CS 4140/6140: Data Mining HW 2

Novella Alvina - u1413401

January 2023

Part I

Creating n-grams

The space counts as a character in character n-grams. Do not do any text pre-processing.

1 Question A

Using the code below and following the instruction to use scikit-learn, do the following:

1.1 Question i

Construct 2-grams based on characters, for all documents. Do not share these n-grams. Shown here is the first 10 2-grams constructed.

```
length 2-gram char D0: 333
[' "' ' (' ' /' ' 1' ' 3' ' 4' ' a' ' b' ' c' ' d']
length 2-gram char D1: 461
[' "' ' (' ' -' ' /' ' 1' ' 2' ' a' ' b' ' c' ' d']
length 2-gram char D2: 401
[' "' ' (' ' /' ' 1' ' 2' ' a' ' b' ' c' ' d' ' e']
length 2-gram char D3: 154
[' 1' ' 4' ' a' ' b' ' c' ' d' ' e' ' f' ' g' ' h']
```

1.2 Question ii

Construct 3-grams based on characters, for all documents. Do not share these n-grams.Shown here is the first 10 3-grams constructed.

```
length 3-gram char D0: 933
[' "w' ' (a' ' (g' ' />' ' 10' ' 30' ' 4/' ' a ' ' ab' ' ac']
length 3-gram char D1: 1714
[' "a' ' "b' ' "d' ' "e' ' "g' ' "h' ' "m' ' "s' ' ("' ' (a']
```

```
length 3-gram char D2: 1428
[’ "c’ ’ "l’ ’ "o’ ’ "p’ ’ "r’ ’ ("’ ’ (i’ ’ (s’ ’ (t’ ’ />’]
length 3-gram char D3: 256
[’ 1 ’ ’ 40’ ’ a ’ ’ af’ ’ al’ ’ an’ ’ at’ ’ av’ ’ ba’ ’ be’]
```

1.3 Question iii

Construct 2-grams based on words, for all documents. Do not share these n-grams. Shown here is the first 10 2-grams constructed.

```
length 2-gram word D0: 349
[’106 minutes’ ’30 minutes’ ’4 10’ ’a beautiful’ ’a complete’ ’a girl’
 ’a lot’ ’a nice’ ’a pretty’ ’a promising’]
length 2-gram word D1: 908
[’1930s among’ ’1937 and’ ’1937 but’ ’20th century’ ’a bad’ ’a bit’
 ’a brawl’ ’a break’ ’a conclusion’ ’a contest’]
length 2-gram word D2: 596
[’1960 s’ ’2005 qualified’ ’a blonde’ ’a burning’ ’a cause’ ’a cry’
 ’a few’ ’a half’ ’a heavy’ ’a magnificent’]
length 2-gram word D3: 63
[’1 only’ ’40 minutes’ ’a 1’ ’a single’ ’after 40’ ’all costs’ ’and after’
 ’and there’ ’at all’ ’avoid this’]
```

