تحلیل داده‌های حجیم

تمرین دوم

نام و نام خانوادگی: ناصر کاظمی
شماره دانشجویی: ۹۹۱۰۲۰۵۹

- **سوال ۱**

الف)

Conviction نسبت فرکانس مورد انتظار مشاهده A بدون B -یا به عبارتی فرکانس پیش‌بینی اشتباه- با فرض استقلال A و B به فرکانس پیش‌بینی‌های نادرست را بیان می‌کند. Conviction احتمال وقوع A بدون B با فرض استقلال را با احتمال واقعی وقوع A بدون B را مقایسه می‌کند. برای مثال اگر $conv(A \to B) = 1.5$ به این معنی است که اگر A و B از هم مستقل بودند، یا به عبارتی وابستگی‌شان کاملا تصادفی بود، $50\,\%$ بیشتر پیش‌بینی نادرست صورت می‌گرفت.

ب)

از سه کمیت تعریف شده تنها $lift$ نسبت به A و B متقارن است. اثبات:

$confidence$:

$$conf(A \to B) = P(B|A) = \frac{S(A \cap B)}{S(B)}$$

به وضوح عبارت بالا نسبت به A و B متقارن نیست.

$lift$:

$$lift(A \to B) = \frac{conf(A \to B)}{S(B)} = \frac{S(A \cap B)}{S(A)S(B)}$$

پس نسبت به A و B متقارن است.

$conviction$:

$$conv(A \to B) = \frac{1 - S(B)}{1 - conf(A \to B)} = \frac{S(A) - S(A)S(B)}{S(A) - S(A \cap B)}$$

عبارت بالا نسبت به A و B متقارن نیست.

ج)

در تعریف confidence فراوانی خود B در نظر گرفته نشده است. فرض کنید فرکانس B بسیار بالا باشد. در این صورت به احتمال خیلی زیاد فرکانس $A \cap B$ نیز بالا می‌رود. پس مقدار confidence نیز افزایش می‌یابد. این در حالی است که زیاد بودن این کمیت به خاطر وابستگی میان A و B نیست. درواقع در این حالت کمیت confidence به ما اطلاعاتی نمی‌دهد.

- **سوال ۲**

از آنجایی که ساپورت ترشولد را ۵ در نظر گرفته‌ایم، هر maximal frequent itemset باید در حداقل ۵ سبد آمده باشد و هر superset مستقیمش در کمتر از ۴ سبد. می‌دانیم که هر iteemsetای که مشاهده شده است، در سبدی آمده‌است که شماره‌اش مضرب مشترکی از آیتم‌های موجود در آن itemset است.
اگر هر تک آیتم را بررسی کنیم می‌بینیم که تنها آیتم‌های ۱ تا ۲۰ در حداقل ۵ سبد آمده‌اند. پس آیتم‌های بیشتر از ۲۰ نمی‌توانند عضوی از یک maximal frequent itemset باشند.

حال فرض کنید $I = \{i_1, i_2, \cdots, i_k\}$ یک itemset باشد که ک.م.م عناصرش از ۲۰ بیشتر است. در این صورت بزرگترین عنصر هر سبدی که این itemset زیرمجوعه‌ای از آن است از ۲۰ بیشتر است. پس نمی‌تواند این itemset در حداقل ۵ سبد بیاید. پس یک maximal frequent itemset نیست. پس ک.م.م عناصر هر maximal frequent itemset باید حداکثر ۲۰ باشد.

پس آیتم‌های هر maximal frequent itemset باید از ۱ تا ۲۰ باشد.

سبدهای ۱۱ تا ۲۰ را در نظر بگیرید. کل itemset هر کدام از این سبدها maximal است. زیرا فرکانس هر کدام حداقل ۵ است و اگر هر superset از آنها در نظر بگیریم، ک.م.م آن از ۲۰ بیشتر می‌شود. پس فرکانسش از ۵ کمتر می‌شود. اگر هر سابست از آیتم‌های این سبد را در نظر بگیریم به وضوح maximal نخواهد بود.
سبدهای ۱ تا ۱۰ نیز هر کدام زیرمجموعه سبدهای ۱۱ تا ۲۰ هستند پس نمی‌توانند maximal باشند. از طرفی اگر هر itemset دلخواه از آیتم‌های ۱ تا ۲۰ را درنظر بگیریم که ک.م.مشان کوچکتر یا مساوی ۲۰ است، زیرمجموعه‌ای از سبدی از سبدهای ۱۱ تا ۲۰ خواهد بود. پس نمی‌تواند maximal باشد.

