Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

ЗВІТ

про виконання практикуму № 5

«Побудова асоціативних правил за алгоритмами Apriori та FP-росту»

з дисципліни «Інтелектуальний аналіз даних»

Виконав:

Студент III курсу
Групи КА-76
Хомич О. Р.
Варіант № 18

Перевірила:
Недашківська Н. І.

Київ – 2020

Мета роботи

Отримати навички роботи з бібліотекою **Mlextend** Python, що надає можливість працювати з такими алгоритмами, як *Apriori* та *FP-growth*, що відсутні у Scikit-learn, навчитися будувати асоціативні правила за допомогою вищенаведених алгоритмів, що були опрацьовані на лекційних заняттях. Окрім цього, засвоїти поняття метрик, що використовуються для визначення надійності отриманих правил, та навчитися підбирати параметри алгоритму, щоб отримувати бажані правила.

Задача

За заданими транзакціями провести **аналіз ринкових кошиків**, тобто визначити набори товарів, які часто купуються разом, або ніколи не купляються разом. Для виконання цього аналізу використаємо метод побудови **асоціативних правил**, які допомагають віднайти логічні закономірності у великих масивах даних.

Початкові дані:

`Groceries Market Basket Dataset` (доступ за посиланням: https://www.kaggle.com/irfanasrullah/groceries).

Набір даних містить 9835 транзакцій та 169 унікальних елементів (товарів або категорій товарів). Це дані про транзакції з переліком товарів, які купують клієнти. З лівого боку розміщено кількість товарів у кошику, тоді колонки *Item 1, 2, 3* тощо містять самі товари, кількість яких відповідає числу зліва.

Хід виконання роботи

1. Завантажити початкові дані та виконати їх первинну обробку

Після прочитання файлу CSV ми отримаємо наступний об'єкт *DataFrame* бібліотеки *pandas*, перші 5 транзакцій якого зображені на рисунку 1:

	Item(s)	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7	Item 8	Item 9	...	Item 23	Item 24	Item 25	Item 26	Item 27	Item 28	Item 29	Item 30	Item 31	Item 32
0	4	citrus fruit	semi-finished bread	margarine	ready soups	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	3	tropical fruit	yogurt	coffee	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1	whole milk	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	4	pip fruit	yogurt	cream cheese	meat spreads	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	4	other vegetables	whole milk	condensed milk	long life bakery product	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Рис. 1. Початкові дані після завантаження

Як видно, максимальна кількість товарів у кошику може сягати 32 Причому якщо кількість товарів менша, то відсутні поля заповнюються значенням **NaN**.

Після завантаження даних необхідно провести їх первинну обробку. Виконаємо її в два послідовних етапи:

1) Так як поставлена задача потребує знаходження таких товарів, які купуються **разом**, то немає необхідності розглядати транзакції, в яких фігурує лише один товар. Як виявилося, таких транзакцій була майже чверть від загальної кількості (2159 від 9835). Видалення таких

транзакцій, по-перше, суттєво зменшить об’єм даних, які необхідно обробляти кожного разу при виконанні алгоритму пошуку частих наборів; а по-друге, зменшить підтримку і без того частих наборів (якщо видаляються часті товари) та збільшить підтримку товарів, що купляються рідше. Тепер перші 5 транзакцій дещо змінилися: транзакцію з індексом **2** було видалено. Ці зміни можна побачити на рис. 2:

	Item(s)	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7	Item 8	Item 9	...	Item 23	Item 24	Item 25	Item 26	Item 27	Item 28	Item 29	Item 30	Item 31	Item 32
0	4	citrus fruit	semi-finished bread	margarine	ready soups	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	3	tropical fruit	yogurt	coffee	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	4	pip fruit	yogurt	cream cheese	meat spreads	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	4	other vegetables	whole milk	condensed milk	long life bakery product	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	5	whole milk	butter	yogurt	rice	abrasive cleaner	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Рис. 2. Набір даних після першого етапу первинної обробки

