Asmt 2: Document Similarity and Hashing

Han Ambrose
Turn in through Canvas by 2:45pm, then come to class:
Wednesday, January 29
100 points

Overview

In this assignment you will explore the use of k-grams, Jaccard distance, min hashing, and LSH in the context of document similarity.

You will use four text documents for this assignment:

- `http://www.cs.utah.edu/~jeffp/teaching/cs5140/A2/D1.txt`
- `http://www.cs.utah.edu/~jeffp/teaching/cs5140/A2/D2.txt`
- `http://www.cs.utah.edu/~jeffp/teaching/cs5140/A2/D3.txt`
- `http://www.cs.utah.edu/~jeffp/teaching/cs5140/A2/D4.txt`

As usual, it is recommended that you use LaTeX for this assignment. If you do not, you may lose points if your assignment is difficult to read or hard to follow. Find a sample form in this directory: `http://www.cs.utah.edu/~jeffp/teaching/latex/`

1 Creating k-Grams (50 points)

You will construct several types of k-grams for all documents. All documents only have at most 27 characters: all lower case letters and space. *Yes, the space counts as a character in character k-grams.*

[G1] Construct 2-grams based on characters, for all documents.

[G2] Construct 3-grams based on characters, for all documents.

[G3] Construct 2-grams based on words, for all documents.

Remember, that you should only store each k-gram once, duplicates are ignored.

A: (25 points) How many distinct k-grams are there for each document with each type of k-gram? You should report $4 \times 3 = 12$ different numbers.

D1.txt : 2-grams based on character: 266
D2.txt : 2-grams based on character: 264
D3.txt : 2-grams based on character: 296
D4.txt : 2-grams based on character: 249

D1.txt : 3-grams based on character: 770
D2.txt : 3-grams based on character: 759
D3.txt : 3-grams based on character: 978
D4.txt : 3-grams based on character: 770

D1.txt : 2-grams based on word: 289
D2.txt : 2-grams based on word: 297
D3.txt : 2-grams based on word: 390

D4.txt : 2-grams based on word: 364

```
[13]  1 import numpy as np
      2 import os
      3 from google.colab import drive
      4 import matplotlib as mpl
      5 import matplotlib.pyplot as plt
      6 import seaborn as sns
      7 import math
      8
      9 #mount your Google drive into this notebook
     10 drive.mount('/content/gdrive')
     11 #find the path to your Google drive root
     12 os.getcwd()+"/gdrive/My Drive"
     13 os.chdir('/content/gdrive/My Drive/Colab Notebooks/Data Mining/HW2')
```

```
[23]  1 def two_gram_char (data_set):
      2   with open(data_set,'r') as data_set:
      3     text = data_set.read()
      4     k_gram =set()
      5     for i in range(len(text)-1):
      6       if (text[i] + text[i+1]) not in k_gram:
      7         k_gram.add(text[i] + text[i+1])
      8   return k_gram
```

```
[24]  1 def three_gram_char (data_set):
      2   with open(data_set,'r') as data_set:
      3     text = data_set.read()
      4     k_gram =set()
      5     for i in range(len(text)-2):
      6       if (text[i] + text [i+1] + text[i+2]) not in k_gram:
      7         k_gram.add(text[i] + text [i+1] + text[i+2])
      8   return k_gram
```

```
[25]  1 def two_gram_word (data_set):
      2   with open(data_set, 'r') as data_set:
      3     word = str.split(data_set.read())
      4     k_gram = set()
      5     for i in range(len(word)-1):
      6       if (word[i]+ ' ' + word[i+1]) not in k_gram:
      7           k_gram.add(word[i]+ ' ' + word[i+1])
      8   return k_gram
```

```
[26]  1 doc_set = ['D1.txt','D2.txt','D3.txt','D4.txt']
      2 for data_set in doc_set:
      3   print(data_set + ' : 2_grams based on character: %d' % len(two_gram_char(data_set)))
```

```
[29]  1 doc_set = ['D1.txt','D2.txt','D3.txt','D4.txt']
      2 for data_set in doc_set:
      3   print(data_set + ' : 3_grams based on character: %d' % len(three_gram_char(data_set)))
```

```
[28]  1 doc_set = ['D1.txt','D2.txt','D3.txt','D4.txt']
      2 for data_set in doc_set:
      3   print(data_set + ' : 2_grams based on word: %d' %len(two_gram_word(data_set)))
```

B: (25 points) Compute the Jaccard similarity between all pairs of documents for each type of k-gram. You should report $3 \times 6 = 18$ different numbers.

