دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده مدیریت، علم و فناوری

پروژه۴ درس داده کاوی

مقایسه روش‌های الگو های تکراری

نگارش
فاطمه سادات علیخانی

استاد درس
دکتر مهدی قطعی

چکیده

در این گزارش به بررسی دیتاست retail پرداخته و الگوهای تکراری و آیتم های پرتکرار را در آن با استفاده از الگوریتم‌های Apriori ، FP-growth ، Eclat پیدا و بررسی می‌کنیم هم چنین در آخر عملکرد این سه الگوریتم با هم را مقایسه می‌کنیم.

کلمات پر کاربرد : apriori, fp-growth, الگوهای تکراری ، آیتم های پرکاربرد ، eclot و ...

فهرست

مقدمه

یافتن الگوهای پرتکرار در بسیاری از مسائل کاربرد زیادی دارد، منظور از الگوی پرتکرار الگویی است که در تعداد زیادی از transaction ها دیده شود و با استفاده از آن اطلاعات و ارتباط زیادی بین آیتم‌های مختلف میتواند بدست آورد، برای این کار الگوریتم های زیادی وجود دارد در این گزارش با توضیح دادن الگوریتم هایی مانند Apriori و FP-Growth و EClat آشنا میشویم و عملکرد آنها را بررسی و با هم مقایسه خواهیم کرد همینطور برای بررسی این الگوریتم ها از دیتاست retail استفاده شده است که مربوط به یک فروشگاه زنجیره اییست که آیتم های مختلف در سبدهای خرید مختلف خریداری شده اند و با استفاده از این الگوریتم ها بررسی میکنیم چه آیتم هایی بیشتر استفاده شده و یا با هم خریده داری شده اند.

Dataset

در این گزارش از دیتاست retail استفاده شده است، که در مورد فروش محصولات مختلف از یک فروشگاه زنجیره‌ایی است هر سطر از آن نشان دهنده‌ی یک سبد خرید است که در آن یک سری عدد قرار داده شده که هر عدد نشان دهنده ی یک محصول است،

تعداد کل سطرها ۸۸۱۶۲ سطر و تعداد کل محصولات ۱۶۴۷۰ است.

نمونه‌ای از دو ردیف آن در جدول رو به رو نشان داده شده است. این جدول نشان میدهد در تراکنش شماره یک محصولات ۳۰ و ۳۱ و ۳۲ خریداری شده اند و در تراکنش شماره ۲ محصولات ۳۳ و ۳۴ و ۳۵ خریداری شده اند.

TID	items
1	30 31 32
2	33 34 35

پیش پردازش داده

با استفاده از TransactionEncoder در کتابخانه mlxtend دیتاها به صورت یک آرایه ایی از trueو false تبدیل میکنیم که هر ستون مربوط به یک محصول است و هر سطر مربوط به یک تراکنش یا سبد خرید اگر در سبد خریدی محصول i ام خریده شده باشد، در ستون i مقدار True قرار میگرد و اگر در سبد خریدی محصول i ام خریده داری نشده باشد در این ستون مقدار false قرار میگیرد. در نهایت ۵ سطر اول از این ساختار به صورت زیر است :

	0	1	10	100	1000	10000	10001	10002	10003	10004	...	9990	9991	9992	9993	9994	9995	9996	9997	9998	9999
0	True	True	True	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

5 rows × 16470 columns

شکل ۱ one hot encoded data-

به طور مثال از شکل بالا میتوانیم دریابیم که آیتم شماره‌ی 0 در سبد خرید شماره صفر وجود دارد ولی در تراکنش‌های بعدی وجود ندارد.

پیاده سازی الگوریتم‌ها

الگوریتم Apriori

توضیح الگوریتم

یکی از الگوریتم های معروف بحث کاوش الگوهای تکراری (frequent pattern mining) الگوریتمApriori است. روش کار این الگوریتم به این صورت است که که ابتدا یک میزان کمترین حد پشتیبانی مشخص کرده (minimum support) بعد از آن آیتم‌هایی که فراوانی و تعداد آن‌ها از این تعداد ساپورت بیشتر است را انتخاب می‌کنیم و بقیه را از لیست مورد بررسی حذف می‌کنیم، دوباره هر ترکیب دوتایی از آیتم های موجود در لیست مورد بررسی را بررسی میکنیم و آن هایی که ساپورت آنها از minimum support بیشتر است را انتخاب می‌کنیم و این عملیات را آنقدر ادامه میدهیم تا دیگر ترکیبی با ساپورت بیشتر از minimum support پیدا نشود.

اجرای الگوریتم

برای این کار از کتابخانه‌ی mxtend استفاده می‌کنیم. این کتابخانه یک متد apriori دارد که مقدار relative minimum support که به معنی حداقل احتمال وجود یک عنصر در دیتاست است و دیتاست به فرم one-hot encode شده میگیرد و الگوهایی که از مینیمم ساپورت مشخص شده بیشتر هستند را میدهد. در ابتدا مینیمم ساپورت را مقدار 0.1 قرار دادیم و نتیجه به صورت زیر است در جدول کشیده شده آیتم های مختلف را مشاهده میکنیم مثلا محصول شماره ۳۹ است که مقدار ساپورت آن ۰,۵۷ است و بیشترین ساپورت را دارد مشاهده میکنیم که محصول ۳۹ که خود بیشترین ساپورت را دارد در اکثر الگوهای دوتایی دیده میشود. همینطور زمان اجرا شدن این الگوریتم ۲ ثانیه و ۵۱۵ صدم ثانیه می‌باشد.

```
frequent_itemsets1 = apriori(retail_onehotdf,
                             min_support=0.1,
                             use_colnames=True)
frequent_itemsets1
```

```
Duration: 0:00:02.515455
memory (159802, 3707258)
```

	support	itemsets
0	0.172036	(32)
1	0.176902	(38)
2	0.574794	(39)
3	0.169517	(41)
4	0.477927	(48)
5	0.117341	(38, 39)
6	0.129466	(41, 39)
7	0.330551	(48, 39)
8	0.102289	(41, 48)