پس تمام maximal frequent itemsetها عبارت‌اند از:

[1,11], [1,2,3,4,6,12], [1,13], [1,2,7,14], [1,3,5,15],
[1,2,4,8,16], [1,17], [1,2,3,6,9,18], [1,19], [1,2,3,4,5,10,20]

- **سوال ۳**

جزئیات پیاده‌سازی الگوریتم A-Priori

مرحله 1: مجموعه آیتم های کاندید را ایجاد می‌کنیم.

در این مرحله مجموعه آیتم‌های کاندید را تولید می کنیم. ما با اولین مجموعه اقلام کاندید شروع می‌کنیم، که مجموعه‌ای از تمام itemهای منحصر به فرد در transactionها است.

برای تولید مجموعه آیتم‌های کاندید بعدی، روی مجموعه آیتم‌های کاندید قبلی تکرار می کنیم و با پیوستن به مجموعه آیتم‌ها، مجموعه آیتم‌های نامزد بعدی را ایجاد می‌کنیم. سپس روی تراکنش‌ها iterate می‌کنیم و بررسی می کنیم که آیا مجموعه آیتم‌های کاندید زیر مجموعه ای از تراکنش است یا خیر. اگر چنین باشد، تعداد مجموعه آیتم‌های نامزد را افزایش می‌دهیم. سپس روی مجموعه آیتم‌های کاندید این کار را تکرار می‌کنیم و اگر تعداد آن‌ها از support threshold بیشتر یا مساوی باشد، مجموعه آیتم‌ها را به مجموعه آیتم‌های frequent اضافه می‌کنیم.

مرحله 2: مجموعه آیتم‌های frequent را ایجاد می‌کنیم.

در این مرحله مجموعه آیتم‌های مکرر را تولید می‌کنیم. ما روی مجموعه آیتم‌های نامزد iterate می‌کنیم و اگر تعداد آن‌ها از آستانه حمایت بیشتر یا مساوی باشد، مجموعه آیتم‌ها را به مجموعه آیتم‌های frequent اضافه می‌کنیم. سپس مجموعه آیتم‌های کاندید بعدی را با join کردن مجموعه آیتم‌های frequent ایجاد می‌کنیم. این روند را تا زمانی تکرار می‌کنیم که مجموعه آیتم‌های کاندید دیگری نداشته باشیم.

اجرای الگوریتم A-Priori

برای تولید همه مجموعه‌های frequent، الگوریتم A-Priori را روی تراکنش‌ها اجرا می کنیم. در هر iteration، مجموعه آیتم‌های کاندید و مجموعه آیتم‌های مکرر را تولید می‌کنیم. سپس مجموعه آیتم‌های frequent را به لیست همه مجموعه‌های اقلام frequent اضافه می‌کنیم. سپس مجموعه آیتم‌های کاندید بعدی را با پیوستن به مجموعه آیتم‌های frequent ایجاد می‌کنیم. این روند را تا زمانی تکرار می‌کنیم که دیگر مجموعه آیتم‌های کاندیدی نداشته باشیم یا به حداکثر تعداد آیتم‌ها در مجموعه آیتم‌ها برسیم.

```python
def generate_next_candidate_set(prev_frequent_set, length):
    next_candidate_set = [var1 | var2 for index, var1 in enumerate(prev_frequent_set) for var2 in prev_frequent_set[index + 1:]
                          if list(var1)[:length - 2] == list(var2)[:length - 2]]
    return next_candidate_set
```

```python
def generate_frequent_itemset_k(sc, Ck, shared_itemset, min_supp):
    def get_supp(x):
        x_supp = len([1 for itemset in shared_itemset.value if x.issubset(itemset)])
        if x_supp >= min_supp:
            return x, x_supp
        else:
            return ()
    freq_itemset_k = sc.parallelize(Ck).map(get_supp).filter(lambda x: x != ()).collect()
    return freq_itemset_k
```

```python
def apriori(sc, itemset_rdd, min_sup, max_k):
    # share the whole itemset with all workers
    shared_itemset = sc.broadcast(itemset_rdd.map(lambda x: set(x)).collect())
    # store for all freq_k
    frequent_itemset = []

    # prepare candidate_1
    k = 1
    c_k = itemset_rdd.flatMap(lambda x: set(x)).distinct().collect()
    c_k = [{x} for x in c_k]

    # when candidate_k is not empty
    while len(c_k) > 0 and k <= max_k:
        # generate freq_k
        f_k = generate_frequent_itemset_k(sc, c_k, shared_itemset, min_sup)

        frequent_itemset.append(f_k)
        # generate candidate_k+1
        k += 1
        c_k = generate_next_candidate_set([set(item) for item in map(lambda x: x[0], f_k)], k)

    sc.stop()
    return frequent_itemset
```

```
1  frequentItemSets[1]
```