1.4 Question (iv

```
length 2-gram char D0: 333
[’ "’ ’ (’ ’ /’ ’ 1’ ’ 3’ ’ 4’ ’ a’ ’ b’ ’ c’ ’ d’]
length 2-gram char D1: 461
[’ "’ ’ (’ ’ -’ ’ /’ ’ 1’ ’ 2’ ’ a’ ’ b’ ’ c’ ’ d’]
length 2-gram char D2: 401
[’ "’ ’ (’ ’ /’ ’ 1’ ’ 2’ ’ a’ ’ b’ ’ c’ ’ d’ ’ e’]
length 2-gram char D3: 154
[’ 1’ ’ 4’ ’ a’ ’ b’ ’ c’ ’ d’ ’ e’ ’ f’ ’ g’ ’ h’]

length 3-gram char D0: 933
[’ "w’ ’ (a’ ’ (g’ ’ />’ ’ 10’ ’ 30’ ’ 4/’ ’ a ’ ’ ab’ ’ ac’]
length 3-gram char D1: 1714
[’ "a’ ’ "b’ ’ "d’ ’ "e’ ’ "g’ ’ "h’ ’ "m’ ’ "s’ ’ ("’ ’ (a’]
length 3-gram char D2: 1428
[’ "c’ ’ "l’ ’ "o’ ’ "p’ ’ "r’ ’ ("’ ’ (i’ ’ (s’ ’ (t’ ’ />’]
length 3-gram char D3: 256
[’ 1 ’ ’ 40’ ’ a ’ ’ af’ ’ al’ ’ an’ ’ at’ ’ av’ ’ ba’ ’ be’]

length 2-gram word D0: 349
[’106 minutes’ ’30 minutes’ ’4 10’ ’a beautiful’ ’a complete’ ’a girl’
```

```
 'a lot' 'a nice' 'a pretty' 'a promising']
length 2-gram word D1: 908
['1930s among' '1937 and' '1937 but' '20th century' 'a bad' 'a bit'
 'a brawl' 'a break' 'a conclusion' 'a contest']
length 2-gram word D2: 596
['1960 s' '2005 qualified' 'a blonde' 'a burning' 'a cause' 'a cry'
 'a few' 'a half' 'a heavy' 'a magnificent']
length 2-gram word D3: 63
['1 only' '40 minutes' 'a 1' 'a single' 'after 40' 'all costs' 'and after'
 'and there' 'at all' 'avoid this']
```

1.5 Question(v)

hich parameters (and their values) and methods of CountVectorizer did you use?

For k-gram based on characters, the parameters for CountVectorizer are

```
analyzer="char", ngram_range=(k,k)
```

. In this case, k for **2-gram** = 2, and k for **3-gram** is 3.

For k-gram based on word, the parameters for CountVectorizer are

```
token_pattern=r'(?u)\b\w+\b', ngram_range=(k,k)
```

. By default, the analyzer is set to 'word', so it doesn't have to be specified. Additionally, by using **token_pattern**, it allows to specify which word or punctuation is counted. For this case, it allows single-letter word but no punctuation. k = 2 for this 2-gram.

2 Question B

2.1 Question i

```python
[16] def jaccard_similarity(text, comb):

       for c in comb:
         d_a = c[0]
         d_b = c[1]

         k_grams_char_2_a = set(n_grams_char([text[d_a]], 2))
         k_grams_char_2_b = set(n_grams_char([text[d_b]], 2))

         k_grams_char_3_a = set(n_grams_char([text[d_a]], 3))
         k_grams_char_3_b = set(n_grams_char([text[d_b]], 3))

         n_grams_word_a = set(n_grams_word([text[d_a]], 2))
         n_grams_word_b = set(n_grams_word([text[d_b]], 2))

         intersection_2_char = len(k_grams_char_2_a.intersection(k_grams_char_2_b))
         intersection_3_char = len(k_grams_char_3_a.intersection(k_grams_char_3_b))
         intersection_word = len(n_grams_word_a.intersection(n_grams_word_b))

         union_2_char = len(k_grams_char_2_a.union(k_grams_char_2_b))
         union_3_char = len(k_grams_char_3_a.union(k_grams_char_3_b))
         union_word = len(n_grams_word_a.union(n_grams_word_b))

         print(f"Jaccard Similarity {c} for k-gram for 2 characters: {intersection_2_char/union_2_char}")
         print(f"Jaccard Similarity {c} for k-gram for 3 characters: {intersection_3_char/union_3_char}")
         print(f"Jaccard Similarity {c} for k-gram for 2 word: {intersection_word/union_word}")
```

