به نام خدا

درس تحلیل کلان داده‌ها

تمرین سری ۱

سنا موحدین ۹۹۳۱۰۸۰

سوال ۱

- بخش ۱-۱:

(عنوان فایل کد مربوط به این بخش q1-1 است.)

a) خروجی مربوطه:

```
Product Category: Electronics
Total Revenue: 101324330.20000139

Product Category: Books
Total Revenue: 100772199.18999891

Product Category: Sports & Outdoors
Total Revenue: 101433008.11000131

Product Category: Home & Kitchen
Total Revenue: 101214302.76999967

Product Category: Clothing
Total Revenue: 100953812.93999816
```

b) خروجی مربوطه:

```
Top 10 Customers by Purchase Volume:
1. Customer ID: 9179, Purchase Volume: $79828.49000000002
2. Customer ID: 2959, Purchase Volume: $79175.12999999996
3. Customer ID: 8818, Purchase Volume: $78446.82000000002
4. Customer ID: 9161, Purchase Volume: $78006.31
5. Customer ID: 8907, Purchase Volume: $76459.56999999999
6. Customer ID: 2973, Purchase Volume: $75656.47000000004
7. Customer ID: 3870, Purchase Volume: $75642.15999999999
8. Customer ID: 9637, Purchase Volume: $75586.39
9. Customer ID: 5820, Purchase Volume: $75461.57999999999
10. Customer ID: 6763, Purchase Volume: $75371.32000000004
```

c) برای انجام فرایند خواسته شده، می‌توانیم از map و reduce استفاده کنیم. هنگامی که دیتا را از فایل می‌خوانیم. هر خط را به بخش‌های مختلفی که دارد، split کرده و به عنوان خروجی تابع read_data قرار می‌دهیم. سپس کل این دیتا را به عنوان ورودی تابع map می‌گذاریم. بخش a و b در این سوال بسیار شبیه به یکدیگرند و تنها کلید و مقداری که در هر بخش نیاز است متفاوت است. این تفاوت در تابع map موثر است. به همین دلیل در ورودی این تابع یک پارامتر با عنوان question_part هم قرار داده‌ام و در ادامه مشخص شده که اگر مربوط به بخش a بود، دو تایی‌های مربوطه چه مواردی باشند و در غیر این صورت چطور. در بخش a، product category به عنوان کلید و در بخش b، costumer_id باید به این عنوان قرار داشته باشد. همچنین مقادیر در هر دو قسمت، مربوط به قیمت و هزینه است. خروجی این تابع به ازای هر خط دیتا به صورت یک دو تایی که شامل موارد گفته شده است، می‌باشد. این تابع به ازای هر entry در فایلمان صدا زده می‌شود. در ادامه خروجی تابع map برای هر دو بخش را می‌بینیم. (در هر دو بخش مشخص شده که خروجی تابع برای ده خط اول پرینت شود.)

خروجی تابع map برای بخش a:

```
Data after Map Function: (for part a)
[('Electronics', 590.18), ('Books', 537.2), ('Sports & Outdoors', 130.1), ('Home & Kitchen', 12
7.13), ('Home & Kitchen', 686.1), ('Books', 827.38), ('Clothing', 198.82), ('Books', 564.84), (
'Books', 858.33), ('Home & Kitchen', 263.86)]
```

خروجی تابع map برای بخش b:

```
Data after Map Function: (for part b)
[('5844', 590.18), ('1221', 537.2), ('1462', 130.1), ('8885', 127.13), ('7888', 686.1), ('8520'
, 827.38), ('9569', 198.82), ('6536', 564.84), ('8936', 858.33), ('9310', 263.86)]
```

در ادامه تابع group_by را داریم. این تابع خروجی تابع map را به عنوان ورودی دریافت می‌کند. فرآیندی که باید در این تابع صورت بگیرد این است که به ازای هر کلید، مقادیر متناظرش همه در کنار هم قرار بگیرند. پس به این صورت عمل می‌کنیم که به ازای هر دوتایی (key, value) که داریم، اگر کلید قبلا اضافه نشده، کلید را به همراه مقدارش اضافه می‌کنیم و اگر کلید از دوتایی‌های قبلی اضافه شده باشد، تنها مقدار این دوتایی جدید را به مقادیر مربوط به کلید، append می‌کنیم. پس در نهایت به ازای هر کلید، مجموعه‌ای از تمام مقادیر متناظرش خواهیم داشت. در ادامه برای هر دو بخش a و b نمونه‌ای از خروجی این تابع قرار داده می‌شود.