2-grams char between D1 and D2 is 0.99248
2-grams char between D1 and D3 is 0.78413
2-grams char between D1 and D4 is 0.66667
2-grams char between D2 and D3 is 0.78344
2-grams char between D2 and D4 is 0.66019
2-grams char between D3 and D4 is 0.67178

3-gram char between D1 and D2 is 0.95524
3-gram char between D1 and D3 is 0.50301
3-gram char between D1 and D4 is 0.30619
3-gram char between D2 and D3 is 0.49871
3-gram char between D2 and D4 is 0.30350
3-gram char between D3 and D4 is 0.31330

2-gram word between D1 and D2 is 0.79205
2-gram word between D1 and D3 is 0.19542
2-gram word between D1 and D4 is 0.00772
2-gram word between D2 and D3 is 0.17637
2-gram word between D2 and D4 is 0.00916
2-gram word between D3 and D4 is 0.01208

```
def jac_sim(d1,d2,d3,d4,type):
  print(type +' between D1 and D2 is %.5f' % (len(d1.intersection(d2))/len(d1.union(d2))))
  print(type +' between D1 and D3 is %.5f' % (len(d1.intersection(d3))/len(d1.union(d3))))
  print(type +' between D1 and D4 is %.5f' % (len(d1.intersection(d4))/len(d1.union(d4))))
  print(type +' between D2 and D3 is %.5f' % (len(d2.intersection(d3))/len(d2.union(d3))))
  print(type +' between D2 and D4 is %.5f' % (len(d2.intersection(d4))/len(d2.union(d4))))
  print(type +' between D3 and D4 is %.5f' % (len(d3.intersection(d4))/len(d3.union(d4))))
```

```
jac_sim(two_gram_char('D1.txt'), two_gram_char('D2.txt'), two_gram_char('D3.txt'), two_gram_char('D4.txt'),'2-grams char')
print('')
jac_sim(three_gram_char('D1.txt'), three_gram_char('D2.txt'), three_gram_char('D3.txt'), three_gram_char('D4.txt'),'3-gram char')
print('')
jac_sim(two_gram_word('D1.txt'), two_gram_word('D2.txt'), two_gram_word('D3.txt'), two_gram_word('D4.txt'),'2-gram word')
```

2 Min Hashing (50 points)

We will consider a hash family $\mathcal{H}$ so that any hash function $h \in \mathcal{H}$ maps from $h : \{k\text{-grams}\} \to [m]$ for m large enough (To be extra cautious, I suggest over $m \geq 10{,}000$; but should work with smaller m too).

A: (35 points) Using grams G2, build a min-hash signature for document `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:

$$\hat{\mathsf{JS}}_t(a,b) = \frac{1}{t}\sum_{i=1}^{t} \begin{cases} 1 & \text{if } a_i = b_i \\ 0 & \text{if } a_i \neq b_i. \end{cases}$$

You should report 5 numbers.

By using MinHash, t = 20 : approximate JS between D1 and D2 is 0.95
By using MinHash, t = 60 : approximate JS between D1 and D2 is 0.9833333333333333
By using MinHash, t = 150 : approximate JS between D1 and D2 is 0.9666666666666667
By using MinHash, t = 300 : approximate JS between D1 and D2 is 0.9733333333333334
By using MinHash, t = 600 : approximate JS between D1 and D2 is 0.9716666666666667

```
D1_gram = three_gram_char('D1.txt')
D2_gram = three_gram_char('D2.txt')
D_total = list(D1_gram.union(D2_gram))

for k in [20,60,150,300,600]:
    coll = 0
    for i in range (k):
        v = [math.inf, math.inf]
        for j in range (len(D_total)):
            h = hash(str(i)+D_total[j]+str(i)) % 10000
            if D_total[j] in D1_gram:
                if (h < v[0]):
                    v[0] = h
            if D_total[j] in D2_gram:
                if (h < v[1]):
                    v[1] = h
        if v[0] == v[1]:
            coll = coll+1
    print("By using MinHash, t = %d"%k, ": approximate JS between D1 and D2 is ", coll/k)
```

B: (15 point) What seems to be a good value for t? You may run more experiments. Justify your answer in terms of both accuracy and time.

Based on the Jaccard similarity we calculated in question 1, 3-gram char between D1 and D2 is 0.95524
By using MinHash, t = 500 : approximate JS between D1 and D2 is 0.976 and runtime is 0.42 second
By using MinHash, t = 1000 : approximate JS between D1 and D2 is 0.963 and runtime is 0.86 second
By using MinHash, t = 2000 : approximate JS between D1 and D2 is 0.9615 and runtime is 1.73 second
By using MinHash, t = 5000 : approximate JS between D1 and D2 is 0.9592 and runtime is 4.31 second
By using MinHash, t = 10000 : approximate JS between D1 and D2 is 0.9573 and runtime is 8.64 second
By using MinHash, t = 100000 : approximate JS between D1 and D2 is 0.9572 and runtime is 86.48 second
It seems like t = 10000 seeems to be a close approximation and does not take a lot of runtime

3 Bonus (3 points)

Describe a scheme like Min-Hashing over a domain of size n for the *Andberg* Similarity, defined $\mathsf{Andb}(A, B) = \frac{|A \cap B|}{|A \cup B| + |A \triangle B|}$. That is so given two sets A and B and family of hash functions, then $\mathbf{Pr}_{h \in \mathcal{H}}[h(A) = h(B)] = \mathsf{Andb}(A, B)$. Note the only randomness is in the choice of hash function h from the set $\mathcal{H}$, and $h \in \mathcal{H}$ represents the process of choosing a hash function (randomly) from $\mathcal{H}$. The point of this question is to design this process, and show that it has the required property.

Or show that such a process cannot be done.