دانشکده مهندسی کامپیوتر

پایان‌نامه دوره کارشناسی

مهندسی کامپیوتر – نرم‌افزار

عنوان پروژه:

طراحی و پیاده‌سازی سیستمی جهت کاوش سودمندترین K مجموعه‌آیتم

دانشجویان:

احمدرضا رستمانی- فاطمه شیری

استاد راهنما:

دکتر نگین دانشپور

بهمن ۱۴۰۰

بسم الله الرحمن الرحيم

دانشگاه تربیت دبیر شهید رجائی

تاییدیه اتمام پروژه

گواهی می‌شود که خانم/آقای ..... با شماره دانشجویی ..... دانشجوی رشته مهندسی کامپیوتر گرایش ..... دانشکده مهندسی کامپیوتر پایان نامه مقطع کارشناسی که دارای ..... واحد بوده است را با عنوان ..... تحت نظارت استاد ..... با نمره‌ی ..... در تاریخ ..... به اتمام رسانده‌اند. نسخه نهایی پایان‌نامه و فایل آن (به‌صورت DOCX و PDF) مطابق با ساختار کلی و دستورالعمل مصوب دانشکده تهیه و تحویل آموزش دانشکده شده است .

ردیف	عنوان	نام و نام خانودگی	امضا
۱	استاد راهنمای پروژه		
۲	مسئول پروژه‌ها گروه		
۳	مدیرگروه		
۴	رئیس یا معاون آموزشی و پژوهشی دانشکده		

تشکر و قدردانی

بدین وسیله از تمام عزیزانی که در مسیر به ثمر رساندن پروژه حاضر و نگارش این نوشتار همراهمان بوده‌اند تشکر کرده و دستشان را به مهر می‌فشاریم. علی الخصوص خانم دکتر دانشپور که به عنوان استاد راهنما در مراحل مختلف این پروژه همواره با سعه صدر و گشاده‌رویی در کنارمان بودند، آقای مهندس هاشمی بزرگوار و آقای مهندس مالتی عزیز، که بی‌شک بدون زحمات بی‌دریغ و راهنمایی‌های این عزیزان انجام این پروژه و به ثمر رساندن آن ممکن نبود.

چکیده

امروزه با وجود حجم بالای داده، یافتن رابطه‌ای با معنی میان داده‌ها و استخراج دانش مفید از آن‌ها امری ضروری و کارآمد برای تصمیم‌گیری در کسب وکارها به شمار می‌آید. در اغلب روش‌هایی که در گذشته به استخراج این روابط بامعنی در داده‌های مفید به‌ویژه در موضوع تحلیل سبدخرید مشتریان پرداخته‌اند، به سود آیتم‌ها توجهی نشده است. حال آن که داشتن اطلاعات درباره داده‌های سودمند می‌تواند در تصمیم‌گیری‌ها دید بهتری به صاحبان کسب وکارها بدهد. در پژوهش حاضر، ضمن تعریف برخی مفاهیم مورد نیاز و مرتبط با مسئله کاوش سودمندترین K مجموعه آیتم، به بررسی و معرفی روش‌هایی پرداخته شده است که با توجه به میزان سودمندی هر آیتم، مجموعه آیتم‌های دارای سود بیشتر نسبت به یک آستانه سودمندی از پیش تعیین شده را کشف می‌کنند.

از میان روش‌های معرفی شده، الگوریتم TKO، به‌عنوان سیستم مطرح شده برای نیل به هدف فوق پیاده‌سازی شده است. در این روش که از الگوریتم‌های یک مرحله‌ای به شمار می‌رود، K به عنوان پارامتر ورودی از کاربر دریافت شده و بر اساس آن سودمندترین K مجموعه آیتم ممکن از مجموعه داده موجود استخراج شده و به عنوان خروجی نمایش داده می‌شود. از مزایای این روش می‌توان به استفاده کمتر از منابع در زمان اجرا و عملکرد مناسب در مقایسه با سایر روش‌ها اشاره کرد.

کلیدواژه‌ها

داده‌کاوی، مجموعه آیتم‌های سودمند، مجموعه آیتم‌های پرتکرار، آستانه سودمندی

فهرست مطالب

فهرست علائم اختصاری و نشانه ها

عنوان کامل	علامت اختصاری
High Utility Itemset	HUI
High Utility Itemset Mining	HUIM
Top-k High Utility Itemset Mining in One Phase	TKO
Top-k High Utility Itemset Mining	TKU

فهرست جداول

فهرست شکل‌ها

فصل اول: مقدمه

با مورد استفاده قرار گرفتن رایانه‌ها در تحلیل و ذخیره‌سازی داده‌ها و نیز پیشرفت روزافزون فناوری اطلاعات، حجم اطلاعات ذخیره شده در پایگاه‌های داده، رشد بی‌سابقه‌ای را تجربه کرده است. وجود منابع اطلاعاتی نظیر شبکه جهانی وب، سیستم‌های یکپارچه اطلاعاتی، سیستم‌های یکپارچه بانکی و تجارت الکترونیک منجر به افزایش حجم داده در پایگاه‌های داده و ایجاد انبارهای عظیمی از داده‌ها شده است. امروزه نیاز به طراحی سیستم‌هایی که با تأکید بر حداقل مداخله انسانی، قادر به استخراج سریع اطلاعات مورد علاقه کاربران باشد و نیز روی آوردن به روش‌های تحلیل متناسب با حجم داده‌های حجیم به خوبی احساس شده و از این رو استخراج سریع دانشی دقیق و قابل اطمینان از داده‌های حجیم موجود در پایگاه‌های داده، بیش از پیش مورد توجه قرار گرفته است. [۱]

داده‌های ذخیره شده در پایگاه‌های داده اغلب حجیم بوده و به تنهایی قابل استفاده نیستند. بلکه صرفا دانش نهفته در آن‌ها قابل استفاده می‌باشد. داده‌کاوی، در حال حاضر مهم‌ترین فناوری به منظور بهره‌وری مؤثر، صحیح و سریع از داده‌های حجیم بوده که در حجم وسیعی از داده، به شناسایی الگوها و مدل‌های صحیح، جدید و بالقوه مفید پرداخته و ترسیم این الگوها و مدل‌ها را، به گونه‌ای که برای انسان قابل درک باشد، دنبال می‌کند. در این تعریف مقصود از الگوی مفید، مدلی در داده‌ها است که ارتباط میان یک زیرمجموعه از داده‌ها را توصیف کرده و ساده، معتبر، قابل فهم و جدید است.

در مسئله مجموعه آیتم‌های سودمند که یکی از زیرشاخه‌های علم داده‌کاوی محسوب می‌شود، هدف نهایی، کاوش مجموعه آیتم‌هایی با میزان سودمندی بالا در پایگاه داده است. کاوش بهینه این مجموعه آیتم‌ها، نقش مهمی را در بسیاری از مسائل کاربردی مانند تحلیل خرید مشتریان ایفا می‌کند. الگوریتم‌های پیشین این حوزه، با ایجاد مجموعه آیتم‌های سودمند کاندید و سپس محاسبه میزان سودمندی دقیق این مجموعه آیتم‌ها، سعی در حل این مسائل دارند. این الگوریتم‌ها ناچار به ایجاد تعداد زیادی مجموعه‌آیتم سودمند کاندید بوده که در عمل بسیاری از آن‌ها سودمند نبوده و ایجاد آن‌ها عملا بی‌فایده محسوب می‌شود.

در این پژوهش، به معرفی روش‌هایی پرداخته می‌شود که بدون نیاز به تولید مجموعه آیتم‌های کاندید، سعی در کاوش مجموعه آیتم‌های سودمند دارند. این الگوریتم‌ها ضمن هرس کردن فضای جست و جو، ساختمان داده‌های جدیدی را معرفی می‌کنند که به کمک آن‌ها جست و جو در میان داده‌ها را سریع‌تر و بهینه‌تر انجام داده و چالش‌های موجود در الگوریتم‌های پیشین را تا حد قابل قبولی رفع می‌کنند. سپس الگوریتم‌هایی تحت عنوان کاوش K مجموعه آیتم سودمند معرفی می‌شوند که بر حسب نیاز صاحب محصول، تعداد معینی از سودمندترین مجموعه آیتم‌های موجود در پایگاه داده را کشف می‌کنند.

در فصل اول، به بررسی پیش‌زمینه‌ها و سرفصل‌های بالاتر این موضوع و معرفی صورت مسئله کلی آن پرداخته می‌شود که شامل تحلیل سبد خرید، کاوش الگوهای پرتکرار، کاوش قواعد وابستگی موجود میان داده‌ها و شرح کلی الگوریتم‌های مطرح موجود در هر کدام است. در فصل دوم، ضمن مروی بر مطالعات و پژوهش‌های پیشین، مسئله کاوش مجموعه آیتم‌های سودمند به‌طور جزئی‌تر و دقیق‌تر شرح داده شده و به بررسی و مقایسه روش‌های مختلف به کار گرفته شده در الگوریتم‌های مختلف مرتبط با این موضوع پرداخته می‌شود. فصل سوم به معرفی الگوریتم استفاده شده در پژوهش حاضر، کاوش سودمندترین k مجموعه آیتم، اختصاص خواهد یافت. سرانجام، در فصل چهارم به بررسی جزییات پیاده‌سازی و نتایج حاصله پرداخته می‌شود.

فصل دوم: مروری بر منابع و معرفی پیش‌زمینه‌ها

۲-۱- مقدمه

پیدایش ایده‌ی مسئله کاوش مجموعه آیتم‌های سودمند، به مسئله تحلیل سبد خرید مشتریان در پایگاه داده‌های تراکنشی بازمی‌گردد. در این فصل ابتدا به شرح این مسئله و سپس به معرفی روش‌های موجود و پایه‌ای برای آن در بحث داده‌کاوی پرداخته می‌شود.

۲-۲- تحلیل سبد خرید مشتریان

فرض کنید به عنوان مدیر فروش یک فروشگاه لوازم الکترونیکی، در حال صحبت با یکی از مشتریانی هستید که به تازگی اقدام به خرید یک لپ‌تاپ و یک دوربین دیجیتال نموده است. چه چیزی را به عنوان خرید بعدی به او پیشنهاد می‌کنید؟ در این شرایط اطلاعات موجود از مشتریان سابق شما که اقدام به خرید لپ‌تاپ و دوربین دیجیتال نموده‌اند، می‌تواند در پیشنهاد شما موثر باشد. کاوش الگوهای پرتکرار و قواعد وابستگی، دانشی هستند که می‌توانند در این سناریوها مفید واقع شوند [۲].

کاوش مجموعه آیتم‌های پرتکرار، منجر به کشف ارتباطات و وابستگی‌های موجود میان آیتم‌های مجموعه داده‌های تراکنشی یا رابطه‌ای می‌گردد. کشف این موارد در میان انبوه رکوردهای تراکنشی کسب و کارها، می‌تواند تاثیر به‌سزایی روی فرایندهایی نظیر تصمیم‌گیری، بازاریابی و تحلیل رفتار مشتریان داشته باشد. یکی از مسائل شاخص کاوش مجموعه آیتم‌های پرتکرار، تحلیل سبد خرید است. این فرایند با بررسی آیتم‌های موجود در سبد خرید مشتریان، به تحلیل عادت‌های مشتریان در هنگام خرید و یافتن وابستگی‌ها و ارتباطات میان آیتم‌ها می‌پردازد و با در اختیار فروشندگان قرار دادن این اطلاعات، به آنان کمک کرده تا میزان فروش خود را با در کنار هم قرار دادن کالاهایی که معمولا با هم به فروش می‌رسند، بهبود بخشند. به عنوان مثال، اگر با تحلیل سبد خرید مشتریان یک فروشگاه الکترونیکی این نتیجه حاصل شود که مشتریانی که اقدام به خرید کامپیوتر نموده‌اند تمایل به خرید

آنتی ویروس[1] نیز داشته اند، در این‌صورت در کنار هم قرار دادن این دو محصول، منجر به افزایش میزان فروش هر دو آن‌ها خواهد شد [۲].

۲-۳- کاوش مجموعه آیتم‌های پرتکرار

با وجود پایگاه‌های داده امروزی، ذخیره‌سازی و مورد استفاده قرار دادن داده‌های حجیم شرکت‌های تجاری، سازمان‌های علمی و دولت‌ها به امری آسان مبدل شده است. در سال‌های اخیر، تحقیقات در خصوص روش‌های به دست آوردن اطلاعات ارزشمند از پایگاه‌های داده مختلف مورد توجه خاصی قرار گرفته و در نتیجه این موضوع، مسائل داده‌کاوی متعددی مطرح شده است. یکی از معروف‌ترین این مسئله‌ها، مسئله کاوش مجموعه آیتم‌های پرتکرار[2] است. یک مجموعه ناتهی از آیتم‌های موجود در یک پایگاه داده، یک مجموعه آیتم[3] نامیده شده و تعداد تراکنش‌های شامل این مجموعه آیتم در آن پایگاه داده، Support آن مجموعه نام دارد. چنانچه Support یک مجموعه آیتم، بیشتر از آستانه تعریف شده توسط کاربر[4] باشد، آنگاه به عنوان یک مجموعه آیتم پرتکرار شناخته خواهد شد. در یک پایگاه داده و به ازای یک آستانه تعریف شده توسط کاربر، این مسئله معادل یافتن تمامی مجموعه آیتم‌های پرتکرار است. [۲]

چالش اساسی در کاوش مجموعه آیتم‌های پر تکرار در مجموعه داده‌های بزرگ این است که منجر به ایجاد تعداد بسیار زیادی مجموعه آیتم پرتکرار خواهد شد. به‌خصوص اگر این مقدار عدد کوچکی بوده باشد. این موضوع به این دلیل است که اگر یک مجموعه آیتم پرتکرار محسوب شود، تمام زیرمجموعه‌های آن نیز پرتکرار محسوب می‌شوند. برای مثال تعداد زیرمجموعه‌های یک مجموعه آیتم پرتکرار ۱۰۰ عضوی طبق رابطه (۲-۱) به دست می‌آید که همگی این زیرمجموعه‌ها نیز پرتکرار محسوب می‌شوند.

