# Solution of Gradiance Homework – 2

– By Anirban Haldar

## Similarity (LSH)

1. **Consider the following three vectors u, v, w in a 6-dimensional space:**
   **u = [1, 0.25, 0, 0, 0.5, 0]**
   **v = [0.75, 0, 0, 0.2, 0.4, 0]**
   **w = [0, 0.1, 0.75, 0, 0, 1]**

   **Suppose we construct 3-bit sketches of the vectors by the random hyperplane method, using the randomly generated normal vectors r1, r2, and r3, in that order:**
   **r1 = [1, -1, 1, -1, 1, -1]**
   **r2 = [-1, -1, 1, 1, -1, 1]**
   **r3 = [1, 1, 1, 1, 1, 1]**

   **Construct the sketches of the three vectors u, v, w. Then identify the correct vector, sketch pair from the following:**

   **a) u, [-1. +1, +1]**
   **b) v, [+1, +1, -1]**
   **c) u, [+1, -1, +1]**
   **d) w, [-1, +1, -1]**

   ⇨ To construct sketches of each vector u, v, w, we first have to perform dot product of each vector with all the random normal vectors (i.e. r1, r2, r3). Now, if the value of the dot product of a vector for a particular random normal vector is > 0, then we can say, the vector lies above the plain, and we will set "+1" as bit value, otherwise, the vector lies under the plain and bit value will be "-1".

   So, for our vectors and normal, the sketches will be like,

   So, the correct answer is, u, [+1, -1, +1]

```
:
u = np.array([1, 0.25, 0, 0, 0.5, 0])
v = np.array([0.75, 0, 0, 0.2, 0.4, 0])
w = np.array([0, 0.1, 0.75, 0, 0, 1])

r1 = np.array([1, -1, 1, -1, 1, -1])
r2 = np.array([-1, -1, 1, 1, -1, 1])
r3 = np.array([1, 1, 1, 1, 1, 1])

def sketch(x):
    res = []
    for i in [r1, r2, r3]:
        if x.dot(i) > 0: res.append(1)
        else: res.append(-1)
    return res

print("u ->", sketch(u))
print("v ->", sketch(v))
print("w ->", sketch(w))
```

```
u -> [1, -1, 1]
v -> [1, -1, 1]
w -> [-1, 1, 1]
```