شکل ۲- الگوریتم apriori و min_sup=0.1

همینطور که در شکل صفحه قبل مشاهده شد با مینیمم ساپورت یک دهم فقط الگوهای با اندازه ۲ پیدا شدند برای پیدا کردن الگوهایی با طول بیشتر

میزان مینیمم ساپورت را کمتر می‌کنیم و مقدار آن را ۰,۰۵ قرار می‌دهیم و نتیجه به صورت شکل رو به رو می‌شود. در ابتدا مشاهده می‌کنیم که زمان اجرای کد به اندازه‌ی دو دهم ثانیه بیشتر شده است و هم میزان حداکثر مموری که استفاده شده بیشتر شده و میتوان مشاهده کرد که الگوهایی ۳تایی که با هم تکرار شده اند به لیست اضاف شده اند.

همینطور که در قسمت قبل گفته شد محصول شماره‌ی ۳۹ بیشتر از بقیه خریداری شده است و مینیمم ساپورت آن ۰,۵ است و مشاهده می‌کنیم که در در هر سه الگوی سه تایی این محصول تکرار شده است، این به دلیل این است که خود بیشترین تکرار را داشته به طور مثال در الگوهای ۳ تایی مشاهده میکنیم که اعداد (۴۸،۳۲،۳۹) در سبدهای خرید زیادی با هم خریداری شده‌اند.همینطور مشاهده میکنیم محصولی با ساپورت کمتر نمیتواند در الگوهای بزرگتر قرار بگیرد. مثلا محصول شماره‌ی۶۵ میزان ساپورتش به اندازه ۰,۰۵ است و آن را در الگوهای بزرگتر نمی‌توانیم مشاهده کنیم زیرا ساپورت خودش کم است و احتمال اینکه در دسته های بزرگتر قرار بگیرد بسیار کم است.

بعد از آن مینیم ساپورت ۰,۰۱را بررسی کرده و می‌بینیم که با این میزان ساپورت ۱۵۹ آیتم یا آیتم‌ست پیدا شده اند همینطور که در خط آخر آن مشاهده میشود کالاهای (۴۸،۴۱،۳۸،۳۹) که در قسمت‌های قبل به صورت تکی دیده میشدند و ساپورت بالایی داشتند در این قسمت با هم قابل مشاهده هستند و همینطور که مشاهده میشود زمان اجرای کد نسبت به حالت قبل دو ثانیه افزایش یافته است.

```
Duration: 0:00:02.758765
memory (426769, 5293664)
```

	support	itemsets
0	0.172036	(32)
1	0.176902	(38)
2	0.574794	(39)
3	0.169517	(41)
4	0.477927	(48)
5	0.050725	(65)
6	0.095903	(32, 39)
7	0.091128	(48, 32)
8	0.117341	(38, 39)
9	0.090107	(38, 48)
10	0.129466	(41, 39)
11	0.330551	(48, 39)
12	0.102289	(48, 41)
13	0.061274	(48, 32, 39)
14	0.069213	(38, 48, 39)
15	0.083551	(48, 41, 39)

شکل ۳- الگوریتم apriori و min_sup=0.05

```
Duration: 0:00:09.911106
memory (196698, 639815564)
```

	support	itemsets
0	0.012500	(1004)
1	0.025374	(101)
2	0.010004	(10515)
3	0.031692	(110)
4	0.016175	(1146)
...	...	...
154	0.013532	(48, 38, 170, 39)
155	0.014020	(48, 32, 38, 39)
156	0.018670	(48, 41, 32, 39)
157	0.012250	(48, 38, 39, 36)
158	0.022583	(48, 41, 38, 39)

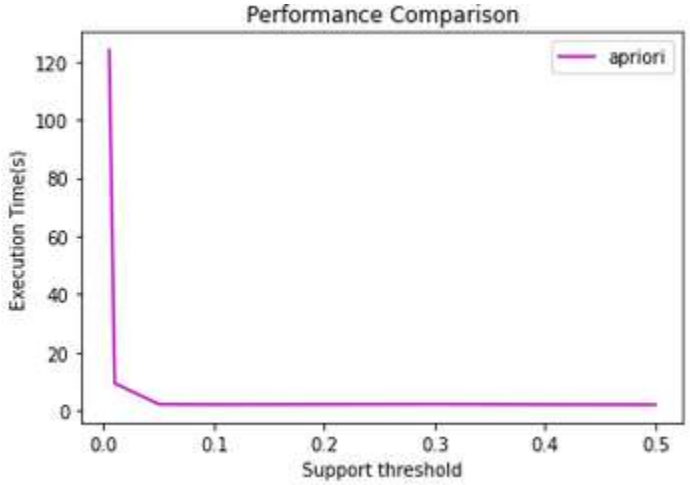
159 rows × 2 columns

شکل ٤- الگوریتم apriori و min_sup=0.01

پس به طور کلی نتیجه میگیریم که هر چه مقدار مینیمم ساپورت کمتر شود تعداد الگوهای مشاهده شده بیشتر میشود و در پی آن زمان اجرا و حافظه به کار برده شده نیز بیشتر می‌شود به این دلیل که در هر مرحله تعداد الگوها بیشتری وجود دارند که بررسی شوند.