Output exceeds the size limit. Open the full output data in a text editor

```
[({'ELE17451', 'SNA90258'}, 113),
 ({'GRO99222', 'SNA90258'}, 156),
 ({'DAI62779', 'SNA90258'}, 114),
 ({'DAI22896', 'GRO73461'}, 304),
 ({'GRO73461', 'SNA69641'}, 150),
 ({'ELE59935', 'GRO73461'}, 116),
 ({'DAI22177', 'GRO73461'}, 248),
 ({'ELE66810', 'GRO73461'}, 228),
 ({'GRO36567', 'GRO73461'}, 117),
 ({'GRO73461', 'SNA55952'}, 117),
 ({'DAI48891', 'GRO73461'}, 117),
 ({'ELE11111', 'GRO73461'}, 158),
 ({'FRO16142', 'GRO73461'}, 197),
 ({'FRO24098', 'GRO73461'}, 112),
 ({'GRO73461', 'SNA59903'}, 123),
 ({'DAI55911', 'GRO73461'}, 116),
 ({'FRO31317', 'GRO73461'}, 395),
 ({'GRO73461', 'SNA72163'}, 285),
 ({'DAI63921', 'GRO73461'}, 219),
 ({'GRO73461', 'SNA18336'}, 121),
 ({'DAI91290', 'GRO73461'}, 161),
 ({'ELE12792', 'GRO73461'}, 116),
 ({'GRO73461', 'GRO85051'}, 147),
 ({'DAI73122', 'GRO73461'}, 146),
 ({'FRO73056', 'GRO73461'}, 195),
...
 ({'GRO30386', 'SNA99873'}, 112),
 ({'SNA93860', 'SNA99873'}, 105),
 ({'GRO59710', 'SNA99873'}, 124),
 ({'GRO15017', 'SNA99873'}, 154),
 ...]
```

```
1  frequentItemSets[2]
```

Output exceeds the size limit. Open the full output data in a text editor

```
[({'DAI22896', 'DAI62779', 'GRO73461'}, 101),
 ({'DAI62779', 'FRO31317', 'GRO73461'}, 100),
 ({'DAI88807', 'GRO73461', 'SNA72163'}, 110),
 ({'FRO40251', 'GRO73461', 'GRO85051'}, 147),
 ({'FRO73056', 'GRO44993', 'GRO73461'}, 106),
 ({'ELE32164', 'GRO59710', 'GRO73461'}, 137),
 ({'DAI62779', 'ELE32164', 'GRO73461'}, 131),
 ({'DAI43223', 'ELE32164', 'GRO73461'}, 111),
 ({'DAI88079', 'FRO40251', 'GRO73461'}, 144),
 ({'DAI75645', 'GRO73461', 'SNA80324'}, 230),
 ({'DAI62779', 'GRO73461', 'SNA80324'}, 198),
 ({'FRO40251', 'GRO73461', 'SNA80324'}, 232),
 ({'DAI75645', 'FRO47962', 'GRO73461'}, 111),
 ({'DAI75645', 'ELE17451', 'GRO73461'}, 121),
 ({'DAI62779', 'DAI75645', 'GRO73461'}, 261),
 ({'DAI75645', 'FRO40251', 'GRO73461'}, 293),
 ({'DAI75645', 'GRO21487', 'GRO73461'}, 114),
 ({'DAI75645', 'GRO46854', 'GRO73461'}, 101),
 ({'DAI62779', 'GRO73461', 'SNA96271'}, 114),
 ({'FRO40251', 'GRO56726', 'GRO73461'}, 103),
 ({'DAI62779', 'GRO71621', 'GRO73461'}, 153),
 ({'DAI62779', 'DAI85309', 'GRO73461'}, 179),
 ({'DAI62779', 'FRO19221', 'GRO73461'}, 142),
 ({'ELE17451', 'GRO30386', 'GRO73461'}, 103),
 ({'DAI62779', 'ELE17451', 'GRO73461'}, 245),
...
 ({'DAI62779', 'ELE92920', 'FRO40251'}, 152),
 ({'DAI62779', 'ELE92920', 'GRO81087'}, 134),
 ({'DAI62779', 'DAI83733', 'ELE92920'}, 103),
 ({'DAI42493', 'DAI62779', 'ELE92920'}, 112),
 ({'DAI23334', 'DAI62779', 'ELE92920'}, 143)]
```