2) Для застосування алгоритмів пошуку частих наборів необхідно перетворити цей набір даних до дещо іншого вигляду: в шапці колонок мають міститися назви товарів, а в самих колонках значення ***True***, якщо товар присутній у відповідній транзакції, або ***False***, якщо товару у відповідній транзакції немає (можна також використовувати ***1*** або ***0***). Для подібного перетворення ідеально підходить *TransactionEncoder*, що міститься у бібліотеці **Mlextend**. Він перетворює дані транзакцій бази даних у вигляді звичайного списку у масив **NumPy**. Отже, все, що нам залишається зробити перед застосуванням цього інструменту, це записати список транзакцій у вигляді вкладеного списку. Використовуючи *TransactionEncoder*, ми можемо перетворити цей набір даних у формат масиву, що підходить для типових API машинного навчання. За допомогою методу `fit` *TransactionEncoder* дізнається унікальні мітки (товари) в наборі даних, а за допомогою методу `transform` він перетворює вхідний набір даних (список списків) в однозначно кодований булевий масив **NumPy**. Масив **NumPy** є логічним інструментом для ефективного використання пам’яті при роботі з великими наборами даних. Для зручності ми перетворимо його у *DataFrame*, перші 5 елементів якого зображені на рис. 3.

	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	baby food	bags	baking powder	bathroom cleaner	beef	...	turkey	vinegar	waffles	whipped/sour cream	whisky	white bread	white wine
0	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
2	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
4	False	False	True	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False

Рис. 3. Набір даних після другого етапу первинної обробки

Як можна побачити, тепер набір даних дійсно має бажаний вигляд і готовий до застосування алгоритмів пошуку частих наборів. Також можна помітити, що порядок слідування транзакцій не змінився, тому що товар *abrasive cleaner* для неї відмічений значенням ***True***. Для наявності це явище виділено на рис. 2 та 3 червоним прямокутником.

Хоч наступний етап і не є обов'язковим, але для кращого розуміння даних, з якими нам доведеться працювати, виконаємо пошук 20 найбільш популярних товарів серед покупців. Будемо обчислювати загальну кількість відповідного товару серед усіх транзакцій. Відповідний результат зображено на рис. 4:

