s.

1.1.2 b) Compute exact similarity scores between the documents. Are these the expected results?

Notes: - You shouls choose or explore different values of k for your shinglesand report the results for multiple values of k. Which values create the largest **range** of similarity scores? - You may choose to shingle on words and create an n-gram model, but it is recommended you shingle on letters as described in class - You may construct your characteristic matrix or characteristic sets with or without hash functions (e.g. by using Python's `set` methods). Note that choice of hash function should change heavily with k!

[5]:
```
def shingler(text, k):
    # I use a list comprehension to shingle a document with size k.
    ## Then I take the set of the list so we don't have any duplicates.
    ### We shingle on characters as described in class, but I would be␣
 ↪interested in seeing how this
    ### would compare with n-grams.
    return set([text[0+i:k+i] for i in range(0, len(text))])

def jaccard_similarity(text1, text2, k=5):
    # Shingle each document using our shingle function.
    shingles1 = shingler(text1, k)
    shingles2 = shingler(text2, k)

    # The intersection of these two sets are all of the shared shingles. Then I␣
 ↪find the length of this new set.
    intersection = len(shingles1.intersection(shingles2))
    union        = len(shingles1.union(shingles2))

    return intersection / union
```

[6]:
```
range_dictionary = dict()
keys = list(documents.keys())
for k in range(1, 11):
    x = 0
    scores = list()
    for i in range(len(keys)):
```

```
        for j in range(i, len(keys)):
            y = ' ' * x
            if keys[i] != keys[j]:
                score = jaccard_similarity(documents[keys[i]],␣
↪documents[keys[j]], k)
                scores.append(score)
                print('{}k = {} - Jaccard similarity between {} and {} is: {:0.
↪5f}'.format(y, k, keys[i], keys[j], score, k=k))
        x+=1
    range_dictionary[k] = max(scores) - min(scores)
    print('\n')

print(range_dictionary)
```

```
k = 1 - Jaccard similarity between countmc.txt and lesmis.txt is: 1.00000
k = 1 - Jaccard similarity between countmc.txt and leagues.txt is: 0.92593
k = 1 - Jaccard similarity between countmc.txt and odyssey.txt is: 0.92593
k = 1 - Jaccard similarity between countmc.txt and awaken.txt is: 0.92593
k = 1 - Jaccard similarity between countmc.txt and BB.txt is: 0.92593
 k = 1 - Jaccard similarity between lesmis.txt and leagues.txt is: 0.92593
 k = 1 - Jaccard similarity between lesmis.txt and odyssey.txt is: 0.92593
 k = 1 - Jaccard similarity between lesmis.txt and awaken.txt is: 0.92593
 k = 1 - Jaccard similarity between lesmis.txt and BB.txt is: 0.92593
  k = 1 - Jaccard similarity between leagues.txt and odyssey.txt is: 1.00000
  k = 1 - Jaccard similarity between leagues.txt and awaken.txt is: 1.00000
  k = 1 - Jaccard similarity between leagues.txt and BB.txt is: 1.00000
   k = 1 - Jaccard similarity between odyssey.txt and awaken.txt is: 1.00000
   k = 1 - Jaccard similarity between odyssey.txt and BB.txt is: 1.00000
    k = 1 - Jaccard similarity between awaken.txt and BB.txt is: 1.00000


k = 2 - Jaccard similarity between countmc.txt and lesmis.txt is: 0.82036
k = 2 - Jaccard similarity between countmc.txt and leagues.txt is: 0.64322
k = 2 - Jaccard similarity between countmc.txt and odyssey.txt is: 0.61616