بعد از آن مقدار مینیمم ساپورت ۰,۰۰۵ محاسبه شده و زمان اجرای آن به صورت چشم گیری افزایش پیدا کرد و برابر دو دقیقه و ۲۴ ثانیه شد.

که این نشان دهنده آن است که الگوریتم Apriori برای مینیمم ساپورت های بالا خوب عمل میکند و هر چه مقدار مینیمم ساپورت کمتر شود که در پی آن باید تعداد بیشتر الگو پیدا شود زمان اجرای الگوریتم نیز بیشتر میشود.



شکل ۵ زمان اجرایی الگوریتم apriori

Min sup	۰,۰۰۵	۰,۰۱	۰,۰۵	۰,۱	۰,۳	۰,۵
Time	۱۲۴,۱۱۸۷۴۹	۰۹,۳۳۲۱	۰۲,۷۵۸۷	۰۲,۵۱۵۴۵	۰۲,۱۵۹۰۰	۰۱,۹۷۸۴۹

جدول بالا مقدار دقیق مدت زمان اجرا شدن الگوریتم apriori با مینیمم ساپورت‌های مختلف را نشان می‌دهد و مشاهده می‌شود که با کم شدن مینیمم ساپورت مدت زمان اجرا شدن الگوریتم به صورت چشمگیری زیاد می‌شود. این به این علت است که در این الگوریتم در هر مرحله از پیدا کردن الگوهای تکراری اطلاعات بیشتری باید بررسی شود و همینطور در هر مرحله از اطلاعات قبلی خود استفاده‌ای ندارد.

الگوریتم FP-growth

توضیح الگوریتم

در این الگوریتم ابتدا یک تعداد تکرار هر آیتم را بدست آورده و یک frequent item list درست میکنیم و آنها را به ترتیب ساپورت مرتب کرده و در هر transaction ترتیب آیتم ها را به ترتیب آیتم هایی که در frequent item list آمده مرتب می‌کنیم و همراه با آن مانند الگوریتم apriori آیتم‌هایی ساپورت آن ها از یک مقدار مشخص کمتر بود را حذف میکنیم به دلیل اینکه این آیتم‌ها نمیتواند الگوهای پرتکرار بزرگتر بسازند. اگر لیست آیتم‌های ما به صورت زیر باشد :

TID	items
1	30 31 32 34
2	30 33 34
3	30 31 34

frequent item list ما به صورت رو به رو است :

TID	frequency
30	3
31	2
32	1
33	1
34	3

لیست آیتم ها با حذف ایتم های کم تکرار به صورت زیر خواهد شد :

TID	items
1	30 34 31
2	30 34
3	30 34 31

ساخت درخت FP

بعد از آن درخت fp را می‌سازیم، به این صورت که در ابتدا ریشه‌ی درخت را تهی قرار می‌دهیم و بعد از آن با آیتم هایی که در transaction ها به صورت مرتب شده قرار دارند اولین آیتم را فرزند ریشه قرار داده و به همین صورت به ترتیبی که آیتم ها را در یک transaction سورت شده می‌بینیم آنها را در درخت به هم متصل کرده و در مقابل آن تعداد تکرار را مینوسیم که اگر همین مسیر دوباره تکرار شد به تعداد تکرار آن یک عدد اضافه می‌کنیم پس در نهایت شماره‌ی آخرین آیتمی که در یک پترن قرار دارد نشان دهنده ساپورت آن پترن است.

این الگوریتم به این علت که در هر مرحله از اطلاعات گذشته استفاده می‌کند نسبت به الگوریتم Apriori میتواند بهتر عمل کند. ضمن اینکه با یک بار ساختن درخت میتوان در دفعات بعدی از آن استفاده کرد و همینطور با داشتن درخت fp میشود به جای اینکه به دنبال الگوهای تکراری بگردیم الگوهایی را پیدا کنیم که یک آیتم خاص در آن‌ها وجود دارند.

اجرای الگوریتم

برای این کار از کتابخانه‌ی mxtend استفاده شده است. این کتابخانه یک متد fpgrowth دارد که مقدار relative minimum support که به معنی حداقل احتمال وجود یک عنصر در دیتاست است را ورودی میگیرد به همراه دیتاست به فرم one-hot encode شده و الگوهایی که از مینیمم ساپورت مشخص شده بیشتر هستند را میدهد.

در ابتدا min_sup = 0.1 را بررسی می‌کنیم و نتیجه آن به صورت شکل زیر است:

```
frequent_itemsetsg1 = fpgrowth(retail_onehotdf,
                               min_support=0.1,
                               use_colnames = True)
end_time = datetime.now()
frequent_itemsetsg1
```

```
Duration: 0:00:06.065012
memory (168996, 1457598)
```

	support	itemsets
0	0.172036	(32)
1	0.574794	(39)
2	0.176902	(38)
3	0.169517	(41)
4	0.477927	(48)
5	0.117341	(38, 39)
6	0.129466	(41, 39)
7	0.102289	(41, 48)
8	0.330551	(48, 39)

شکل ۶- fp growth و min sup = 0.1

همانطور که مشاهده میشود و با شکل شماره ۲ (*الگوریتم apriori و min_sup=0.1*) میتوان مقایسه کرد که نتایج دو الگوریتم یکسان است و ساپورت های یکسانی نشان داده میشود و در نتیجه آیتم‌های یکسانی هم در جدول مشاهده می‌شود و مقدار حافظه استفاده شده در این الگوریتم کمتر از الگوریتم Apriori است و البته زمان اجرای این الگوریتم با در این مرحله کمی بیشتر از الگوریتم Apriori شده است. و همانطور که در قسمت قبل گفته شد آیتم شماره۳۹ که بیشترین مقدار ساپورت را دارد در اکثر الگوهای دوتایی دیده می‌شود و آیتمی که کمترین ساپورت را دارد (آیتم ۳۲) در الگوهای دوتایی دیده نمی‌شود زیرا تکرار آن به صورت تنها کم بوده احتمال وجود آن در ترکیب‌های بزرگتر کمتر می‌شود.