- **سوال ۴**

ابتدا داده‌ها را می‌خوانیم. برای این کار schem زیر را تعریف می‌کنیم و سپس براساس آن با استفاده از تابع read.csv رکوردها را از فایل csv می‌خوانیم.

```
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)
    ])
```

قسمتی از داده‌ها که برای این سوال مورد نیاز است را با map زیر جدا می‌کنیم:

```
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], sorted(list(set(list(x[1]))))))\
                .filter(lambda x: len(x[1]) < 20)\
                .map(lambda x: x[1])
```

رکوردها را براساس روز و پلاک هر ماشین گروه‌بندی می‌کنیم. این کار با استفاده از متد groupByKey انجام می‌شود. سپس لیست دوربین‌های یکتای هر کلید را مرتب می‌کنیم تا همه عناصر در کل فرآیند ترتیب مشخص داشته باشند. سپس داده‌های مربوط به ماشین‌هایی اند که در یک روز از بیشتر از ۲۰ دوربین متفاوت گذشته‌اند را دور می‌اندازیم.

در این سوال از آنجایی که حجم داده‌ها بیشتر بود، پیاده‌سازی قبلی جواب نداد. به همین خاطر برای بهبود عملکرد الگوریتم از هیوریستیک‌هایی مانند الزام freqeunt بودن subsetهای هر frequent itemset استفاده شده‌است. برای این کار از تابع prune_candidates استفاده می‌کنیم. الگوریتم را نیز به‌صورت بازگشتی پیاده کرده‌ایم.

```
def prune_candidates(x, Ck, n):
    combs = list(combinations(x, n))
    return all(i in Ck for i in combs)
```

```python
def apriori(traffic_rdd, support_threshold, n):
    if n == 1:
        f_1 = traffic_rdd.flatMap(lambda x: x)\
                .map(lambda x: (x, 1))\
                .reduceByKey(lambda x, y: x + y)\
                .filter(lambda x: x[1] >= support_threshold)\
                .map(lambda x : ([x[0]], x[1]))\
                .collect()
        f_1 = {tuple(x[0]): x[1] for x in f_1}
        frequent_paths[1] = f_1
        return f_1

    Cn_1 = apriori(traffic_rdd, support_threshold, n - 1)
    f_n = traffic_rdd.flatMap(lambda x: combinations(x, n))\
                .filter(lambda x: prune_candidates(x, Cn_1, n - 1))\
                .map(lambda x: (x, 1))\
                .reduceByKey(lambda x, y: x + y)\
                .filter(lambda x: x[1] >= support_threshold)\
                .collect()
    f_n = {tuple(x[0]): x[1] for x in f_n}
    frequent_paths[n] = f_n
    return f_n
```