---

[1] Antivirus

[2] Frequent Itemsets Mining

[3] Itemset

[4] min_support

$$\binom{۱۰۰}{۱} + \binom{۱۰۰}{۲} + \binom{۱۰۰}{۳} + \cdots \binom{۱۰۰}{۱۰۰} = ۲^{۱۰۰} - ۱ = ۱.۲۷ * ۱۰^{۳۰}$$

رابطه (۲-۱) تعداد زیرمجموعه های یک مجموعه ۱۰۰ عضوی

۲-۴- کاوش قواعد وابستگی

فرض می‌کنیم $I = \{I_۱, I_۲, \dots, I_m\}$ مجموعه تمامی آیتم‌های موجود در پایگاه داده باشد. هر تراکنش غیرتهی در پایگاه داده D را با T نشان می‌دهیم. $T \subset I$ بوده و دارای یک مشخصه TID می‌باشد. A و B نیز هر کدام یک مجموعه آیتم هستند. یک قاعده وابستگی که به‌صورت $A \rightarrow B$ بیان می‌شود، دارای دو مولفه confidence و support بوده که به ترتیب با c و s مشخص می‌شوند. مولفه support بیانگر درصد تراکنش‌هایی است که شامل حداقل یکی از دو مجموعه آیتم‌های A و B باشند. همچنین مولفه confidence بیانگر تراکنش‌های شامل مجموعه آیتم A که شامل مجموعه آیتم B نیز باشند، می‌باشد. به بیان معادل:

$$support(A \rightarrow B) = P(A \cup B)$$

$$confidence(A \rightarrow B) = P(B \mid A)$$

قواعد وابستگی که دارای support و confidence بزرگ‌تر یا مساوی آستانه‌های تعریف شده برای هر یک از این دو مولفه باشند، قواعد وابستگی قوی[۱] نامیده می‌شوند. به طور کلی، کاوش قواعد وابستگی را می‌توان به عنوان یک فرایند ۲ مرحله‌ای در نظر گرفت. [۳]

۱- پیدا کردن تمامی مجموعه آیتم‌های پرتکرار

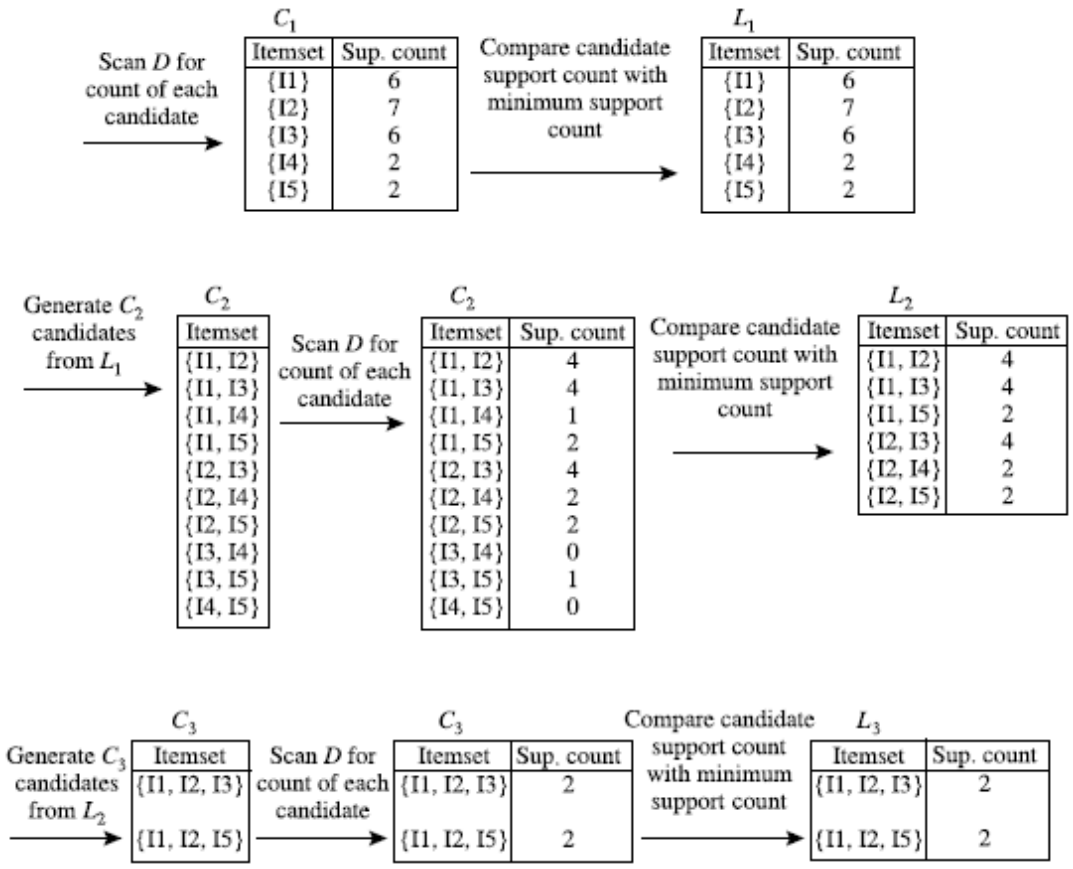
۲- تولید قواعد وابستگی قوی از مجموعه آیتم‌های پرتکرار تولید شده در مرحله قبل

[۱] Strong

ابتدا به معرفی Apriori به عنوان یکی از الگوریتم‌های پایه‌ای برای کاوش مجموعه آیتم‌های پرتکرار پرداخته، سپس ضمن بحث در خصوص تولید قواعد وابستگی از این مجموعه آیتم‌ها، روش‌های مطرح برای بهینه‌سازی این موضوع را معرفی می‌نماییم.

۲-۴-۱- الگوریتم Apriori

این الگوریتم یکی از الگوریتم‌های پایه کاوش مجموعه آیتم‌های پرتکرار به شمار می‌رود. با اجرای چندباره یک روال که جست و جوی مرحله به مرحله نام دارد، مجموعه‌آیتم‌های پرتکرار استخراج می‌شوند. در این روال یک مجموعه آیتم k عضوی، برای استخراج مجموعه آیتم k+1 عضوی مورد استفاده قرار می‌گیرد. ابتدا با جست و جوی پایگاه داده، تمامی مجموعه آیتم‌های یک عضوی پرتکرار استخراج می‌شوند. این مجموعه‌آیتم‌ها که L_1 نامیده می‌شوند، در مرحله بعد برای استخراج L_2 مورد استفاده قرار می‌گیرند که معادل مجموعه آیتم‌های دو عضوی است. این روال تا جایی ادامه پیدا می‌کند که دیگر امکان استخراج مجموعه آیتم‌های بزرگ‌تر وجود نداشته باشد. در هر مرحله و برای به دست آوردن هر L_k، یک بار جست و جوی کامل پایگاه داده صورت می‌گیرد. به منظور بهبود عملکرد این الگوریتم، با کمک یک اصل مهم که اصل Apriori نام دارد، فضای جست و جو محدود شده و کاهش می‌یابد. بر طبق این اصل، چنانچه یک مجموعه آیتم پرتکرار نباشد، هیچ فوق‌مجموعه‌ای از آن نیز نمی‌تواند پرتکرار باشد. شکل (۲-۱) نحوه اجرای این الگوریتم را نشان می‌دهد. [۲]



شکل (۲-۱) نحوه اجرای الگوریتم Apriori [۲].

۲-۴-۲- نحوه ایجاد قواعد وابستگی

پس از استخراج مجموعه آیتم‌های پرتکرار موجود در پایگاه داده D، می‌توان قواعد وابستگی را به شرح زیر ایجاد کرد:

الف) به ازای هر مجموعه آیتم پرتکرار I، تمام زیرمجموعه‌های ناتهی آن را تولید کنید.

ب) به ازای هر زیرمجموعه ناتهی s از مجموعه I، قاعده $s \rightarrow (I - s)$ را با شرط $confidence > min\_conf$ تولید کنید.

با توجه به حاصل شدن این قواعد از مجموعه آیتم‌های پرتکرار، تمامی آن‌ها پرتکرار بوده و نیازی به بررسی این مطلب نمی‌باشد. این مجموعه آیتم‌های پرتکرار می‌توانند برای سهولت دسترسی و سرعت بیشتر در جداول هش ذخیره گردند. [۳]

۲-۴-۳- معرفی روش‌های بهینه‌سازی و الگوریتم‌های مرتبط

به منظور بهبود عملکرد الگوریتم اولیه، تا کنون پیاده‌سازی‌های متعددی با بهره‌گیری از روش‌هایی همچون Hash-based، Transaction reduction، Partitioning و Sampling ارائه شده‌اند. به عنوان نمونه، در پیاده‌سازی مبتنی بر Transaction reduction، با تکیه بر این اصل که یک تراکنش فاقد مجموعه آیتم پرتکرار k عضوی، نمی‌تواند شامل یک مجموعه آیتم پرتکرار k+1 عضوی باشد، مرحله به مرحله تعداد تراکنش‌های نیازمند برررسی کاهش می‌یابد. و یا در پیاده‌سازی بهره‌گیرنده از تکنیک Sampling، با نمونه‌گیری تصادفی از پایگاه داده، جست و جو برای مجموعه آیتم‌های پرتکرار در مقیاسی کوچک‌تر و در درون حافظه RAM صورت پذیرفته و از این طریق، ضمن افزایش سرعت جست و جو، تعداد جست و جوهای کمتری مورد نیاز است. اما طی این حالت ممکن است برخی از مجموعه آیتم‌های پرتکرار استخراج نشوند که برای غلبه بر این چالش، آستانه support کمتری نسبت به آستانه support تعیین شده، در نظر گرفته می‌شود تا در حد امکان هیچ مجموعه آیتم پرتکراری از دست نرود. این پیاده‌سازی برای سناریوهایی نظیر محاسبات حساس پرتکرار که طی آن‌ها، بازدهی الگوریتم از اهمیت بالایی برخوردار است، می‌تواند انتخاب مناسبی باشد. [۲]

۲-۵- مشکلات مربوط به کاوش مجموعه آیتم‌های پرتکرار

فرایند کاوش مجموعه داده‌های پرتکرار با یک هرس رو به پایین همراه است. به سادگی قابل اثبات است که تمامی زیرمجموعه‌های یک مجموعه آیتم پرتکرار، پرتکرار بوده و تمامی فوق‌مجموعه‌های یک مجموعه آیتم غیر پرتکرار، غیر پرتکرار هستند. این ویژگی، امکان بهره‌مندی از رویکرد قدرتمندی را برای الگوریتم به ارمغان می‌آورد. بلافاصله

پس از مشخص شدن یک مجموعه آیتم به عنوان مجموعه آیتم غیر پرتکرار، تمامی فوق‌مجموعه‌های مرتبط با آن نیز به عنوان مجموعه آیتم غیر پرتکرار شناخته خواهند شد.

به عنوان مثال در یک پایگاه داده شامل n آیتم، پس از این‌که الگوریتم یک مجموعه آیتم K عضوی را به عنوان یک مجموعه آیتم غیر پرتکرار شناسایی کرد، دیگر نیازی به بررسی $۲^{n-k} - ۱$ فوق‌مجموعه آن نخواهد بود. با این وجود، کاوش مجموعه آیتم‌های پر تکرار صرفا به حضور یا عدم حضور آیتم‌ها اهمیت داده و سایر اطلاعات مربوط به یک آیتم مانند سود مطلق و یا سود حاصل از آن در فاکتور را در نظر نمی‌گیرد.

در پایگاه داده یک فروشگاه، هر آیتم دارای یک سود مشخص بوده و در هر تراکنش، با تعداد خریداری شده از آن آیتم در ارتباط است. به عنوان مثال فرض کنید جدول سود آیتم‌ها در پایگاه داده دارای ۷ آیتم شامل نام و سود هر آیتم، و جدول تراکنش‌ها دارای ۸ تراکنش شامل مشخصه تراکنش، آیتم‌های موجود در آن، تعداد خریداری شده از هر یک و مجموع کل سود آن تراکنش می‌باشد. برای محاسبه support یک مجموعه‌آیتم، الگوریتم فقط از دو ستون اول جدول تراکنش‌ها استفاده کرده و اطلاعات موجود در جدول سود و دو ستون آخر جدول تراکنش‌ها مورد استفاده قرار نمی‌گیرد. از این رو ممکن‌است یک مجموعه داده پرتکرار استخراج شود در حالی که سود اندکی داشته باشد و یا یک مجموعه آیتم که سود بالایی دارد، به عنوان یک مجموعه پرتکرار شناسایی نشود. [۴]

در برخی سناریوها مانند تحلیل خرید، سود یک مجموعه آیتم، نسبت به تعداد دفعات تکرار آن، از اهمیت بیشتری برای صاحبان کسب و کار برخوردار است. در حالی که الگوریتم‌های کاوش مجموعه آیتم‌های پرتکرار، قادر به تعیین سود مجموعه آیتم‌ها نبوده و از این رو نیاز به الگوریتم جدیدی که سود آیتم‌ها را در کاوش خود مد نظر قرار دهد، احساس می‌شود. به مجموعه آیتم‌هایی که مجموع سود آنها از مقدار مشخصی بیشتر باشد، مجموعه آیتم‌های سودمند می‌گویند. کاوش مجموعه آیتم‌های سودمند به سادگی کاوش مجموعه آیتم‌های پرتکرار نیست. چرا که روش هرس بستار بالا به پایین در خصوص آنها صدق نمی‌کند. طبق این روش، با افزودن یک آیتم به یک مجموعه آیتم دیگر، میزان تکرار مجموعه آیتم ایجاد شده، لزوما کاهش یافته و یا ثابت می‌ماند اما میزان سودمندی

آن از قاعده مشخصی پیروی نکرده و ممکن است دچار کاهش و یا افزایش شده و یا ثابت بماند. در بخش بعد به بیان راه حل و شرح مسئله مطرح شده می‌پردازیم.

۲-۶- تعریف مسئله مجموعه داده‌های پر سود HUIM

همان‌طور که در بخش پیش عنوان شد، هدف از مسئله مجموعه آیتم‌های سودمند، یافتن مجموعه آیتم‌هایی با سود بالا در پایگاه داده می‌باشد. به تازگی الگوریتم‌های متعددی برای یافتن مجموعه آیتم‌های سودمند مطرح شده‌اند که اغلب آن‌ها، یک رویکرد یکسان اتخاذ می‌کنند. این الگوریتم‌ها ابتدا مجموعه آیتم‌های پرسود کاندید را استخراج کرده و در مرحله بعد با محاسبه دقیق سود هر یک از آن ها، سعی در کاوش مجموعه آیتم های پرسود دارند. این الگوریتم‌ها غالبا به‌دلیل ایجاد تعداد زیادی از مجموعه آیتم‌های کاندید، با دو مشکل عمده کمبود حافظه برای ذخیره و زمان اجرای بالا برای ایجاد و محاسبه سود این مجموعه آیتم‌ها مواجه هستند. در سوی مقابل، الگوریتم‌هایی نظیر HUI-Miner و یا *HUI-Miner با بهره‌گیری از ساختارهای جدید و بدون ایجاد مجموعه آیتم‌های کاندید، سعی در ارائه راه حل مناسبی برای دو مشکل ذکر شده الگوریتم‌های قبلی دارند [۵].

۲-۷- الگوریتم‌های با تولید کاندید - الگوریتم‌های بدون نیاز به تولید کاندید

الگوریتم‌های پیشین، با تولید مجموعه‌آیتم‌های کاندید و سپس محاسبه سود دقیق این مجموعه‌ها، سعی در حل این مسائل داشتند. این الگوریتم‌ها ناچار به ایجاد تعداد زیادی مجموعه کاندید با سود پایین بوده که عملا بی‌فایده محسوب می‌شدند. در پژوهش حاضر روش HUI-Miner و نیز روش بهینه‌تر آن *HUI-Miner معرفی می‌گردند که بدون نیاز به ایجاد مجموعه آیتم‌های کاندید، سعی در یافتن مجموعه آیتم‌های سودمند دارند. این الگوریتم‌ها ضمن معرفی ساختمان داده‌های جدید به منظور جست و جوی سریع‌تر و راحت‌تر در میان آیتم‌ها، با هرس کردن فضای جست و جو سعی در انجام این کار به بهینه‌ترین شکل ممکن دارند. HUI-Miner برای ذخیره اطلاعات مورد نیاز درباره سود مجموعه آیتم‌ها و مراحل هرس، از ساختمان داده جدید لیست سودمندی[۱] بهره می‌برد.

[۱] Utility list

لیست سودمندی آیتم‌ها، امکان دسترسی مستقیم به لیست سودمندی سایر آیتم‌ها و نیز محاسبه میزان سود آن‌ها بدون جست و جوی پایگاه داده را ممکن می‌سازد. بدون نیاز به تولید مجموعه آیتم‌های کاندید، این الگوریتم با عملکرد بهینه‌ای اقدام به کاوش مجموعه آیتم‌های سودمند می‌نماید. به منظور هر چه بالاتر بردن سرعت اجرای این الگوریتم، الگوریتم *HUI-Miner از ساختمان داده ارتقا یافته *utility list بهره می‌برد. نتایج بررسی‌ها نشان می‌دهد الگوریتم *HUI-Miner نسبت به الگوریتم HUI-Miner در خصوص پایگاه داده‌های خلوت[1] عملکرد بهتری از خود نشان می‌دهد. این الگوریتم‌ها با وجود مصرف کمتر حافظه، تا چندین برابر، نسبت به الگوریتم‌های فعلی سریع‌تر عمل می‌کنند [۴].

۲-۸- معرفی الگوریتم HUI-Miner

در این بخش به شرح مراحل مختلف الگوریتم HUI-Miner پرداخته و در انتها مقایسه مختصری میان این الگوریتم و الگوریتم *HUI-Miner صورت می‌دهیم.