Figure 1: Jaccard Similarity for each combination of D1, D2, D3 and D4

2.2 Question ii

```
Jaccard Similarity (0, 1) for k-gram for 2 characters: 0.591182364729459
Jaccard Similarity (0, 1) for k-gram for 3 characters: 0.3268170426065163
Jaccard Similarity (0, 1) for k-gram for 2 word: 0.018638573743922204
Jaccard Similarity (0, 2) for k-gram for 2 characters: 0.6203090507726269
Jaccard Similarity (0, 2) for k-gram for 3 characters: 0.3102108768035516
Jaccard Similarity (0, 2) for k-gram for 2 word: 0.020518358531317494
Jaccard Similarity (0, 3) for k-gram for 2 characters: 0.407514450867052
Jaccard Similarity (0, 3) for k-gram for 3 characters: 0.17839444995044598
Jaccard Similarity (0, 3) for k-gram for 2 word: 0.007334963325183374
Jaccard Similarity (1, 2) for k-gram for 2 characters: 0.6640926640926641
Jaccard Similarity (1, 2) for k-gram for 3 characters: 0.40080249665626394
Jaccard Similarity (1, 2) for k-gram for 2 word: 0.0273224043715847
Jaccard Similarity (1, 3) for k-gram for 2 characters: 0.30851063829787234
Jaccard Similarity (1, 3) for k-gram for 3 characters: 0.11931818181818182
Jaccard Similarity (1, 3) for k-gram for 2 word: 0.004136504653567736
Jaccard Similarity (2, 3) for k-gram for 2 characters: 0.3470873786407767
Jaccard Similarity (2, 3) for k-gram for 3 characters: 0.12566844919786097
Jaccard Similarity (2, 3) for k-gram for 2 word: 0.001519756838905775
```

Part II
Min Hashing

3 Question A

```
jaccard hashing for k = 20 : 0.95
duration =  0.043097734451293945
jaccard hashing for k = 60 : 0.9333333333333333
duration =  0.1267080307006836
jaccard hashing for k = 150 : 0.9666666666666667
duration =  0.3112325668334961
jaccard hashing for k = 300 : 0.9533333333333334
duration =  0.614356279373169
jaccard hashing for k = 600 : 0.9483333333333334
duration =  1.2512729167938232
```

4 Question B

```
jaccard hashing for k = 20 : 0.95
duration =  0.043097734451293945
jaccard hashing for k = 60 : 0.9333333333333333
duration =  0.1267080307006836
jaccard hashing for k = 150 : 0.9666666666666667
duration =  0.3112325668334961
jaccard hashing for k = 300 : 0.9533333333333334
duration =  0.614356279373169
jaccard hashing for k = 450 : 0.9488888888888889
duration =  0.9396347999572754
jaccard hashing for k = 600 : 0.9483333333333334
duration =  1.2512729167938232
jaccard hashing for k = 750 : 0.944
duration =  1.507951259613037
jaccard hashing for k = 800 : 0.945
duration =  1.6488940715789795
jaccard hashing for k = 1000 : 0.946
duration =  2.0190625190734863
```

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 t = [20, 60, 150, 300, 450, 600, 750, 800, 1000], the result shown above shows that the best accuracy is when t = 150 which produces Jaccard Similarity of 0.967 and as the t increases the duration it takes to produce the result also increases. Overall, the accuracy across the t seemed to be at almost constant(around 0.95) with only insignificant difference. This may be because as t increases, it runs more iteration and it takes longer time.

Part III
Code Appendix

```
[3] def n_grams_char(text, k):
      vectorizer = CountVectorizer(analyzer="char", ngram_range=(k,k))
      X = vectorizer.fit_transform(text)
      return vectorizer.get_feature_names_out()
```

Figure 2: n-gram based on characters

```
[2] def n_grams_word(text, k):
      vectorizer = CountVectorizer(token_pattern=r'(?u)\b\w+\b', ngram_range=(k,k))
      X = vectorizer.fit_transform(text)
      return vectorizer.get_feature_names_out()
```