k = 2 - Jaccard similarity between countmc.txt and awaken.txt is: 0.62010
k = 2 - Jaccard similarity between countmc.txt and BB.txt is: 0.64303
 k = 2 - Jaccard similarity between lesmis.txt and leagues.txt is: 0.64706
 k = 2 - Jaccard similarity between lesmis.txt and odyssey.txt is: 0.64500
 k = 2 - Jaccard similarity between lesmis.txt and awaken.txt is: 0.63614
 k = 2 - Jaccard similarity between lesmis.txt and BB.txt is: 0.66667
  k = 2 - Jaccard similarity between leagues.txt and odyssey.txt is: 0.77778
  k = 2 - Jaccard similarity between leagues.txt and awaken.txt is: 0.78133
  k = 2 - Jaccard similarity between leagues.txt and BB.txt is: 0.80392
   k = 2 - Jaccard similarity between odyssey.txt and awaken.txt is: 0.77307
   k = 2 - Jaccard similarity between odyssey.txt and BB.txt is: 0.79602
    k = 2 - Jaccard similarity between awaken.txt and BB.txt is: 0.79469
```

```
k = 3 - Jaccard similarity between countmc.txt and lesmis.txt is: 0.57348
k = 3 - Jaccard similarity between countmc.txt and leagues.txt is: 0.37045
k = 3 - Jaccard similarity between countmc.txt and odyssey.txt is: 0.31627
k = 3 - Jaccard similarity between countmc.txt and awaken.txt is: 0.32705
k = 3 - Jaccard similarity between countmc.txt and BB.txt is: 0.33739
 k = 3 - Jaccard similarity between lesmis.txt and leagues.txt is: 0.36221
 k = 3 - Jaccard similarity between lesmis.txt and odyssey.txt is: 0.32385
 k = 3 - Jaccard similarity between lesmis.txt and awaken.txt is: 0.33690
 k = 3 - Jaccard similarity between lesmis.txt and BB.txt is: 0.34169
  k = 3 - Jaccard similarity between leagues.txt and odyssey.txt is: 0.49107
  k = 3 - Jaccard similarity between leagues.txt and awaken.txt is: 0.49897
  k = 3 - Jaccard similarity between leagues.txt and BB.txt is: 0.49719
   k = 3 - Jaccard similarity between odyssey.txt and awaken.txt is: 0.49957
   k = 3 - Jaccard similarity between odyssey.txt and BB.txt is: 0.51343
    k = 3 - Jaccard similarity between awaken.txt and BB.txt is: 0.53948


k = 4 - Jaccard similarity between countmc.txt and lesmis.txt is: 0.35335
k = 4 - Jaccard similarity between countmc.txt and leagues.txt is: 0.14589
k = 4 - Jaccard similarity between countmc.txt and odyssey.txt is: 0.11841
k = 4 - Jaccard similarity between countmc.txt and awaken.txt is: 0.12746
k = 4 - Jaccard similarity between countmc.txt and BB.txt is: 0.13479
 k = 4 - Jaccard similarity between lesmis.txt and leagues.txt is: 0.14471
 k = 4 - Jaccard similarity between lesmis.txt and odyssey.txt is: 0.11699
 k = 4 - Jaccard similarity between lesmis.txt and awaken.txt is: 0.13005
 k = 4 - Jaccard similarity between lesmis.txt and BB.txt is: 0.13237
  k = 4 - Jaccard similarity between leagues.txt and odyssey.txt is: 0.26614
  k = 4 - Jaccard similarity between leagues.txt and awaken.txt is: 0.27421
  k = 4 - Jaccard similarity between leagues.txt and BB.txt is: 0.27382
   k = 4 - Jaccard similarity between odyssey.txt and awaken.txt is: 0.28702
   k = 4 - Jaccard similarity between odyssey.txt and BB.txt is: 0.30486
    k = 4 - Jaccard similarity between awaken.txt and BB.txt is: 0.30920


k = 5 - Jaccard similarity between countmc.txt and lesmis.txt is: 0.21405
k = 5 - Jaccard similarity between countmc.txt and leagues.txt is: 0.04292
k = 5 - Jaccard similarity between countmc.txt and odyssey.txt is: 0.02835