۲-۸-۱- تعاریف پیش نیاز

فرض کنیم $\varphi = \{I_1, I_2, I_3, \dots, I_m\}$ برابر مجموعه آیتم‌های موجود در پایگاه داده DB متشکل از دو جدول سود و تراکنش باشد. هر آیتم موجود در φ دارای یک مقدار مثبت به عنوان سود در جدول سودها و هر تراکنش موجود در جدول تراکنش‌ها، دارای یک مشخصه منحصر به فرد به نام TID بوده و یک زیرمجموعه از φ می‌باشد. یک زیرمجموعه k عضوی از φ یک مجموعه آیتم k عضوی نامیده می‌شود. تعاریف زیر برای یک آیتم، یک مجموعه آیتم و یک تراکنش برقراراست:

تعریف ۱: سود خارجی یک آیتم که با $eu(i)$ نشان داده می‌شود، برابر با میزان سود آیتم i در جدول سود می‌باشد.

[1] Sparse databases

تعریف ۲: سود داخلی یک آیتم i در تراکنش T که با iu(i, T) نشان داده می‌شود برابر با تعداد آیتم i در تراکنش T می‌باشد.

تعریف ۳: سود یک آیتم i در یک تراکنش T که با u(i, T) نشان داده می‌شود برابر eu(i)*iu(i,T) می‌باشد.

تعریف ۴: سود یک مجموعه آیتم X در یک تراکنش T که با u(X,T) نشان داده می‌شود، برابر است با مجموع سود همه آیتم‌های موجود در X، اگر آن تراکنش شامل مجموعه آیتم X باشد و در غیر این صورت برابر صفر خواهد بود.

تعریف ۵: سود یک مجموعه آیتم X که با u(X) نشان داده می‌شود برابر با مجموع سود آن مجموعه آیتم در همه تراکنش‌های شامل آن است.

تعریف ۶: سود یک تراکنش T که با tu(T) نشان داده می‌شود برابر مجموع سود همه آیتم‌های موجود در آن تراکنش است.

تعریف ۷: میزان TWU یک مجموعه‌آیتم X که با twu(X) نشان داده می‌شود برابر است با مجموع سود همه تراکنش‌های شامل آن مجموعه آیتم.

تعریف ۸: چنانچه میزان TWU یک مجموعه آیتم، کمتر از آستانه سود تعریف شده توسط کاربر باشد، آنگاه تمامی فوق‌مجموعه‌های آن، غیر سودمند خواهند بود.

تعریف ۹: برای یک مجموعه آیتم k عضوی به نام X، تعمیم آن شامل (K + i) آیتم خواهد بود که به آن i امین تعمیم مجموعه آیتم X گفته می‌شود.

تعریف ۱۰: در صورت برقراری دو شرط زیر، یک تراکنش به عنوان revised در نظر گرفته می‌شود. نخست آن که همه آیتم‌های دارای TWU ≤ min_util از تراکنش حذف شده باشند و آیتم‌های باقیمانده به ترتیب صعودی مرتب شده باشند.

تعریف ۱۱: برای هر مجموعه آیتم X و تراکنش T که شرط $X \subset T$ برای آن برقرار باشد، مجموعه تمامی آیتم‌های موجود در T که پس از آخرین آیتم عضو X ظاهر شده‌اند را با T/X نشان می‌دهیم.

تعریف ۱۲: سود باقیمانده یک مجموعه آیتم X در یک تراکنش T که با نماد ru(X,T) نشان داده می‌شود، برابر با مجموع سود همه آیتم‌های موجود در T/X است.

۲-۸-۲- تحقیقات مرتبط

الگوریتم‌های ZP، ZSP، FSH، shFSH، DCG و ... با رویکردی مشابه الگوریتم Apriori که برای کاوش مجموعه آیتم‌های پرتکرار به کار می‌رود، اقدام به کاوش مجموعه آیتم‌های سودمند می‌کنند. ابتدا تمامی مجموعه آیتم‌های یک عضوی به عنوان مجموعه آیتم‌های کاندید در نظر گرفته شده و پس از اعمال جست و جو روی پایگاه داده، میزان سودمندی هر یک از آن‌ها مشخص می‌شود. در مرحله بعد و پس از حذف مجموعه آیتم‌های غیر سودمند، مجموعه آیتم‌های باقیمانده، برای ایجاد مجموعه آیتم‌های دو عضوی مورد استفاده قرار می‌گیرند. این روال تا جایی ادامه پیدا می‌کند که دیگر هیچ مجموعه آیتم جدیدی ایجاد نشود. علاوه بر چالش‌های مطرح شده در بخش قبل، این الگوریتم‌ها در هر مرحله ناچار به جست و جوی پایگاه داده هستند.

در این میان، الگوریتم‌های مبتنی بر FP-Growth همچون UP-Growth، UP-Growth+ عملکرد بهتری از خود نشان می‌دهند. این الگوریتم‌ها با بهره‌گیری از ساختار درخت پیشوندی، قادر هستند تا ضمن ایجاد تعداد کمتری از مجموعه آیتم‌های کاندید، سرعت ایجاد آن‌ها را نیز افزایش دهند. با این حال، کماکان تعداد مجموعه آیتم‌های کاندید تولید شده توسط این الگوریتم‌ها به مراتب بیشتر از تعداد مجموعه آیتم‌های سودمند بوده که این امر موید هدر رفتن زمان و حافظه می‌باشد.

۲-۸-۳- الگوریتم

الگوریتم‌های قدیمی کاوش مجموعه آیتم‌های سودمند، مستقیما روی پایگاه داده اعمال می‌شدند. حتی الگوریتم‌های مبتنی بر FP-Growth که مجموعه آیتم‌های کاندید را با استفاده از درخت پیشوندی ایجاد می‌کنند،

باز هم برای محاسبه سود مجموعه آیتم‌های کاندید، ناگزیر به جست و جوی پایگاه داده هستند. در این بخش سعی می‌شود ضمن معرفی ساختمان داده لیست سودمندی، به ارائه پاسخی مناسب برای این سوال که چگونه می‌توان بدون جست و جوهای مکرر پایگاه داده، مجموعه آیتم‌های سودمند را مورد کاوش قرار داد، پرداخته شود[۴] .

۲-۸-۳-۱- ساختمان داده لیست سودمندی

در الگوریتم HUI-Miner هر مجموعه آیتم با یک لیست سودمندی در ارتباط است. در این بخش به بررسی لیست سودمندی و نحوه ایجاد آن در سه بخش برای مجموعه آیتم‌های دارای ۱، ۲ و K $(K \geq 3)$ آیتم خواهیم پرداخت.

۲-۸-۳-۲- لیست سودمندی اولیه[۱]

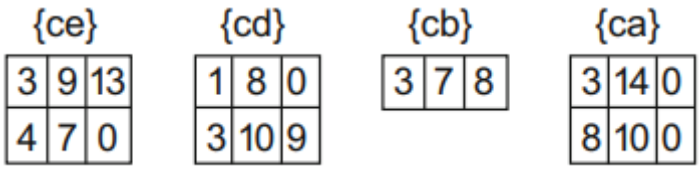
لیست سودمندی مجموعه آیتم‌های یک عضوی، لیست سودمندی اولیه نامیده می‌شود که با دو بار جست و جوی پایگاه داده ساخته می‌شود. در طول جست و جوی اولیه، TWU تمام آیتم‌ها محاسبه شده و بنا به اصل ۱ چنانچه TWU یک آیتم کمتر از آستانه تعیین شده باشد، این آیتم دیگر در فرآیند کاوش مورد بررسی قرار نخواهد گرفت. سپس آیتم‌های با $TWU \geq min\_util$، به ترتیب صعودی TWU مرتب می‌شوند.

۲-۸-۳-۳- لیست سودمندی مجموعه آیتم‌های دو عضوی

لیست سودمندی مجموعه آیتم {xy} را می‌توان بدون جست و جوی پایگاه داده و از طریق اشتراک لیست‌های سودمندی مجموعه آیتم‌های {x} و {y} به دست آورد. الگوریتم با مقایسه مشخصه TID موجود در دو لیست سودمندی، اقدام به شناسایی تراکنش‌های مشترک می‌کند. به ازای هر تراکنش مشترک t، الگوریتم یک عنصر E ایجاد نموده و آن را به لیست سودمندی مجموعه آیتم {xy} اضافه می‌کند. مقدار TID این عنصر، برابر TID تراکنش t خواهد بود. همچنین برای این عنصر، دو مولفه سود داخلی و سود باقیمانده نیز تعریف می‌شود که به

[۱] Initial utility list

ترتیب برابر مجموع سود داخلی دو تراکنش مشترک و سود باقیمانده تراکنش موخر است. شکل (۲-۲) لیست سودمندی مجموعه آیتم‌های دو عضوی را نشان می‌دهد.



شکل (۲-۲) لیست های سودمندی مجموعه آیتم‌های دو عضوی [۴].

به عنوان مثال، برای ایجاد لیست سودمندی مجموعه آیتم {ce} الگوریتم از اشتراک لیست‌های سودمندی مجموعه آیتم‌های {c} و {e} استفاده می‌کند که به ایجاد {(۴،۷،۰),(۳،۹،۱۳)} منجر می‌شود. با توجه به داده‌های نشان داده شده در شکل (۲-۳) می‌توان دریافت که {ce} تنها در تراکنش‌های T_3 و T_4 ظاهر شده‌است.

$$u(\{ce\},T_3) = u(c,T_3) + u(e,T_3) = 6 + 3 = 9$$

به طریق مشابه، $u(\{ce\},T_4) = 4 + 3 = 7$ می‌باشد. همچنین خواهیم داشت:

$$ru(\{ce\},T_3) = u(d,T_3) + u(b,T_3) + u(a,T_3) = 4 + 1 + 8 = 13$$

$$ru(\{ce\},T_4) = 0$$

که درستی توضیحات بالا را نشان می‌دهد [۴].

Tid	Item	Util.	Item	Util.	Item	Util.	Item	Util.	Item	Util.
T1	c	4	d	4						
T2	d	4	b	1						
T3	c	6	e	3	d	4	b	1	a	8
T4	c	4	e	3						
T5	d	4								
T6	e	6	d	4	b	2	a	10		
T7	b	4	a	6						
T8	c	2	a	8						

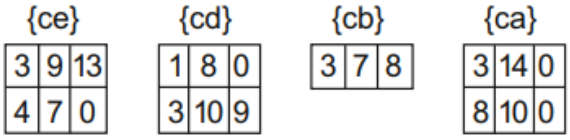
شکل (۲-۳) داده‌های موجود در پایگاه داده تراکنشی [۴].

۲-۸-۳-۴- لیست سودمندی مجموعه آیتم‌های K عضوی ($K \geq 3$)

برای ایجاد لیست سودمندی مجموعه آیتم‌های k عضوی ($K \geq 3$) شامل آیتم‌های $\{i_1, \dots, i_{(k-1)}, i_k\}$، همچون مجموعه آیتم‌های ۲ عضوی می‌توان از اشتراک لیست سودمندی مجموعه آیتم‌های شامل $\{i_1, \dots, i_{(k-2)}, i_k\}$ و $\{i_1, \dots, i_{(k-2)}, i_{(k-1)}\}$ بهره گرفت. با این تفاوت که این بار ممکن است این دو مجموعه، در یک یا چند آیتم مشترک باشند. در این حالت میزان سود داخلی از رابطه (۲-۲) محاسبه می‌شود.

$$u(\{i_1, \dots, i_{(k-1)}, i_k\}, T) = u(\{i_1, \dots, i_{(k-1)}, i_k\}, T) + u(\{i_1, \dots, i_{(k-2)}, i_k\}, T) - u(\{i_1, \dots, i_{(k-2)}\}, T)$$

رابطه (۲-۲) محاسبه سود داخلی



شکل (۲-۴) لیست سودمندی مجموعه آیتم‌های دو عضوی. [۴]

به عنوان مثال چنانچه بخواهیم برای ایجاد لیست پیوندی مجموعه‌آیتم {ced} از لیست پیوندی مجموعه آیتم‌های {ce} و {cd} استفاده کنیم، با توجه به مشترک بودن آیتم c در این دو مجموعه‌آیتم، خواهیم داشت:

$$u(\{ced\},T_3) = u(\{ce\},T_3) + u(\{cd\},T_3) - u(c,T_3) = 9 + 10 - 6 = 13$$

شکل (۲-۵) شبه کد روال ایجاد لیست سودمندی این گونه مجموعه‌آیتم‌ها را نشان می‌دهد.

Algorithm 1 *Construct(P.UL, Px.UL, Py.UL)*

Input: *P.UL*, the utility-list of itemset *P*;
Px.UL, the utility-list of itemset *Px*;
Py.UL, the utility-list of itemset *Py*.
Output: *Pxy.UL*, the utility-list of itemset *Pxy*.
Pxy.UL = *NULL*
foreach element *Ex* ∈ *Px.UL* **do**
  if ∃*Ey*∈*Py.UL* **and** *Ex.tid*==*Ey.tid* **then**
    if *P.UL* is not empty **then**
      search such *E*∈*P.UL* that *E.tid*==*Ex.tid* *Exy* = <*Ex.tid*, *Ex.iutil*+*Ey.iutil*-*E.iutil*, *Ey.rutil*>
    else
      Exy = <*Ex.tid*, *Ex.iutil*+*Ey.iutil*, *Ey.rutil*>
    end
    append *Exy* to *Pxy.UL*
  end
end
return *Pxy.UL*

شکل (۲-۵) شبه کد روال ایجاد لیست سودمندی مجموعه آیتم‌ها [۴].

۲-۸-۴- استراتژی هرس

این الگوریتم، مجموعه آیتم‌های سودمند را، در یک درخت شمارش مجموعه‌ای در مرتبه اول عمق مورد جست و جو قرار می‌دهد. پس از اولین جست و جو در پایگاه داده و ایجاد لیست‌های سودمندی اولیه، الگوریتم با یک مرحله تعمیم آن‌ها، لیست‌های سودمندی جدیدی ساخته و با صرف نظر کردن از تعمیم‌های غیرسودمند، با تکرار این روال و تعمیم مجموعه آیتم‌های سودمند، به جست و جو ادامه می‌دهد. به منظور محدودسازی فضای جست و جو، الگوریتم HUI-Miner از سودهای داخلی و سودهای باقیمانده لیست‌های سودمندی بهره می‌گیرد.

بر طبق قاعده‌ای، چنانچه مجموع همه سودهای داخلی و سودهای باقیمانده لیست سودمندی یک مجموعه آیتم از آستانه سود تعریف شده کمتر باشد، در این‌صورت دیگر هیچ تعمیمی از این مجموعه آیتم، سودمند نبوده و نیازی به بررسی آن‌ها نخواهد بود [۵].

۲-۸-۵- الگوریتم بهبود یافته *HUI-Miner

الگوریتم HUI-Miner از طریق ایجاد لیست‌های سودمندی، به کاوش مجموعه آیتم‌های سودمند می‌پردازد. همان طور که پیش‌تر بررسی شد، ایجاد لیست‌های سودمندی از طریق مقایسه TIDها صورت گرفته و با وجود ساده بودن این مقایسه، الگوریتم ناچار به انجام تعداد زیادی از این مقایسه‌ها در حین فرایند کاوش می‌باشد. الگوریتم *HUI-Miner با بهره گیری از ساختار *utility-list قادر است تا بدون مقایسه TIDها، *utility-list ها را ایجاد کند و در نتیجه عملکرد بهتری نسبت به الگوریتم HUI-Miner دارد. این الگوریتم همواره میزان حافظه مصرفی کمتری نسبت به الگوریتم HUI-Miner داشته و در پایگاه داده‌های خلوت، به مراتب زمان اجرای پایین‌تری نسبت به الگوریتم HUI-Miner دارد [۴].

فصل سوم : معرفی مسئله مطرح شده در پژوهش حاضر: کاوش سودمندترین k مجموعه آیتم

۳-۱- معرفی مسئله و مقدمه

طی سال‌های اخیر، پژوهش‌های متعددی در زمینه مسائلی همچون کاوش برترین K قاعده وابستگی، برترین K قاعده ترتیبی، پرتکرارترین K مجموعه آیتم و نیز سودمندترین K مجموعه آیتم صورت گرفته است. آن چه این مسائل را از یکدیگر متمایز می‌سازد، نوع الگوی مورد کاوش قرار گرفته، ساختارهای داده و رویکردهای جست و جوی به کار گرفته شده در الگوریتم‌های طراحی شده برای آن‌ها است. انتخاب ساختار داده و رویکردهای جست و جو، بازدهی عملکرد را از لحاظ زمان اجرا و میزان حافظه مصرفی تحت تاثیر قرار می‌دهد. در مسئله قدیمی کاوش سودمندترین K الگوی برتر، که نزدیک‌ترین مسئله به موضوع مورد بحث می‌باشد، صرفا سود هر آیتم در نظر گرفته می‌شد و میزان تکرار آن‌ها در تراکنش در نظر گرفته نمی‌شد اما در مسئله حاضر، هر دو مورد در نظر گرفته خواهند شد [۶].