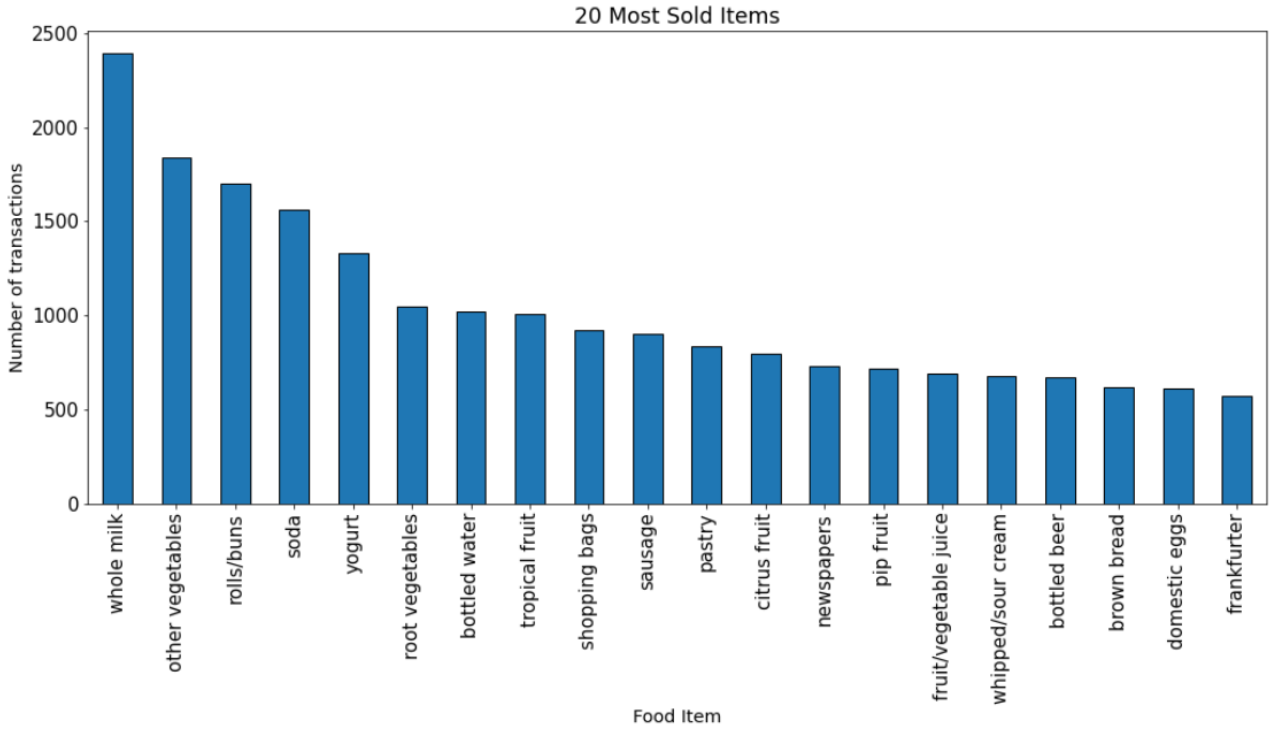

Рис. 4. Візуалізація 20 найбільш популярних товарів серед усіх транзакцій

Можна помітити, що товар *whole milk* купляють найчастіше (кількість покупок сягає 2400 одиниць). Із суттєвим відставанням покупці надають перевагу товару *other vegetables*, після якого різниця не становить більше 100 покупок відповідного товару.

2. Побудувати часті набори та асоціативні правила (АП), використовуючи алгоритм Apriori

Спочатку розглянемо даний алгоритм детально, щоб мати змогу при отриманні певних результатів вірно трактувати їх.

Теоретична основа та опис алгоритму Apriori

Алгоритм Apriori вивчає правила асоціації та застосовується до бази даних, що містить велику кількість транзакцій. Застосовуючи алгоритм Apriori, ми можемо вивчити продуктові предмети, які купуються разом, використовуючи правила асоціації. Ми маємо можливість знайти ті предмети, які, як правило,

купуються разом частіше, ніж інші предмети. Але кінцевою метою цього аналізу є потреба змусити покупців купувати більше.

Напевно, можемо логічно припустити, що овочі і фрукти чи ванільний цукор і дріжджі, здається, часто трапляються разом. Вони називаються наборами з двох предметів. З достатньо великим набором даних буде набагато важче передбачити такі відносини, особливо коли маємо справу з наборами з трьох і більше елементів. Саме тут може допомогти алгоритм Apriori!

Основний алгоритм Apriori складається з трьох етапів:

1. **Приєднання.** Скануємо всю базу даних на наявність частотних наборів з 1 предмета.
2. **Обрізання.** Ті набори предметів, які задовольняють мінімальним пороговим значенням підтримки та достовірності, переходять до наступного етапу для двохелементних наборів.
3. **Повторення.** Два попередні етапи повторюються для кожного рівня наборів елементів, поки ми не досягнемо визначеного раніше розміру або не досягнемо мінімальних порогових значень підтримки та достовірності.

Перед тим, як більш детально описати алгоритм, визначимо деякі поняття та опишемо постановку задачі. Дано:

1. $I = \{i_1, i_2, \dots, i_n\}$ – множина об'єктів;
2. $T = \{i_k \mid i_k \in I\} \subset I$ – транзакція (множина подій, що відбуваються разом);
3. $D = \{T_1, T_2, \dots, T_n\}$ – множина транзакцій.

Необхідно:

a) Знайти набори об'єктів, які зустрічаються в I найчастіше – часті набори;
b) Побудувати множину асоціативних правил (АП).

Визначимо наступне:

$D_{i_k} = \{T_r \mid i_k \in T_r, k = 1, \dots, n,\ r = 1, \dots, m\} \subseteq D$ – множина транзакцій, що містять об'єкт i_k.

$F \subseteq I$ – довільний набір об'єктів.

$D_F = \{T_r \mid F \in T_r, r = 1, \dots, m\} \subseteq D$ – множина транзакцій, що містять набір F.

Підтримкою набору F називається $Supp(F) = \frac{|D_F|}{|D|}$. $\forall E, F \subseteq I$ з умови $E \subseteq F$ випливає $Supp(E) \geq Supp(F)$ – **властивість антимонотонності**. Набір називається **частим**, якщо $Supp(F) \geq Supp_{min}$, де $Supp_{min}$ – мінімальна підтримка набору (порогове значення підтримки).

Асоціативне правило (АП) – це висловлювання виду «якщо X, тоді Y» ($X \Rightarrow Y$), де $X \subset I, Y \subset I, X \cap Y = \emptyset$.