k = 5 - Jaccard similarity between countmc.txt and awaken.txt is: 0.03453
k = 5 - Jaccard similarity between countmc.txt and BB.txt is: 0.03620
 k = 5 - Jaccard similarity between lesmis.txt and leagues.txt is: 0.04262
 k = 5 - Jaccard similarity between lesmis.txt and odyssey.txt is: 0.02736
 k = 5 - Jaccard similarity between lesmis.txt and awaken.txt is: 0.03609
 k = 5 - Jaccard similarity between lesmis.txt and BB.txt is: 0.03572
  k = 5 - Jaccard similarity between leagues.txt and odyssey.txt is: 0.13933
  k = 5 - Jaccard similarity between leagues.txt and awaken.txt is: 0.14590
  k = 5 - Jaccard similarity between leagues.txt and BB.txt is: 0.13839
   k = 5 - Jaccard similarity between odyssey.txt and awaken.txt is: 0.16412
```

```
   k = 5 - Jaccard similarity between odyssey.txt and BB.txt is: 0.17169
    k = 5 - Jaccard similarity between awaken.txt and BB.txt is: 0.17915


k = 6 - Jaccard similarity between countmc.txt and lesmis.txt is: 0.12042
k = 6 - Jaccard similarity between countmc.txt and leagues.txt is: 0.01256
k = 6 - Jaccard similarity between countmc.txt and odyssey.txt is: 0.00479
k = 6 - Jaccard similarity between countmc.txt and awaken.txt is: 0.00738
k = 6 - Jaccard similarity between countmc.txt and BB.txt is: 0.00906
 k = 6 - Jaccard similarity between lesmis.txt and leagues.txt is: 0.01367
 k = 6 - Jaccard similarity between lesmis.txt and odyssey.txt is: 0.00576
 k = 6 - Jaccard similarity between lesmis.txt and awaken.txt is: 0.00991
 k = 6 - Jaccard similarity between lesmis.txt and BB.txt is: 0.00817
  k = 6 - Jaccard similarity between leagues.txt and odyssey.txt is: 0.07947
  k = 6 - Jaccard similarity between leagues.txt and awaken.txt is: 0.08562
  k = 6 - Jaccard similarity between leagues.txt and BB.txt is: 0.07679
   k = 6 - Jaccard similarity between odyssey.txt and awaken.txt is: 0.10260
   k = 6 - Jaccard similarity between odyssey.txt and BB.txt is: 0.10213
    k = 6 - Jaccard similarity between awaken.txt and BB.txt is: 0.11361


k = 7 - Jaccard similarity between countmc.txt and lesmis.txt is: 0.06610
k = 7 - Jaccard similarity between countmc.txt and leagues.txt is: 0.00409
k = 7 - Jaccard similarity between countmc.txt and odyssey.txt is: 0.00101
k = 7 - Jaccard similarity between countmc.txt and awaken.txt is: 0.00283
k = 7 - Jaccard similarity between countmc.txt and BB.txt is: 0.00261
 k = 7 - Jaccard similarity between lesmis.txt and leagues.txt is: 0.00498
 k = 7 - Jaccard similarity between lesmis.txt and odyssey.txt is: 0.00175
 k = 7 - Jaccard similarity between lesmis.txt and awaken.txt is: 0.00371
 k = 7 - Jaccard similarity between lesmis.txt and BB.txt is: 0.00224
  k = 7 - Jaccard similarity between leagues.txt and odyssey.txt is: 0.04481
  k = 7 - Jaccard similarity between leagues.txt and awaken.txt is: 0.05318
  k = 7 - Jaccard similarity between leagues.txt and BB.txt is: 0.04803
   k = 7 - Jaccard similarity between odyssey.txt and awaken.txt is: 0.06306
   k = 7 - Jaccard similarity between odyssey.txt and BB.txt is: 0.06250
    k = 7 - Jaccard similarity between awaken.txt and BB.txt is: 0.07340


k = 8 - Jaccard similarity between countmc.txt and lesmis.txt is: 0.03693
k = 8 - Jaccard similarity between countmc.txt and leagues.txt is: 0.00146
k = 8 - Jaccard similarity between countmc.txt and odyssey.txt is: 0.00029