خروجی تابع group_by برای بخش a:

این اسکرین‌شات تنها برخی از مقادیر مربوط به کلید electronics را نشان می‌دهد. (در نهایت ۵ دیکشنری خواهیم داشت که کلیدهایشان همان ۵ دسته هستند و valueهای هر کدام هم به صورت زیر قرار داده شده.)

```
Data after Group-By Function: (for part a)
dict_items([('Electronics', [590.18, 762.74, 558.02, 442.24, 159.57, 607.83, 398.59, 338.74,
620.37, 243.27, 731.95, 502.33, 197.02, 479.81, 919.68, 589.09, 35.33, 446.85, 998.8, 647.42
865.22, 136.8, 512.21, 314.1, 581.16, 690.9, 777.61, 534.96, 311.81, 718.97, 851.73, 785.76,
1.8, 777.82, 639.1, 327.72, 660.12, 885.35, 833.4, 713.89, 474.8, 825.79, 456.25, 162.45, 17
.36, 534.36, 58.26, 174.8, 493.55, 298.98, 950.65, 97.08, 563.46, 777.87, 580.59, 748.66, 60
96, 297.29, 214.98, 876.9, 899.72, 929.09, 26.49, 321.19, 589.01, 383.83, 446.52, 502.68, 52
.5, 464.27, 305.08, 846.39, 606.52, 348.09, 158.95, 273.98, 804.35, 438.52, 179.15, 203.86,
15, 962.14, 645.95, 527.63, 680.38, 337.47, 687.5, 829.85, 132.41, 734.32, 247.54, 45.03, 96
```

خروجی تابع group_by برای بخش b:

به ازای هر کاربر، چنین لیستی از مخارجش خواهیم داشت. البته این اسکرین‌شات تنها برخی از مقادیر مربوط به کاربر 5844 را نشان می‌دهد.

```
Data after Group-By Function: (for part b)
dict_items([('5844', [590.18, 356.74, 529.85, 574.2, 348.07, 235.29, 800.51, 609.51, 278.6,
77, 137.28, 842.48, 505.08, 698.31, 732.13, 325.97, 287.87, 246.41, 486.96, 155.76, 908.52,
89.51, 367.94, 298.45, 423.25, 395.06, 704.2, 488.41, 756.13, 210.1, 243.44, 243.15, 176.18,
52.85, 248.18, 282.29, 357.69, 61.24, 923.83, 680.39, 226.78, 301.97, 958.05, 488.77, 449.56
1.04, 817.05, 515.04, 352.42, 286.85, 249.08, 741.4, 499.03, 860.26, 122.69, 994.88, 168.8,
, 842.47, 43.12, 906.93, 560.29, 125.43, 382.96, 166.32, 878.93, 173.47, 951.89, 286.14, 938
, 361.64, 980.85, 296.82, 512.45, 241.75, 94.67, 382.91, 680.05, 946.36, 490.76, 420.75, 246
5, 482.9, 442.17, 181.73, 516.44, 147.14, 236.12, 729.84, 336.03, 115.95, 802.4, 316.28, 414
```

در نهایت از تابع reduce استفاده می‌کنیم. در هر دو بخش a و b، نوع تابعی که در این مرحله نیاز است یک جور است. در هر دو قرار است جمع مقادیر حساب شود. پس این تابع خروجی تابع group_by را دریافت کرده و برای هر کلید، مقادیر را جمع می‌کند.

در نهایت برای بخش a، این مقادیر را برای هر دسته‌بندی برمی‌گرداند. برای بخش b هم با توجه به مقدار بدست آمده، کاربران را سورت کرده و ۱۰ کاربر اول از نظر میزان خرج را برمی‌گرداند. اسکرین شات مربوط به خروجی این مراحل بالاتر قرار داده شده بود.

d) تابع map برای رعایت و افزایش efficincy، روی یک ورودی در هر زمان عمل می‌کند، فیلدهای لازم را استخراج کرده و آنها را به یک فرمت مناسب برای پردازش بیشتر نگاشت می‌کند. این تابع با عدم بارگیری کل مجموعه داده به یکباره در حافظه، به بهره‌وری حافظه اطمینان می‌دهد. تابع group_by برای بهینه‌سازی استفاده از حافظه، داده‌های گروه‌بندی شده را در یک دیکشنری ذخیره می‌کند. و در نهایت تابع reduce بدون هیچ پیچیدگی خاصی به طور بهینه مقادیر نهایی را محاسبه می‌کند. رویکردهای پیش گرفته شده باعث کاهش استفاده ناپایدار از حافظه شده و از حجم محاسبات غیرضروری هم جلوگیری می‌کند.

- بخش ۲-۱:

(عنوان فایل کد مربوط به این بخش q1-2 است.)

a) خروجی مربوطه:

```
Top 10 users:
1. User ID: 909087, Number of actions: 9
2. User ID: 917845, Number of actions: 8
3. User ID: 852131, Number of actions: 8
4. User ID: 819422, Number of actions: 8
5. User ID: 787417, Number of actions: 8
6. User ID: 749904, Number of actions: 8
7. User ID: 741832, Number of actions: 8
8. User ID: 627332, Number of actions: 8
9. User ID: 435727, Number of actions: 8
10. User ID: 403101, Number of actions: 8
```

یکی از نکاتی که در این سوال وجود داشت، این بود که برخی کاربرها تعداد actionهای یکسانی داشتند. با توجه به این که محدودیتی از این نظر در صورت سوال قرار داده نشده بود، این طور تعریف کردم که وقتی این حالت پیش آمد، کاربر با آیدی بزرگتر اول قرار بگیرد. پس با توجه به نوع سورت شدن و محدودیت‌هایی که می‌توان تعریف کرد، آیدی یوزرهایی که از شماره ۲ تا ۱۰ گزارش شده می‌تواند بین افراد متفاوت باشد.
همچنین در این سوال کل بازه زمانی برای محاسبه‌ی فعالترین کاربر در نظر گرفته شده.