در شکل ۷ نتیجه الگوریتم fp-growth را با مینیمم ساپورت 0.05 مشاهده میکنیم که تعداد پترن ها نسبت به شکل قبل بیشتر شده به این دلیل که مینیمم ساپورت را کمتر کردیم و آیتم هایی با ساپورت کمتر را هم مشاهده می‌کنیم و نتیجه آن دقیقا مانند شکل ۳ در قسمت apriori است که الگوهایی با سه آیتم هم قابل مشاهده هستند و آیتمی مانند آیتم ۶۵ که خود ساپورت آن کم است در الگوهای تکراری دیگر مشاهده نمیشود اما آیتم ۳۹ که بیشترین ساپورت را دارد در اکثر الگوهای چندتایی دیده میشود. همینطور آیتم‌های ۳۹ و ۳۸ و ۴۸ با هم خریداری میشوند و یا همینطور آیتم های ۴۱و ۳۹ و ۳۸ ، ۴۸ و۳۲ و ۳۹ الگوهای پرتکراری هستند. هر یک از این آیتم ها به تنهایی ساپورت بالاتر از یک دهم را دارند.

```
Duration: 0:00:05.868634
memory (179200, 1457414)
```

	support	itemsets
0	0.172036	(32)
1	0.574794	(39)
2	0.176902	(38)
3	0.169517	(41)
4	0.477927	(48)
5	0.050725	(65)
6	0.095903	(32, 39)
7	0.091128	(48, 32)
8	0.061274	(48, 32, 39)
9	0.117341	(38, 39)
10	0.090107	(48, 38)
11	0.069213	(48, 38, 39)
12	0.129466	(41, 39)
13	0.102289	(41, 48)
14	0.083551	(41, 39, 48)
15	0.330551	(48, 39)

شکل ۷- fp grwoth و min sup = 0.05

اگر میزان مینیمم ساپورت را تا مقدار 0.01 کاهش دهیم نتیجه‌ای به صورت شکل مقابل میبینیم مانند شکل ۴ که در قسمت Apriori بود ۱۵۹ الگو با مینیمم ساپورت بیشتر از یک صدم میبینیم اما تفاوت در زمان اجرای این دو الگوریتم دیده میشود که الگوریتم Apriori در ۱۰ ثانیه اجرا شده (در حالی که این الگوریتم با مینیم ساپورت بیشتر در ۲ ثانیه اجرا میشده) اما الگوریتم fp-growth در ۶ ثانیه اجرا شده است.

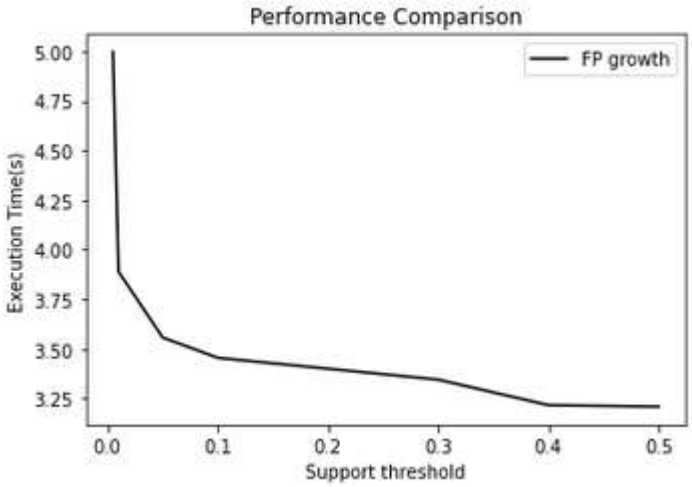
```
Duration: 0:00:06.901396
memory (13502476, 14894594)
```

	support	itemsets
0	0.015562	(9)
1	0.011399	(19)
2	0.172036	(32)
3	0.010435	(31)
4	0.574794	(39)
...	...	...
154	0.013112	(39, 1327)
155	0.010980	(48, 1327)
156	0.010832	(2238, 48)
157	0.014598	(2238, 39)
158	0.010640	(12925, 39)

159 rows × 2 columns

شکل ۸- fpgrowth و min sup = 0.01

پس به طور کلی نتیجه می‌گیریم که در الگوریتم fp-gwoth به اندازه الگوریتم apriori تفاوت در زمان اجزا وجود ندارد ینی با کمتر شدن مینیمم ساپورت و اضاف شدن الگوهای تکراری زمان اجرای الگوریتم fp-growth به طور چشمگیری تغیر نمیکند(فقط یک ثانیه) برعکس الگوریتم apriori که با کمتر شدن مینیمم ساپورت زمان اجرای آن به طور چشم گیری افزایش پیدا میکرد.



Min sup	۰,۰۰۵	۰,۰۱	۰,۰۵	۰,۱	۰,۳	۰,۵
Time	۰۴,۹۹۸۰۱	۰۳,۸۸۹۵	۰۳,۵۵۷۷۸	۰۳,۴۵۴۸	۳,۳۴۵	۰۳,۲۱۶۱۳

جدول بالا مقدار دقیق مدت زمان اجرا شدن الگوریتم fp-growth با مینیمم ساپورت‌های مختلف را نشان می‌دهد و مشاهده می‌شود که با کم شدن مینیمم ساپورت مدت زمان اجرا شدن فقط کمی افزایش پیدا میکند و مانند الگوریتم Apriori تفاوت چشمگیر نداریم، استفاده از درخت و عمقی پیش رفتن در آن باعث این اتفاق شده است.