Підтримка АП $Supp(X \Rightarrow Y) = p(X \cup Y) = \frac{|D_{F=X \cup Y}|}{|D|}$, де $|D_{F=X \cup Y}|$ – це кількість транзакцій, що містять X і Y.

Достовірність (значущість) АП $Conf(X \Rightarrow Y) = p(Y/X) = \frac{p(X \cap Y)}{p(X)} = \frac{Supp(X \Rightarrow Y)}{Supp(X)}$.

Покращення (Improvement) АП $Improv(X \Rightarrow Y) = \frac{Supp(X \Rightarrow Y)}{Supp(X)Supp(Y)}$. АП R_k будемо вважати значущим, якщо $Improv(R_k) > 1$.

Більше значення достовірності свідчить про більшу корисність правила. Проте це нежорстка імплікація, тобто не завжди із X слідує Y, але часто. Наскільки часто – це нам показує параметр $Conf_{min}$. Наприклад, в половині випадків, якщо $Conf_{min} = 0.5$. Цього часто буває достатньо. $Supp(Y)$ – безумовна підтримка, $Conf(X \Rightarrow Y)$ – умовна підтримка набору Y.

Тепер задача зводиться до наступного: знайти такі АП, що:

1. Набори X і Y часто зустрічаються разом: $Supp(X \Rightarrow Y) \geq Supp_{min}$;
2. Якщо зустрічається X, то часто зустрічається також і Y: $Conf(X \Rightarrow Y) \geq Conf_{min}$.

Де: $Supp_{min}$ – мінімальне порогове значення підтримки; $Conf_{min}$ – мінімальне порогове значення достовірності.

Тепер перейдемо до самого алгоритму Apriori:

$L_k = \{F_j \mid F_j = \{i_1, i_2, \dots, i_k\}, Supp(F_j) \geq Supp_{min}\}$ – множина k-елементних частих наборів.

C_k – множина кандидатів в k-елементні часті набори, отримана шляхом зв'язування множини L_{k-1} з собою. k-елементні набори є **зв'язуваними**, якщо вони мають $k-1$ перших спільних елементів.

Ідейне визначення алгоритму Apriori було наведено вище, тепер наведемо більш формальне (етапи будуть збігатися, лише будуть використані формальні визначення):

1. Побудова множини одноелементних частих наборів: $L_1 = \{i \mid i \in I, Supp(i) \geq Supp_{min}\}$;
2. Для всіх $k = 2, \dots, n$ генеруємо k-елементні часті набори $L_k = \{F \cup \{i\} \mid F \in L_{k-1},\ i \in L_1 \backslash F,\ Supp(F \cup \{i\}) \geq Supp_{min}\}$;
3. Якщо $L_k = \emptyset$, то виходимо із циклу по k.

Тобто ми перебираємо всі часті підмножини деякої множини. На **першому кроці** беремо одноелементні часті набори L_1. Далі додаємо до них по одному об'єкту, тобто формуємо двоелементні множини і так далі. Нечасті набори ми відкидаємо і так поступово нарощуємо потужність наборів, фільтруючи від нечастих наборів. Об'єм перебору суттєво залежить від параметру $Supp_{min}$: ***якщо значення $Supp_{min}$ зменшуємо, то кількість частих наборів буде більшою, якщо $Supp_{min}$ збільшуємо – то меншою.*** Це дуже критичний параметр щодо кількості знайдених частих наборів. k – це потужність наборів, які аналізуються на одній ітерації циклу.

На **другому кроці** шукаємо часті набори потужності k, за умови, що часті набори потужності $k-1$ нам відомі. Тобто Apriori – це рекурсивний метод. Беремо по одному елементу F з $k-1$ -елементного частого набору L_{k-1}, та по

одному об'єкту i з множини L_1 без F та намагаємося їх об'єднати. Якщо $Supp(F \cup \{i\}) \geq Supp_{min}$, то записуємо це об'єднання в L_k.

На **третьому кроці**, якщо не знайшли жодного частого набору потужності k, то виходимо з циклу по k, тому що вже жодного частого набору в подальшому не отримаємо.

Побудова АП за частими наборами

Нехай часті набори вже знайдено. Тоді побудова АП вважається простою і ефективною процедурою в оперативній пам'яті.