b) این سوال بسیار شبیه به بخش قبلی است. ابتدا از تابع map استفاده می‌کنیم و به ازای هر entry، یک دو تایی از user_id و action تشکیل می‌دهیم. با توجه به این که صورت سوال تغییر کرد و بنا بر این شد که فعالترین کاربر از جهت تعداد like, share, comment گزارش شود، در مرحله group_by یوزر آیدی‌ها به عنوان کلید و اکشن‌های صورت گرفته به عنوان مقادیر آن کلید قرار داده می‌شوند. در نهایت در تابع reduce، تعداد اکشن‌ها شمرده و برای هر کاربر برگردانده می‌شود. این تنها تفاوت در تابع reduce این بخش و بخش قبلی است که در اینجا تعداد اکشن‌ها و در سوال قبل جمع مقادیر مهم بود.

c) در بخش b تا حدودی این موارد توضیح داده شد. در این بخش اسکرین‌شات خروجی map و group_by را هم برای بخشی از دیتا قرار می‌دهم.

خروجی map:

```
Data after Map Function:
[('963962', 'share'), ('558390', 'post'), ('192442', 'comment'), ('792113', 'like'), ('507769
', 'comment'), ('566401', 'like'), ('203339', 'share'), ('422252', 'comment'), ('593306', 'po
st'), ('335538', 'share')]
```

خروجی group_by:

```
Data after Group-By Function:
dict_items([('963962', ['share']), ('558390', ['post', 'comment', 'share', 'comment']),
e']), ('203339', ['share', 'share']), ('422252', ['comment', 'post']), ('593306', ['post
'like']), ('951379', ['comment']), ('211513', ['comment']), ('39293', ['share', 'post'])
821', ['share', 'post', 'comment']), ('471538', ['comment', 'post', 'share', 'like']), (
ost', 'post']), ('50059', ['comment', 'comment']), ('492855', ['like', 'share']), ('8737
9343', ['post']), ('385081', ['post', 'comment', 'like']), ('336907', ['post', 'like', '
```

d) مهمترین مساله‌ای که باید در چنین سوالی مدنظر قرار بگیرد، همان حجم زیاد دیتا است. انتخاب روش MapReduce برای یافتن پاسخ این سوال و بررسی دیتاست، خود یکی از موثرترین پیش‌بینی‌ها برای رویارویی با آن است. در این مراحل، bottleneck ما تابع group_by است. پس باید سعی کنیم تا حد امکان این تابع را ساده پیاده کرده و از هر پیچیدگی جلوگیری کنیم. مثلا یکی از کارهایی که صورت گرفته، انجام این مرحله تنها با یک iteration است تا از حجم محاسبات اضافه و پیچیده جلوگیری شود.

در این سوال یکی از چالش‌های مهم مربوط به دیتاست داده شده بود که حتی باعث شد صورت سوال تغییر کند. در ابتدا خواسته شده بود که تاثیرگذارترین کاربر گزارش شود و این طور تعیین شده بود که منظور از تاثیرگذارترین، تعداد تعاملاتی است که پست آنها دریافت می‌کند. در حالی که در دیتاستی که در اختیار ما قرار گرفت، بنظر می‌رسد منظور از post_id، پستی است که هر کاربر روی آن اکشن خود (لایک، کامنت یا شیر) را اعمال کرده، نه پستی که خودش آپلود کرده است. از طرف دیگر ممکن است تصور شود وقتی در دیتاست entry شبیه به مورد زیر وجود دارد، منظور این است که پست شماره ۱۱۲۱ را کاربر ۱۸۷۳۴۸ آپلود کرده و در تاریخ بیان شده، یک عملیات share روی آن صورت گرفته.

(187348,1121,share,2022-10-24 23:58:17)

پس اطلاعات درون این دیتاست واضح و مشخص نیست.

چیزی که باعث می‌شود فرض اول درست‌تر بنظر برسد، این است که دیده می‌شود آیدی بعضی پست‌ها در بیش از دو entry وجود دارد. این در حالی می‌تواند درست باشد که آیدی پست، پستی که روی آن اکشن صورت گرفته را نشان دهد (یعنی نمی‌دانیم چه کسی آن را آپلود کرده.)

سوال ۲

- بخش ۱-۲:

(a

basket ID	Items	pairs with their count (support)
1	1,2,3,5	{(1,2):4}, {(1,3):4}, {(1,5):8}, {(2,3):7}, {(2,5):2}, {(3,5):3}
2	1,2,3	{(1,2):4}, {(1,3):4}, {(2,3):7}
3	2,3	{(2,3):7}
4	1,5	{(1,5):8}
5	1,2,3	{(1,2):4}, {(1,3):4}, {(2,3):7}
6	1,3,5	{(1,3):4}, {(1,5):8}, {(3,5):3}
7	1,2,5	{(1,2):4}, {(1,5):8}, {(2,5):2}
8	1,5	{(1,5):8}
9	2,3	{(2,3):7}
10	1,5	{(1,5):8}
11	2,3	{(2,3):7}
12	1,5	{(1,5):8}
13	3,5	{(2,3):7}
14	2,3	{(2,3):7}
15	1,5	{(1,5):8}

item	count (support)
1	10
2	8
3	9
4	6
5	9

جدول item و count در حقیقت به pass 1 و جدول دیگر به pass 2 مربوط هستند. در pass 1، همان طور که دیده می‌شود ساپورت 4 از 7 کمتر است، پس تنها این item حذف می‌شود. در pass 2 باید بررسی کنیم که این مقدار نسبت به threshold که مقدارش 7 است چگونه است. با انجام این کار می‌فهمیم کدام جفت‌ها را باید نگه داریم. طبق جدول تنها دو جفت (2,3) و (1,5) فرکانس کمتر از ۷ ندارند.

b) با توجه به آنچه در بخش قبل توضیح داده شد، hash شدن به bucketها را تنها برای دو جفت گفته شده باید انجام دهیم.

