**سوال سوم:**

**قسمت الف:**

مهمترین محدودیت LSH این است که نمی‌تواند تضمین کند که دو بردار مشابه در فضای ورودی به یک دسته از هش‌ها تبدیل شوند. به عبارت دیگر، LSH ممکن است برای برخی از جفت بردارهای مشابه، هش‌های مختلف را تولید کند و در نتیجه آن‌ها را در گروه یکسان قرار ندهد. این محدودیت باعث می‌شود LSH مناسب برای بعضی از کاربردهای پردازش تصویر و صوت باشد، اما نمی‌توان آن را به عنوان یک روش قطعی برای جستجو در پایگاه داده‌های بزرگ استفاده کرد.

به عنوان مثال، برای استفاده LSH در داده‌هایی مانند تصاویر، می‌توان از توابع هش مختلفی مانند L2 distance، Euclidean distance، Cosine similarity و ... استفاده کرد. اما مشکل اصلی این است که پیدا کردن پارامترهای مناسب برای هر کدام از این توابع هش، بسته به خصوصیات داده های مورد استفاده و برخی شرایط دیگر ممکن است سخت یا غیرممکن باشد. این در حالیست که استفاده از پارامترهای نامناسب می‌تواند باعث کاهش دقت و کارایی LSH در تحلیل داده ها شود. لذا انتخاب توابع و پارامترهای مناسب برای هش، محدودیت اساسی در استفاده از LSH است.

از سوی دیگر این روش برای داده های با ابعاد بالا کارآمد نیست. برای مثال، اگر داده ها شامل تصاویر با ابعاد بالا باشند، LSH به دلیل افزایش تعداد بیت های مورد نیاز برای نمایش هر تصویر، به شدت کارآمدی خود را از دست می دهد.مساله دیگر، مشکلات در حالت‌های پراکنده‌ی داده‌ها میباشد، به علت اینکه در این حالت‌ها پیدا کردن هش‌های مشابه برای داده‌ها سخت‌تر است و ممکن است باعث کاهش دقت الگوریتم شود.موضوع پایانی نیز درمورد کارایی در داده‌های دارای شواهدی عمقی است. LSH به‌خوبی برای داده‌های پراکنده و خطی در مقیاس‌های کوچک و متوسط عملکرد خوبی دارد، اما برای داده‌هایی با ساختار عمقی نظیر داده‌های ژرف، به دلیل پیچیدگی کار انجام داده کاهش می‌یابد.

در مجموع، LSH یک الگوریتم مناسب برای محاسبه تطابق‌های در مجموعه‌های داده‌های بزرگ است، با این حال بهتر است که مشخصات داده‌ها و شرایط مورد استفاده دقیقاً بررسی شوند تا مطمئن شد که الگوریتم LSH برای حل مسائل، به خوبی جوابگو است.

**قسمت ب:**

برای ثابت کردن قابلیتSimilarity preserving Min-Hashing ، فرض کنید دو مجموعه A و B با اندازه n و m به ترتیب دارای مجموعه‌ای از signature های S(A) و S(B) هستند. حال با استفاده از Min-Hashing ، signature های S(A) و S(B) را به صورت تصادفی تولید کرده و در یک جدول به نام MinHash Matrix ذخیره می‌کنیم. حال برای تخمین شباهت جاکارد بین A و B ، فقط کافیست signature های S(A) و S(B) را با یکدیگر مقایسه کنید. در صورتی که دو signature یکسان باشند، به عنوان 1 شمارش خواهند شد. در غیر این صورت، به عنوان 0 شمارش خواهند شد. با توجه به اینکه signature های S(A) و S(B) توسط Min-Hashing به صورت تصادفی تولید شده‌اند، احتمال بسیار کمی وجود دارد که دو signature یکسان نباشند. بنابراین، این روش قابلیت Similarity preservingدارد و می‌توان از آن برای تخمین شباهت جاکارد بین دو مجموعه استفاده کرد.