k = 8 - Jaccard similarity between countmc.txt and awaken.txt is: 0.00137
k = 8 - Jaccard similarity between countmc.txt and BB.txt is: 0.00109
 k = 8 - Jaccard similarity between lesmis.txt and leagues.txt is: 0.00188
 k = 8 - Jaccard similarity between lesmis.txt and odyssey.txt is: 0.00035
 k = 8 - Jaccard similarity between lesmis.txt and awaken.txt is: 0.00159
 k = 8 - Jaccard similarity between lesmis.txt and BB.txt is: 0.00076
  k = 8 - Jaccard similarity between leagues.txt and odyssey.txt is: 0.02305
```

```
  k = 8 - Jaccard similarity between leagues.txt and awaken.txt is: 0.03015
  k = 8 - Jaccard similarity between leagues.txt and BB.txt is: 0.02710
   k = 8 - Jaccard similarity between odyssey.txt and awaken.txt is: 0.03574
   k = 8 - Jaccard similarity between odyssey.txt and BB.txt is: 0.03440
    k = 8 - Jaccard similarity between awaken.txt and BB.txt is: 0.04257


k = 9 - Jaccard similarity between countmc.txt and lesmis.txt is: 0.02100
k = 9 - Jaccard similarity between countmc.txt and leagues.txt is: 0.00064
k = 9 - Jaccard similarity between countmc.txt and odyssey.txt is: 0.00006
k = 9 - Jaccard similarity between countmc.txt and awaken.txt is: 0.00085
k = 9 - Jaccard similarity between countmc.txt and BB.txt is: 0.00057
 k = 9 - Jaccard similarity between lesmis.txt and leagues.txt is: 0.00092
 k = 9 - Jaccard similarity between lesmis.txt and odyssey.txt is: 0.00006
 k = 9 - Jaccard similarity between lesmis.txt and awaken.txt is: 0.00079
 k = 9 - Jaccard similarity between lesmis.txt and BB.txt is: 0.00036
  k = 9 - Jaccard similarity between leagues.txt and odyssey.txt is: 0.01243
  k = 9 - Jaccard similarity between leagues.txt and awaken.txt is: 0.01682
  k = 9 - Jaccard similarity between leagues.txt and BB.txt is: 0.01534
   k = 9 - Jaccard similarity between odyssey.txt and awaken.txt is: 0.01985
   k = 9 - Jaccard similarity between odyssey.txt and BB.txt is: 0.01894
    k = 9 - Jaccard similarity between awaken.txt and BB.txt is: 0.02569


k = 10 - Jaccard similarity between countmc.txt and lesmis.txt is: 0.01216
k = 10 - Jaccard similarity between countmc.txt and leagues.txt is: 0.00028
k = 10 - Jaccard similarity between countmc.txt and odyssey.txt is: 0.00000
k = 10 - Jaccard similarity between countmc.txt and awaken.txt is: 0.00055
k = 10 - Jaccard similarity between countmc.txt and BB.txt is: 0.00030
 k = 10 - Jaccard similarity between lesmis.txt and leagues.txt is: 0.00034
 k = 10 - Jaccard similarity between lesmis.txt and odyssey.txt is: 0.00000
 k = 10 - Jaccard similarity between lesmis.txt and awaken.txt is: 0.00049
 k = 10 - Jaccard similarity between lesmis.txt and BB.txt is: 0.00020
  k = 10 - Jaccard similarity between leagues.txt and odyssey.txt is: 0.00644
  k = 10 - Jaccard similarity between leagues.txt and awaken.txt is: 0.00898
  k = 10 - Jaccard similarity between leagues.txt and BB.txt is: 0.00840
   k = 10 - Jaccard similarity between odyssey.txt and awaken.txt is: 0.01075
   k = 10 - Jaccard similarity between odyssey.txt and BB.txt is: 0.01027
    k = 10 - Jaccard similarity between awaken.txt and BB.txt is: 0.01510


{1: 0.07407407407407407, 2: 0.20419766527550964, 3: 0.2572165224055272, 4:
0.23635481323887436, 5: 0.1866897047174948, 6: 0.11563206084270758, 7:
0.07239334497985266, 8: 0.04228427480479304, 9: 0.02563425686976935, 10:
0.015098314606741573}
```

