Massive Data Analysis HW3

Ahmadreza Tavana: 98104852

Professor: Dr. Gholampour

Winter 2023

Question 1

Part a)

If we choose any k-row and has value of 0, the result of minHash would be equal to “don’t know”, because we don’t have the place of first 1. For this, we have to calculate the probability of choosing k rows which all of them be equal to 1. So we have:

$$P = \left(\frac{n-m}{n}\right) \times \left(\frac{n-m-1}{n-1}\right) \dots \left(\frac{n-m-k+1}{n-k+1}\right) \approx \left(\frac{n-m}{n}\right)^k$$

In fact, each fraction is obtained by dividing number of remaining zeros to all of the elements.

Part b)

For this we have:

$$\left(\frac{n-m}{n}\right)^k \leq e^{-10} \rightarrow \left(1 - \frac{1}{\frac{n}{m}}\right)^{\frac{n}{m} \times \frac{mk}{n}} \leq e^{-10}$$

As $m \ll n$ we have:

$$e^{-\frac{mk}{n}} \leq e^{-10} \rightarrow -\frac{mk}{n} \leq -10 \rightarrow k \geq \frac{10n}{m}$$

So the lowest amount of k to have “don’t know” values equal to e^{-10} is equal to $\frac{10n}{m}$.

Part c)

Let's suppose the below vector: (1's colored with blue and 0's colored with red for better intuition.)

S_1	S_2
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
1	0
0	0
1	1
1	0

$$Jaccard\ similarity = J(S_1, S_2) = \frac{1}{3}$$

$$Probabiliy\ of\ cyclic\ permutation = \frac{1}{15}$$

As we can see, Jaccard similarity is not close to probability of cyclic permutation and is far from that and cyclic permutation cannot be a good estimation of Jaccard similarity.

Question 2

Part a)

Suppose that x belongs to T and also $|T| = m$ and $|T| \leq n$. So we have:

$$P\big(g(x) = g(z)\big) = P\big(h(x) = h(z)\big)^k \leq p_2^k = p_2^{\log_{\frac{1}{p2}} n} = \frac{1}{n}$$

Now we have to calculate the expected value of $\big|T \cap W_j\big|$ so we have:

$$E\big(\big|T \cap W_j\big|\big) = m \times P\big(g(x) = g(z)\big) = \frac{m}{n}$$

As $m < n$ so $E\big(\big|T \cap W_j\big|\big) < 1$.

Now by using Markov inequality we have:

$$P\left[\sum_{j=1}^{L}\big|T \cap W_j\big| > 3\right] \leq \frac{E\left[\sum_{j=1}^{L}\big|T \cap W_j\big| > 3\right]}{3L} = \frac{E\left[\big|T \cap W_j\big|\right] \times L}{3L} < \frac{1}{3}$$

Part b)

$$P\big[\forall 1 \leq j \leq L\,, g_j(x^*) \neq g_j(z)\big] = \Big(P\big(g_1(x^*) \neq g_1(z)\big)\Big)^L$$

$$\rightarrow P\big[\forall 1 \leq j \leq L\,, g_j(x^*) \neq g_j(z)\big] = \Big(1 - P\big(g_1(x^*) = g_1(z)\big)\Big)^L$$

$$\rightarrow P\big[\forall 1 \leq j \leq L\,, g_j(x^*) \neq g_j(z)\big] \leq \big(1 - p_1^k\big)^L$$

As we know $k = \log(\frac{1}{p_2})\, n$, we have:

$$P\big[\forall 1 \leq j \leq L\,, g_j(x^*) \neq g_j(z)\big] \leq \left(1 - p_1^{\log_{\frac{1}{p2}} n}\right)^L$$

As we know $\rho = \frac{(\log(\frac{1}{p_1}))}{(\log(\frac{1}{p_2}))}$ by getting $L = n^p$, we have:

$$P\left[\forall 1 \ \leq j \leq L\ , g_j(x^*) \neq g_j(z)\right] \ \leq (1 - n^{-\rho})^L$$

From basic mathematics we know that $(1 - x) \leq e^{-x}$. Now we have:

$$(1 - n^{-\rho})^L \leq e^{-\frac{L}{n\rho}} = e^{-1}$$
$$P\left[\forall 1 \ \leq j \leq L\ , g_j(x^*) \neq g_j(z)\right] \ \leq e^{-1}$$