حال الگوریتم A-priori را اجرا می‌کنیم. برای ترشولد ۱۰۰۰ مجموعه مسیرهای پرتردد عبارت‌اند از:

```
1  frequent_paths[4]
```

```
{(900102, 900142, 900212, 900244): 1799,
 (900101, 900212, 900244, 100700839): 1105,
 (22010087, 22010088, 22010094, 22010095): 2083,
 (900142, 900202, 900212, 900244): 1017,
 (900142, 900212, 900249, 100700853): 1295,
 (900193, 900212, 900244, 100700839): 1463,
 (900142, 900212, 900273, 100700853): 1044,
 (900212, 900244, 100700839, 100700853): 1299,
 (142, 900215, 900234, 900256): 1055,
 (900142, 900152, 900212, 100700853): 1299,
 (900142, 900212, 900244, 100700853): 4784,
 (900102, 900212, 900244, 100700853): 1134,
 (900139, 900212, 900244, 100700826): 1315,
 (900142, 900193, 900212, 900244): 1126,
 (900142, 900212, 900244, 900249): 1309,
 (900142, 900152, 900212, 900244): 1549,
 (900193, 900212, 900244, 100700853): 1071,
 (209103, 900265, 100700804, 100700834): 1276,
 (900139, 900212, 900244, 100700839): 1093,
 (900101, 900212, 900244, 100700841): 1467,
 (900102, 900142, 900212, 100700853): 1088,
 (631633, 900142, 900212, 900244): 1281,
 (900142, 900212, 900244, 100700839): 2251,
 (900215, 900234, 900256, 22010118): 1341,
 (900152, 900212, 900244, 100700853): 1264,
 (900225, 900259, 900268, 900269): 1094,
 (231, 900101, 900236, 100700841): 1076,
 (231, 900236, 900255, 100700841): 1307}
```

الگوریتم SON

این الگوریتم درواقع یک الگوریتم 2-pass است. در pass اول ابتدا داده‌های را بین ۳تا RDD تقسیم می‌کنیم. سپس روی هر کدام الگوریتم A-Priori را با ترشولد $\frac{1}{3}$ ترشولد اصلی اجرا می‌کنیم و نتایج بدست‌آمده را ادغام می‌کنیم.

برای تقسیم داده‌ها بین ۳ RDD به این شکل عمل می‌کنیم:

```
traffic_rdd = traffic_rdd.zipWithIndex()
```

تابع فوق به هر عنصر RDD یک index می‌دهد. حال برای تقسیم این عناصر بین ۳ RDD از باقی‌مانده index هر عنصر به پیمانه ۳ استفاده می‌کنیم. سپس روی هر RDD الگوریتم A-Priori را اجرا و نتایج را ادغام می‌کنیم. در اینجا ما itemsetهای ۳تایی را بررسی می‌کنیم:

```
traffic_rdd_1 = traffic_rdd.filter(lambda x : x[1]%3==0).map(lambda x : x[0])
traffic_rdd_2 = traffic_rdd.filter(lambda x : x[1]%3==1).map(lambda x : x[0])
traffic_rdd_3 = traffic_rdd.filter(lambda x : x[1]%3==2).map(lambda x : x[0])

support_threshold = support_threshold / 3

f_n_1 = apriori(traffic_rdd_1, support_threshold, 3)
f_n_2 = apriori(traffic_rdd_2, support_threshold, 3)
f_n_3 = apriori(traffic_rdd_3, support_threshold, 3)
```

```
f_n = {}
for key in f_n_1:
    f_n[key] = f_n_1[key]
for key in f_n_2:
    if key in f_n:
        f_n[key] += f_n_2[key]
    else:
        f_n[key] = f_n_2[key]
for key in f_n_3:
    if key in f_n:
        f_n[key] += f_n_3[key]
    else:
        f_n[key] = f_n_3[key]

f_n
```