The two french texts (countmc and lesmis) and the two texts written by the same author (BB

and awaken) are consistently more similar than other texts regardless of k. Further, the translated french text shares a slightly higher similarity score to the other two french texts than with the other texts. Finally, all of the english texts share more similarity with each other than with the french texts. This all makes perfect sense. The only thing I was slightly surprised with, was that with lower values of k, the texts written by the same author are less similar to each other than the two french texts written by different authors (although this does make sense upon more thought because the french alphabet is slightly smaller). As we increase k though, the same author's texts become the most similar.

Also, I computed the k value with the largest range, and $k = 3$ is what produced the largest range. Although this might be the case, I don't necessarily think this is the optimal k value. Something between 6 and 9 seems better; moving forward I'll use $k = 5$ because minhashing is an estimate of similarity so larger values of k might lose information (much like using too large of a k value with jaccard similarities). In other words, too small of a k value will give us incredibly similar documents regardless, and using too large of a k value will provide diminishing returns.

1.1.3 c) Implement minhashing with 1000 hash functions on the 6 documents, checking your results against those in part b).

- You may choose your own value of p as the modulus of the hash functions. You are encouraged to use the example code from the minhashing in class notebook to start you out.

[7]:
```
def hash_func(row, nhash):
    # use the "universal hash":  (a*x+b) mod p, where a, b are random ints and
 ↪p > N (= 10 here) is prime
    np.random.seed(4022)
    A = np.random.choice(range(0,10000), size=nhash)
    B = np.random.choice(range(0,10000), size=nhash)
    p = 999983
    return [(A[i]*row + B[i]) % p for i in range(nhash)]

def minhash_signature(k, documents, nhash=1000):
    # Create a list of all of the different sets of shingles (i.e. if we have 5
 ↪documents, we should have 5 sets of shingles)
    document_keys = documents.keys()
    shingle_list = [shingler(documents[key], k) for key in document_keys]
    # sanity check
#     print(shingle_list)
#     print(len(shingle_list))

    # Create a set of all the different shingles found in the documents.
    shingles = set()
    for shingle in shingle_list:
        shingles = shingles.union(shingle)
    # sanity check
#     print(shingles)
```

```
#     print(len(shingles))

    # Create characteristic matrix
    characteristic = np.full([len(shingles), len(shingle_list)], fill_value=0)
    # iterate through the set of all the shingles found in all of the documents
    for row, shingle in enumerate(shingles):
        # iterate through the list of shingle sets to figure out if a given␣
 ↪document contains a given shingle
        # here, col is synonomous with document
        for col, shingle_set in enumerate(shingle_list):
            if shingle in shingle_set:
                characteristic[row, col] = 1
    # sanity check
#     print(characteristic)

    # Create signature matrix
    signature = np.full([nhash, len(shingle_list)], fill_value=np.inf)

    # I kinda just grabbed the code from the in-class notebook and retooled it␣
 ↪in my style of coding
    # and so it would work with my characteristic matrix I create above.

    # For each row of the characteristic matrix...
    for row in range(len(shingles)):
        # STEP 3:  Compute hash values (~permuted row numbers) for that row␣
 ↪under each hash function
        hash_vals = hash_func(row, nhash)
        # For each column in the characteristic matrix
        for col in range(len(shingle_list)):
            # ... but if there is a 1, replace signature matrix element in that␣
 ↪column for each hash fcn
            # with the minimum of the hash value in this row, and the current␣
 ↪signature matrix element
            if characteristic[row, col]==1:
                for h in range(nhash):
                    if hash_vals[h] < signature[h, col]:
                        signature[h, col] = hash_vals[h]
    return signature, document_keys

def minhash_similarity(k, signature_matrix, documents):
    for i1, key1 in enumerate(documents):
        for i2, key2 in enumerate(documents):
            if key1 != key2:
                sim = np.sum(signature_matrix[:, i1] == signature_matrix[:,␣
 ↪i2]) / len(signature_matrix[:, 1])
```

```
                print('k = {} - Minhash similarity between {} and {} is: {:0.
↪5f}'.format(k, key1, key2, sim))
            print('\n')
```