الگوریتم FP-max

توضیح الگوریتم

همانطور که در دو قسمت قبل مشاهده کردیم در هر مرحله از اینکه بخواهیم آیتم های پرتکرار را ببینیم بعضی آیتم ها هم به صورت تکی نمایش داده میشدند هم به صورت اجتماع آنها با یک آیتم دیگر، به طور مثال با وجود اینکه الگوی ۴۸، ۳۹ و ۳۲ را در الگوهای پرتکرار خود می‌دیدیم هر کدام از آنها را به صورت تنها یا ترکیب دوتایی آنها را در مجموعه الگوهای پرتکرار مشاهده میکردیم در صورتی که علاوه بر اینکه ما میدانیم اگر مجموعه‌ی (۳۲، ۳۹ ، ۴۸) پرتکرار باشند پس هر کدام از آنها به تنهایی ساپورت بالایی دارند، ما نیاز داریم که الگوهای با تعداد عضو بیشتر پیدا کنیم.

الگوریتم FP-max در واقع الگوریتم توسعه یافته‌ایی برای الگوریتم fp-growth است که در آن اگر الگویی در یک مجموعه الگوی بزرگتر قرار بگیرد آن الگو به تنهایی را نشان نمیدهد و پیاده سازی آن مانند الگوریتم fp-growth است.

اجرای الگوریتم

```
maxfp1 = fpmax(retail_onehotdf,
               min_support=0.1,
               use_colnames=True)
maxfp1
```

```
Duration: 0:00:04.485791
```

	support	itemsets
0	0.102289	(41, 48)
1	0.129466	(41, 39)
2	0.172036	(32)
3	0.117341	(38, 39)
4	0.330551	(48, 39)

شکل ۹- الگوریتم fpmax و min_sup= 0.1

برای پیاده سازی این الگوریتم نیز از کتابخانه mltend و متد fpmax استفاده کردیم.همانطور که در شکل شماره ۹ قابل مشاهده است بزرگترین الگویی که هر آیتم در آن قرار دارد را نشان داده شده است مثلا آیتم ۴۱ که در دو الگوی بزرگتر و به طول ۲ قابل مشاهده است به صورت تکی به کار نرفته فقط تنها آیتمی که به صورت تکی دیده میشود کالای شماره ۳۲ است و آن به این دلیل است که در مجموعه‌ی بزرگتری قرار ندارد.

و به راحتی میتوان مشاهده کرد که کالاهای ۳۸ و ۳۹ ، ۴۱ و ۳۹ و .. با هم خریداری شده اند.

نتیجه الگوریتم fp-max با مینیمم ساپورت 0.01 به صورت شکل مقابل است ، در شکل ۴ مربوط به اجرای الگوریتم fp-growth با مینیم ساپورت 0.01 مشاهده کردیم که تعداد ۱۵۹ الگوی تکراری پیدا شده بود و مسلما یک سری از آنها تکراری بود و بررسی کردن آنها کاری تکراری بوده،در حالی که در این شکل فقط ۷۸ الگو دیده میشود، که محصولات (۴۸،۴۱،۳۲،۳۹) و (۴۸،۴۱،۳۸،۳۹) با هم خریداری شده‌اند و میزان ساپورت آنها بیشتر از یک صدم است.

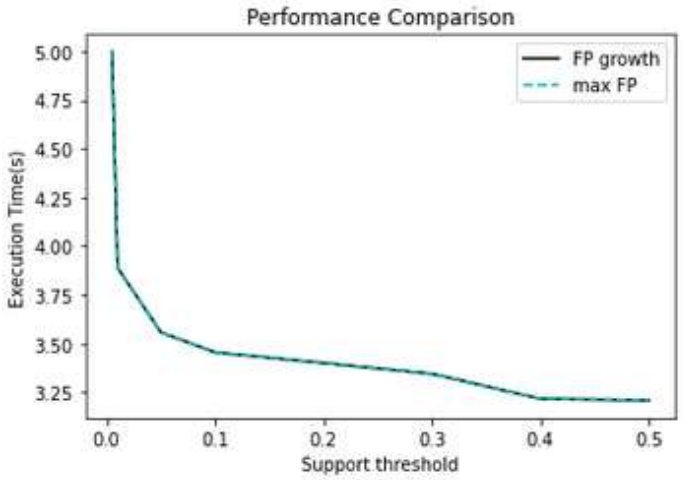
```
Duration: 0:00:04.781850
```

	support	itemsets
0	0.010004	(10515)
1	0.010152	(264)
2	0.010254	(2958)
3	0.010333	(45)
4	0.010333	(242)
...	...	...
73	0.011286	(41, 65)
74	0.020383	(48, 65, 39)
75	0.018670	(48, 41, 32, 39)
76	0.022583	(48, 41, 38, 39)
77	0.014020	(48, 32, 38, 39)

78 rows × 2 columns

شکل ۱۰- الگوریتم fpmax با min_sup=0.01

نمودار زیر زمان اجرای الگوریتم fp-max و fp-growth را با هم نشان میدهد و مشاهده میکنیم که زمان اجرایی این دو الگوریتم به بسیار به هم نزدیک است و هر دو از لحاظ زمانی شبیه هم هستند و بهتر از apriori عمل میکنند.