(i * j) mod 7

(2,3): (2 * 3) mod 7 = 6 mod 7 = 6

(1,5): (1 * 5) mod 7 = 5 mod 7 = 5

بنابراین (2,3) به bucket شماره 6 و (1,5) به bucket شماره 5، hash می‌شوند.

c)

دو جفت (2,3) و (1,5)

bit vector	bucket no.	count	pair	candidate set
1	6	7	(2,3)	(2,3)
1	5	8	(1,5)	(1,5)

- بخش ۲-۲:

به طور قطعی می‌توان گفت که شرط ها برای a, c, d برقرار است.

1 A, B, C -> D, E, F, G, H conf = support (A B C D E F G H) / support (A B C)

2 A, B, C, D, E -> H conf = support (A B C D E H) / support (A B C D E)

a) A, B, C, D -> E, G, H conf = support (A B C D E G H) / support (A B C D)

ساپورت صورت a از ساپورت ۱ بزرگتر یا مساوی است. ساپورت مخرج a از مخرج ۱ کمتر است (محدودیت بیشتری دارد پس ساپورتش یا کمتر است یا مساوی است) پس شرط اول برقرار است.

ساپورت صورت از ساپورت صورت ۲ کمتر یا مساوی است. همچنین ساپورت مخرج از ساپورت مخرج ۲ بیشتر یا مساوی است. پس در کل conf a از conf 2 کمتر بوده و این شرط هم برقرار است.

b) C, D, E -> A, B, G, H conf = support (A B C D E G H) / support (C D E)

ساپورت صورت b با ساپورت صورت ۱ مساوی است. ولی درمورد مقایسه‌ی مخرج این دو، نمی‌توان چیزی گفت پس لزوما شرط گفته شده برقرار نیست.

c) A, B, C, D, E -> G, H conf = support (A B C D E G H) / support (A B C D E)

ساپورت صورت c از ساپورت ۱ بزرگتر یا مساوی است. ساپورت مخرج c از مخرج ۱ کمتر است (محدودیت بیشتری دارد پس ساپورتش یا کمتر است یا مساوی است). در کل کسر c بزرگتر بوده و شرط اول برقرار است.

ساپورت صورت c از ساپورت صورت ۲ کمتر یا مساوی است. همچنین ساپورت مخرج از ساپورت مخرج ۲ بیشتر یا مساوی است. پس در کل conf c از conf 2 کمتر بوده و این شرط هم برقرار است.

d) A, B, C, E -> D, G, H conf = support (A B C D E G H) / support (A B C E)

ساپورت صورت d از ساپورت ۱ بزرگتر یا مساوی است. ساپورت مخرج d از مخرج ۱ کمتر است (محدودیت بیشتری دارد پس ساپورتش یا کمتر است یا مساوی است). در کل کسر d بزرگتر بوده و شرط اول برقرار است.

ساپورت صورت d از ساپورت صورت ۲ کمتر یا مساوی است. همچنین ساپورت مخرج از ساپورت مخرج ۲ بیشتر یا مساوی است. پس در کل conf d از conf 2 کمتر بوده و این شرط هم برقرار است.

e) A, B, C -> F conf = support (A B C F) / support (A B C)

ساپورت مخرج e با ساپورت مخرج ۱ برابر است. ساپورت صورت آن هم از ساپورت صورت ۱ بزرگتر یا مساوی است، پس شرط اول برقرار است.

ساپورت مخرج e از ساپورت مخرج ۲ بزرگتر یا مساوی است ولی درمورد صورت آنها نمی‌تواند مقایسه‌ای انجام داد و شرط ۲ لزوما برقرار نخواهد بود.

- بخش ۳-۲:

(عنوان فایل کد مربوط به این بخش q2 است.)

توضیح روند کلی کد:

پس از خواندن دیتا (توابع read_data و preprocess_data)، اولین مرحله پیدا کردن آیتم‌های پرتکرار است. برای این کار باید تعداد هر آیتم را در مجموعه داده‌ها پیدا کنیم (generate_candidate_itemsets). سپس ساپورت هر کدام را با ترشولد داده شده (۲۰۰) مقایسه می‌کنیم و آیتم‌های پرتکرار را بدست می‌آوریم (تابع generate_frequent_itemsets). در نهایت برای نمایش در خروجی، آن ها را سورت کرده و ۱۰ تای اول را پرینت می‌کنیم. برای تشکیل جفت‌های پرتکرار باید از آیتم‌های پرتکرار استفاده کنیم. پس آنها را به عنوان ورودی به تابع generate_pairs می‌دهیم. سپس بررسی می‌کنیم که آیا این جفت در هیچ تراکنشی وجود دارد یا نه. اگر وجود داشت، یکی به تعداد تکرارهای آن اضافه می‌کنیم. در نهایت با ترشولد ۱۰۰ مقایسه کرده و برای نمایش در خروجی، تنها ۱۰ تای اول را از لیست جفت‌های پرتکرار نمایش می‌دهیم (تابع generate_frequent_pairs). برای ساخت سه‌تایی های پرتکرار هم شبیه به مرحله قبل عمل کرده و از جفت‌های پرتکرار بدست آمده استفاده می‌کنیم. تابع generate_triples، تمام سه تایی‌ها را می‌سازد و در تابع generate_frequent_triples، ساپورت آن ها را محاسبه و با ترشولد ۷۵ مقایسه می‌کنیم.