**اثبات موجود در اسلایدهای کلاس:**

برای اثبات شباهت نگهدار بودن LSH درواقع داریم که:

$$\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$$

حال فرض کنید x یک داکیومنت از دیتاست باشد و Z نیز یک shingle از این مجموعه باشد، بنابراین تفاسیر داریم که:

$$\Pr[\pi(z) = \min(\pi(X))] = 1/|X|$$

حال خود این عبارت معادل این است که Z مساله به یک min element نگاشت شده است.

حال فرض کنید y به گونه ای باشد که $\pi(y) = \min(\pi(C_1 \cup C_2))$

دراینصورت داریم که:

$$\pi(y) = \min(\pi(C_1)) \quad \text{if } y \in C_1$$
$$\pi(y) = \min(\pi(C_2)) \quad \text{if } y \in C_2$$

بنابراین احتمال اینکه هردو مورد درست باشند عبارت است از $y \in C_1 \cap C_2$

حال با این احتمال گفته شده داریم که:

$$\Pr[\min(\pi(C_1))=\min(\pi(C_2))]=|C_1 \cap C_2|/|C_1 \cup C_2| = sim(C_1, C_2)$$

بنابراین خواسته مساله اثبات میگردد.

**قسمت ب:**

create_shingle

اولین تابع برای ساخت shingle ها بوده و طی آن Shingle های با سایز K که مقدار آن در این پیاده سازی ۵ است. در انتها نیز نتایج این مجموعه کاراکترهای ۵ تایی استخراج شده برای جمله زیر نمایش داده شده :

'The lazy dog is jumped over by a quick brown fox.'

```
def create_shingle(text: list, k: int):
    shingle_set = []
    for i in range(len(text) - k+1):
        shingle_set.append(text[i:i+k])
    return shingle_set
```

```
documents_shingles = [None] * len(documents)
for i in range(len(documents)):
    documents_shingles[i] = create_shingle(documents[i], k=5)
documents_shingles = list(documents_shingles)
print(documents_shingles[2])
```

```
['The l', 'he la', 'e laz', ' lazy', 'lazy ', 'azy d', 'zy do', 'y dog', ' dog ', 'dog i', 'og is', 'g is ', ' is j', 'is ju',
's jum', ' jump', 'jumpe', 'umped', 'mped ', 'ped o', 'ed ov', 'd ove', ' over', 'over ', 'ver b', 'er by', 'r by ', ' by a',
'by a ', 'y a q', ' a qu', 'a qui', ' quic', 'quick', 'uick ', 'ick b', 'ck br', 'k bro', ' brow', 'brown', 'rown ', 'own f',
'wn fo', 'n fox', ' fox.']
```

**نتیجه تست اول:**

ضمنا به دلیل list بودن جنس خروجی و برای مقایسه آن با rest1 ناچار به sort کردن آن قبل از بررسی تست بودیم. در ادامه با بررسی assert درمورد جمله اول دیتاست تایید تست بابت مطابقت دو مجموعه دریافت گردید.

```
res1 = sorted({'of Am', ' of A', 'ty of', 'f Ami', 'irKab', ' Amir', 'unive', 'mirKa', 'y of ', 'ivers', 'rsity', 'sity ',
       'versi', 'ersit', 'ity o', 'rKabi', 'niver', 'Kabir', 'AmirK'})
documents_shingles[0] = sorted(documents_shingles[0])
# documents_shingles[0]
assert res1 == documents_shingles[0], "Test 1 Failed!"
print("Test 1 Successful!")
```

```
Test 1 Successful!
```