کاوش مجموعه آیتم‌های سودمند به یافتن مجموعه آیتم‌هایی که سود آنها از آستانه تعیین شده توسط کاربر بیشتر باشد، می‌پردازد. در این مسئله تعیین آستانه سود از اهمیت بالایی برخوردار می‌باشد. چرا که اندازه خروجی مسئله را تعیین کرده و روی بازدهی الگوریتم تاثیر چشم‌گیری می‌گذارد. از این رو، تعیین بهینه و مناسب این آستانه، برای کاربران امری دشوار به شمار می‌رود. اگر آستانه تعیین شده بسیار پایین باشد، تعداد زیادی مجموعه آیتم سودمند شناسایی خواهد شد که ضمن افزایش میزان حافظه مصرفی، فهم نتایج را برای کاربر دشوار می‌سازد. از سوی دیگر چنانچه این مقدار، بیش از حد بزرگ باشد، ممکن‌است هیچ مجموعه آیتم سودمندی شناسایی نشود. برای حل این چالش، یک آستانه سود داخلی تعریف می‌شود که آن را در ابتدا با صفر یا یک مقداردهی می‌کنند. سپس با استفاده از راهبردهای مناسب، این مقدار را افزایش داده تا سودمندترین K مجموعه آیتم مطلوب، مورد کاوش قرار گیرند.

دو الگوریتم بهینه کاوش سودمندترین K مجموعه آیتم[1] و کاوش سودمندترین K مجموعه آیتم در یک فاز[2] برای حل این مسئله پیشنهاد شده‌اند. این الگوریتم‌ها بدون نیاز به تعیین آستانه سود از سوی کاربر، به شناسایی k مجموعه آیتم دارای بیشترین سود در پایگاه داده‌های تراکنشی می‌پردازند. نتایج بررسی‌ها نشان می‌دهد که این الگوریتم‌ها در مقایسه با الگوریتم‌های مدرن امروزی، عملکرد مناسبی از خود نشان داده‌اند. در این فصل به بررسی این الگوریتم‌ها و بحث در خصوص مزایا و محدودیت‌های هر یک می‌پردازیم.

۳-۲- چالش‌های موجود در کاوش سودمندترین K مجموعه آیتم

با وجود این‌که کاوش سودمندترین K مجموعه آیتم، در برخی از سناریوها و کاربردها بسیار حائر اهمیت است، اما برای توسعه الگوریتم‌هایی بهینه به منظور کاوش این مجموعه آیتم‌ها، چالش‌هایی مطرح است. اولین چالش موجود، یکنواخت نبودن سود مجموعه آیتم‌هاست. بدین معنی که میزان سود یک مجموعه آیتم، رابطه مشخصی با میزان سود فوق مجموعه‌های خود ندارد. دومین چالش، چگونگی ترکیب مفهوم کاوش برترین K الگو با مدل TWU است. اگر چه این مدل به طور گسترده در کاوش سود مورد استفاده قرار می‌گیرد، اما با توجه به مشخص نبودن سود دقیق مجموعه آیتم‌ها در فاز اول اجرای الگوریتم، استفاده از آن برای مسئله موجود، کار دشواری است. سومین چالش، مشخص نبودن آستانه سود می‌باشد. الگوریتم‌های قبلی با در اختیار داشتن این آستانه، قادر بودند فضای جست و جو را به طور بهینه‌ای کاهش داده و محدود سازند. از این رو میزان این آستانه در ابتدا صفر در نظر گرفته می‌شود و الگوریتم طراحی شده می‌بایستی رفته‌رفته این میزان را افزایش داده تا بتواند فضای جست و جو را محدود سازد.

تعیین و به کارگیری رویکردهایی موثر و بهینه به منظور افزایش هر چه سریع‌تر و بیشتر آستانه سود، به گونه‌ای که در طول اجرای الگوریتم، کمترین میزان حافظه و زمان جهت ایجاد مجموعه آیتم‌های غیرنهایی مورد نیاز

[1] Top-k high utility itemset mining

[2] Top-k high utility itemset mining in one phase

باشد، یکی از چالش‌های اساسی و مهم در توسعه این گونه الگوریتم‌ها به شمار می‌رود. چهارمین و آخرین چالش موجود، افزایش بهینه آستانه سود بدون از دست دادن هیچ یک از K مجموعه آیتم‌های سودمند می‌باشد. در صورت استفاده از یک روش نامناسب جهت افزایش این آستانه، ممکن است برخی از این مجموعه آیتم‌ها در حین کاوش، به اشتباه دچار هرس شده و در نتیجه از مجموعه آیتم‌های سودمند نهایی حذف شوند. در طراحی الگوریتم‌های پیشنهادی میبایستی راه حل مناسبی برای هر یک از چالش‌های عنوان شده، اندیشیده شود [۶].

۳-۳- معرفی روش‌های ارائه شده

الگوریتم‌های کاوش مجموعه آیتم‌های سودمند، عموما به دو دسته الگوریتم‌های یک مرحله‌ای و الگوریتم‌های دو مرحله‌ای تقسیم می‌شوند. الگوریتم‌های دو مرحله ای، ابتدا اقدام به ایجاد مجموعه آیتم های سودمند کاندید کرده و سپس در مرحله بعدی با محاسبه سود دقیق هر یک این مجموعه آیتم‌ها، به شناسایی مجموعه آیتم‌های سودمند می‌پردازند. الگوریتم‌های IHUP، IIDS و UP-Growth جزو این دسته الگوریتم‌ها به شمار می‌روند. برای مثال UP-Growth یکی از جدیدترین و بهینه‌ترین الگوریتم‌های موجود بوده که با بهره گیری از استراتژی‌های بهینه و موثری همچون DGU، DGN، DLU و DLN در مرحله اول، اقدام به هرس کردن و محدود سازی فضای جست و جو نموده و سپس با محاسبه سود دقیق مجموعه آیتم‌های کاندید، به کاوش مجموعه آیتم‌های سودمند می‌پردازد. از مزایای این روش‌ها می‌توان به سهولت در فهم و پیاده‌سازی، و از معایب آن‌ها می‌توان به مصرف زیاد حافظه به دلیل زیاد بودن تعداد کاندیدهای تولید شده، بالا بودن تعداد دفعات جست و جوی پایگاه داده برای محاسبه سود هر مجموعه آیتم، استفاده از رویکردهای نه چندان دقیق برای هرس فضای جست و جو و بالا بودن زمان اجرا به دلیل جست و جوهای متعدد پایگاه داده اشاره کرد [۷].

در سوی مقابل، الگوریتم‌های یک مرحله‌ای تنها در یک مرحله و بدون ایجاد مجموعه آیتم‌های سودمند کاندید، به کاوش مجموعه آیتم‌های سودمند می‌پردازند. الگوریتم‌های d^2 HUP و HUI-Miner جزو این دسته الگوریتم‌ها به شمار می‌روند. به عنوان مثال، الگوریتم d^2 HUP با تبدیل یک پایگاه داده افقی، به یک ساختار مبتنی بر درخت

به نام CAUL، به کاوش مستقیم مجموعه آیتم‌های سودمند از پایگاه داده می‌پردازد. کاهش تعداد دفعات جست و جوی پایگاه داده به‌دلیل بهره‌مندی از ساختمان داده‌های جدیدی مانند لیست سودمندی، کاهش میزان حافظه مصرفی و مدت زمان اجرا نسبت به روش‌های دوفاز با توجه به عدم ایجاد مجموعه‌های کاندید و عدم نیاز به جست و جوهای متعدد پایگاه داده و نیز، کران‌های بالای دقیق‌تر با وجود عملکرد مناسب این الگوریتم‌ها در برخی سناریوها را می‌توان از مزایای این دسته الگوریتم‌ها به شمار آورد [۶].

۳-۳-۱- معرفی الگوریتم‌های TKO و TKU

در ادامه دو الگوریتم TKU(کاوش سودمندترین K مجموعه آیتم بدون مشخص کردن آستانه سود) به عنوان یک نمونه از الگوریتم‌های دو مرحله‌ای و TKO(کاوش مجموعه‌داده‌های سودمند در یک فاز) به عنوان یک نمونه از الگوریتم‌های یک مرحله‌ای مورد بررسی قرار گرفته و به مقایسه آن‌ها پرداخته می‌شود.

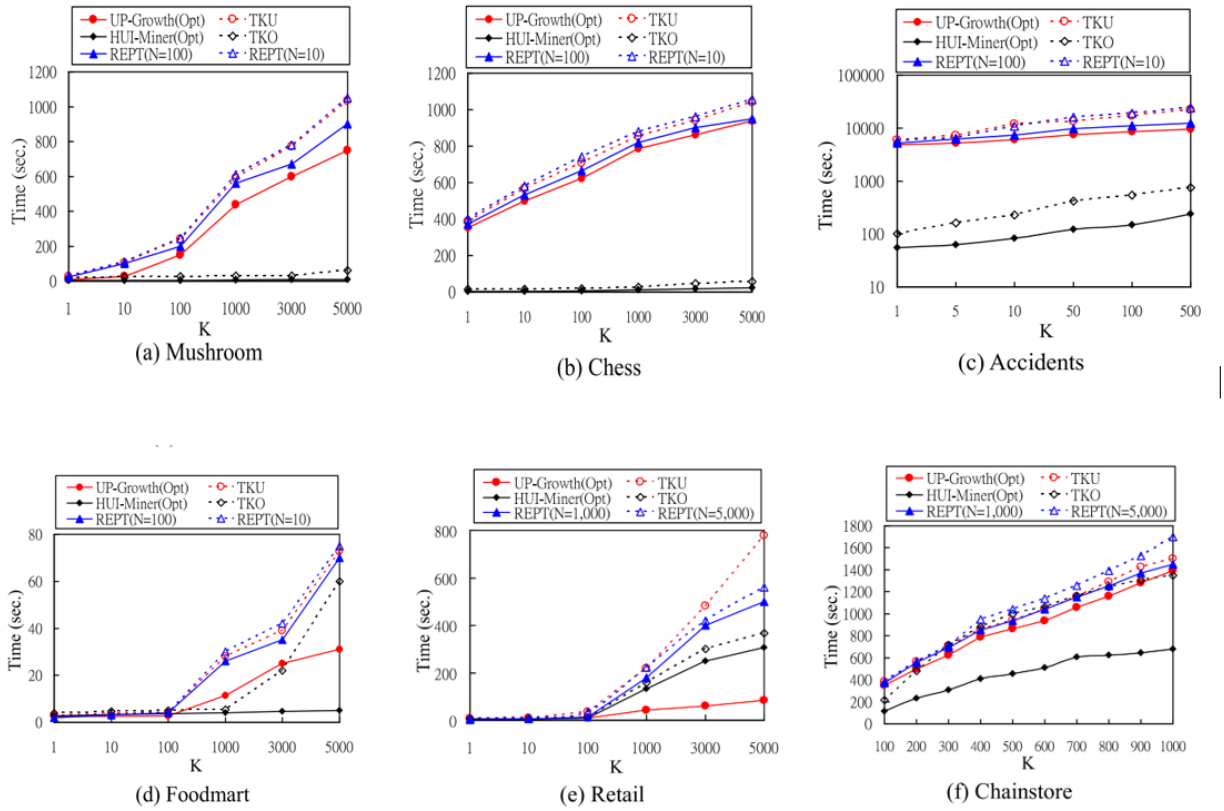
TKUBase که پیاده‌سازی ساده TKU محسوب می‌شود، بسطی از UP-Growth، که یکی از الگوریتم‌های مبتنی بر درخت است، می‌باشد. TKUBase ساختار UP-Tree از الگوریتم UP-Growth را برای نگهداری اطلاعات تراکنش‌ها و ذخیره سودمندترین K مجموعه آیتم‌ها، مورد استفاده قرار می‌دهد. این الگوریتم در سه مرحله اجرا می‌شود: (۱) ساختن UP-Tree، (۲) کاوش K مجموعه‌آیتم سودمند بالقوه[۱] از UP-Tree، و (۳) شناسایی سودمندترین K مجموعه آیتم از مجموعه سودمندترین K مجموعه آیتم بالقوه [۸]

TKOBase نیز که پیاده‌سازی ساده TKO محسوب می‌شود، بسطی از الگوریتم HUI-Miner بوده که در فصل گذشته به تفصیل به بررسی آن پرداختیم. این الگوریتم از ساختمان داده لیست سودمندی برای نگهداری اطلاعات تراکنش‌ها و اطلاعات مورد نیز برای ایجاد مجموعه آیتم‌های سودمند استفاده می‌کند. جزئیات بیشتر مربوط به الگوریتم TKO در ادامه آورده شده است.

[۱] Potential top-k high utility itemsets mining

خروجی حاصل از الگوریتم TKU بسیار دقیق بوده اما مدت زمان اجرا و میزان استفاده منابع در آن بسیار بالا است. در سوی مقابل، TKO خروجی کاملا دقیقی نداشته و حتی ممکن است مجموعه آیتم‌های اشتباهی در خروجی دیده شود. اما از سرعت اجرای بالایی برخوردار است. یکی از بهترین روش‌ها، ترکیب این دو الگوریتم با هم است؛ به نحوی که نتیجه نهایی حاصل از TKO به TKU داده شود تا در زمان کمتر، خروجی دقیق‌تری حاصل شود [۹].

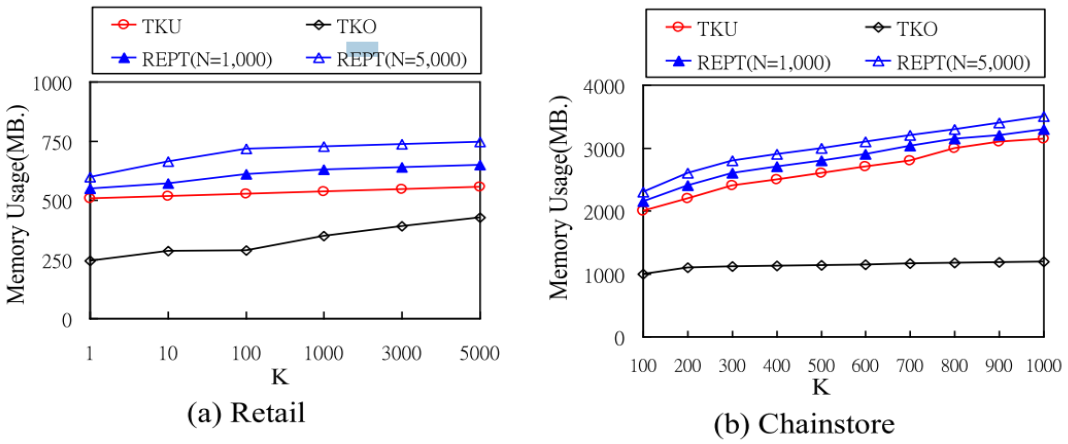
مقایسه این دو الگوریتم از نظر زمان اجرا روی سه مجموعه داده Accidents، Chess و Mushroom به عنوان نمونه‌هایی از مجموعه داده‌های فشرده، و روی سه مجموعه داده Retail، Chainstore و Foodmart به عنوان نمونه‌هایی از مجموعه داده‌های خلوت، به ازای مقادیر متفاوتی از K در شکل‌های زیر آورده شده‌است:



شکل (۳-۱) مقایسه الگوریتم‌های مختلف داده‌کاوی روی مجموعه داده‌های خلوت و فشرده از نظر زمان اجرا.[۶]

همانطور که مشاهده می‌شود، با توجه به یک مرحله‌ای بودن الگوریتم TKO، زمان اجرای این الگوریتم، به مراتب کمتر از الگوریتم‌های دیگر است.