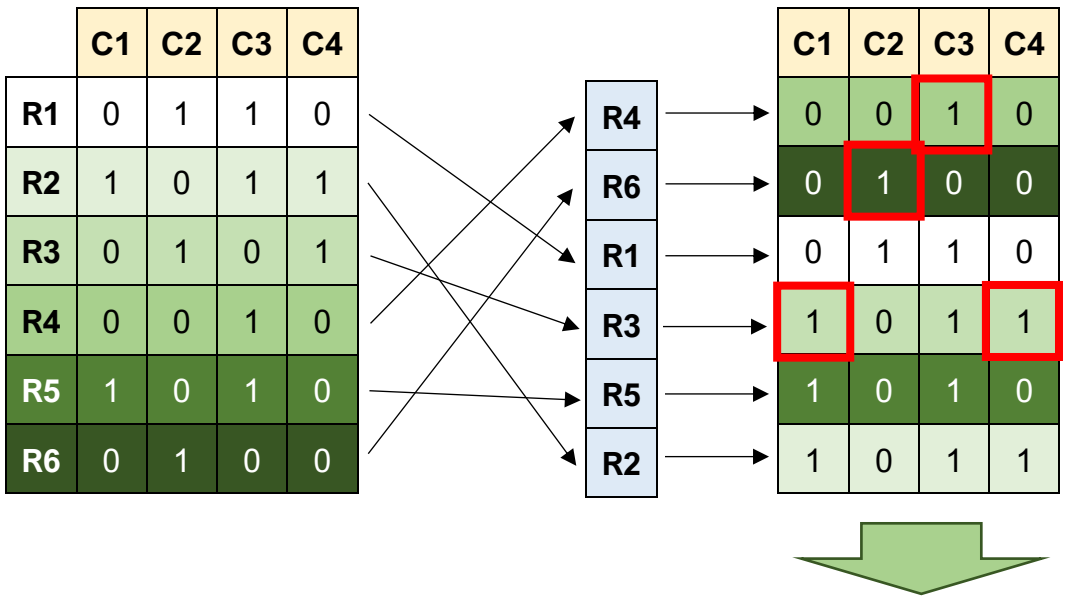
**2. Consider the following matrix:**

| | C1 | C2 | C3 | C4 |
|---|---|---|---|---|
| **R1** | 0 | 1 | 1 | 0 |
| **R2** | 1 | 0 | 1 | 1 |
| **R3** | 0 | 1 | 0 | 1 |
| **R4** | 0 | 0 | 1 | 0 |
| **R5** | 1 | 0 | 1 | 0 |
| **R6** | 0 | 1 | 0 | 0 |

**Perform a minhashing of the data, with the order of rows: R4, R6, R1, R3, R5, R2. Which of the following is the correct minhash value of the stated column? Note: we give the minhash value in terms of the original name of the row, rather than the order of the row in the permutation. These two schemes are equivalent, since we only care whether hash values for two columns are equal, not what their actual values are.**

**a) The minhash value for C3 is R4**
**b) The minhash value for C2 is R4**
**c) The minhash value for C2 is R3**
**d) The minhash value for C3 is R1**

⇨ We know in minhash technique, we randomly order the shingles, and create a signature by marking the 1st shingle of the document which has value 1 for each document. Now, as per the question, we are given a random sequence of shingle as R4, R6, R1, R3, R5, R2. So by finding 1st shingle with value 1, the signature matrix will be…




| C1 | C2 | C3 | C4 |
|---|---|---|---|
| R3 | R6 | R4 | R3 |

So, we can declare the correct answer as,
The minhash value for C3 is R4

**3. Here is a matrix representing the signatures of seven columns, C1 through C7.**

| C1 | C2 | C3 | C4 | C5 | C6 | C7 |
|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 1 | 2 | 5 | 4 |
| 2 | 3 | 4 | 2 | 3 | 2 | 2 |
| 3 | 1 | 2 | 3 | 1 | 3 | 2 |
| 4 | 1 | 3 | 1 | 2 | 4 | 4 |
| 5 | 2 | 5 | 1 | 1 | 5 | 1 |
| 6 | 1 | 6 | 4 | 1 | 1 | 4 |

**Suppose we use locality-sensitive hashing with three bands of two rows each. Assume there are enough buckets available that the hash function for each band can be the identity function (i.e., columns hash to the same bucket if and only if they are identical in the band). Find all the candidate pairs, and then identify one of them in the list below.**

**a) C2 and C3**
**b) C2 and C5**
**c) C1 and C5**
**d) C5 and C7**

⇨ Here, first we divide the signatures into three bands containing two rows each. Now, for each band vector (two shingle values), we create a bucket and hash any document if that document contains same vector under that band.

After hashing, we get the bands & buckets as

| | C1 | C2 | C3 | C4 | C5 | C6 | C7 |
|---|---|---|---|---|---|---|---|
| Band 1 | 1 | 2 | 1 | 1 | 2 | 5 | 4 |
| | 2 | 3 | 4 | 2 | 3 | 2 | 2 |
| Band 2 | 3 | 1 | 2 | 3 | 1 | 3 | 2 |
| | 4 | 1 | 3 | 1 | 2 | 4 | 4 |
| Band 3 | 5 | 2 | 5 | 1 | 1 | 5 | 1 |
| | 6 | 1 | 6 | 4 | 1 | 1 | 4 |

| Band 1 | Band 2 | Band 3 |
|---|---|---|
| Bucket (1, 2) => C1, C4 | Bucket (3, 4) => C1, C6 | Bucket (5, 6) => C1, C3 |
| Bucket (2, 3) => C2, C5 | Bucket (1, 1) => C2 | Bucket (2, 1) => C2 |
| Bucket (1, 4) => C3 | Bucket (2, 3) => C3 | Bucket (1, 4) => C4, C7 |
| Bucket (5, 2) => C6 | Bucket (3, 1) => C4 | Bucket (1, 1) => C5 |
| Bucket (4, 2) => C7 | Bucket (1, 2) => C5 | Bucket (5, 1) => C6 |
| | Bucket (2, 4) => C7 | |

