تحلیل داده‌های حجیم

تمرین سوم

نام و نام خانوادگی: ناصر کاظمی
شماره دانشجویی: ۹۹۱۰۲۰۵۹

- **سوال ۱**

الف)

در صورتی Don’t know اتفاق می‌افتد که در k ردیفی که انتخاب می‌شود همه مقادیر ستون‌هایشان 0 باشد. زیرا هیچ ردیفی برای مپ شدن انتخاب نمی‌شود.

با فرض اینکه $n - m > k$ داریم

$$Pr(\text{don't know}) = \frac{\binom{n-m}{k}}{\binom{n}{k}} = \frac{\frac{(n-m)!}{(n-m-k)!k!}}{\frac{n!}{(n-k)!k!}} = \frac{(n-m)!(n-k)!}{(n-m-k)!n!} = \frac{(n-k)(n-k-1)\cdots(n-k-m+1)}{n(n-1)\cdots(n-m+1)} \leq \left(\frac{n-k}{n}\right)^m$$

ب)

در صورتی که همان شرایط don’t know اتفاق بیافتد، الگوریتم موفق عمل نمی‌کند.

اگر m و n خیلی بزرگ باشند و n خیلی بزرگتر از k باشد، در این صورت داریم:

$$Pr(\text{don't know}) \leq \left(\frac{n-k}{n}\right)^m = (1-\frac{k}{n})^{\frac{n}{k}\frac{km}{n}} = e^{-\frac{km}{n}} \Longrightarrow \frac{km}{n} \geq 10$$

یا $k \geq 10\frac{n}{m}$.

ج)

یک ردیف مانند r رندوم انتخاب می‌کنیم و از آن شروع می‌کنیم تا آخرین ردیف و سپس از اولین ردیف تا ردیف $r - 1$ را انتخاب می‌کنیم. تعداد این جایگشت‌ها برابر است با تعداد ردیف‌ها.

index	S1	S2
1	0	0
2	1	0
3	0	1
4	1	1
5	0	0
6	1	0

مقدار jaccard similarity دو ستون بالا برابر است با $\frac{1}{4}$. از طرفی احتمال اینکه minhash value دو ستون $S1$ و $S2$ برابر است با $\frac{3}{6} = \frac{1}{2}$. یعنی جایگشت‌های دوری که از ردیف اول، چهارم و پنجم شروع می‌شوند. این دو احتمال با هم برابر نیستند. پس این روش نمی‌تواند برای تخمین jaccard distance مناسب باشد.

• سوال ۲

الف)

با استفاده از نامساوی مارکوف داریم

$$\Pr[\sum_{j=1}^{L} |T \cap W_j| \geq 3L] \leq \frac{E[\sum_{j=1}^{L} |T \cap W_j|]}{3L}$$

حال برای هر نقطه $x \in T$ داریم:

$$\begin{aligned} Pr[x \in W_j | x \in T] &= Pr[x \in W_j | d(x,z) \geq c\lambda] \\ &= Pr[h_{ji}(x) = h_{ji}(z) 1 \leq i \leq k | d(x,z) \geq c\lambda] \\ &= p_2^k \end{aligned}$$

زیرا برابر بودن هر یک از توابع هش عضو g_j برای x و z مستقل است و خانواده توابع هش داده شده $(\lambda, c\lambda, p_1, p_2)$-sensitive هستند.

از طرفی داریم:

$$p_2^k = p_2^{\log_{\frac{1}{p_2}} n} = \frac{1}{n}$$

بنابراین:

$$Pr[x \in T \cap W_j] = \frac{1}{n} \Longrightarrow E[|T \cap W_j|] \sum_{x \in A} \mathbf{1}(x \in W_j) Pr[x \in T \cap W_j] \leq \frac{n}{n} = 1$$

و در نهایت:

$$\Pr[\sum_{j=1}^{L} |T \cap W_j| \geq 3L] \leq \frac{E[\sum_{j=1}^{L} |T \cap W_j|]}{3L} \leq \frac{L}{3L} = \frac{1}{3}.$$

ب)

از آنجایی که خانواده توابع هش داده شده $(\lambda, c\lambda, p_1, p_2)$-sensitive هستند، در این صورت داریم:

$$Pr[\forall 1 \leq j < L, g_j(x^*) \neq g_j(z)] < (1 - p_1^k)^L = (1 - \left(p_1^{\log_{\frac{1}{p_2}} n}\right)^{\frac{\log p_1}{\log p_2}} = (1 - \frac{1}{n^{\log_{p_2} p1}})^{n^{\log_{p_2} p_1}} < \frac{1}{e}.$$

ج)

فرض کنید x' یک نقطه گزارش‌شده و S مجموعه (c, λ)-ANN و U مجموعه $3L$ نقطه انتخاب‌شده باشد.