[8]:
```
k = 5

sig_matrix, docs = minhash_signature(k, documents)
```

[9]:
```
minhash_similarity(k, sig_matrix, docs)
```

```
k = 5 - Minhash similarity between countmc.txt and lesmis.txt is: 0.20800
k = 5 - Minhash similarity between countmc.txt and leagues.txt is: 0.05400
k = 5 - Minhash similarity between countmc.txt and odyssey.txt is: 0.03000
k = 5 - Minhash similarity between countmc.txt and awaken.txt is: 0.03500
k = 5 - Minhash similarity between countmc.txt and BB.txt is: 0.03900


k = 5 - Minhash similarity between lesmis.txt and countmc.txt is: 0.20800
k = 5 - Minhash similarity between lesmis.txt and leagues.txt is: 0.04700
k = 5 - Minhash similarity between lesmis.txt and odyssey.txt is: 0.02900
k = 5 - Minhash similarity between lesmis.txt and awaken.txt is: 0.03700
k = 5 - Minhash similarity between lesmis.txt and BB.txt is: 0.04400


k = 5 - Minhash similarity between leagues.txt and countmc.txt is: 0.05400
k = 5 - Minhash similarity between leagues.txt and lesmis.txt is: 0.04700
k = 5 - Minhash similarity between leagues.txt and odyssey.txt is: 0.13900
k = 5 - Minhash similarity between leagues.txt and awaken.txt is: 0.14500
k = 5 - Minhash similarity between leagues.txt and BB.txt is: 0.14100


k = 5 - Minhash similarity between odyssey.txt and countmc.txt is: 0.03000
k = 5 - Minhash similarity between odyssey.txt and lesmis.txt is: 0.02900
k = 5 - Minhash similarity between odyssey.txt and leagues.txt is: 0.13900
k = 5 - Minhash similarity between odyssey.txt and awaken.txt is: 0.15200
k = 5 - Minhash similarity between odyssey.txt and BB.txt is: 0.15300


k = 5 - Minhash similarity between awaken.txt and countmc.txt is: 0.03500
k = 5 - Minhash similarity between awaken.txt and lesmis.txt is: 0.03700
k = 5 - Minhash similarity between awaken.txt and leagues.txt is: 0.14500
k = 5 - Minhash similarity between awaken.txt and odyssey.txt is: 0.15200
k = 5 - Minhash similarity between awaken.txt and BB.txt is: 0.17200


k = 5 - Minhash similarity between BB.txt and countmc.txt is: 0.03900
k = 5 - Minhash similarity between BB.txt and lesmis.txt is: 0.04400
```

```
k = 5 - Minhash similarity between BB.txt and leagues.txt is: 0.14100
k = 5 - Minhash similarity between BB.txt and odyssey.txt is: 0.15300
k = 5 - Minhash similarity between BB.txt and awaken.txt is: 0.17200
```

1.1.4 d) Discussion:

Can we detect expected differences here? Are the two French documents most similar to each other? Are the two documents by the same author, with the same theme, the most similar? Is the French-to-English text the most similar English text when compared to the French texts? What kind of alternatives might have captured the structures between these texts?

I tried a few values of k for this part, and $k = 8$ seemed too high (even though this seemed like a great option going into it). $k = 3$ gave us the largest range of similarity scores so it seemed like a good idea too, although these scores seemed a little too high for my liking. Thus I settled for $k = 5$. It gave me a good balance, and sort of mirrored the jaccard similarities nicely.

The two French documents are indeed the most similar, with the two documents written by the same author slightly behind in 2nd place. Finally, the french-to-english text is more similar to english. When I thought about the French texts more, it makes sense that they're the most similar because the number of unique characters is lower than the english texts so theres a smaller problem-space, and a higher probability of sharing shingles because there are less total shingle combinations. The translated text is similar to this train of thought, and it was unsurprising that it shared more similarity with the english texts. Finally, I think the texts by the same author were a good middle of the road for the results we had, because they will definitely be similar, but of course not 100% similarity.

The only alternative I could really think of was to keep the accented characters, and maybe some punctuation because these might've helped pick up the similarities for the french texts a little bit better (and it would've increased the number of characters for the french alphabet slightly), and the punctuation would've helped us with the same authored texts potentially.