So, finally we get our potential candidates as,
(C1, C4), (C2, C5), (C1, C6), (C1, C3), (C4, C7)
Now, the only matching pair from options is C2 and C5

4. **Find the set of 2-shingles for the "document":**

   **ABRACADABRA**

   **and also for the "document":**

   **BRICABRAC**

   **Answer the following questions:**

   1. **How many 2-shingles does ABRACADABRA have?**
   2. **How many 2-shingles does BRICABRAC have?**
   3. **How many 2-shingles do they have in common?**
   4. **What is the Jaccard similarity between the two documents"?**

   **Then, find the true statement in the list below.**

   a) **BRICABRAC has 6 2-shingles.**
   b) **The Jaccard similarity is 5/9.**
   c) **ABRACADABRA has 10 2-shingles.**
   d) **There are 4 shingles in common.**

⇨ Shingles are nothing but splitting a document, string, paragraph or a blob of any data into small chunks of a fixed length (generally denoted by K or K-grams) and collecting them in a set.

Now, as we know, shingles are set of fixed length values, then we can perform various set operations like intersection, union etc.

Say we have two shingled as S & R then we can define,

Jaccard Similarity between them = $\frac{|S \cap R|}{|S \cup R|}$

```
def shingle(s, k):
    res = set()
    for i in range(len(s)-k+1): res.add(s[i:i+k])
    return list(sorted(list(res)))

def common_shingles(l1, l2):
    i, j, res = 0, 0, []
    while i < len(l1) and j < len(l2):
        if l1[i] == l2[j]:
            res.append(l1[i])
            i, j = i+1, j+1
        elif l1[i] > l2[j]: j += 1
        else: i += 1
    return res

def union_shingles(l1, l2, comm):
    for i in comm: l2.remove(i)
    return list(sorted(l1 + l2))

shin1 = shingle('ABRACADABRA', 2)
shin2 = shingle('BRICABRAC', 2)
common = common_shingles(shin1.copy(), shin2.copy())
union = union_shingles(shin1.copy(), shin2.copy(), common.copy())

print("Shingles in 'ABRACADABRA' :", shin1, '->', len(shin1))
print("Shingles in 'BRICABRAC'   :", shin2, '->', len(shin2))
print("Common Shingles           :", common, '->', len(common))
print("Union Shingles            :", union, '->', len(union))
print("Jaccard Similarity        :", len(common), '/', len(union))
```

```
Shingles in 'ABRACADABRA' : ['AB', 'AC', 'AD', 'BR', 'CA', 'DA', 'RA'] -> 7
Shingles in 'BRICABRAC'   : ['AB', 'AC', 'BR', 'CA', 'IC', 'RA', 'RI'] -> 7
Common Shingles           : ['AB', 'AC', 'BR', 'CA', 'RA'] -> 5
Union Shingles            : ['AB', 'AC', 'AD', 'BR', 'CA', 'DA', 'IC', 'RA', 'RI'] -> 9
Jaccard Similarity        : 5 / 9
```

So, here we can see, the correct answer is The Jaccard similarity is 5/9.