شکل ۱۱_ نمودار زمان اجرایی الگوریتم fpmax و fp growth

الگوریتم ECLAT

توضیح الگوریتم

این الگوریتم کمی متفاوت از الگوریتم دیگر است و به جای اینکه هر transation در سطح قرار بگیرد هر آیتم در یک سطر قرار می‌گیرد و در مقابل آنها اینکه این آیتم‌ها در چه transation هایی استفاده شده‌اند قرار میگیرد مانند جدول زیر :

item	transaction
30	T1, T2 , T3
31	T3
32	T1,T2

پس در ابتدا ساختار دیتا ست را تغیر می‌دهیم و بعد از آن مانند الگوریتم Apriori آیتم‌هایی که min support کمی دارند حذف میکنیم (در واقع آنهایی که در transation های کمتری استفاده شده اند) و بعد از آن بین transaction list آیتم های باقی مانده عملیات اشتراک گیری انجام میدهیم و الگوهای جدید را بدست آورده و دوباره شرط مینیمم ساپورت را روی آنها اجرا می‌کنیم.

اجرای الگوریتم

دراین پروژه کمی الگوریتم ECLAT را تغیر داده و سپس آن را پیاده سازی کردم.

همانطور که در بخش‌های قبل مشاهده شد که در الگوهای ۳ تایی یا ۴ تایی بیشتر آیتم‌هایی دیده شده اند که مینیمم ساپورت آنها نسبت به بقیه آیتم ها بیشتر است پس در ابتدا تک آیتم هایی را پیدا کرده که مینیمم ساپورت آنها بیشتر از ۰,۱ است و بعد از آن الگوهایی با ساپورت کمتر را پیدا میکنیم.

این قسمت را با استفاده از دیتا ست one hot encode شده که در قسمت قبل بدست آمده پیاده سازی کردم ساختار one hot encode به این صورت بود که در ستون ها آیتم ها قرار داشتند و در ردیف ها با true و false مشخص میشد که آیتم در یک transaction وجود

داشته اند یا نه. پس اطلاعات هر ستون به صورت true و false نشان دهنده ی این است که در هر آیتم در چه transaction هایی استفاده شده است.

برای بدست آوردن تعداد true ها در یک ستون از تابع Sum استفاده کرده که در حقیقت این مقدار نشان دهنده این است که این آیتم در چند transaction دیده میشود و همان مقدار support یک آیتم خاص هست و بعد از آن support relative را میتوان بدست آورد تا آیتم ست هایی که support آنها کمتر از 0.1 را بدست بیاوریم. و قطعه کد آن به صورت زیر

```
frequent_list = []
min_sup = .1
for i in range(0,retail_onehotdf.shape[1] ):
    support = sum(retail_onehotdf[str(i)])
    relative_sup = support = support/retail_onehotdf.shape[0]
    if( relative_sup > min_sup ) :
        frequent_list.append(i)
frequent_list
```

شکل ۱۲پیدا کردن item هایی با مینیمم ساپورت مشخص

و نتیجه کد به همراه ساپورت هر آیتم به شکل روبه رو است :

```
32 support :  0.1720355708808784
38 support :  0.176901612939815 35
39 support :  0.5747941289898142
41 support :  0.16951747918604387
48 support :  0.47792699802636057
Duration: 0:00:04.119283
```

و نتیجه کاملا همانند الگوریتم های قبل است.

این الگوریتم در زمان یک دقیقه و ۵۰ ثانیه اجرا شد.و دلیل این زمان این است که تعداد آیتم ها در دیتاست زیاد بوده چک کردن همه آنها باعث بیشتر شدن زمان شده است.

بعد از آن میتوانیم الگوهایی با چند آیتم را با ساپورت ۰,۱ و یا کمتر با الگوریتم eclat پیدا کنیم. که در این جا ساپورت ۰,۰۵ را بررسی کردیم، زیرا همانطور که در قسمت های قبل دیدیم میشود بیشتر آیتم هایی با ساپورت بیشتر در الگوهای پرتکرار آیتم ست ها قرار میگرفتند. این کار اگرچه کمی دقت ما را پایین می‌آورد اما عملیات و محاسبات کمتری دارد.

اجرای این الگوریتم به صورت است که هر دو ستونی که نشان دهنده‌ی آیتم های پرتکرار هستند را از one hot encode با هم and کرد و نتیجه آن در ردیف هایی که مقدار true داریم تراکنش هایی هستند که در هردو مشترک هستند و relative support آن را مقایسه کرده و اگر از یک مینمم ساپورت بیشتر بود آن را جزو دسته‌ی پرتکرار ذخیره میکنیم.

```
intersect = retail_onehotdf[str(new_item[0])]
for  i in range(1, len(item))  :
    intersect = intersect & retail_onehotdf[str(new_item[i])]
support = sum(intersect)/retail_onehotdf.shape[0]
if( support > min_support) :
    dic2[new_ite,] =[i for i, x in enumerate(check_list) if x]
    print(new_item , "support : ", support)
```

شکل ۱۳- اجرای الگوریتم eclat

و نتیجه آن به مینیمم ساپورت پنج صدم به صورت زیر است :

```
min support == 0.05
(32, 39) support :  0.09590299675597196
(32, 48) support :  0.0911276967400921
(38, 39) support :  0.1173408044282117
(38, 48) support :  0.09010684875569973
(39, 41) support :  0.12946620993171662
(39, 48) support :  0.33055057734624893
(41, 48) support :  0.10228896803611533
(32, 39, 48) support :  0.06127356457430639
(38, 39, 48) support :  0.06921349334180259
(39, 41, 48) support :  0.0835507361448243
Duration: 0:00:00.764954
```

شکل ۱۴

و نتیجه آن مانند قسمت های قبل شده است و در حالی که مدت زمان اجرا شدن آن ۰,۷ ثانیه است.