**مجموعه ۳۰۸ vocab ایجاد شده از دیتاست:**

```
vocab = set()
for shingle in documents_shingles:
    vocab.update(shingle)

print(vocab)
```

```
{'the s', 'ivers', 'g tha', 'hat l', 'hat i', 'ty of', 'lor j', 'e dog', 'wn in', 'y a q', ' is l', 'rown.', 'activ', 's jum',
'ck an', 'y has', 'he sl', 'y a f', 'ping ', 'he do', 'is no', 'rgeti', ' is b', 't jum', 'wn le', 'pt ov', 'The f', ' is j',
't is ', 'r a l', ' fox ', 'jumps', 'umped', 'uick ', 'a laz', 'uick.', 'or ju', 'mping', ' leap', 's sle', 'r the', 'ver i',
'vely.', 'not v', 'rsity', 'a sle', 'ick j', 'A sle', 'ick b', 'sity ', 'nd br', ' Amir', 'nerge', 'fox t', ' not ', 'y dog',
'r it.', 'ep is', 'fox j', ' lazy', 'ery e', 'x lea', 'zy ha', 'ox is', 'n-col', ' brow', 'y of ', 'n col', 'leapt', 'the l',
't lea', 'of Am', 'umps ', 'leaps', 'AmirK', 'A fox', ' acti', 'lored', ' a sl', 's not', 'e sle', 'in co', 'niver', 'eep i',
'ively', 'eaps ', 'a fox', 'he la', 'getic', 'ox le', 'eaped', 'er by', 'e fox', 'ry ac', 'a qui', 'aps o', 'ox th', ' has ',
'own j', ' fox.', 'lazy ', 'ry li', ' colo', 'wn-co', 'A bro', 'y act', 'd ove', 'og th', 'k bro', ' a la', 'is as', ' in c',
'fox i', 'is se', 'r lea', 'The l', 'by th', 'eepin', ' over', ' by t', ' a br', ' is a', ' that', 'The q', 'azy d', 'lor i',
's ove', 's lea', 'lazy.', 'at is', 'tive.', 'rKabi', 't ver', 'ng do', 'mirKa', 'fox l', 'ps ov', 'zy is', 'e qui', ' a qu', '
and ', 'the d', 'ver b', 'p is ', 'r a d', 'ored ', 'tharg', 'ity o', 'r jum', 'ethar', 'rgic.', 'ing d', 'is le', 'og ha', 'is
br', 'ing o', 'own-c', 'og is', ' a fo', 'er it', 'is sl', 'dog i', 'is ju', 'very ', 'he qu', ' is s', 'olore', 'eapin', 'k an
d', 'energ', 'f Ami', 'hat j', 'brown', ' dog ', 'e laz', 't ove', 'by a ', ' dog.', 'aped ', 'r by ', 'y a b', 'azy h', 'ck b
r', 's see', 'd bro', 'een j', 'seen ', 'argic', 'x jum', 's asl', 'g has', 'x is ', 'own l', 'g is ', 'ox ju', 'that ', 'azy
i', 'red f', 'a dog', 'wn fo', ' very', 'versi', 'rown-', 'own i', 'ery l', 'umpin', 'n fox', 'as a ', 's qui', 'er th', 'apin
g', 's a b', 'Kabir', 'or is', 'color', 'ed fo', 'olor ', 'eapt ', 'en ju', 'ot ve', 'y is ', ' a do', 'd fox', 'wn ju', 'leep
i', 'eping', 'has a', 'leapi', ' by a', 'mps o', 'ick a', 'lor l', 's a q', 'is qu', 's bro', 'n in ', 'aslee', 'erget', 'own
f', ' quic', 'ed ov', ' ener', 'A dog', 'over ', ' is n', 'a bro', 'at ju', ' is q', ' seen', 's a f', 'k jum', 's laz', 'ck j
u', 'ctive', 'apt o', ' the ', 's let', 'quick', 'The d', 'g dog', ' asle', 'letha', 'ersit', 'livel', ' live', 'x tha', 'A la
z', 'ver a', 'ped o', 'g ove', 'dog t', 'unive', 'mped ', 'r a s', 'jumpi', 'r is ', 'dog h', 'n lea', ' slee', 'irKab', 'pin
g.', 'A qui', ' of A', 'zy do', ' leth', 'y ene', 'ng ov', 'ver t', 'ery a', 'y the', 'sleep', 'and b', 'leep ', ' jump', 'n ju
m', 'er a ', 'rown ', 'is la', 'at le', 'y liv', 'etic.', 'hargi', 'ry en', 'he fo', 'leape', '-colo', 'jumpe', 'or le'}
```