در pass دوم داده اصلی را براساس نتیجه ادغام شده فیلتر می‌کنیم. مسیرهای ۳تایی که فرکانیسشان از ترشولد بیشتر بوده‌اند و در نتیجه pass اول آمده‌اند انتخاب می‌شوند.

```
1  f_3_son
```

```
Output exceeds the size limit. Open the full output data in a text editor
[((900259, 900269, 100700841), 1361),
 ((900235, 100700804, 100700834), 1216),
 ((900207, 900225, 900269), 1181),
 ((230204, 900107, 900276), 2535),
 ((205802, 900215, 900234), 1823),
 ((205802, 900234, 900265), 1229),
 ((212802, 900215, 900234), 1341),
 ((900139, 900268, 100700826), 2691),
 ((631765, 900164, 900276), 1159),
 ((631765, 900164, 100700820), 2598),
 ((631765, 900276, 100700820), 1719),
 ((206602, 900234, 100700845), 1859),
 ((900101, 22010119, 100700841), 1229),
 ((900102, 900142, 100700853), 1293),
 ((900142, 900212, 900259), 1143),
 ((631357, 900212, 900244), 1381),
 ((631357, 900102, 100701130), 1682),
 ((631633, 900212, 900244), 3383),
 ((631829, 900226, 900246), 1037),
 ((205802, 212802, 900233), 1795),
 ((900142, 900212, 22010119), 1198),
 ((900215, 900234, 900256), 2223),
 ((900101, 900212, 900244), 5775),
 ((900142, 900212, 100700839), 2682),
 ((900142, 900244, 100700839), 2494),
...
 ((900142, 900245, 100700853), 1435),
 ((900244, 100700839, 100700853), 1518),
 ((900212, 100700839, 100700853), 1533),
 ((900139, 100700826, 100700841), 1417),
 ((900101, 900244, 100700841), 1705)]
```

الـگوریتم SON عملکرد خـود را در یک کلاسـتر واقعی نـشان می‌دهـد. در این سـوال چـون مـا تـنها از یک مـاشین استفاده می‌کنیم حتی تـا حـدودی از الـگوریتم A-Priori کندتـر عـمل می‌کند. زیرا مـنابع مـا یکسان است و در SON یک pass بیشتر هم داریم.

تـرشولـد انـتخاب شـده صـرفا بـراساس یک rule of thumb اسـت. در واقع مـا چیزی حـدود 0.1% تـعداد رکوردها را به عنوان support انتخاب کرده‌ایم که انتخاب معقولی است.

بخشی از داده‌های دور افتاده درواقع همان‌هایی بودند که در یک روز از بیش از ۲۰ دوربین متفاوت عبور کرده‌اند. همچنین مـا لیست دوربین‌های مـربوط به یک مـاشین را بـه set تـبدیل می‌کنیم. زیرا itemهـای تکراری نباید تاثیری در الگوریتم ما داشته‌باشند.

حـذف شدن دوربین‌های دور از هم به دلیل اصل monotonicity است. اگر دو دوربین C_1 و C_2 نزدیک هم نباشند، آنگاه اگر $\{C_1, C_2\}$ بخواهد frequent باشد، خود C_1 و C_2 نیز باید frequent باشند. که احـتمال چنین چیزی بسیار کم است. مـخصوصـا اگـر انـدازه تـرکیب بـزرگـتر شود که در این صـورت احـتمال frequent بـودن subsetهـای آن‌هـا که مکانـشان کنار هـم نیست نـاچیز می‌شـود و در نتیجه حـذف می‌شود.

امکان بررسی رفتارهای تکرارشونده دیگر نیز وجود دارد. برای مثال اگر بخواهیم زمان‌های پر تردد را پیدا کنیم کافی‌اسـت جـفت‌هـایمان را بـه صـورت <FINAL_CAR_KEY, PASS_DAY_TIME> بگیریم. چنین رفـتار تکرارشـونـده‌ای معنی‌دار نیز هسـت. اگـر بـازه‌هـای مـشخص از زمـان را در نـظر بگیریم، آنـهایی که بیشترین تعداد ماشین در آنها تردد داشته‌اند مطلوب ما هستند.
برای خودروهای پرتردد نیز کافی است جفت‌های
<(FINAL_CAR_KEY, PASS_DAY_TIME), DEVICE_CODE>
را در نـظر بگیریم.سپس groupByKey کنیم و سپس بـراساس FINAL_CAR_KEY این کار را تکرار کنیم و طـول لیست زمـان‌هـا را مـقایسه کنیم. می‌تـوان این groupByKey را نیز انـجام نـداد و صـرفا تـعداد را محاسبه کرد که این بستگی به اطلاعات مورد نظر دارد.