در این صورت:

$$Pr[x' \notin S] = Pr[S \cap (\bigcup W_j) = \phi \ or \ U \cap (S \cap (\bigcup W_j)) = \phi] \leq Pr[S \cap (\bigcup W_j) = \phi] + Pr[U \cap (S \cap (\bigcup W_j)]$$

بنابر قسمت ب داریم:

$$Pr[S \cap (\bigcup W_j) = \phi] \leq Pr[x' \notin \bigcup W_j] < \frac{1}{e}$$

و بنابر قسمت الف داریم:

$$\begin{aligned} Pr[U \cap (S \cap (\bigcup W_j)] &< Pr[\,|T \cap (\bigcup) W_j| \geq |U|\,] \\ &= Pr[\,|\bigcup (T \cap W_j)| \geq |U|\,] \\ &\leq Pr[\,|\sum (T \cap W_j)| \geq 3L] \leq \frac{1}{3} \end{aligned}$$

پس

$$Pr[x' \notin S] \leq \frac{1}{e} + \frac{1}{3} \Longrightarrow Pr[x' \in S] \geq \frac{2}{3} - \frac{1}{e}.$$

- **سوال ۳**

برای بخش الف به صورت زیر عمل می‌کنیم:

```python
schema = StructType([
    StructField("DEVICE_CODE", IntegerType(), True),
    StructField("SYSTEM_ID", IntegerType(), True),
    StructField("ORIGINE_CAR_KEY", StringType(), True),
    StructField("FINAL_CAR_KEY", StringType(), True),
    StructField("CHECK_STATUS_KEY", IntegerType(), True),
    StructField("COMPANY_ID", StringType(), True),
    StructField("PASS_DAY_TIME", TimestampType(), True)
])

```

Python

```python
df = spark_session.read.csv(
    '/content/drive/MyDrive/MDA/HW3/TrafficData.csv', header=True, schema=schema)
df.show(1)

```

Python

```
+-----------+---------+--------------+-------------+---------------+----------+-------------------+
|DEVICE_CODE|SYSTEM_ID|ORIGINE_CAR_KEY|FINAL_CAR_KEY|CHECK_STATUS_KEY|COMPANY_ID|      PASS_DAY_TIME|
+-----------+---------+--------------+-------------+---------------+----------+-------------------+
|   22010122|      284|       97955760|     64111706|              7|       161|2022-01-10 08:58:02|
+-----------+---------+--------------+-------------+---------------+----------+-------------------+
only showing top 1 row
```

```python
traffic_rdd = df.rdd.map(lambda x: ((x["FINAL_CAR_KEY"], x["PASS_DAY_TIME"].date()), x["DEVICE_CODE"]))\
                    .groupByKey()\
                    .map(lambda x: (x[0], set(x[1])))

```

Python

برای بخش ابتدا به این شکل یک مسیر رندوم می‌سازیم.

```python
# hash function to map each device code to a number between 0 and num_device
device_index_map = {}
for i in range(num_device):
    device_index_map[device_codes[i]] = i

```

Python

```python
path_vec = np.zeros(num_device)
indices = np.random.choice(np.arange(num_device), replace=False,
                           size=int(num_device * 0.8))
path_vec[indices] = 1
path = []
for i in range(len(path_vec)):
    if path_vec[i] == 1:
        path.append(device_codes[i])

```

Python

```python
len(path)

```

Python

```
756
```

سپس cosine similarity هر مسیر را با مسیر مذکور محاسبه می‌کنیم و به صورت نزولی مرتب می‌کنیم. در نهایت ۵تای اول را انتخاب می‌کنیم (۵ مسیر با بیشترین تشابه).

```python
def path_similarity(x):
    similarity = 0
    for device_code in x:
        similarity += path_vec[device_index_map[device_code]]
    return similarity / ((len(x) ** 0.5) * len(path) ** 0.5)


most_similar_path = traffic_rdd.map(lambda x: (x[0], path_similarity(x[1]))).sortBy(lambda x: x[-1], ascending=False)
most_similar_path.take(5)
```

Python

```
[(('64111706', datetime.date(2022, 1, 10)), 0.8395095799193328),
 (('64111706', datetime.date(2022, 1, 8)), 0.8390337632522099),
 (('64111706', datetime.date(2022, 1, 11)), 0.8378173112911534),
 (('64111706', datetime.date(2022, 1, 12)), 0.8240384548552595),
 (('64111706', datetime.date(2022, 1, 9)), 0.8232996903942893)]
```

در پیاده‌سازی LSH از banding یا همان AND-OR amplification استفاده می‌کنیم. به این صورت که وکتورهای حاصل از hash کردن بردار embed شده مسیرها با استفاده از cosine similarity را به b دسته r تایی تقسیم می‌کنیم. اگر هر هش هر مسیر یک دسته با دسته متناظر آن در هش مسیر رندوم انتخاب شده برابر بود، آن را در لیست مسیرهای مشابه احتمالی قرار می‌دهیم.