برای محاسبه‌ی confidence از قواعد گفته شده، از این فرمول استفاده می‌کنیم:

(تابع confidence_based_top_rules)

confidence (A, B -> C) = support (A B C) / support (A B)

برای محاسبه‌ی interest می‌توانیم از روش‌هایی استفاده کنیم ولی در اینجا از فرمولی که در اسلایدها هم داشتیم استفاده شده. این فرمول به این صورت است:

(تابع interest_based_top_rules)

interest (A, B -> C) = conf (A, B -> C) – transactions with c / total transactions

درمورد lift در بخش مربوطه توضیح داده خواهد شد.

خروجی‌های مربوط به هر بخش:

(a

```
10 most frequent items:
{'mineral water'}: 1788
{'eggs'}: 1348
{'spaghetti'}: 1306
{'french fries'}: 1282
{'chocolate'}: 1229
{'green tea'}: 991
{'milk'}: 972
{'ground beef'}: 737
{'frozen vegetables'}: 715
{'pancakes'}: 713
```

(b

```
10 most frequent pairs of items:
{'spaghetti', 'mineral water'}: 448
{'chocolate', 'mineral water'}: 395
{'mineral water', 'eggs'}: 382
{'mineral water', 'milk'}: 360
{'ground beef', 'mineral water'}: 307
{'spaghetti', 'chocolate'}: 294
{'spaghetti', 'ground beef'}: 294
{'spaghetti', 'eggs'}: 274
{'french fries', 'eggs'}: 273
{'frozen vegetables', 'mineral water'}: 268
```

(c

```
10 most frequent pairs of items:
{'spaghetti', 'mineral water'}: 448
{'chocolate', 'mineral water'}: 395
{'mineral water', 'eggs'}: 382
{'mineral water', 'milk'}: 360
{'ground beef', 'mineral water'}: 307
{'spaghetti', 'chocolate'}: 294
{'spaghetti', 'ground beef'}: 294
{'spaghetti', 'eggs'}: 274
{'french fries', 'eggs'}: 273
{'frozen vegetables', 'mineral water'}: 268
```

(d

```
Top 5 association rules based on confidence:
('spaghetti', 'milk') -> mineral water: 0.44360902255639095
('chocolate', 'milk') -> mineral water: 0.43568464730290457
('spaghetti', 'ground beef') -> mineral water: 0.43537414965986393
('frozen vegetables', 'spaghetti') -> mineral water: 0.430622009569378
('milk', 'eggs') -> mineral water: 0.42424242424242425
```

(e

```
Top 5 association rules based on interest:
('ground beef', 'mineral water') -> spaghetti: 0.24282799209833913
('spaghetti', 'milk') -> mineral water: 0.20524080498540043
('mineral water', 'olive oil') -> spaghetti: 0.19787055767765585
('chocolate', 'milk') -> mineral water: 0.19731642973191404
('spaghetti', 'ground beef') -> mineral water: 0.1970059320888734
```

f) معیار Lift برای اندازه‌گیری قوت یا اهمیت یک قاعده انجمنی مورد استفاده قرار می‌گیرد. علاوه بر این، Lift میزان وابستگی بین دو آیتم را نسبت به وقوع یکدیگر می‌سنجد.

به طور کلی این قاعده به این صورت تعریف می‌شود:

lift (A -> b) = support (A union B) / (support (A) * support (B))

(معنی صورت، تراکنش‌هایی است که هم A و هم B را دارد.)

اگر این قاعده دارای lift یک باشد، به طور ضمنی دلالت بر این دارد که احتمال پیش‌آمد مقدم و نتیجه از یکدیگر مستقل هستند. هنگامی که دو رویداد مستقل از هم هستند، هیچ قاعده‌ای را نمی‌توان با این دو رویداد ایجاد کرد. اگر lift بزرگ‌تر از یک باشد، باعث می‌شود بدانیم درجه کدام دو پیش‌آمد وابسته به دیگری است و این قواعد را به طور بالقوه برای پیش‌بینی نتیجه در مجموعه داده آینده مورد استفاده قرار دهیم. اگر lift کمتر از یک باشد، یعنی اقلام جایگزین یکدیگر هستند. این بدین معناست که حضور یک آیتم تاثیر منفی بر حضور آیتم دیگر دارد و بالعکس.پس به طور کلی با استفاده از معیار lift، می‌توانیم قواعد انجمنی را بر اساس اهمیت و وابستگی آنها به یکدیگر ارزیابی کرده و قواعد مهم‌تر و معنادارتر را شناسایی کنیم.