نتیجه الگوریتم با ساپورت یک دهم :

```
min support == 0.1
(38, 39) support :  0.1173408044282117
(39, 41) support :  0.12946620993171662
(39, 48) support :  0.33055057734624893
(41, 48) support :  0.10228896803611533
Duration: 0:00:00.419873
```

شکل ۱۵

```
min support == 0.005
(32, 38) support :  0.032134025997595336
(32, 39) support :  0.09590299675597196
(32, 41) support :  0.036251446201311224
(32, 48) support :  0.0911276967400921
(38, 39) support :  0.1173408044282117
(38, 41) support :  0.04420271772418956
(38, 48) support :  0.09010684875569973
(39, 41) support :  0.12946620993171662
(39, 48) support :  0.33055057734624893
(41, 48) support :  0.10228896803611533
(32, 38, 39) support :  0.02087066990313287
(32, 38, 41) support :  0.00913091808262063
(32, 38, 48) support :  0.018670175358998207
(32, 38, 39, 41) support :  0.007055193847689481
(32, 38, 39, 48) support :  0.014019645652321862
(32, 38, 41, 48) support :  0.006125087906354212
(32, 39, 41) support :  0.026757559946462194
(32, 39, 48) support :  0.06127356457430639
(32, 39, 41, 48) support :  0.018670175358998207
(32, 41, 48) support :  0.023400104353349514
(38, 39, 41) support :  0.034606746670901294
(38, 39, 48) support :  0.06921349334180259
(38, 39, 41, 48) support :  0.02258342596583562
(38, 41, 48) support :  0.026927701277194255
(39, 41, 48) support :  0.0835507361448243
(32, 38, 39, 41, 48) support :  0.0050815544111975685
Duration: 0:00:00.696186
```

شکل ۱۶

الگوریتم eclat با آیتم هایی که مینیمم ساپورت آنها یک دهم است و منیمم ساپورت الگوهای چند تایی آن 0.005 است زمان اجرایی آن شش دهم ثانیه است زمان اجرایی آن البته بدون در نظر گرفتن پیدا کردن آیتم هایی با ساپورت یک دهم بسیار پایین است.

به این دلیل که در واقع یک نوع فیلتر اجرا کردیم که ابتدا آیتم هایی با ساپورت بیشتر را فقط انتخاب کنیم و بعد ترکیب آنها را پیدا کنیم.

و فقط ترکیباتی از محصولات , 38 , 32 48 , 41 , 38 را میبینیم و ترکیب همه ی این ۵ محصول با هم ساپورت ۵ صدم را داراست.

Association Rules

support

Support که در هر قسمت می‌دیدیم یک معیار سنجش است که این دو کالا چقدر با هم خریدی داری شده‌اند مثلا ساپورت کالای ۳ و ۳۲ برابر 0.0959 بود به این معنی مه در ۹ صدم درصد مواقع با هم دیده شده اند و همراه هم رخ داده اند.

Support (i39 -> i32) = support(item39 ∪ item32)

Confidence

اما نمیتوان با دقت مناسبی گفت که زمان‌هایی که کالای ۳۹ خریداری شده کالای ۳۲ هم خریداری شده است و از معیار confidence استفاده می‌کنیم به معنی این که اطمینان داریم که در ۵۰ درصد مواقعی که کالای ۳۹ خریداری شده است کالای ۳۲ نیز خریداری میشود و روش محاسبه‌ی آن به صورت زیر است :

Confidence(i39 -> i32) = Support(i39 ∪ i32)/support(i39)

Lift

اگر این پارامتر برابر یک باشد به این معنی است که این دو از هم مجزا هستند و اگر مقادیری بیشتر از یک داشته باشند به این معنی است که میتوانند این دو پارامتر الگوهای مفیدی باشند و به صورت زیر محاسبه میشود :

lift(i39 -> i32) = Support(i39 ∪ i32)/support(i32)

جدول اطلاعات مختلف برای این دیتا ست به صورت زیر است که برای الگوهایی با ساپورت 0.05 با الگوریتم apriori بدست آمده اند است اما چون نتیجه همه الگوریتم ها یکسان بود به صورت یکجا آورده شده اند :