مقایسه این دو الگوریتم از نظر میزان حافظه مصرفی روی دو مجموعه داده Retail و Chainstore به ازای مقادیر متفاوتی از K در نمودارهای زیر آورده شده است که نشان می‌دهد، به طور کلی میزان حافظه مصرفی در الگوریتم TKO، کمتر از میزان حافظه مصرفی در الگوریتم TKU می‌باشد [۶].



شکل (۳-۲) مقایسه عملکرد دو الگوریتم TKO و TKU از نظر حافظه مصرفی به ازای K های مختلف. [۶]

۳-۴- معرفی الگوریتم های TKO و TKOBase

۳-۴-۱ مقدمه

یکی از الگوریتم های پیشنهادی، الگوریتم TKO یا همان کاوش سودمندترین K مجموعه آیتم در یک مرحله می‌باشد. این الگوریتم با بهره گیری از روال جست و جوی اصلی الگوریتم HUI-Miner و ساختمان داده لیست سودمندی، قادر است تا مجموعه آیتم‌های سودمند را تنها در یک مرحله کاوش نماید. به محض ایجاد یک مجموعه

آیتم توسط این الگوریتم، میزان سود آن از طریق لیست سودمندی و بدون نیاز به جست و جوی پایگاه داده محاسبه می‌گردد. در ادامه ابتدا به بررسی نسخه ساده این الگوریتم با عنوان TKOBase پرداخته و سپس نسخه‌های پیشرفته‌تر این الگوریتم را که شامل رویکردهای متعددی به جهت بهینه‌سازی بیشتر هستند، مورد بررسی قرار می‌دهیم.

۳-۴-۲ الگوریتم TKOBase

الگوریتم TKOBase یک پارامتر k و یک پایگاه داده تراکنشی D در قالب افقی به عنوان ورودی دریافت می‌کند. چنانچه پیش‌تر این پایگاه داده به یک قالب عمودی همچون لیست سودمندی تغییر یافته باشد، می‌تواند مستقیما توسط الگوریتم مورد استفاده قرار گیرد. این الگوریتم در ابتدا یک آستانه سودمندی به نام $\min\_util_{Border}$ با مقدار صفر در نظر گرفته و یک ساختمان داده از نوع min-heap به نام TopK-CI-List برای ذخیره سودمندترین K مجموعه آیتم فعلی در طول فرایند کاوش ایجاد می‌کند. سپس با ۲ بار جست و جوی پایگاه داده، لیست‌های سودمندی اولیه را ایجاد می‌کند.

پس از این مراحل، با اعمال روشی که ترکیبی از رویکرد نوین RUC[۱] و روال جست و جوی الگوریتم HUI-Miner است، به کاوش فضای جست و جو می‌پردازد. در زمان اجرای الگوریتم، سودمندترین K مجموعه آیتم موجود در TopK-CI-List همواره به‌روزرسانی شده و مقدار min_util_{Border} به آهستگی افزایش داده می‌شود. پس از اتمام اجرای الگوریتم، TopK-CI-List دربردارنده سودمندترین K مجموعه آیتم ممکن موجود در پایگاه داده خواهد بود. در هر مرحله و برای هر مجموعه‌آیتم L عضوی $\{x_1,x_2,x_l\}$ ایجاد شده، چنانچه میزان سودمندی این مجموعه آیتم از min_util_{Border} کمتر نباشد، آن گاه رویکرد RUC به جهت افزایش min_util_{Border} روی آن اعمال خواهد شد. به این‌صورت که ابتدا مجموعه آیتم فوق به لیست TopK-CI-List اضافه شده و در صورت وجود بیشتر از K مجموعه آیتم در آن، مقدار min_util_{Border} به سود K-امین مجموعه آیتم سودمند موجود در لیست

[۱] Raising threshold by utility of candidates

افزایش می‌یابد و پس از آن مجموعه آیتم‌های دارای میزان سودمندی کمتر از min_util_{Border} از لیست حذف خواهند شد. این موضوع منجر می‌شود تا همواره حداکثر K مجموعه آیتم سودمند در لیست موجود باشد. شکل (۱-۴) شبه کد روال جستحوی این الگوریتم را نشان می‌دهد [۶].

PROCEDURE: TopK-HUI-Search

Input: (1) $u(P)$: utility-list for a prefix P;
(2) $Class[P]$: a set of itemsets w.r.t. the prefix P;
(3) $ULS[P]$: a set of utility-lists w.r.t.the prefix P;
(4) δ: border minimum utility threshold min_util_{Border};
(5) *TopK-CI-List*: a list for storing candidate itemsets;

Results: (1) Use *TopK-CI-List* to capture all the top-k HUIs

```
01.  For each X={x1, x2,…, xL} ∈Class[P] do
02.  {  If (SUM(X.iutils) ≥ δ)
03.     {  //Raise min_util_Border by the strategy RUC;
04.        δ ← RUC(X, TopK-CI-List);
05.     }
06.     If (SUM(X.iutils) + SUM(X.rutils) ≥ δ)
07.     {  Class[X] ← Ø; ULS[X] ← Ø;
08.        For each Y = {y1, y2,…, yL} ∈Class[P] | yL ≻ xL do
09.        {  Z ← X ∪ Y;
10.           ul(Z) ← Construct(ul(P), X, Y, ULS[P]);
11.           Class[X] ← Class[X] ∪ Z;
12.           ULS[X] ← ULS[X] ∪ ul(Z);
13.        }
14.        TopK-HUI-Search(X, ULS[X], Class[X], δ, TopK-CI-List);
15.     }
16.  }
```

شکل (۳-۳) شبه کد روال جست و جوی الگوریتم TKOBase [۶].

۳-۴-۳- الگوریتم TKO

با به کارگیری رویکردهایی موثر و بهینه، می‌توان بازدهی الگوریتم TKOBase را ارتقا بخشید. الگوریتم جدید حاصل از به کارگیری این رویکردها، الگوریتم TKO نام دارد. یکی از این رویکردها، رویکرد[۱]RUZ است که برای بررسی آن ابتدا لازم است به ارائه تعریف‌های زیر بپردازیم.

تعریف ۱: یک عنصر Z در یک لیست سودمندی، عنصری است که میزان سود باقیمانده آن صفر باشد. مجموعه همه عناصر Z موجود در یک لیست سودمندی را با ZE(X) و مجموع سود داخلی همه این عناصر را با NZEU(X) نمایش می‌دهند.

تعریف ۲: مجموع سود باقیمانده تمام عناصر لیست سودمندی یک مجموعه آیتم را سود باقیمانده آن مجموعه آیتم نامیده و با RU(X) نشان می‌دهند.

اصل ۱: چنانچه برای یک مجموعه آیتم X رابطه $NZEU(X) + RU(X) < min\_util_{Border}$ برقرار باشد، هیچ یک از فوق‌مجموعه‌های آن، جزو سودمندترین K مجموعه آیتم‌ها نخواهند بود.

طبق این اصل، چناچه شرط فوق برای هر یک از مجموعه آیتم‌های کاندید تولید شده در طول فرایند کاوش برقرار باشد، دیگر نیازی به بررسی فوق‌مجموعه‌های آن نخواهد بود. یکی دیگر از این رویکردها، رویکرد[۲]EPB می‌باشد. در این رویکرد تلاش می‌شود تا ابتدا مجموعه آیتم‌های کاندید با بیشترین سود ایجاد شوند؛ چرا که در این حالت میزان آستانه سودمندی زودتر و سریع‌تر افزایش یافته و هرس کردن فضای جست و جو، بهتر و بهینه‌تر صورت می‌پذیرد. با استفاده از رویکرد، مجموعه آیتم‌ها به ترتیب نزولی میزان سود تخمین زده شده(به عنوان مثال مجموع سودهای باقیمانده و سودهای داخلی)، مورد بررسی قرار می‌گیرند. رویکردهای PE و DGU از دیگر رویکردهای

[۱] Reducing estimated utility values by using Z-elements

[۲] Exploring the most Promising Branches first

مورد استفاده در الگوریتم TKO به شمار می‌آیند. در فصل بعد به پیاده‌سازی این الگوریتم‌ها در محیط برنامه‌نویسی و جمع‌بندی مطالب ارائه‌شده می‌پردازیم [۶].

فصل چهارم: جزییات پیاده‌سازی و نتایج حاصله

۴-۱- مقدمه

در این فصل ابتدا الگوریتم TKOBase را با زبان برنامه‌نویسی C# پیاده‌سازی کرده و سپس با بهره‌گیری از استراتژی RUZ، به پیاده‌سازی الگوریتم TKO پرداخته شده است. در پایان نیز، نتایج حاصل از اجرای این الگوریتم‌ها روی مجموعه داده‌های مختلف، مورد بررسی و ارزیابی قرار گرفته است [۱۰].

۴-۲- پیاده‌سازی با زبان C#

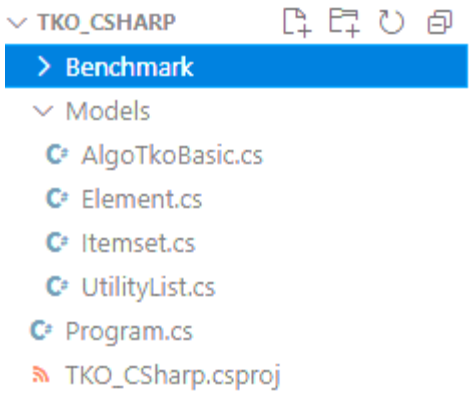
پیاده سازی این الگوریتم به روش شی‌گرا انجام شده است. پیش از بررسی جزئیات مربوط به کد، ابتدا به ارائه توضیح مختصری در خصوص برنامه‌نویسی شی‌گرا و ساختار کلی سیستم می پردازیم.

۴-۳- ساختار شی‌گرای برنامه

با توجه به نیاز به پیاده‌سازی یک ساختمان داده جدید و استفاده بلادرنگ از آن در طول برنامه، ساختارهای عمل‌گرا یا ماژولار، چندان مناسب به نظر نمی‌رسیدند. ساختارهای شی‌گرا به تسهیل عملیات‌هایی مانند دستیابی، درج، حذف، جست و جو و مرتب‌سازی که در طول برنامه مرتبا از آنها استفاده می‌شود، کمک می‌کنند.

ساختمان داده‌های از نوع لیست(در پیاده‌سازی مورد بحث، لیست سودمندی) معمولا در مواردی که حفظ ترتیب آیتم‌های موجود در آن حائز اهمیت باشد، کاربرد بیشتری دارند. لیست‌ها نوع دسترسی ترتیبی را امکان‌پذیر ساخته که با توجه به توضیحات ارائه شده در فصل گذشته، در این پیاده‌سازی کاملا کاربردی هستند.

ساختار فایل‌های برنامه در شکل (۴-۱) آورده شده است که در ادامه هر یک به تفصیل بررسی خواهند شد.



شکل (۴-۱) ساختار فایل‌های برنامه.

۴-۳-۱- کلاس Element

هر یک از سطرهای موجود در لیست سودمندی یک مجموعه آیتم، به مدل کلاس Element نگاشت شده‌اند.

شکل (۴-۲) پیاده‌سازی این کلاس را نشان می‌دهد.

```
16 references
public class Element
{
    // The three variables as described in the paper:
    public int tid = -1;   // transaction id
    public int iutils = 0; // itemset utility
    public int rutils = 0; // remaining utility

    4 references
    public Element(int tid, int iutils, int rutils)
    {
        this.tid = tid;
        this.iutils = iutils;
        this.rutils = rutils;
    }
}
```

شکل (۴-۲) کلاس Element.

۴-۳-۲- کلاس UtilityList

برای بیان مفهوم لیست سودمندی یک مجموعه آیتم، از کلاس UtilityList استفاده شده است. شکل (۴-۳) پیاده‌سازی این کلاس را نشان می‌دهد.

```
34 references
public class UtilityList
{
    public int item;  // the item
    public int sumIutils = 0;  // the sum of item utilities
    public int sumRutils = 0;  // the sum of remaining utilities
    public List<Element> elements = new List<Element>();  // the elements

    3 references
    public UtilityList(int item)
    {
        this.item = item;
    }

    /**
     * Method to add an element to this utility list
       and update the sums at the same time. */
    4 references
    public void addElement(Element element)
    {
        sumIutils += element.iutils;
        sumRutils += element.rutils;
        elements.Add(element);
    }
}
```

شکل (۴-۳) کلاس UtilityList.

۴-۳-۳- کلاس Itemset

همچنین، برای بیان مفهوم یک مجموعه آیتم، از کلاس Itemset استفاده شده است. در شکل (۴-۴) پیاده‌سازی این کلاس، نشان داده شده است.

```
8 references
public class Itemset
{
    public int[] itemset;
    public int item;
    public long utility; // absolute support
    2 references
    public int[] getItemset()
    {
        return itemset;
    }
    0 references
    public int getItem()
    {
        return item;
    }
    1 reference
    public Itemset(int[] itemset, int item, long utility)
    {
        this.itemset = itemset;
        this.item = item;
        this.utility = utility;
    }
    0 references
    public int CompareTo(Itemset o)
    {
        if (o == this)   return 0;
        long compare = this.utility - o.utility;
        if (compare > 0) return 1;
        if (compare < 0) return -1;
        return 0;
    }

}
```

شکل (۴-۴) کلاس Itemset.

۴-۳-۴- کلاس TKO_Algorithm

این کلاس دربردارنده متدها و مولفه‌های مورد نیاز برای پیاده‌سازی الگوریتم است. در این پیاده‌سازی، برای ایجاد یک لیست سودمندی جدید، از لیست‌های سودمندی دو مجموعه آیتم داده‌شده، از متد Construct استفاده می‌شود. شکل (۴-۵) پیاده‌سازی این متد را نشان می‌دهد.

```
2 references
private UtilityList Construct(UtilityList p, UtilityList px, UtilityList py)
{
    // create an empty utility list for pXY
    UtilityList pxyUL = new UtilityList(py.item);
    // for each element in the utility list of pX
    foreach (var ex in px.elements)
    {
        // do a binary search to find element ey in py with tid = ex.tid
        Element ey = FindElementWithTid(py, ex.tid);
        if (ey == null)
        {
            continue;
        }
        // if the prefix p is null
        if (p == null)
        {
            // Create the new element
            Element eXY = new Element(ex.tid, ex.iutils + ey.iutils, ey.rutils);
            // add the new element to the utility list of pXY
            pxyUL.addElement(eXY);

        }
        else
        {
            // find the element in the utility list of p wih the same tid
            Element e = FindElementWithTid(p, ex.tid);
            if (e != null)
            {
                // Create new element
                Element eXY = new Element(ex.tid, ex.iutils + ey.iutils - e.iutils,ey.rutils);
                // add the new element to the utility list of pXY
                pxyUL.addElement(eXY);
            }
        }
    }
    return pxyUL;
}
```

شکل (۴-۵) پیاده‌سازی متد Construct

در بدنه این متد، برای یافتن تراکنش‌های مشترک در دو لیست سودمندی، از متد FindElementsWithTid استفاده شده است. این متد به بررسی وجود و یا عدم وجود یک تراکنش با Tid مشخص در لیست سودمندی

دیگر می‌پردازد. به منظور افزایش سرعت جست و جو، از روش جست و جوی دودویی استفاده شده است. پیاده‌سازی این متد در شکل (۴-۶) نشان داده است.

```
2 references
private Element FindElementWithTid(UtilityList ulist, int tid)
{
    List<Element> list = ulist.elements;
    // perform a binary search to check if  the subset appears in  level k-1.
    int first = 0;
    int last = list.Count - 1;
    while (first <= last)
    {
        int middle = (first + last) >> 1; // divide by 2
        if (list[middle].tid < tid)
            first = middle + 1;  //  the itemset compared is larger than the subset according to the lexical order
        else if (list[middle].tid > tid)
            last = middle - 1; //  the itemset compared is smaller than the subset  is smaller according to the lexical order
        else
            return list[middle];
    }
    return null;
}
```

شکل (۴-۶) پیاده‌سازی متد FindElementsWithTid.