(g

```
Top 5 association rules based on lift:
('spaghetti', 'mineral water') -> ground beef: 0.0003876720294630742
('spaghetti', 'mineral water') -> olive oil: 0.0003479251012145749
('milk', 'mineral water') -> frozen vegetables: 0.00032245532245532246
('ground beef', 'mineral water') -> spaghetti: 0.00031924817055833515
('frozen vegetables', 'mineral water') -> milk: 0.0003186229347091702
```

سوال ۳

(عنوان فایل کد مربوط به این بخش q3 است.)

من در کد این فایل، برای نمونه خروجی برخی توابع از فایل سعدی و رومی استفاده کردم که در کد کامنت شده‌اند. (به طور کلی، برای بررسی توابع بخش a, b, c ابتدا از فایل سعدی و رومی استفاده کردم تا دیتاست کمی سبک‌تر باشد و درستی توابع بررسی شوند.)

(a

در این بخش، ابتدا توسط تابع read_file، فایل‌های مربوط به هر شاعر را می‌خوانیم. در تابع preprocess_data، علاوه بر توکنایز کردن هر entry در فایل (که ما هر entry را معادل یک خط یا همان مصرع در نظر می‌گیریم و طبق آن جلو می‌رویم.)، با استفاده از توابع موجود در کتابخانه‌ی hazm، برخی عملیات‌های normalization, stemming, removing stopwords و چنین مواردی را انجام می‌دهیم. . این عمل باعث افزایش دقت و بهبود نتیجه می‌شود. بین مصراع هر فایل هم عملیات shuffling هم انجام می‌دهیم. همچنین دیتا باید به نسبت ۸:۲ تقسیم شده تا برای train و test مورد استفاده قرار گیرد. این فرآیند در تابع split_data انجام می‌شود. در این جا مثالی از خروجی این بخش می‌بینیم (این خروجی‌ها برای یکی از مصرع‌ها در فایل رومی و سعدی هستند.)

```
saadi:
Original line: سل الجوعان کیف الخبز وحده
Tokens: ['سل', 'الجوع', 'کیف', 'الخبز', 'وحده']

rumi:
Original line: جام از ساقی ربود و انداخت شکست
Tokens: ['جا', 'ساق', 'ربود', 'انداخ', 'شکس']
```

یکی از چالش‌هایی که در این بخش وجود داشت این بود که با این که اعمال این پیش‌پردازش‌ها به نتیجه‌ی نهایی کمک می‌کند و باعث می‌شود دقت بیشتری بگیریم، ولی زمان پردازش را زیاد می‌کند و در چنین پروژه‌ای که با کلان داده‌ها سر و کار داریم، این مساله کامل مشهود است.

(b

با پیاده‌سازی الگوریتم‌های گفته شده، در بخش‌های قبل‌تر هم آشنا شده‌ایم. چیزی که در اینجا مهم است انتخاب مینمم ساپورت‌ها و همچنین این مساله است که ست‌های پرتکرار چندتایی باشند. مسلما هر چقدر تعداد و تنوع آنها بیشتر باشد، نتیجه بهتری خواهیم گرفت ولی از آن طرف زمان و حجم حافظه بیشتری هم گرفته می‌شود. پس در اینجا یک trade off داریم و باید ببینیم چطور عمل کنیم. تابع apriori_and_support طوری نوشته شده که تمام فریکوئنت ست هایی که از توکن‌های هر خط ممکن است بدست بیاید را محاسبه می‌تواند

محاسبه کند. (مثلا اگر یک خط شامل ۵ توکن «سل، الجوع، کیف، الخبز، وحده» است، از آن می‌تواند آیتم‌های تک، جفت، سه‌تایی، چهارتایی و پنج‌تایی بسازیم و این تابع می‌تواند تمام آنها را حساب کند، خودمان تعیین می‌کنیم که تا ست‌های چندتایی را محاسبه کند. این تابع علاوه بر ست‌های پرتکرار، ساپورت یا count هر کدام را هم برمی‌گرداند. (از تابع prune_infrequent درون این تابع استفاده شده تا ست‌های غیرپرتکرار حذف شوند.)

یکی از ورودی‌های این تابع، min support است که همانطور که اشاره کردیم، خودمان باید با ایده‌هایی این مقدار را تعیین کنیم. مسلما یکی از کارها این است که مقادیری را امتحان کرده و با آن خروجی‌ها را بررسی کنیم و ببینیم چه دقتی می‌گیریم. برای این که تخمین اولیه برای این کار کمی منطقی‌تر بوده و خیلی دور از ذهن نباشد، تصمیم گرفتم برای هر دسته از ست‌ها (مثلا برای ست های تکی)، ساپورت آنها را پرینت کرده و مقادیر را ببینم. می‌توانیم از ماکس یا مین هم استفاده کنیم تا ببینیم اعداد در چه بازه‌ای قرار دارند. همچنین تعداد تکرار هر عدد ساپورت که موجود است را هم می‌توانیم پرینت و مشاهده کنیم. من از میانگین هم استفاده کردم. همه‌ی این‌ها باعث می‌شود بتوانیم در ابتدای کار تخمین منطقی‌تری برای میزان min support داشته باشیم و در ادامه با مشاهده‌ی نتایج، دقتی که می‌گیریم و همچنین با توجه به میزان زمان و حجم حافظه، تصمیمات بهتری اخذ کنیم و این مقدار را آپدیت کنیم. توابعی مانند average_support و count_word_occuarences در همین راستا تعریف شده‌اند. برای افزایش دقت، برای هر ست چندتایی، ساپورت متناسب با آن و برای هر شاعر هم متناسب با خودش این مقدار را تعریف می‌کنیم. اسکرین‌شات زیر برخی نتایج عددی بیان شده را برای فایل مربوط به سعدی نشان می‌دهد:

```
Average Support for saadi: 8.166576023618989
Number of words in saadi file: 12871
Number of words in saadi file with support more than 8: 2113
```