Part c)

In this part we have to obtain $P(d(x', z) < x\lambda | illustrated\ approach)$

We know that the probability of having more than 3L wrong points and picking them all or not being x^* in the buckets is equal to:

$$P(d(x', z) > x\lambda | illustrated\ approach)$$

Now we know that the probability of having more than 3L wrong points and picking them all or not being x^* in the buckets is less than summation of probability of these two conditions and it's also less than summation of probability of having more than 3L wrong points at all and probability of not being x^* in the buckets. Know suppose we name these conditions as below. Then we have:

A = "having more than 3L wrong points and picking them all"

B = "not being x^* in the buckets"

C = "having more than 3L wrong points"

$$P(d(x', z) > x\lambda | illustrated\ approach) = P(A\ or\ B)$$
$$\rightarrow\ P(A\ or\ B) \leq P(A) + P(B) \leq P(A) + P(C)$$

From the previous part we have:

$$P(A) = \frac{1}{3} \quad \& \quad P(C) = e^{-1}$$

Now from above explains we have:

$$P(d(x', z) > x\lambda | illustrated\ approach) \leq \frac{1}{3} + e^{-1}$$

$$P(d(x', z) < x\lambda | illustrated\ approach) \geq 1 - \frac{1}{3} + e^{-1} = constant$$

The above conclusion shows that with a probability greater than a constant, the reported point is an actual (c,λ) – ANN.

Question 3

Part a)

At first we do the process for SampleData.csv and after that we do all the process for the original dataset.

```
+-----------+---------+--------------+-------------+---------------+----------+-------------------+
|DEVICE_CODE|SYSTEM_ID|ORIGINE_CAR_KEY|FINAL_CAR_KEY|CHECK_STATUS_KEY|COMPANY_ID|      PASS_DAY_TIME|
+-----------+---------+--------------+-------------+---------------+----------+-------------------+
|  100700853|      283|       9016704|      9016704|              5|       264|2022-01-08 00:00:39|
|     230204|       81|       9017538|      9017538|              5|       161|2022-01-08 00:38:48|
|     631357|       81|       9017538|      9017538|              5|       161|2022-01-08 00:13:50|
|     900164|      283|       9035020|      9035020|              5|       264|2022-01-08 00:27:38|
|     900149|      283|       9020900|      9020900|              5|       264|2022-01-08 00:40:08|
|     900246|      283|       9021792|      9021792|              5|       264|2022-01-08 00:55:51|
|  100700871|      283|       9021792|      9021792|              5|       264|2022-01-08 01:07:52|
|     900235|      283|       9021792|      9021792|              5|       264|2022-01-08 01:06:44|
|  100701059|      283|       9038947|      9038947|              5|       264|2022-01-08 00:47:59|
|     631765|       81|       9024221|      9024221|              5|       161|2022-01-08 01:04:11|
+-----------+---------+--------------+-------------+---------------+----------+-------------------+
only showing top 10 rows
```

By using map, we make each row of the df to car license plate and time of their passing. We put the key of the rdd the code of the device which shows the pathway that is passed. The result rdd has a row for each of pathway in each day. The result is shown in below:

```
[(('9016704', datetime.date(2022, 1, 8)), 100700853),
 (('9017538', datetime.date(2022, 1, 8)), 230204),
 (('9017538', datetime.date(2022, 1, 8)), 631357),
 (('9035020', datetime.date(2022, 1, 8)), 900164),
 (('9020900', datetime.date(2022, 1, 8)), 900149),
 (('9021792', datetime.date(2022, 1, 8)), 900246),
 (('9021792', datetime.date(2022, 1, 8)), 100700871),
 (('9021792', datetime.date(2022, 1, 8)), 900235),
 (('9038947', datetime.date(2022, 1, 8)), 100701059),
 (('9024221', datetime.date(2022, 1, 8)), 631765)]
```