یکی از مهم‌ترین قسمت‌های پیاده‌سازی الگوریتم TKOBase، افزایش تدریجی میزان آستانه سودمندی است. افزایش این آستانه در متد WriteOut صورت می‌پذیرد. این متد، ابتدا مجموعه آیتم سودمند کاندید شناسایی شده را به لیست نهایی اضافه کرده و در صورت موجود بودن بیشتر از K مجموعه آیتم در این لیست، اقدام به حذف مجموعه آیتم های با سود کمتر و افزایش میزان آستانه سودمندی می‌نماید. در شکل (۴-۷) پیاده سازی این متد نشان داده شده است.

```
2 references
private void WriteOut(int[] prefix, int item, long utility)
{
    Itemset itemset = new Itemset(prefix, item, utility);
    kItemsets.Add(itemset);
    if (kItemsets.Count > k)
    {
        if (utility > this.minutility)
        {
            Itemset lower;
            do
            {
                lower = kItemsets.First();
                if (lower == null)
                {
                    break; // / IMPORTANT
                }
                kItemsets.Remove(lower);
            } while (kItemsets.Count > k);
            this.minutility = kItemsets.First().utility;
            Console.WriteLine(this.minutility);
        }
    }
}
```

شکل (۴-۷) پیاده‌سازی متد WriteOut.

برای کاوش مجموعه آیتم‌های سودمند و نیز محدودسازی فضای جست و جو، از یک متد بازگشتی به نام SearchTKOBase استفاده می‌شود که پیاده سازی آن در شکل (۴-۸) نشان داده شده است. در این پیاده‌سازی، از ساده‌ترین رویکرد ممکن که همان بررسی شرط sum(iutil) + sum(rutil) > min_util است، برای هرس و محدودسازی فضای جست و جو استفاده شده است.

```
1 reference
private void SearchTKOBase(int[] prefix, UtilityList pUl, List<UtilityList> uLs)
{

    for (int i = 0; i < uLs.Count; i++)
    {
        UtilityList X = uLs[i];
        if (X.sumIutils >= minutility)
        {
            WriteOut(prefix, X.item, X.sumIutils);
        }
        if (X.sumRutils + X.sumIutils >= minutility)
        {
            List<UtilityList> exULs = new List<UtilityList>();
            for (int j = i + 1; j < uLs.Count; j++)
            {
                UtilityList Y = uLs[j];
                exULs.Add(Construct(pUl, X, Y));
            }
            int[] newPrefix = new int[prefix.Length + 1];
            Array.Copy(prefix, 0, newPrefix, 0, prefix.Length);
            newPrefix[prefix.Length] = X.item;
            SearchTKOBase(newPrefix, X, exULs);
        }

    }
}
```

شکل (۴-۸) پیاده‌سازی متد Search در الگوریتم TKOBase.

اجرای الگوریتم TKOBase روی یک مجموعه داده، با فراخوانی متد RunTKOBaseAlgorithm صورت می‌پذیرد. این متد پس از دو بار جست و جوی پایگاه داده، و با فراخوانی متد SearchTKOBase، سودمندترین K مجموعه آیتم را شناسایی می‌نماید. می‌توان به منظور افزایش سرعت اجرای الگوریتم، داده‌های تراکنشی را پس از اولین جست و جوی پایگاه داده، در حافظه RAM ذخیره کرد. البته باید به این نکته توجه داشت که این مورد ممکن است اجرای الگوریتم را با کمبود حافظه مواجه سازد. در شکل (۴-۹) پیاده‌سازی این متد نشان داده شده است.

```
public void RunTKOBaseAlgorithm(String input, String output, int k)
{
    Stopwatch sw = new Stopwatch();
    sw.Start();
    this.minutility = 1;
    this.k = k;
    this.kItemsets = new List<Itemset>();
    string[] lines;

    try
    {
        lines = File.ReadAllLines(input);
    }
    catch (Exception)
    {
        Console.WriteLine("Something went wrong while reading from file."); return;
    }
    foreach (var line in lines)
    {
        if (string.IsNullOrWhiteSpace(line) || line.StartsWith('#') || line.StartsWith('%') || line.StartsWith('@'))  continue;
        var splits = line.Split(":");
        var items = splits[0].Split(" ");
        int transactionUtility = int.Parse(splits[1]);
        for (int i = 0; i < items.Length; i++)
        {
            int item = int.Parse(items[i]);
            bool containsItem = mapItemToTWU.ContainsKey(item);
            if (containsItem)
                mapItemToTWU[item] = mapItemToTWU[item] + transactionUtility;
            else
                mapItemToTWU.TryAdd(item, transactionUtility);
        }
    }
    List<UtilityList> listItems = new List<UtilityList>();
    Dictionary<int, UtilityList> mapItemToUtilityList = new Dictionary<int, UtilityList>();
    foreach (int item in mapItemToTWU.Keys)
    {
        UtilityList uList = new UtilityList(item);
        listItems.Add(uList);
        mapItemToUtilityList.Add(item, uList);
    }

    listItems.Sort((o1, o2) =>...);
    int tid = 0;
    foreach (var line in lines)...
    SearchTKOBase(Array.Empty<int>(), null, listItems);
    sw.Stop();
    totalTime = sw.ElapsedMilliseconds / 1000;
}
```

شکل (۴-۹) پیاده‌سازی متد RunTKOBaseAlgorithm.

۴-۴- پیاده‌سازی الگوریتم TKO

همان طورکه در فصل پیش عنوان شد، یکی از رویکردهایی که می‌توان با بهره‌گیری از آن، سریع‌تر و بهینه‌تر میزان آستانه سودمندی را افزایش داد و به بازدهی بهتری نسبت به الگوریتم TKOBase دست یافت، رویکرد RUZ می‌باشد. این رویکرد تنها با یک تغییر ساده و با کمک عناصر Z موجود در یک لیست سودمندی قابل اعمال

است. برای اعمال این رویکرد، لازم است تا متد مورد استفاده در الگوریتم TKOBase برای محدودسازی فضای جست و جو، به گونه نشان داده شده در شکل (۴-۲۰) تغییر یابد.

```
2 references
private void SearchTKO_RUZ(int[] prefix, UtilityList pUl, List<UtilityList> uLs)
{
    for (int i = 0; i < uLs.Count; i++)
    {
        UtilityList X = uLs[i];
        if (X.sumIutils >= minutility)
            WriteOut(prefix, X.item, X.sumIutils);
        var zElementsIutilSummation = X.elements.Where(e => e.rutils == 0).Sum(x => x.iutils);
        if (zElementsIutilSummation + X.sumRutils >= minutility)
        {
            List<UtilityList> exULs = new List<UtilityList>();
            for (int j = i + 1; j < uLs.Count; j++)
            {
                UtilityList Y = uLs[j];
                exULs.Add(Construct(pUl, X, Y));
            }
            int[] newPrefix = new int[prefix.Length + 1];
            Array.Copy(prefix, 0, newPrefix, 0, prefix.Length);
            newPrefix[prefix.Length] = X.item;
            SearchTKO_RUZ(newPrefix, X, exULs);
        }

    }
}
```

شکل (۴-۱۰) پیاده‌سازی متد Search در الگوریتم TKO و بهره‌گیری از رویکرد RUZ.

۴-۵- صحت‌سنجی پیاده‌سازی

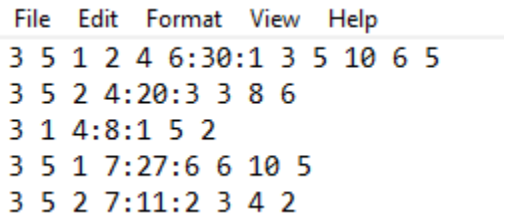
پیش از آغاز آزمون روی مجموعه داده‌ها، نتایج پیاده‌سازی ارائه‌شده، با نمونه‌های داده‌شده توسط نویسنده [۶] که در [۱۰] آورده شده است مورد مقایسه قرار گرفته است. داده ورودی، یک جدول شامل ۵ تراکنش و ۷ آیتم بوده و K برابر با ۸ در نظر گرفته شده است.

جدول (۴-۱) اطلاعات مجموعه داده‌های مورد آزمایش

	آیتم ها	سود تراکنش	سود هر آیتم در تراکنش
T۱	۳۵۱۲۴۶	۳۰	۱۳۵۱۰۶۵
T۲	۳۵۲۴	۲۰	۳۳۸۶
T۳	۳۱۴	۱	۱۵۲

T۴	۳ ۵ ۱ ۷	۲۷	۶ ۶ ۱۰ ۵
T۵	۳ ۵ ۲ ۷	۱۱	۲ ۳ ۴ ۲

نمونه جدول بالا در قالب یک فایل متنی به صورت شکل (۴-۱۱) به عنوان ورودی به برنامه داده می‌شود.



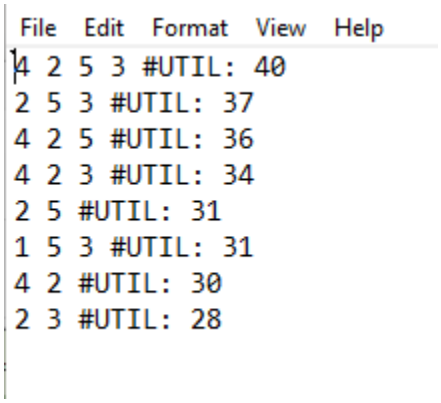
شکل (۴-۱۱) نمونه فایل متنی ورودی.

مطابق [۱۰] خروجی این مجموعه داده باید مطابق جدول زیر باشد:

جدول (۴-۲) خروجی حاصل از اجرای الگوریتم

سود	مجموعه آیتم ها
۳۰	{ ۲ ۴}
۳۱	{ ۲ ۵}
۳۱	{ ۱ ۳ ۵}
۳۴	{ ۲ ۳ ۴}
۳۷	{ ۲ ۳ ۵}
۳۶	{ ۲ ۴ ۵}
۴۰	{ ۲ ۳ ۴ ۵}
۳۰	{۱ ۲ ۳ ۴ ۵ ۶ }

خروجی برنامه پیاده‌سازی شده در قالب یک فایل متنی در شکل (۴-۱۲) آورده شده است که تطابق دو خروجی و در نتیجه صحت عملکرد الگوریتم پیاده‌سازی شده را نشان می‌دهد.



شکل (۴-۱۲) نمونه فایل متنی خروجی.

۴-۶- اجرای الگوریتم TKOBase روی مجموعه داده‌های آزمایشی

تمامی اجراها بر روی سیستمی با مشخصات زیر انجام شده است:

CPU: Core i۵ ۷۴۰۰ ۳۰۰۰ GHZ

۱ CPU ٤ logical ۲ physicals

Os: windows 10

.NET SDK: ۵.۰.۱۴

برای یکسان بودن شرایط آزمون، در تمامی آزمون‌ها مقدار k برابر ۲ در نظر گرفته شده است.

۴-۶-۱- مجموعه داده‌های مورد استفاده

تمامی مجموعه داده‌ها از [۱۲] گرفته شده‌اند. از میان مجموعه داده‌های موجود، ۳ مجموعه داده Retail، به عنوان نمونه ای از مجموعه داده‌های خلوت، Chainstore به عنوان نمونه ای از مجموعه داده‌های فشرده با سود تعیین شده واقعی و Accidents به عنوان نمونه ای از مجموعه داده‌های فشرده برای آزمون انتخاب شده‌اند. مشخصات هر کدام از مجموعه داده‌ها در جدول زیر آورده شده است.

جدول (۴-۳) اطلاعات مجموعه داده‌های مورد آزمایش

نام مجموعه داده	تعداد تراکنش‌ها	تعداد آیتم‌ها (I)	میانگین تعداد آیتم در هر تراکنش (A)	تراکم (A/I)	حجم	توضیحات
Chainstore	۱۱۱۲۹۴۹	۴۶۰۸۶	۷.۲۳	۰.۰۲٪	۸۰ KB	تراکنش‌های مشتریان در یکی از شعبه‌های یک فروشگاه خواروبار در کالیفرنیا، آمریکا.
Reatil	۸۸۱۶۲	۱۶۴۷۰	۱۰	۰.۰۶ %	۶ KB	تراکنش‌های فروشگاهی ناشناس در بلژیک [۱۳]
Accidents	۳۴۰۱۸۳	۴۶۸	۳۳.۸	۷.۲۲٪	۶۴ KB	تصادفات ترافیکی ناشناس شده [۱۳]

۴-۶-۲- اجرای الگوریتم TKOBase و TKO روی مجموعه داده Chinstore

قطعه کدهای نشان داده شده در شکل‌های (۴-۱۳) و (۴-۱۴) به ترتیب منجر به کاوش سودمندترین ۲ مجموعه آیتم ممکن روی داده‌های موجود در مجموعه داده Chainstore با الگوریتم TKOBase و الگوریتم TKO می‌گردند. برای اجرای این الگوریتم لازم است تا داده‌های ورودی در یک فایل متنی و با قالبی به شکل قالب نشان داده شده در شکل (۴-۱۵) به آن داده شود. هر داده موجود در فایل متنی، دارای سه قسمت بوده که به ترتیب و از راست به چپ، بیانگر آیتم‌های موجود در تراکنش، سود آن تراکنش و سود حاصل از حضور هر یک از آیتم‌های موجود در آن تراکنش است. نتایج حاصل از اجرا، که سودمندترین مجموعه آیتم‌های شناسایی شده و نیز میزان سود آن‌ها است، در یک فایل متنی نوشته می‌شود که نمونه آن در شکل (۴-۱۷) برای الگوریتم TKOBase و در شکل (۴-۱۸) برای الگوریتم TKO نشان داده شده است. نتایج مربوط به مصرف منابع به ازای هر دو روش نیز در شکل‌های (۴-۱۹)، (۴-۲۰)، (۴-۲۱) و (۴-۲۲) نشان داده شده است. نحوه اجرای الگوریتم برای تمام آزمون‌های بعدی به همین شکل بوده و از این رو، از اشاره به آن در قسمت‌های بعد صرف نظر می‌شود.

```
class Program
{

    static void Main(string[] args)
    {
        var  algorithm = new AlgoTkoBasic();
        string input = @"C:\Users\barman\Desktop\Codes\chainstore.txt";
        string output = @"C:\Users\barman\Desktop\Codes\result.txt";
        algorithm.RunTKOBaseAlgorithm(input, output, 2);
        algorithm.WriteResultTofile(output);
        algorithm.PrintStats();


        var summary = BenchmarkRunner.Run<TKO_AlgorithmBenchmarks>();
    }
}
```

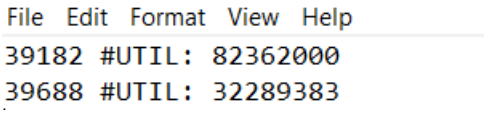
شکل (۴-۱۳) قطعه کد اجرای الگوریتم TKOBase.

```
class Program
{

    static void Main(string[] args)
    {
        var  algorithm = new AlgoTkoBasic();
        string input = @"C:\Users\barman\Desktop\Codes\chainstore.txt";
        string output = @"C:\Users\barman\Desktop\Codes\result_tko.txt";
        algorithm.RunTKO_RUZAlgorithm(input, output, 2);
        algorithm.WriteResultTofile(output);
        algorithm.PrintStats();


        var summary = BenchmarkRunner.Run<TKO_AlgorithmBenchmarks>();
    }
}
```

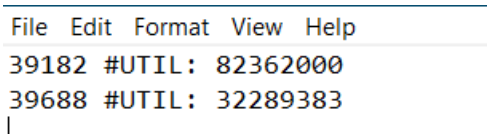
شکل (۴-۱۴) قطعه کد اجرای الگوریتم TKO.

```
39684                    :   50    :50
39610                    :   222   :222
8275 16890 39388 45968   :   688   :198 139 51 300
39307 44006              :   246   :50 196
16369:476:476
```