این موارد مربوط به تک‌آیتم‌های پرتکرار است. همانطور که دیده می‌شود تعداد کلماتی که ساپورتش از ۸ بیشتر است، حدود ۱۶ درصد از کل کلمات این فایل است. بنظر میرسد این مقدار همچنان زیاد باشد. چون این درصد تنها برای یک دسته از آیتم‌های پرتکرار یک شاعر است! می‌توانیم یک درصد (مثلا ۵ درصد از کل را برای هر دسته از آیتم‌های پرتکرار هر شاعر در نظر بگیریم و طبق آن ساپورت را تعیین کنیم. می‌توانیم تعداد تکرار هر ساپورت را در نظر بگیریم و طبق آن از ساپورت را تعیین کنیم. یعنی **مثلا** اگر برای تک‌آیتم‌های فریکوینت فایلی این مقادیر را داشته باشیم:

(۵۰:۳) یعنی ساپورت ۵۰ با تعداد تکرار ۳. (۲۰:۵)، (۲:۸) را داشته باشیم، ترشولد را مثلا حدود ۲۰ قرار دهیم.

مثالی از تک‌آیتم تا آیتم‌های چهارتایی پرتکرار: (برای فایل اشعار سعدی)

```
Frequent Itemsets (Size 1): {frozenset({'فیروزی'}), frozenset({'سر'}), frozenset({'خراب'}), frozenset({'نیافت'}), frozenset({
}), frozenset({'اب'}), frozenset({'کز'}), frozenset({'عقل'}), frozenset({'آموخ'}), frozenset({'خلق'}), frozenset({'م'}), froze
set({'عاشق'}), frozenset({'ع'}), frozenset({'دس'}), frozenset({'خو'}), frozenset({'گر'}), frozenset({'سعد'}), frozenset({'برخاس
}), frozenset({'دوس'}), frozenset({'چو'}), frozenset({'چشم'}), frozenset({'لب'}), frozenset({'صور'}), frozenset({'روز'}), f
frozenset({'س'}), frozenset({'صبر'}), frozenset({'آب'}), frozenset({'درد'}), frozenset({'مگر'}), frozenset({'عشق'}), frozense
t({''}), frozenset({'نوس'}), frozenset({'غ'}), frozenset({'با'}), frozenset({'مزار'}), frozenset({'دیده'}), frozenset({'سرو'}
```

```
Frequent Itemsets (Size 2): {frozenset({'دوس', 'س'}), frozenset({'دگر', 'نمی'}), frozenset({'نیافت\u200c'}), frozenset({'پیرمن', 'چن'}), frozenset(
خو'}), frozenset({'ب', 'گو'}), frozenset({'قدر', 'سعد'}), frozenset({'عمر', 'س'}), frozenset({'نیافت', 'نصل'}), frozenset({'دید
et({'یار', 'ابرو'}), frozenset({'لب', 'غلا'}), frozenset({'برخاس', 'گل'}), frozenset({'آیینه', 'یرود'}), frozenset({'مهرب', 'دل'})
یس', 'ا'}), frozenset({'آب', 'کوفه'}), frozenset({'نشود', 'گر'}), frozenset({'فریاد', 'دس'}), frozenset({'مانند', 'عقل'}), frozens
'بیند'}), frozenset({'خو', 'دوس'}), frozenset({'توس', 'دمن'}), frozenset({'خرسند', 'دوس'}), frozenset({'چندین', 'بند'}), frozense
rozenset({'می', 'یار', 'کند\u200c'}), frozenset({'بند', 'سیمین'}), frozenset({'بیند', 'چو'}), frozenset({'خو', 'سوال'}), frozenset({'ک
'خو', 'صحرا'}), frozenset({'ک', 'صحرا'}), frozenset({'نجاتم', 'نیا'}), frozenset({'دگر', 'دیده'}), frozenset({'سعدی', 'آیینه'}), fr
```

```
Frequent Itemsets (Size 3): {frozenset({'امروز', 'خار', 'منیل'}), frozenset({'گدا', '.', 'فقیر'}), frozenset({'حرا', 'پاسبان'
frozenset({'اذن', 'دس', 'عفو'}), frozenset({'موج\u200c', 'مرا', 'غم', 'ابی'}), frozenset({'مرد', 'خاک', 'پوسه'}), frozenset({'صبح', '
'}), frozenset({'جول', 'زلف', 'زخ'}), frozenset({'سرو', 'بلند', 'دس'}), frozenset({'مرده', 'کن', 'دد'}), frozenset({'ز', 'خصم'
اره'}), frozenset({'هزار', 'اصفهار', 'خطا'}), frozenset({'برنجد', 'وه', 'دل'}), frozenset({'چو', 'ساق', 'شراب'}), frozenset({
, 'نرسد', 'کس'}), frozenset({'دین', '.', 'فرومایه'}), frozenset({'مشیار', 'رو', 'رحم'}), frozenset({'کمر', 'چو', 'بندگ'}), fro
', 'فرمانده', 'سر'}), frozenset({'نخور', 'شکر', 'ناب'}), frozenset({'بیاراس', 'خانه', 'بازار'}), frozenset({'لگدکوب', 'س', 'پا
```

```
Frequent Itemsets (Size 4): {frozenset({'بیمو', 'جا', 'صوف', 'ساق'}), frozenset({'بلا', 'برکنده', 'منیاد', 'صبر'}), frozenset({
et({'طاق', 'آه', 'قرس', 'تمانند'}), frozenset({'اوتس', 'زیاد', 'عشق', 'مگر'}), frozenset({'آ', 'سخن', 'پدانند', 'اندر'}), froze
frozenset({'ز', 'پرواز', 'بنمودم', 'بشکست'}), frozenset({'جزع', 'گر', 'بخوانند', 'دعاص'}), frozenset({'درکش', 'د', 'درک', 'سعد'
, 'ی\u200cن'}), frozenset({'ساق', 'جا', 'صوف', 'جان'}), frozenset({'و', 'فیاس', 'آن\u200c', 'کن', 'گر\u200cکه'}), frozenset({'گر
ی', 'چوبین', 'پوس', 'می\u200cچد'}), frozenset({'آرا', 'نی', 'ببرد', 'مخو\u200cن'}), frozenset({'خون', 'فروبرده', 'خناس', '
'امشب'}), frozenset({'', 'کن', 'یاور', 'آیینه'}), frozenset({'بگف', 'خو', 'روز', 'دوش'}), frozenset({'و', 'برود', 'دل', 'پیر'
```