If we do group_by for the above rdd, the rdd will have all the pathways for each car in each specific day in a list. In this part we use set to not have repeated itemset. This approach will not have huge effect on the problem and will not change the main results. The result is shown in below:

```
[(('9020900', datetime.date(2022, 1, 8)), {900149}),
 (('9024374', datetime.date(2022, 1, 8)),
  {103002, 631366, 631829, 900215, 900234, 900251, 22009972}),
 (('9153235', datetime.date(2022, 1, 8)), {900142, 900156, 900199}),
 (('9160867', datetime.date(2022, 1, 8)),
  {900142, 900199, 22010039, 100700853}),
 (('9234928', datetime.date(2022, 1, 8)), {137, 212802}),
 (('9332040', datetime.date(2022, 1, 8)),
  {200301,
   205802,
   631354,
   900212,
   900215,
   900221,
   900223,
   900225,
   900256,
   900259,
   900273,
   900277,
   100700826}),
 (('9385850', datetime.date(2022, 1, 8)), {900212, 900222, 900244, 100700839}),
 (('9433390', datetime.date(2022, 1, 8)), {145}),
 (('9423518', datetime.date(2022, 1, 8)), {900101}),
 (('9435137', datetime.date(2022, 1, 8)), {900101, 900156, 900211, 100701090})]
```

Part b)

For some applications and process, we don't need to have date and license plate of cars. So we make a new rdd without them and work with it whenever we need. The result is shown in below:

```
[{900149},
 {103002, 631366, 631829, 900215, 900234, 900251, 22009972},
 {900142, 900156, 900199},
 {900142, 900199, 22010039, 100700853},
 {137, 212802},
 {200301,
  205802,
  631354,
  900212,
  900215,
  900221,
  900223,
  900225,
  900256,
  900259,
  900273,
  900277,
  100700826},
 {900212, 900222, 900244, 100700839},
 {145},
 {900101},
 {900101, 900156, 900211, 100701090}]
```

In this part the "similarity" function is defined which it gets a pathway as input and return similarity between that pathway to sample pathway. It subscripts the sample pathway with the input pathway and it calculates the cosine distance between the sample pathway and the input pathway by dividing the length of intersection of them to sqrt of the multiple of that input and sample pathway. The function is shown in below:

```
def similarity(input_pathway):
    pathway = input_pathway[1]
    distance = len(sample_pathway.intersection(pathway)) / math.sqrt((len(pathway)*len(sample_pathway)))

    return (input_pathway, distance)
```

the most 5 similar pathways to the sample pathway is collected. In each tuple, the first parameter is license plate and then is datetime and after that is pathway and the last one is the amount of similarity. The result is shown in below:

```
[((('9153235', datetime.date(2022, 1, 8)), {900142, 900156, 900199}), 1.0),
 ((('103814654', datetime.date(2022, 1, 8)), {900142, 900156, 900199}), 1.0),
 ((('92099066', datetime.date(2022, 1, 8)), {900142, 900156, 900199}), 1.0),
 ((('100913501', datetime.date(2022, 1, 8)), {900142, 900156, 900199}), 1.0),
 ((('11251721', datetime.date(2022, 1, 8)), {900142, 900156, 900199}), 1.0)]
```

Part c)

In this part, "generate_vectors" function is defined which it gets an input as number of hash functions and then it generates n vectors of 1 and -1. These vectors are made as dictionary because in the next parts, intersections will be calculated much easier and faster. The function is shown in below.

```
import random

def generate_vectors(n):
    all_path = dict.fromkeys(path_rdd_without_key.flatMap(lambda x: x).distinct().collect(), 1)

    vectors = [None]*n
    for i in range(n):
        v = all_path.copy()
        for key in all_path:
            v[key] = [1,-1][random.randint(0, 1)]

        vectors[i] = v

    return vectors
```

An output of this function is shown in the notebook which you can see that.

Next, we set the n, AND and OR parameters. "AND" and "OR" shows the width and number of hyperplanes respectively. It is also abvious that multiple of "AND" and "OR" is equal to n. Also we call generate_vectors function to generate the random vectors. The selected variables are as below:

n = 500

AND = 25

OR = 20

In this part the "min_hash" is defined which gets a list of pathways as input and return signature of that. it intersects the vector of this list to all of the random vectors which are generated in the last part, for each vector. As the lists are small rather than the pathways, in this approach we just consider random vectors which exist on the main list and we calculate the intersection easily. If the result of intersection is positive, the value of the hash will be equal to 1 and if the result is negative the value of the hash will be equal to 0. The function is shown in below:

```
def min_hash(path):
    column = path[1]
    n = len(vectors)
    signature = [1]*n
    for i, v in enumerate(vectors):
        sum = 0

        for key in column:
            sum += v[key]

        if sum < 0:
            signature[i] = 0

    return (path[0], path[1], signature)
```

We can obtain signature from this function that is shown in the notebook of this question which you can see that.