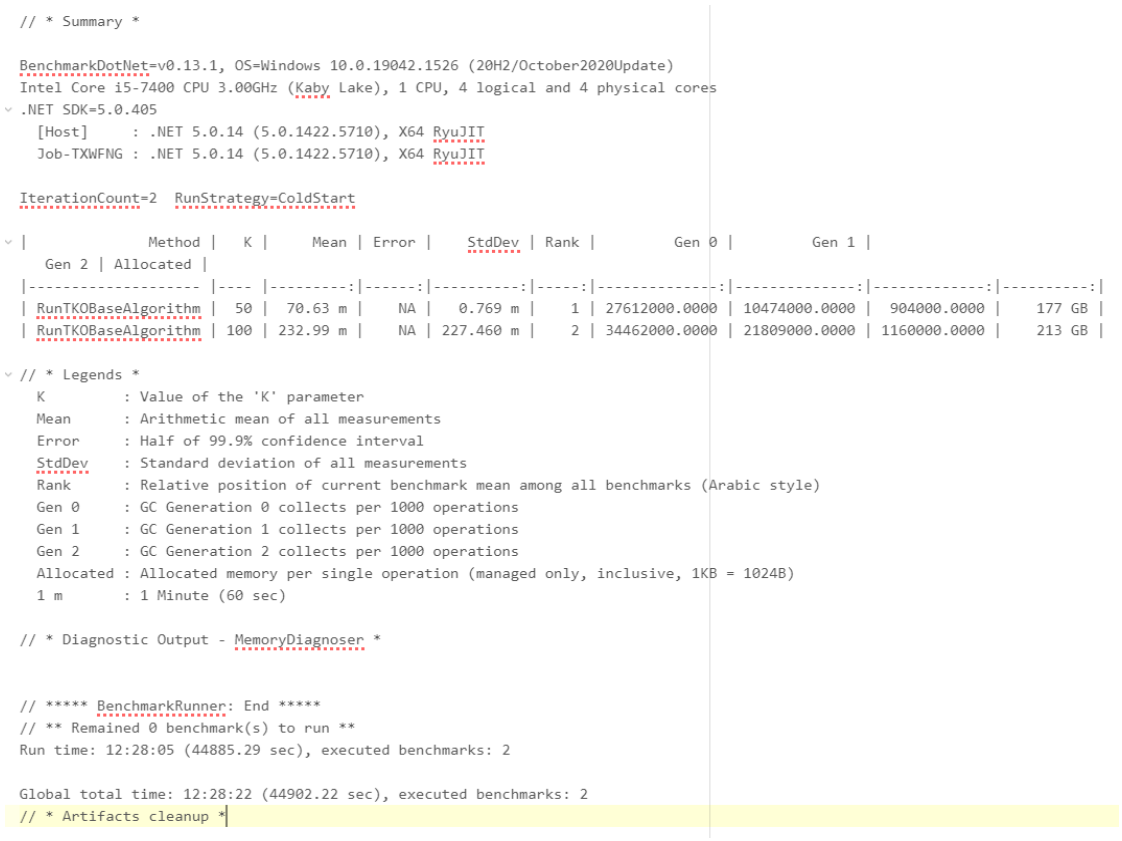
شکل (۴-۱۵) قالب صحیح داده‌های ورودی برای اجرای الگوریتم TKOBase.



شکل (۴-۱۶) نمونه خروجی از فایل result.txt.



شکل (۴-۱۷) نمونه خروجی از فایل result_tko.txt.



```
// * Summary *

BenchmarkDotNet=v0.13.1, OS=Windows 10.0.19042.1526 (20H2/October2020Update)
Intel Core i5-7400 CPU 3.00GHz (Kaby Lake), 1 CPU, 4 logical and 4 physical cores
.NET SDK=5.0.405
  [Host]     : .NET 5.0.14 (5.0.1422.5710), X64 RyuJIT
  Job-TXWFNG : .NET 5.0.14 (5.0.1422.5710), X64 RyuJIT

IterationCount=2  RunStrategy=ColdStart

|             Method |   K |     Mean | Error |    StdDev | Rank |         Gen 0 |         Gen 1 |
   Gen 2 | Allocated |
|------------------- |---- |---------:|------:|----------:|-----:|--------------:|--------------:|-------------:|----------:|
| RunTKOBaseAlgorithm |  50 |  70.63 m |    NA |   0.769 m |    1 | 27612000.0000 | 10474000.0000 |  904000.0000 |    177 GB |
| RunTKOBaseAlgorithm | 100 | 232.99 m |    NA | 227.460 m |    2 | 34462000.0000 | 21809000.0000 | 1160000.0000 |    213 GB |

// * Legends *
  K          : Value of the 'K' parameter
  Mean       : Arithmetic mean of all measurements
  Error      : Half of 99.9% confidence interval
  StdDev     : Standard deviation of all measurements
  Rank       : Relative position of current benchmark mean among all benchmarks (Arabic style)
  Gen 0      : GC Generation 0 collects per 1000 operations
  Gen 1      : GC Generation 1 collects per 1000 operations
  Gen 2      : GC Generation 2 collects per 1000 operations
  Allocated  : Allocated memory per single operation (managed only, inclusive, 1KB = 1024B)
  1 m        : 1 Minute (60 sec)

// * Diagnostic Output - MemoryDiagnoser *


// ***** BenchmarkRunner: End *****
// ** Remained 0 benchmark(s) to run **
Run time: 12:28:05 (44885.29 sec), executed benchmarks: 2

Global total time: 12:28:22 (44902.22 sec), executed benchmarks: 2
// * Artifacts cleanup *
```

شکل (۴-۱۸) گزارش استفاده از منابع اجرای الگوریتم TKOBase روی مجموعه داده Chainstore

```
// * Summary *

BenchmarkDotNet=v0.13.1, OS=Windows 10.0.19042.1526 (20H2/October2020Update)
Intel Core i5-7400 CPU 3.00GHz (Kaby Lake), 1 CPU, 4 logical and 4 physical cores
.NET SDK=5.0.405
  [Host]     : .NET 5.0.14 (5.0.1422.5710), X64 RyuJIT
  Job-WDDCOY : .NET 5.0.14 (5.0.1422.5710), X64 RyuJIT

IterationCount=2  RunStrategy=ColdStart

|              Method |   K |    Mean | Error |  StdDev | Rank |         Gen 0 |        Gen 1 |       Gen 2 | Allocated |
|-------------------- |---- |--------:|------:|--------:|-----:|--------------:|-------------:|------------:|----------:|
| RunTKO_RUZAlgorithm | 100 | 60.41 m |    NA | 0.889 m |    1 | 18770000.0000 | 7066000.0000 | 586000.0000 |    126 GB |
| RunTKO_RUZAlgorithm |  50 | 60.54 m |    NA | 0.918 m |    1 | 18786000.0000 | 7046000.0000 | 609000.0000 |    126 GB |

// * Legends *
  K          : Value of the 'K' parameter
  Mean       : Arithmetic mean of all measurements
  Error      : Half of 99.9% confidence interval
  StdDev     : Standard deviation of all measurements
  Rank       : Relative position of current benchmark mean among all benchmarks (Arabic style)
  Gen 0      : GC Generation 0 collects per 1000 operations
  Gen 1      : GC Generation 1 collects per 1000 operations
  Gen 2      : GC Generation 2 collects per 1000 operations
  Allocated  : Allocated memory per single operation (managed only, inclusive, 1KB = 1024B)
  1 m        : 1 Minute (60 sec)

// * Diagnostic Output - MemoryDiagnoser *


// ***** BenchmarkRunner: End *****
// ** Remained 0 benchmark(s) to run **
Run time: 06:01:13 (21673.11 sec), executed benchmarks: 2

Global total time: 06:01:30 (21690.14 sec), executed benchmarks: 2
// * Artifacts cleanup *
```

شکل (۴-۱۹) گزارش استفاده از منابع اجرای الگوریتم TKO روی مجموعه داده Chainstore

```
=============  TKO-BASIC - v.2.28 =============
 High-utility itemsets count : 2
 Total time ~ 4315 s
=====================================================
WorkloadResult   1: 1 op, 23629552995200.00 ns, 393.8259 m/op
WorkloadResult   2: 1 op, 4328897522000.00 ns, 72.1483 m/op
GC:  34462 21809 1160 229183850736 1
Threading:  2 0 1

// AfterAll
// Benchmark Process 15196 has exited with code 0.

Mean = 232.987 m, StdErr = 160.839 m (69.03%), N = 2, StdDev = 227.460 m
Min = 72.148 m, Q1 = 152.568 m, Median = 232.987 m, Q3 = 313.406 m, Max = 393.8
IQR = 160.839 m, LowerFence = -88.691 m, UpperFence = 554.665 m
ConfidenceInterval = [NaN m; NaN m] (CI 99.9%), Margin = NaN m (NaN% of Mean)
Skewness = 0, Kurtosis = 0.25, MValue = 2
```

شکل (۴-۲۰) گزارش استفاده از منابع اجرای الگوریتم TKOBase روی مجموعه داده Chainstore

```
=============  TKO-BASIC - v.2.28 =============
 High-utility itemsets count : 2
 Total time ~ 3559 s
================================================
WorkloadResult   1: 1 op, 3662511949500.00 ns, 61.0419 m/op
WorkloadResult   2: 1 op, 3587041702700.00 ns, 59.7840 m/op
GC:  18770 7066 586 134791266560 1
Threading:  2 0 1

// AfterAll
// Benchmark Process 15140 has exited with code 0.

Mean = 60.413 m, StdErr = 0.629 m (1.04%), N = 2, StdDev = 0.889 m
Min = 59.784 m, Q1 = 60.098 m, Median = 60.413 m, Q3 = 60.727 m, Max = 61.042 m
IQR = 0.629 m, LowerFence = 59.155 m, UpperFence = 61.671 m
ConfidenceInterval = [NaN m; NaN m] (CI 99.9%), Margin = NaN m (NaN% of Mean)
Skewness = 0, Kurtosis = 0.25, MValue = 2

// ***** BenchmarkRunner: Finish  *****
```

شکل (۴-۲۱) گزارش استفاده از منابع اجرای الگوریتم TKO روی مجموعه داده Chainstore

۴-۶-۳- اجرای الگوریتم TKOBase و TKO روی مجموعه داده Retail

مشابه بخش قبل مجموعه داده Retail با قالب صحیح به برنامه داده می‌شود. نتایج حاصل از اجرا، که سودمندترین مجموعه آیتم‌های شناسایی شده و نیز میزان سود آن‌ها است، در یک فایل متنی که قسمتی از آن در شکل (۴-۲۲) نشان داده شده است، نوشته می‌شود. نتایج مصرف منابع در طول اجرای برنامه به ازای هر روش در شکل های (۴-۲۳)، (۴-۲۴)، (۴-۲۵) و (۴-۲۶) آورده شده است.

```
File  Edit  Format  View  Help
49 40 #UTIL: 481021
49 #UTIL: 463274
```

شکل (۴-۲۲) نمونه خروجی از فایل result.txt.

```
// * Summary *

BenchmarkDotNet=v0.13.1, OS=Windows 10.0.19042.1526 (20H2/October2020Update)
Intel Core i5-7400 CPU 3.00GHz (Kaby Lake), 1 CPU, 4 logical and 4 physical cores
.NET SDK=5.0.405
  [Host]     : .NET 5.0.14 (5.0.1422.5710), X64 RyuJIT
  Job-VUSKQT : .NET 5.0.14 (5.0.1422.5710), X64 RyuJIT

IterationCount=2  RunStrategy=ColdStart

|              Method |   K |     Mean | Error | StdDev | Rank |        Gen 0 |        Gen 1 |       Gen
2 | Allocated |
|-------------------- |---- |---------:|------:|-------:|-----:|-------------:|-------------:|------------:|----------:|
| RunTKOBaseAlgorithm |  50 | 189.2 s |    NA | 0.50 s |    1 | 6088000.0000 | 2403000.0000 | 469000.0000 |     32 GB |
| RunTKOBaseAlgorithm | 100 | 198.7 s |    NA | 0.56 s |    2 | 7487000.0000 | 2874000.0000 | 518000.0000 |     40 GB |

// * Legends *
  K          : Value of the 'K' parameter
  Mean       : Arithmetic mean of all measurements
  Error      : Half of 99.9% confidence interval
  StdDev     : Standard deviation of all measurements
  Rank       : Relative position of current benchmark mean among all benchmarks (Arabic style)
//  Remained 0 benchmark(s) to run
Run time: 00:19:30 (1170.38 sec), executed benchmarks: 2

Global total time: 00:19:47 (1187.28 sec), executed benchmarks: 2
```

شکل (۴-۲۳) گزارش استفاده از منابع اجرای الگوریتم TKOBase روی مجموعه داده Retail.

```
// * Summary *

BenchmarkDotNet=v0.13.1, OS=Windows 10.0.19042.1526 (20H2/October2020Update)
Intel Core i5-7400 CPU 3.00GHz (Kaby Lake), 1 CPU, 4 logical and 4 physical cores
.NET SDK=5.0.405
  [Host]     : .NET 5.0.14 (5.0.1422.5710), X64 RyuJIT
  Job-XQUKZP : .NET 5.0.14 (5.0.1422.5710), X64 RyuJIT

IterationCount=2  RunStrategy=ColdStart

|              Method |   K |     Mean | Error | StdDev | Rank |        Gen 0 |        Gen 1 |       Gen 2 | Allocated |
|-------------------- |---- |---------:|------:|-------:|-----:|-------------:|-------------:|------------:|----------:|
| RunTKO_RUZAlgorithm |  50 | 164.8 s |    NA | 0.01 s |    1 | 3477000.0000 | 1536000.0000 | 326000.0000 |     18 GB |
| RunTKO_RUZAlgorithm | 100 | 165.3 s |    NA | 0.30 s |    1 | 3466000.0000 | 1514000.0000 | 325000.0000 |     18 GB |

// * Legends *
  K          : Value of the 'K' parameter
  Mean       : Arithmetic mean of all measurements
  Error      : Half of 99.9% confidence interval
  StdDev     : Standard deviation of all measurements
  Rank       : Relative position of current benchmark mean among all benchmarks (Arabic style)
  Gen 0      : GC Generation 0 collects per 1000 operations
  Gen 1      : GC Generation 1 collects per 1000 operations
  Gen 2      : GC Generation 2 collects per 1000 operations
  Allocated  : Allocated memory per single operation (managed only, inclusive, 1KB = 1024B)
  1 s        : 1 Second (1 sec)

// * Diagnostic Output - MemoryDiagnoser *


// ***** BenchmarkRunner: End *****
// ** Remained 0 benchmark(s) to run **
Run time: 00:16:31 (991.16 sec), executed benchmarks: 2

Global total time: 00:16:48 (1008.5 sec), executed benchmarks: 2
```

شکل (۴-۲۴) گزارش استفاده از منابع اجرای الگوریتم TKO روی مجموعه داده Retail.

```
============  TKO-BASIC - v.2.28 =============
 High-utility itemsets count : 2
 Total time ~ 199 s
=================================================
WorkloadResult   1: 1 op, 199122581000.00 ns, 199.1226 s/op
WorkloadResult   2: 1 op, 198335083400.00 ns, 198.3351 s/op
GC:  7487 2874 518 43065276216 1
Threading:  2 0 1

// AfterAll
// Benchmark Process 6704 has exited with code 0.

Mean = 198.729 s, StdErr = 0.394 s (0.20%), N = 2, StdDev = 0.557 s
Min = 198.335 s, Q1 = 198.532 s, Median = 198.729 s, Q3 = 198.926 s, Max = 199.123 s
IQR = 0.394 s, LowerFence = 197.941 s, UpperFence = 199.516 s
ConfidenceInterval = [NaN s; NaN s] (CI 99.9%), Margin = NaN s (NaN% of Mean)
Skewness = 0, Kurtosis = 0.25, MValue = 2

// ***** BenchmarkRunner: Finish  *****
```

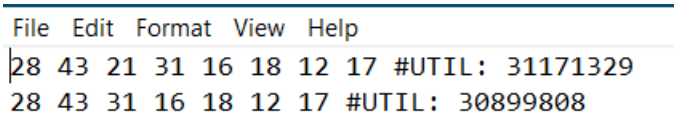
شکل (۴-۲۵) گزارش استفاده از منابع اجرای الگوریتم TKOBase روی مجموعه داده Retail.

```
============  TKO-BASIC - v.2.28 =============
 High-utility itemsets count : 2
 Total time ~ 165 s
=================================================
WorkloadResult   1: 1 op, 165470167500.00 ns, 165.4702 s/op
WorkloadResult   2: 1 op, 165044140000.00 ns, 165.0441 s/op
GC:  3466 1514 325 19598329968 1
Threading:  2 0 1

// AfterAll
// Benchmark Process 12404 has exited with code 0.

Mean = 165.257 s, StdErr = 0.213 s (0.13%), N = 2, StdDev = 0.301 s
Min = 165.044 s, Q1 = 165.151 s, Median = 165.257 s, Q3 = 165.364 s, Max = 165.470 s
IQR = 0.213 s, LowerFence = 164.831 s, UpperFence = 165.683 s
ConfidenceInterval = [NaN s; NaN s] (CI 99.9%), Margin = NaN s (NaN% of Mean)
Skewness = 0, Kurtosis = 0.25, MValue = 2

// ***** BenchmarkRunner: Finish  *****
```

شکل (۴-۲۶) گزارش استفاده از منابع اجرای الگوریتم TKO روی مجموعه داده Retail.