(c

تابع generate_association_rules برای تولید قواعد انجمنی است. در این تابع اندازه ستی که قرار است توسط آن قاعده درست شود را به عنوان ورودی می‌دهیم و به این صورت امکان این که با یک تابع قاعده‌هایی با اندازه‌های مختلف سمت چپ بسازیم. در این جا باید یک معیار انتخاب کنیم که برای ساخت قواعد از آن استفاده کنیم. با توجه به ویژگی‌هایی که معیار confidence دارد و با درنظر گرفتن هدف نهایی که در این مساله داریم، بهتر است از بین معیارهای confidence, interest, lift، معیار confidence را انتخاب کنیم. در ادامه به توضیح بعضی ویژگی‌هایی که منجر به این انتخاب شده اشاره می‌کنیم:

قدرت پیش‌بینی: اطمینان احتمال شرطی نتیجه به شرط قبلی را اندازه‌گیری می‌کند. در این مورد، مقدار اطمینان بالا نشان می‌دهد که حضور کلمات یا الگوهای خاص (شرط قبلی) در یک شعر به طور قابل اعتماد هویت شاعر (نتیجه) را پیش‌بینی می‌کند. این با هدف شناسایی شاعر پشت یک مجموعه داده از بیت‌ها همخوانی دارد.

قابلیت تفسیر: اطمینان یک اندازه مستقیم و آشکار از قدرت ارتباط بین شرط قبلی و نتیجه در یک قانون انجمن فراهم می‌کند. این به مدیران موزه و تحلیلگران اجازه می‌دهد که به راحتی قوانین را تفسیر و درک کنند، که برای کشف اثرات منحصر به فرد هر شاعر بسیار حیاتی است.

کاربردی بودن اطمینان یک معیار رایج در استخراج قوانین انجمن است و تفسیرهای خوبی دارد. این به رایانه امکان محاسبه بهینه‌ای را فراهم می‌کند و می‌تواند به آسانی در الگوریتم‌ها برای شناسایی الگوهای مهم در مجموعه داده‌های بزرگ گنجانده شود، که این امر برای وظیفه در دست بسیار عملی است.

در ادامه باید یک مینیمم کانفیدنس تعریف کنیم تا قواعد مهمتر را نگه داریم. یا می‌توانیم شبیه به سوال دوم بگوییم که مثلا X قاعده مهمتر را با توجه به معیار تعریف شده انتخاب کنیم.

تمامی این مواردی که گفته شد، مانند ترشولد یا مینیمم ساپورت که در بخش‌های قبلی توضیح داده شد، به عنوان یک تخمین اولیه مناسب است و به هرحال باید با داده‌ها امتحان شود و طبق دقتی که بدست می‌آید ممکن است مینیمم کانفیدنس یا حتی معیار انتخاب شده متفاوت شود و تغییر کند.

در نهایت توسط تابع save_rules_to_files، قواعد بدست آمده برای هر شاعر را در فایلی با نام خودش سیو می‌کنیم.

در اینجا بخش training به پایان می‌رسد.

d)

حالا نوبت کار با داده‌های تست است. این داده‌ها در ابتدای کار از ۲۰ درصد کل دیتا جدا شده بود و حالا نوبت به کارگیری آنهاست. در استفاده از قوانین برای بررسی نتیجه و بدست آوردن دقت، می‌توانیم به قاعده‌ها با سمت چپ بزرگتر وزن بیشتری دهیم و با توجه به آنها و درصدی که برای هر شاعر بدست می‌آید، نتیجه نهایی را برای هر مصرع اعلام کنیم.