ابتدا به تعداد $b \times r$ ابر صفحه رندوم میسازیم، که در واقع همان توابع hash ما هستند. سپس با استفاده از این ابرصفحه‌ها، هر مسیر را به یک وکتور $b \times r$بعدی از 0 و 1 هش می‌کنیم. و به صورت زیر مسیرهای کاندید را بدست می‌آوریم و در نهایت از بین آن‌های مسیرهای که واقعا cosine similarity نزدیکی (برای مثال بالاتر از 0.7) دارند را انتخاب می‌کنیم.

```python
b = 10
r = 15
num_planes = b * r

random_planes = []
for i in range(num_planes):
    random_planes.append(np.random.choice([-1.0, 1.0], size=num_device))
random_planes = np.array(random_planes)
random_planes
```

Python

```
array([[-1., -1., -1., ...,  1.,  1., -1.],
       [-1., -1., -1., ..., -1.,  1., -1.],
       [ 1.,  1.,  1., ...,  1., -1.,  1.],
       ...,
       [ 1.,  1., -1., ..., -1., -1., -1.],
       [-1.,  1.,  1., ..., -1.,  1.,  1.],
       [-1., -1., -1., ..., -1.,  1.,  1.]])
```

برای محاسبه b و r برای یک $r \times b$ ثابت، داریم که دو مسیر با تشابه s را با احتمال $1 - (1 - s^r)^b$ به یک باکت می‌فرستد. برای مثال $b = 10$ و $r = 15$ برای 150 ابرصفحه می‌تواند خوب باشد. زیرا هم false positiveها را کم میکند و هم false negativeها را.

```python
def calculate_hash(x):
    items = x
    lst = ""
    for plane in random_planes:
        i = 0
        for item in items:
            i += plane[device_index_map[item]]
        z = 1
        if i < 0:
            z = 0
        lst += str(z)
    return lst

hashed = traffic_rdd.map(lambda x : (x[0], calculate_hash(x[1])))
```

Python

```python
def hash_vector(x):
    hash_values = []
    for plane in random_planes:
        hash_value = 0
        for code in x:
            hash_value += plane[device_index_map[code]]
        hash_values.append(hash_value)
    sig = "".join(["1" if x > 0 else "0" for x in hash_values])
    return sig

```

Python

```python
hashed_path = hash_vector(path)
```

Python

```python
def match_hash(x):
    first = 0
    last = r-1
    for i in range(b):
        if hashed_path[first:last] == x[first:last]:
            return True
        first += r
        last += r
    return False
```

Python

```python
candidates = hashed.filter(lambda x: match_hash(x[1])).collect()
len(candidates)
```

Python

```
5237
```

```python
unique_candidates = set(map(lambda x: tuple(x[0]), candidates))
```

Python

```python
# get the most similar path from the candidates
threshold = 0.8

most_similar_path = traffic_rdd.map(lambda x: (x[0], (tuple(x[1]), path_similarity(x[1]))))\
    .filter(lambda x: x[0] in unique_candidates)\
    .filter(lambda x: x[1][1] > threshold)\
    .collect()

```

Python

+ Code + Markdown

```python
len(most_similar_path)
```

Python

```
4
```

```python
for item in most_similar_path:
    print(item[0], item[1][1])
```

Python

```
('64111706', datetime.date(2022, 1, 8)) 0.8390337632522099
('64111706', datetime.date(2022, 1, 10)) 0.8395095799193328
('64111706', datetime.date(2022, 1, 13)) 0.8087361135779054
('64111706', datetime.date(2022, 1, 12)) 0.8240384548552595
```

جالب است توجه کنید مسیرهای مشابه با مسیر رندوم ساخته شده همگی مربوط به یک شماره پلاک و در روزهای مختلف است. این یعنی یک مسیر که این ماشین به طور مرتب از آن عبور می‌کند.

با افزایش تعداد هایپرپلین‌ها میزان دقت ابتدا بهبود قابل توجهی می‌کند. اما برای از جایی به بعد دقت با نرخی کم به یک مقدار حدی میل می‌کند.