5. **Suppose we have computed signatures for a number of columns, and each signature consists of 24 integers, arranged as a column of 24 rows. There are N pairs of signatures that are 50% similar (i.e., they agree in half of the rows). There are M pairs that are 20% similar, and all other pairs (an unknown number) are 0% similar.**

   **We can try to find 50%-similar pairs by using Locality-Sensitive Hashing (LSH), and we can do so by choosing bands of 1, 2, 3, 4, 6, 8, 12, or 24 rows. Calculate approximately, in terms of N and M, the number of false positive and the number of false negatives, for each choice for the number of rows. Then, suppose that we assign equal cost to false positives and false negatives (an atypical assumption). Which number of rows would you choose if M:N were in each of the following ratios: 1:1, 10:1, 100:1, and 1000:1? Identify the correct choice from the list below.**

   a) **For M = 10N, pick r = 8.**
   b) **For M = 10N, pick r = 6.**
   c) **For M = 100N, pick r = 6.**
   d) **For M = 1000N, pick r = 4.**

⇨ **False Negative:** Say, we have two signatures or documents which are very similar, but they ended up into two different hash buckets, this error is termed as False Negative. We can actually calculate the approx.. false negative for a situation.

Say, we have two documents $D_1$ and $D_2$ , Now say
similarity($D_1$ , $D_2$) = s (large similarity)
rows/band = r
bands = b

then we can say,
probability all rows in a band are same = $s^r$
probability at least one row in a band is different = $(1 – s^r)$
probability at least one row in every band is different = $(1 - s^r)^b$
now, in **false negative, none of the bands should match i.e. $(1 - s^r)^b$ …… (1)**

**False Positive:** Again say, we have two documents having very less similarity, but they ended up in the same hash bucket, this error is termed as false positive. Now to calculate approx.. false positive error,

Say, we have two documents $D_1$ and $D_2$ , Now say
similarity($D_1$ , $D_2$) = s (less similar)
rows/band = r
bands = b

then we can say,
probability all rows in a band are same = $s^r$
probability at least one row in a band is different = $(1 – s^r)$
probability every band is different = $(1 - s^r)^b$
probability at least one band is matching = $1 - (1 - s^r)^b$
now, in **false positive, at least one of the bands should match i.e. $1 - (1 - s^r)^b$ …. (2)**

Now, according to our question,
false positive similarity = 0.2
false negative similarity = 0.5
again, if we have 0.5 similarity and 0.2 similarity documents in the ratio N:M, then the cost will be

$$\text{cost} = \text{false\_positive} * \frac{M}{M+N} + \text{false\_negative} * \frac{N}{M+N}$$

so for each combination of M, N the cost will be

```
s_fn = 0.5
s_fp = 0.2
rows = [1, 2, 3, 4, 6, 8, 12, 24]
MN = [[1, 1], [10, 1], [100, 1], [1000, 1]]
print("        1:1         10:1       100:1      1000:1")
for r in rows:
    b = 24 // r
    fn = (1 - (s_fn ** r)) ** b
    fp = 1 - (1 - s_fp ** r) ** b
    costs = []
    for i in MN:
        cost = fp * (i[0] / sum(i)) + fn * (i[1] / sum(i))
        costs.append(cost)
    print("{:2d}  {:.8f}  {:.8f}  {:.8f}  {:.8f}"
          .format(r, costs[0], costs[1], costs[2], costs[3]))
```

```
        1:1         10:1       100:1      1000:1
 1  0.49763885  0.90479785  0.98542340  0.99428335
 2  0.20948330  0.35496171  0.38376931  0.38693498
 3  0.20292265  0.08781571  0.06502225  0.06251748
 4  0.34424792  0.07041373  0.01618913  0.01023039
 6  0.46960281  0.08559176  0.00954997  0.00119373
 8  0.49416732  0.08985489  0.00979302  0.00099501
12  0.49975589  0.09086471  0.00989616  0.00099852
24  0.49999997  0.09090909  0.00990099  0.00099900
```

Now, our goal is to minimise the cost caused by error, so from the given options,

For M = 10N, pick r = 8. Cost -> 0.08985489
For M = 10N, pick r = 6. Cost -> 0.08559176
**For M = 100N, pick r = 6. Cost -> 0.00954997 <- Minimum**
For M = 1000N, pick r = 4. Cost -> 0.01023039