Ідея побудови АП: ми розщепляємо частий набір F на умову і наслідок, перевіряючи достовірність $Conf$ отриманого правила. Процедура є рекурсивною. Коли вона викликається перший раз, то список правил R порожній, F – деякий непорожній частий набір, Y – порожня.

При знаходженні частих наборів використовуємо тільки параметр $Supp_{min}$. При побудові АП використовується ще й параметр $Conf_{min}$.

Більш формальне визначення алгоритму:

Дано $\{L_k \mid k = 1, 2, \dots, n\}$ – k-елементні часті набори.

Знайти R – список асоціативних правил.

$R := \emptyset$

$\forall k = 2, \dots, n, \forall F \in L_k \; Y := \emptyset$

Процедура $BuildAssocRules(R, F, Y)$:

1. $\forall f \in F$ – можливих піднаборів f набору F

1.1. $X := F\{f\}$ $Y := Y \cup \{f\}$ – розщепляємо частий набір F на умову X і наслідок Y.

1.2. Якщо $Conf(X \Rightarrow Y) \geq Conf_{min}$, то:

1.2.1. Додати АП $X \Rightarrow Y$ в R.

1.2.2. Якщо $|X| > 1$, то $BuildAssocRules(R, F, Y)$.

Варіювання параметрів алгоритму Apriori

Будемо варіювати наступні параметри з відповідними значеннями:

```
min_support = [0.001, 0.005, 0.01]
max_len = [2, 3, 4]
min_confidence = [0.3, 0.4, 0.5]
```

`min_support` – це мінімальне порогове значення підтримки $Supp_{min}$.

`max_len` – це максимальна довжина набору при пошуку частих наборів.

`min_confidence` – це мінімальне порогове значення достовірності $Conf_{min}$.

При цьому будемо відстежувати кількість правил, які мають $Improv(R_k) > 1$, $Improv(R_k) > 2$ та $Improv(R_k) > 3$. Тільки відразу необхідно зауважити, що висока кількість таких правил НЕ є мірою якості алгоритму з відповідними параметрами.

Результати варіювання параметрів наведено у таблиці 1. Дані, що використовувалися для побудови цієї таблиці та отримані безпосередньо внаслідок виконання коду, наведені в додатку А в кінці роботи. На мою думку, таке представлення набагато наочніше за «сирі», неопрацьовані дані.

Таблиця 1. Результати варіювання параметрів для алгоритму Apriori

min_support	max_len	min_confidence	n_imp>1	n_imp>2	n_imp>3	Time, sec
0.001	2	0.3	273	41	13	0.4455
		0.4	94	16	5	0.4289
		0.5	15	8	1	0.3917
	3	0.3	6861	3393	1090	7.6159
		0.4	3625	1864	541	7.7037
		0.5	1976	1243	287	7.6679
	4	0.3	19590	13044	5689	16.5235
		0.4	11842	8216	3320	16.4485
		0.5	7584	5818	1988	16.6042
0.005	2	0.3	134	9	2	0.2480
		0.4	56	6	2	0.2459
		0.5	6	3	0	0.2457
	3	0.3	691	150	6	1.0300
		0.4	401	81	5	1.0288
		0.5	170	58	1	1.0288
	4	0.3	819	241	29	1.1141
		0.4	478	131	14	1.1147
		0.5	244	92	4	1.1144
0.01	2	0.3	91	2	0	0.1572
		0.4	33	0	0	0.1570
		0.5	2	0	0	0.1572
	3	0.3	220	30	0	0.2907
		0.4	111	12	0	0.2926
		0.5	35	9	0	0.2914
	4	0.3	225	33	1	0.2973
		0.4	113	13	0	0.2987
		0.5	37	10	0	0.2985

Відразу кидається в очі підтвердження **властивості антимонотонності**: зі збільшенням параметрів $Supp_{min}$ та $Conf_{min}$ кількість значущих АП зменшується. Аналогічна ситуація спостерігається і з часом побудови АП: чим більше знайдено частих наборів, тим більше АП, а, отже, більше час їх побудови. Єдине, що можна зауважити, час не сильно варіюється від зміни значення параметру $Conf_{min}$. І цьому є логічне пояснення: більша частина часу витрачається не на побудову АП, а на пошук частих наборів. А процедура побудови АП за частими наборами, що вже побудовані, не дуже важка з точки зору кількості та складності обчислень. Саме тому на час побудови істотно впливають лише два перших параметри.