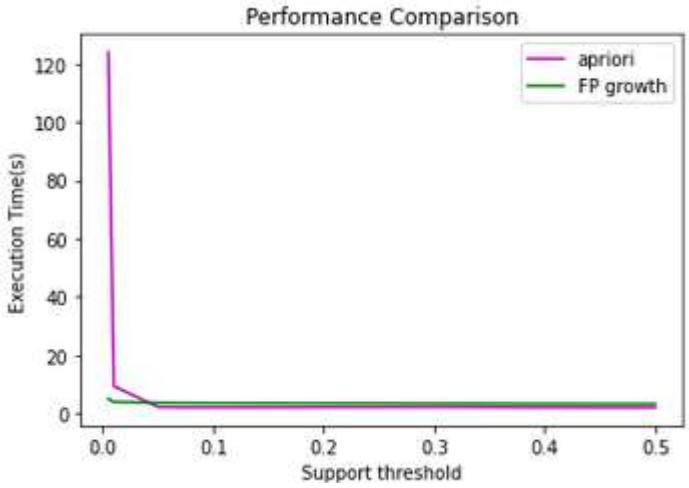
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(32)	(39)	0.172036	0.574794	0.095903	0.557460	0.969843	-0.002982	0.960831
1	(32)	(48)	0.172036	0.477927	0.091128	0.529703	1.108334	0.008907	1.110091
2	(38)	(39)	0.176902	0.574794	0.117341	0.663311	1.153998	0.015659	1.262904
3	(38)	(48)	0.176902	0.477927	0.090107	0.509361	1.065772	0.005561	1.064068
4	(41)	(39)	0.169517	0.574794	0.129466	0.763734	1.328708	0.032029	1.799689
5	(48)	(39)	0.477927	0.574794	0.330551	0.691634	1.203273	0.055841	1.378900
6	(39)	(48)	0.574794	0.477927	0.330551	0.575076	1.203273	0.055841	1.228628
7	(41)	(48)	0.169517	0.477927	0.102289	0.603413	1.262562	0.021272	1.316413
8	(32, 48)	(39)	0.091128	0.574794	0.061274	0.672392	1.169797	0.008894	1.297912
9	(32, 39)	(48)	0.095903	0.477927	0.061274	0.638912	1.336840	0.015439	1.445833
10	(38, 48)	(39)	0.090107	0.574794	0.069213	0.768127	1.336351	0.017421	1.833787
11	(38, 39)	(48)	0.117341	0.477927	0.069213	0.589850	1.234185	0.013133	1.272884
12	(41, 48)	(39)	0.102289	0.574794	0.083551	0.816811	1.421049	0.024756	2.321130
13	(41, 39)	(48)	0.129466	0.477927	0.083551	0.645348	1.350306	0.021675	1.472070

شکل ۱۷- association rules

بیشترین ساپورت مربوط به خط ۶م است که ساپورت Support (i39 -> i48) را حساب کرده است و مقدار ساپورت برابر 0.33 است و مقدار اطمینان این قانون برابر 0.6 است پس تقریبا میشود نتیجه گرفت که اگر آیتم ۳۹ خریداری شود با اطمینان ۶۰ درصد آیتم ۴۸ نیز خریداری میشود و همینطور مقدار lift آن دو نیز بیشتر یک است که نشان دهنده ارتباط خوب بین این دو آیتم است. همینطور سطر ۱۳ هم میزان اطمینان زیادی دارد که اگر آیتم های ۳۹ و ۴۱ با هم خریداری شوند آیتم ۴۸ هم با احتمال زیادی خریداری میشود.سطرهای دیگر را میشود به همین ترتیب بررسی کرد.

مقایسه الگوریتم ها از لحاظ زمانی

همانطور که مشاهده شد بین الگوریتم های apriori و fp-growth برای زمان اجرای آنها در ابتدا هر دو نزدیک به هم بودند اما زمانی که مینیمم ساپورت زیاد میشد زمان اجرای الگوریتم apriori به صورت چشم گیری زیاد میشد و این موضوع به این دلیل است که در الگوریتم apriori محاسبات چند بار انجام میشود و ذخیره نمیشود تا در مراحل بعد از آن استفاده کرد شکل زیر نمودار زمان اجرای این دو الگوریتم را با مینیمم ساپورت های مختلف نشان میدهد :



شکل ۱۸مقایسه الگوریتم apriori و fp-growth

Min sup	۰,۰۰۵	۰,۰۱	۰,۰۵	۰,۱	۰,۳	۰,۵
Time(fp_growth)	۰۴,۹۹۸۰۱	۰۳,۸۸۹۵	۰۳,۵۵۷۷۸	۰۳,۴۵۴۸	۳,۳۴۵	۰۳,۲۱۶۱۳
Time(apriori)	۱۲۴,۱۱۸۷۴۹	۰۹,۳۳۲۱	۰۲,۷۵۸۷	۰۲,۵۱۵۴۵	۰۲,۱۵۹۰۰	۰۱,۹۷۸۴۹

همینطور برای الگوریتم ***eclat*** به علت محاسبات اولیه و به این دلیل که کمی حجم دیتاست زیاد بود در ابتدا مقداری زمان اجرای آن بیشتر شد اما اگر از روش یک بار محاسبه آیتم های پرتکرار و سپس پیدا کردن الگوهایی که این آیتم‌ها در آن ها قرار دارند استفاده کنیم برای پیدا کردن الگوهای پرتکرار اجرا در دفعات دوم به بعد زمان کمتری میبرد.

نتیجه گیری

در تمامی قسمت‌ها مشاهده کردیم که آیتم‌هایی که ساپورت آنها در ابتدا بیشتر است در نهایت در بیشتر الگوهای چندتایی دیده میشوند نه آیتم هایی که ساپورت کمتری دارند به علت اینکه آیتم‌هایی با ساپورت کمتر فراوانی آنها به صورت تکی کمتر است و مسلما در الگوهای با طول بیشتر هم کمتر دیده میشوند.

همینطور با اجرای الگوریتم ها مشاهده کردیم که الگوریتم *apriori* زمانی که مینیمم ساپورت زیاد میشود در زمان مناسبی اجرا نمیشود و استفاده از الگوریتم *fp*-growth معقول تر است چون در زمان کمتری محاسبه میشود و الگوریتم eclat اگر حجم دیتاست زیاد باشد مدت زمان بیشتر نسبت به دو الگوریتم دیگر زمان اجرای آن طول می‌کشد و مناسب دیتاست با حجم بالا نیست.

منابع

(Association Rules Generation from Frequent Itemsets, 2014)

(twards datascience, n.d.)

(Frequent Itemsets via the FP-Growth Algorithm, n.d.)

(Frequent Itemsets via Apriori Algorithm, n.d.)

)Frequent Itemset Mining Dataset Repository, 2014)