Now, the below function gets a signature as input and generate a tuple for each element of the signature which keep all information about hash number, the value of the hash and the primary information. The reason of keeping this information is that we can show the results better. The combination of the hash number and the hash value gives us a specific basket of a hash. We use this in few next parts. The function is shown in below:

```
def hash_band(sign):
    signature = sign[2]
    output = []
    for i in range(OR):
        band = signature[i*AND:(i+1)*AND]
        band_binary = ''.join(str(digit) for digit in band)
        band_hash = int(band_binary, 2)
        output.append(
            (sign[0], i, band_hash, sign[1], sign[2])
        )

    return output
```

We can obtain hash baskets by using above function that is shown in the notebook of this question which you can see that.

Now, we do all of the process which are explained above for our sample pathway. The result is shown in below:

```
('sample car', {900156, 900142, 900199}, [0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,
1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0,
0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1,
0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0,
0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0,
0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1,
1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1,
0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1])
('sample car', 0, 2762006, {900156, 900142, 900199}, [0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0,
0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0,
1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0,
1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0,
0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0,
0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1,
1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1,
0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1])
```

In this part which is actually the most important part of the code, we collect the final candidates. For this, as we understood in above parts, the combination of the hash number and the hash value gives us a specific basket of a hash. Now for each of these hashes, we only keep baskests which the hash of the sample pathway exists on that basket. It means that, the remaining pathways has similar hash with the pathway in at least one of the hyperplanes. The result is shown in the notebook which you can see and I refuse to show it in this report because of having too long output.

```
candidates = hash_baskets.filter(lambda x: any([ (x[1]==i) and (x[2] == sample_baskets[i][2]) for i in range(OR)]))
candidates.take(1)
```

Next, we use groupBy and map to remove repeated pathways. The remaining pathways which are the final candidates are specific and unique. Then we count the number of unique candidates. The results are shown in notebook of this question which you can see.

Now we count the number of final candidate pathways and show them in below:

```
final_candidates_pathways = candidates.map(lambda x: (x[0], x[3])).groupByKey().mapValues(lambda x: list(x)[0])
final_candidates_count = final_candidates_pathways.count()
final_candidates_count
```

```
15
```

```
final_candidates_pathways.take(final_candidates_count)
```

```
[(('9153235', datetime.date(2022, 1, 8)), {900142, 900156, 900199}),
 (('84810729', datetime.date(2022, 1, 8)),
  {900142, 900156, 900199, 900212, 100700841}),
 (('103814654', datetime.date(2022, 1, 8)), {900142, 900156, 900199}),
 (('8821973', datetime.date(2022, 1, 8)),
  {900142, 900156, 900199, 900212, 900244}),
 (('92099066', datetime.date(2022, 1, 8)), {900142, 900156, 900199}),
 (('65668312', datetime.date(2022, 1, 8)),
  {900132, 900142, 900156, 900199, 100700841}),
 (('100913501', datetime.date(2022, 1, 8)), {900142, 900156, 900199}),
 (('93846333', datetime.date(2022, 1, 8)),
  {900142, 900156, 900199, 900212, 900244}),
 (('86947187', datetime.date(2022, 1, 8)),
  {900142, 900156, 900199, 22010095, 100701090}),
 (('11251721', datetime.date(2022, 1, 8)), {900142, 900156, 900199}),
 (('7660802', datetime.date(2022, 1, 8)), {900142, 900156, 900199}),
 (('8086190', datetime.date(2022, 1, 8)), {900142, 900156, 900199}),
 (('90846416', datetime.date(2022, 1, 8)), {900142, 900156, 900199}),
 (('9649673', datetime.date(2022, 1, 8)), {900142, 900156, 900199, 100701092}),
 (('32716193', datetime.date(2022, 1, 8)),
  {900142, 900212, 900244, 100701298})]
```

Final results and parameters for SimpleData:

Actually as the number of hyperplane increases, we can decrease false negative and false positive. With testing different parameters, I figured out that the 25 number of "AND" and 20 number of "OR" would have the best result.

Now we do all of the above process for the original data, means TrafficData.csv. All of the results are shown in notebook of this question and all of the explains and approaches are like what we did for sample data. You can refer to the notebook and see the results.