۴-۶-۴- اجرای الگوریتم TKOBase و TKO روی مجموعه داده Accidents

مشابه قسمت قبل مجموعه داده Accidents با قالب صحیح به برنامه داده می‌شود. نتایج حاصل از اجرا، که سودمندترین مجموعه آیتم‌های شناسایی شده و نیز میزان سود آن‌ها است، در یک فایل متنی که قسمتی از آن در شکل (۴-۲۷) نشان داده شده است، نوشته می‌شود. نتایج مصرف منابع در طول اجرای برنامه به ازای هر روش در شکل‌های (۴-۲۹)، (۴-۲۹)، (۴-۳۰) و (۴-۳۱) آورده شده است.



شکل (۴-۲۷) نمونه خروجی از فایل result.txt.

```
// * Summary *

BenchmarkDotNet=v0.13.1, OS=Windows 10.0.19042.1526 (20H2/October2020Update)
Intel Core i5-7400 CPU 3.00GHz (Kaby Lake), 1 CPU, 4 logical and 4 physical cores
.NET SDK=5.0.405
  [Host]     : .NET 5.0.14 (5.0.1422.5710), X64 RyuJIT
  Job-WVBHRT : .NET 5.0.14 (5.0.1422.5710), X64 RyuJIT

IterationCount=2  RunStrategy=ColdStart

|              Method |   K |    Mean | Error |  StdDev | Rank |         Gen 0 |         Gen 1 |       Gen 2 | Allocated |
|-------------------- |---- |--------:|------:|--------:|-----:|--------------:|--------------:|------------:|----------:|
| RunTKOBaseAlgorithm |  50 | 35.80 m |    NA | 0.065 m |    1 | 51609000.0000 | 18320000.0000 | 889000.0000 |    421 GB |
| RunTKOBaseAlgorithm | 100 | 35.99 m |    NA | 0.084 m |    1 | 51605000.0000 | 17983000.0000 | 886000.0000 |    421 GB |

// * Legends *
  K          : Value of the 'K' parameter
  Mean       : Arithmetic mean of all measurements
  Error      : Half of 99.9% confidence interval
  StdDev     : Standard deviation of all measurements
  Rank       : Relative position of current benchmark mean among all benchmarks (Arabic style)
  Gen 0      : GC Generation 0 collects per 1000 operations
  Gen 1      : GC Generation 1 collects per 1000 operations
  Gen 2      : GC Generation 2 collects per 1000 operations
  Allocated  : Allocated memory per single operation (managed only, inclusive, 1KB = 1024B)
  1 m        : 1 Minute (60 sec)

// * Diagnostic Output - MemoryDiagnoser *


// ***** BenchmarkRunner: End *****
// ** Remained 0 benchmark(s) to run **
Run time: 03:35:32 (12932.45 sec), executed benchmarks: 2

Global total time: 03:35:49 (12949.49 sec), executed benchmarks: 2
```

شکل (۴-۲۸) گزارش استفاده از منابع اجرای الگوریتم TKOBase روی مجموعه داده Accidents.

```
// * Summary *

BenchmarkDotNet=v0.13.1, OS=Windows 10.0.19042.1526 (20H2/October2020Update)
Intel Core i5-7400 CPU 3.00GHz (Kaby Lake), 1 CPU, 4 logical and 4 physical cores
.NET SDK=5.0.405
  [Host]     : .NET 5.0.14 (5.0.1422.5710), X64 RyuJIT
  Job-ARJZXR : .NET 5.0.14 (5.0.1422.5710), X64 RyuJIT

IterationCount=2  RunStrategy=ColdStart

|              Method |   K |     Mean | Error |  StdDev | Rank |          Gen 0 |         Gen 1 |       Gen 2 | Allocated |
|-------------------- |---- |---------:|------:|--------:|-----:|---------------:|--------------:|------------:|----------:|
| RunTKO_RUZAlgorithm | 100 | 35.79 m |    NA | 0.095 m |    1 | 51667000.0000 | 20082000.0000 | 892000.0000 |    421 GB |
| RunTKO_RUZAlgorithm |  50 | 36.01 m |    NA | 0.040 m |    1 | 51629000.0000 | 18351000.0000 | 904000.0000 |    421 GB |

// * Legends *
  K          : Value of the 'K' parameter
  Mean       : Arithmetic mean of all measurements
  Error      : Half of 99.9% confidence interval
  StdDev     : Standard deviation of all measurements
  Rank       : Relative position of current benchmark mean among all benchmarks (Arabic style)
  Gen 0      : GC Generation 0 collects per 1000 operations
  Gen 1      : GC Generation 1 collects per 1000 operations
  Gen 2      : GC Generation 2 collects per 1000 operations
  Allocated  : Allocated memory per single operation (managed only, inclusive, 1KB = 1024B)
  1 m        : 1 Minute (60 sec)

// * Diagnostic Output - MemoryDiagnoser *


// ***** BenchmarkRunner: End *****
// ** Remained 0 benchmark(s) to run **
Run time: 03:35:36 (12936.47 sec), executed benchmarks: 2

Global total time: 03:35:53 (12953.58 sec), executed benchmarks: 2
// * Artifacts cleanup *
```

شکل (۴-۲۹) گزارش استفاده از منابع اجرای الگوریتم TKO روی مجموعه داده Accidents.

```
=============  TKO-BASIC - v.2.28 =============
 High-utility itemsets count : 2
 Total time ~ 2156 s
================================================
WorkloadResult   1: 1 op, 2162704681200.00 ns, 36.0451 m/op
WorkloadResult   2: 1 op, 2155615926100.00 ns, 35.9269 m/op
GC:  51605 17983 886 452556679432 1
Threading:  3 0 1

// AfterAll
// Benchmark Process 21336 has exited with code 0.

Mean = 35.986 m, StdErr = 0.059 m (0.16%), N = 2, StdDev = 0.084 m
Min = 35.927 m, Q1 = 35.956 m, Median = 35.986 m, Q3 = 36.016 m, Max = 36.045 m
IQR = 0.059 m, LowerFence = 35.868 m, UpperFence = 36.104 m
ConfidenceInterval = [NaN m; NaN m] (CI 99.9%), Margin = NaN m (NaN% of Mean)
Skewness = 0, Kurtosis = 0.25, MValue = 2

// ***** BenchmarkRunner: Finish  *****
```

شکل (۴-۳۰) گزارش استفاده از منابع اجرای الگوریتم TKOBase روی مجموعه داده Accidents.

```
=============  TKO-BASIC - v.2.28 =============
 High-utility itemsets count : 2
 Total time ~ 2155 s
================================================
WorkloadResult   1: 1 op, 2143413106600.00 ns, 35.7236 m/op
WorkloadResult   2: 1 op, 2151508506300.00 ns, 35.8585 m/op
GC:  51667 20082 892 452556785016 1
Threading:  3 0 1

// AfterAll
// Benchmark Process 3920 has exited with code 0.

Mean = 35.791 m, StdErr = 0.067 m (0.19%), N = 2, StdDev = 0.095 m
Min = 35.724 m, Q1 = 35.757 m, Median = 35.791 m, Q3 = 35.825 m, Max = 35.858 m
IQR = 0.067 m, LowerFence = 35.656 m, UpperFence = 35.926 m
ConfidenceInterval = [NaN m; NaN m] (CI 99.9%), Margin = NaN m (NaN% of Mean)
Skewness = 0, Kurtosis = 0.25, MValue = 2

// ***** BenchmarkRunner: Finish  *****
```

شکل (۴-۳۱) گزارش استفاده از منابع اجرای الگوریتم TKO روی مجموعه داده Accidents.

۴-۷- مقایسه عملکرد دو الگوریتم TKOBase و TKO

مشخصات سیستم میزبان، میزان حافظه مصرفی توسط هر یک از الگوریتم‌ها، میانگین و انحراف معیار مدت زمان اجرای الگوریتم‌ها از جمله متغیرهایی بودند که در آزمون‌های انجام شده در سه قسمت قبل مورد اندازه‌گیری قرار گرفتند. همانطور که در شکل های بخش ۴-۶-۲ و ۴-۶-۳ ملاحظه می‌شود، مطابق ادعای مقاله [۶]، الگوریتم TKO با توجه به افزایش سریع‌تر آستانه سودمندی و ایجاد مجموعه آیتم‌های کاندید کمتر، عملکرد بهتری نسبت به الگوریتم TKOBase دارد.

عملکرد بهتر به معنی زمان اجرای حدود ۲۰ درصد کمتر، ۴۰ درصد مصرف حافظه کمتر و تقریبا ۴۰ درصد درگیری کمتر CPU در مجموعه داده‌های فشرده Chainstore و Retail با خروجی یکسان بوده است. اما در مجموعه داده خلوت Accidents تغییر محسوسی در متغیرهای مورد بررسی میان دو الگوریتم مشاهده نمی‌شود.

۴-۸- جمع‌بندی و نتیجه‌گیری

در این پژوهش، سعی شد از شاخه‌های بالاتر، شروع به معرفی مسئله کرده تا در نهایت به بررسی مسئله سودمندترین K مجموعه آیتم که درآن K تعداد مجموعه آیتم های سودمند مطلوب است، پرداخته شود. دو الگوریتم بهینه TKO (یک مرحله ای) و TKU (دو مرحله ای) معرفی شده و مورد بررسی قرار گرفتند. این دو الگوریتم بدون تعیین آستانه سودمندی، اقدام به ایجاد مجموعه آیتم‌های مورد نظر می‌نمایند. در نهایت الگوریتم TKO را در زبان #C پیاده‌سازی کرده و سعی کردیم عملکرد آن را روی مجموعه داده‌های مختلف با هم مقایسه نماییم. به طور کلی روش‌های یک مرحله‌ای از نظر سرعت و حافظه بهتر از روش‌های دو مرحله‌ای عمل می‌کنند. اما کماکان در صورت استفاده از لیست سودمندی، هزینه ادغام لیست‌ها یک چالش به شمار می‌آید. برای بهبود این نقیصه، از پایگاه داده‌های واکشی شده (نمایش پایگاه داده به‌صورت افقی)، لیست‌های سودمندی بافرشده و یا لیست‌های سودمندی فشرده شده استفاده می‌شود. برای کاهش فضای جست و جو نیز استفاده از راهبردهای دقیق‌تر پیشنهاد می‌گردد [۶].

۴-۹- پژوهش‌های آتی

با توجه به این‌که در طول این پژوهش، فضای آزمایش مبتنی بر مجموعه داده‌های آزمایشی مطرح در این زمینه بود، بررسی عملکرد الگوریتم روی پایگاه‌های داده واقعی مربوط به کسب و کارها می‌تواند اطلاعات مفیدی را در اختیار قرار دهد که مقدمات این موضوع نیز فراهم شده است.

در ادامه قصد بر آن است سیستم توصیه‌دهنده‌ای را توسعه داده که در آن با استفاده از نتایج بدست آمده از مجموعه آیتم‌های سودمند و ترکیب آن با روش‌های فیلترینگ مشارکتی و محتوا محور[۱][۱۳] و ارائه یک روش

[۱] collaborative and content-based filtering

جدید که ضمن در نظر گرفتن رضایت کاربران، کسب رضایت صاحبان کسب و کار را نیز دنبال کند. نمونه های مشابه این موضوع با روش های کاوش قواعد وابستگی [۱۴] وجود دارد.

همچنین با اضافه کردن امکان اتصال مستقیم سیستم به پایگاه داده و ساختن فایل ورودی از روی آن، می‌توان عملکرد سیستم را ارتقا داده و آن را کاربردی‌تر ساخت.

فصل پنجم: منابع و مراجع

[۱] پ. تقوی، کاوش مجموعه‌داده‌های سودمند، سمینار دوره کارشناسی‌ارشد دانشگاه شهید رجایی، صفحات ۲-۳۶، ۱۳۹۹.

[2] J. Han, M. Jiawei. “Data Mining: Concepts and Techniques”, Morgan Kaufmann is an imprint of Elsevier, ISBN 978-0-12-381479-1, pp. 243-263, 2011.

[۳] م. امینی، " دوره آموزش رایگان داده‌کاوی"، دانشگاه صنعتی شریف

[4] J. Qu, M. Liu, P. Fournier-Viger, “Efficient Algorithms for High Utility Itemset Mining Without Candidate Generation”, Springer Nature Switzerland, pp. ۱۳۲-۱۵۲, ۲۰۱۹

[5] J. Qu, M. Liu, “Mining High Utility Itemsets without Candidate Generation”, 21st ACM international conference on Information and knowledge management, pp. 55-65, 2012

[۶] V. S. Tseng, Ch. Wu, P. Fournier-Viger, P. S. Yu, “Efficient Algorithms for Mining Top-K High Utility Itemsets”, IEEE Transactions on Knowledge and Data Engineering, VOL. 28, NO. ۱, ۲۰۱۶

[7] V. S. Tseng, B. Shie, Ch. Wu, P. S. Yu, “Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases”, IEEE Transactions on Knowledge and Data Engineering, VOL. 25, NO. 8, pp. 1772-1775, 2012

[۸] Q. Duonga, B. Liaoa, P. Fournier-Vigerc, T. Dam, “An Efficient Algorithm for Mining the Top-k High Utility Itemsets, Using Novel Threshold Raising and Pruning Strategies”, pp. 8-12, ۲۰۱۶.

[9] K. P N, Sh. Shinde, “Mining Algorithm for Mining High Utility Itemset Using TKO with TKU”, International Journal of Innovative Research in Science, Engineering and Technology, Vol. 8, Issue 9, pp. 9335-9340, 2019

[10] https://www.philippe-fournier-viger.com/spmf/TKO.php

[11] http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

[12] Frequent Itemset Mining Dataset Repository (uantwerpen.be)

[13] F. Xue, J. Xu, K. Liu and R. Hong, , “Deep Item-based Collaborative Filtering for Top-N Recommendation”, ACM Transactions on Information Systems, 25,2018

[14] R. Smetsers, “Association rule mining for recommender systems”, Matster’s Thesis Tilburg University

Abstract

With the rapid growth of data and information, deriving meaningful relations between datasets and convert it into useful knowledge for businesses seems to be crucial. In the most of the past literatures, although they find meaningful information from data, specifically in market basket analysis, they do not consider the utility of the items. So having information about the high utility itemsets can be useful to make right decisions and give deeper insight to businesses. To meet the requirements, in this literature, we investigate some solutions to mine itemsets that have utilities more than a utility threshold based on a user specified value named K for the number of itemsets need to be found.

Between proposed methods, TKO algorithm, implemented to satisfy the purpose described. In this method which is one-phase algorithm, K as an input is taken from user and based on given K, K high utility itemset are mined as an output of program. Advantages of this method are using less resources compared to other one-phase algorithms and better functionality.

Keywords: Data mining, high utility itemsets, frequent itemset, minimum utility, HUIM

Shahid Rajaee Teacher Training University

Faculty of Computer Science

Thesis B.Sc.

Title:

Top-K High Utility Itemset Mining

By:

Ahmadreza Rostamani

Fatemeh Shiri

Supervisor:

Dr. Negin Daneshpour

Winter 2022