create_sparse_vectors

دومین تابع برای ساخت وکتورهای خلوت مبنی بر وجود هرکدام از vocab در هر جمله است که به شکل زیر طراحی گردید. نتیجه حاصل یک ماتریس با ۳۰۸ سطر(تعداد لغات) و ۵۲ ستون که تعداد داکیومنت است میباشد.

```
def create_sparse_vectors(documents_shingles, vocab):

    """
    :param documents_list:
    :param vocab: Union of all shingles sets (type: List)
    :return: sparse vector matrix: on-hot ecoded of all sentences
    desired result for two sentences could be:
    [
     [0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
     [0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0]
    ]
    """
    counter = 0
    counter_v = 0
    sparse_vector_matrix = [[0 for i in range(len(documents))] for j in range (len(vocab))]
    for i in range(len(documents_shingles)):
        for shingle in documents_shingles[i]:
            for j in range(len(list(vocab))):
                if shingle == list(vocab)[j]:
                    sparse_vector_matrix[j][i] = 1

    return sparse_vector_matrix
```

```
In [154]:
import numpy as np
input_matrix = create_sparse_vectors(documents_shingles, vocab)
input_matrix = np.matrix(np.array(input_matrix))
np.shape(input_matrix)
```

```
Out[154]: (308, 51)
```

```
In [155]:
input_matrix
```

```
Out[155]: matrix([[0, 0, 0, ..., 0, 0, 0],
                  [1, 0, 0, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 1, 0],
                  ...,
                  [0, 0, 0, ..., 0, 0, 1],
                  [0, 0, 1, ..., 0, 0, 0],
                  [0, 0, 0, ..., 0, 0, 0]])
```

create_hash_fucntion

صرفا با شافل کردن ساده خواستع ما برای هر permutation را فراهم میکند

create_minhash_functions

به تعداد موردنیاز از minhash توسط این تابع با فراخوانی مکرر تابع بالا صورت میگیرد

```
# minhash_func = create_minhash_functions(len(vocab), 20)
min_hashes = create_minhash_functions(num_minhash, len(vocab))
len(min_hashes)
```

```
20
```

```
np.array(min_hashes)
```

```
array([[145, 259,  12, ..., 248,  84, 125],
       [304, 211, 179, ..., 280,  82, 209],
       [  5, 200, 210, ...,  27, 294,  94],
       ...,
       [194, 140, 109, ..., 260,   4,  49],
       [238, 278, 167, ..., 234, 148, 121],
       [162, 102,  70, ..., 207, 150,  56]])
```

### create_signature_matrix

در ادامه برای سهولت کار از numpy برای طراحی و کار با ماتریس استفاده گردیده است و خروجی signature برای یک جمله خاص(جمله ۱۰ ام) نمایش داده شده است.

```
input_matrix = np.matrix(np.array(input_matrix))
```

```
def create_signature_matrix(min_hashes,input_matrix):
#     print(np.shape(input_matrix))
    # use this function for creating our signatures (eg the matching)
    signature = []
    for func in min_hashes:
        for i in range(1, len(vocab)+1):
            idx = func.index(i)
#             print("idx:",idx)
#             input_matrix.shape()
            signature_val = input_matrix[idx][0]

#             print(signature_val)
            if signature_val == 1:
                signature.append(idx)
                break
    return signature

# np.shape(new_input_matrix[0,:])
signature_matrix = create_signature_matrix(min_hashes,input_matrix[:,10])

np.array(signature_matrix)
```

```
array([153, 181,  70, 211, 162,   2,  34, 153, 196, 100, 153,  70,  25,
        25, 100,  39,  73,  39,  25, 135])
```