Figure 3: n-gram based on word

```
[16] def jaccard_similarity(text, comb):

      for c in comb:
        d_a = c[0]
        d_b = c[1]

        k_grams_char_2_a = set(n_grams_char([text[d_a]], 2))
        k_grams_char_2_b = set(n_grams_char([text[d_b]], 2))

        k_grams_char_3_a = set(n_grams_char([text[d_a]], 3))
        k_grams_char_3_b = set(n_grams_char([text[d_b]], 3))

        n_grams_word_a = set(n_grams_word([text[d_a]], 2))
        n_grams_word_b = set(n_grams_word([text[d_b]], 2))

        intersection_2_char = len(k_grams_char_2_a.intersection(k_grams_char_2_b))
        intersection_3_char = len(k_grams_char_3_a.intersection(k_grams_char_3_b))
        intersection_word = len(n_grams_word_a.intersection(n_grams_word_b))

        union_2_char = len(k_grams_char_2_a.union(k_grams_char_2_b))
        union_3_char = len(k_grams_char_3_a.union(k_grams_char_3_b))
        union_word = len(n_grams_word_a.union(n_grams_word_b))

        print(f"Jaccard Similarity {c} for k-gram for 2 characters: {intersection_2_char/union_2_char}")
        print(f"Jaccard Similarity {c} for k-gram for 3 characters: {intersection_3_char/union_3_char}")
        print(f"Jaccard Similarity {c} for k-gram for 2 word: {intersection_word/union_word}")
```

Figure 4: Jaccard Similarity for each combination of D1, D2, D3 and D4

```
[171] def mult_hashing(x, a, i, m = 10000):
        return (x*(a+i)/2**(x % 1)) % m
```

```
def min_hashing(k_gram, k):
  v = np.full((1, k), np.inf)
  for g in k_gram:
    for j in range(k):
      s = int(g.encode('utf-8').hex(),16)
      # a_lst = [10, 42, 313, 739, 852]
      # a = 0.7867
      # a = 57683
      # a = 97683
      a = 852
      # s = sha1_hashing(g,j)
      h = mult_hashing(s,a,j)
      if (h < v[0][j]):
        v[0][j] = h
  return v
```

```
[114] def jaccard_hashing(m_a, m_b, k):
        c = []
        for i, x in enumerate(m_a):
          if x == m_b[i]:
            c.append(1)
        return sum(c)/k
```

Figure 5: Fast min hashing for D1 and D2

```
[176] # main
      url = "https://raw.githubusercontent.com/koaning/icepickle/main/datasets/imdb_subset.csv"
      df = pd.read_csv(url) # This is how you read a csv file to a pandas frame
      corpus = list(df['text'])
      corpus_small = corpus[:4] # This is a list of 4 movie reviews

      k_grams_word = []
      k_grams_char_2 = []
      k_grams_char_3 = []

      for c in corpus_small:
        c_lst = [c]
        k_grams_char_2.append(n_grams_char(c_lst, 2))
        k_grams_char_3.append(n_grams_char(c_lst, 3))
        k_grams_word.append(n_grams_word(c_lst, 2))

      for i, k in enumerate(k_grams_char_2):
        print(f"2 char D{i}: {len(k)}")
      print('\n')
      for i, k in enumerate(k_grams_char_3):
        print(f"3 char D{i}: {len(k)}")
      print('\n')
      for i, k in enumerate(k_grams_word):
        print(f"2 word D{i}: {len(k)}")

      r = np.arange(0,len(corpus_small))
      comb = list(combinations(r, 2))
      jaccard_similarity(corpus_small, comb)
```

Figure 6: Q1

```python
r = np.arange(0,len(corpus_small))
comb = list(combinations(r, 2))
jaccard_similarity(corpus_small, comb)

d1 = open('/content/drive/MyDrive/Spring2023/CS4140/D1.txt', 'r')
d2 = open('/content/drive/MyDrive/Spring2023/CS4140/D2.txt', 'r')

corpus_docs = [d1.read(), d2.read()]
kgrams_hash = []

for c in corpus_docs:
  c_lst = [c]
  kgrams_hash.append(n_grams_char(c_lst, 3))

k_lst = [20, 60, 150, 300, 450, 600, 750, 800, 1000]
for k in k_lst:
  fast_mh_a = min_hashing(kgrams_hash[0], k)
  fast_mh_b = min_hashing(kgrams_hash[1], k)
  jh = jaccard_hashing(fast_mh_a[0], fast_mh_b[0], k)
  print(f"jh {k} : {jh}")
```

Figure 7: Q2