### نتیجه Task5

```
def jaccard(a, b):
    """

    :param a:
    :param b:
    :return: Number of common values in a and b
    """
    return len(set(a).intersection(set(b))) / len(set(a).union(set(b)))
```

```
t1 = {'a', 'b', 'c'}
t2 = {'a', 'b', 'd'}
assert jaccard(t1, t2) == 0.5, "Task5 faild"
print("task5 successful")
```

```
task5 successful
```

get_topk_similar

محقق در این قسمت به طور دقیق متوجه خواسته سوال برای ورودی تابع نشد ولی فرض بر این بود که این تابع قرار است با دریافت عدد k  نسبت به نمایش k جمله مشابه با ورودی به ترتیب معیار مشابهت jacard اقدام می نماید.

```
def get_topk_similar(trget_id, k):
    """

    :param trget_id:
    :param candidates:
    :return: k most similar sentences from candidates to target sentence
    """
    result = [(0,1)]
    #TODO-Task6: start your code
    current_doc = create_signature_matrix(min_hashes,new_input_matrix[:,trget_id])
    for i in range(0,51):
        if i == trget_id:
            continue
        loop_doc = create_signature_matrix(min_hashes,new_input_matrix[:,i])
        result.append((jaccard(current_doc, loop_doc),i))

    #end your code
    result.sort(key = lambda i: i[0],reverse = True)

    print("Input sentence: \n", documents[trget_id])
    print("_____________________________________________________")
    for j in range (0,K):
        print(f'{j+1}th similar sentence: \n\n {documents[result[j][1]]} ')
        print(" Jacard value: ",result[j][0])
        print("_____________________________________________________")
    return None
```

باتوجه به لزوم وجود band ها برای امور مربوط به باکت ها و پیاده سازی این موارد توسط طراح محترم سوال، ابهاماتی بابت ورودی تابع وجود داشت و لذا برای نمایش عملکرد کد نسبت به مقایسه جملات با سایر جملات دیتابیس اقدام کردیم.

عملکرد کد برای چند مثال:

```
Input sentence:
 The lazy dog is jumped over by a quick brown fox.
_____________________________________________________________
1th similar sentence:

 The lazy dog is leaped over by a quick and brown fox.
 Jacard value:  0.5652173913043478
_____________________________________________________________
2th similar sentence:

 The lazy dog is jumped over by a quick and brown-colored fox.
 Jacard value:  0.5217391304347826
_____________________________________________________________
3th similar sentence:

 The quick and brown fox is jumped over by the lazy dog.
 Jacard value:  0.48
_____________________________________________________________
4th similar sentence:

 The dog that is asleep is jumped over by a quick brown fox.
 Jacard value:  0.25925925925925924
_____________________________________________________________
5th similar sentence:

 A lazy dog has a quick brown fox jumping over it.
 Jacard value:  0.21428571428571427
_____________________________________________________________
```

```
1 get_topk_similar(12, k=5)
2
```

```
Input sentence:
 A brown-colored fox jumps over a dog that is not very active.
_____________________________________________________________
1th similar sentence:

 A brown-colored fox jumps over a dog that is lazy.
 Jacard value:  0.6875
_____________________________________________________________
2th similar sentence:

 A quick and brown-colored fox jumps over a dog that is lethargic.
 Jacard value:  0.5789473684210527
_____________________________________________________________
3th similar sentence:

 The quick and brown-colored fox jumps over the dog that is not very lively.
 Jacard value:  0.4
_____________________________________________________________
4th similar sentence:

 A quick and brown-colored fox jumps over the dog that is not very lively.
 Jacard value:  0.4
_____________________________________________________________
5th similar sentence:

 A brown-colored fox jumps over the dog that is not very energetic.
 Jacard value:  0.38095238095238093
_____________________________________________________________
```