З параметром *max_len* ситуація кардинально відрізняється. Тут простежується пряма пропорційність: зі збільшенням значення параметру збільшується й кількість знайдених АП, а також, що зрозуміло, збільшується витрачений на виконання алгоритму час.

З перерахованих вище параметрів, що використовувалися для дослідження впливу варіювання параметрів на множину АП, було вирішено обрати наступні (у таблиці 1 вони виділені кольором):

```
min_support = 0.01
max_len = 3
min_confidence = 0.5
```

Це рішення було прийнято з причин, які будуть детальніше висвітлені у пункті 4 даної роботи. Але в цілому була віддана перевага невеликій результуючій множині АП серед інших, тому що при її аналізі буде легше орієнтуватися в обмеженій кількості АП. Хоча всі відмінності результуючих множин також буде детально досліджено в пункті 4.

Отже, побудуємо часті набори за допомогою алгоритму Apriori, використовуючи параметри $Supp_{min} = 0.01$ та `max_len` $= 3$. Їх частину зображено на рисунку 5:

	support	itemsets	length
0	0.041558	(UHT-milk)	1
1	0.022277	(baking powder)	1
2	0.063835	(beef)	1
3	0.041298	(berries)	1
4	0.029964	(beverages)	1
...	...	...	...
493	0.010031	(soda, tropical fruit, whole milk)	3
494	0.013418	(yogurt, soda, whole milk)	3
495	0.010162	(whipped/sour cream, tropical fruit, whole milk)	3
496	0.019411	(yogurt, tropical fruit, whole milk)	3
497	0.013940	(yogurt, whipped/sour cream, whole milk)	3

Рис. 5. Часті набори, побудовані за допомогою алгоритму Apriori

Як ми бачимо, параметри дійсно обмежують результуючу множину отриманих наборів (всього їх вийшло 498). А отримані набори, як-от *yougurt, whipped/sour cream, whole milk* дійсно здається, що мають купуватися разом, тому що належать до молочних продуктів, хоча їх підтримка і поступається одиночним наборам.

Як видно з таблиці 1, збільшення параметру `max_len` до 4 не має особливого сенсу, тому що вони відрізняються один від одного всього на 2 значущих АП. Але частих наборів було побудовано значно більше. Саме тому часу витрачено аж на 0.007 секунди більше. Отже, було прийнято рішення про вибір такого значення параметру.

Хоча цей факт є зрозумілим, але треба також додати, що із збільшенням елементів у наборі його підтримка зменшується. Тобто якщо сир купували 5 разів з 10 транзакцій, то сир з ковбасою купували, наприклад, 2 рази. Внаслідок цього і зменшується підтримка такого набору.

Тепер за допомогою побудованих частих наборів знайдемо значущі АП. Отримані АП, відсортовані за спаданням величини ліфту, зображені на рис. 6. Треба зазначити, що на рисунку зображені не всі АП, лише певна кількість з 35 побудованих.

Перше, що кидається в очі, це наявність двох стовпчиків з однаковими значеннями. Це зроблено навмисно для того, щоб продемонструвати ідентичність понять ***lift*** та ***improvement***, так як цей, здавалося б, очевидний факт, не був настільки очевидним на початку виконання роботи. Тому можна вважати

підтвердженим на практиці той факт, що $Lift(X \Rightarrow Y) = Improv(X \Rightarrow Y) = \frac{Supp(X \Rightarrow Y)}{Supp(X)Supp(Y)} = \frac{Conf(X \Rightarrow Y)}{Supp(Y)}$.

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	improvement
6	(root vegetables, citrus fruit)	(other vegetables)	0.022668	0.239838	0.013288	0.586207	2.444174	0.007851	1.837057	2.444174
18	(root vegetables, tropical fruit)	(other vegetables)	0.026967	0.239838	0.015763	0.584541	2.437228	0.009296	1.829691	2.437228
21	(whipped/sour cream, tropical fruit)	(other vegetables)	0.017718	0.239838	0.010031	0.566176	2.360658	0.005782	1.752237	2.360658
15	(pip fruit, root vegetables)	(other vegetables)	0.019932	0.239838	0.010422	0.522876	2.180117	0.005642	1.593215	2.180117
17	(rolls/buns, root vegetables)	(other vegetables)	0.031136	0.239838	0.015633	0.502092	2.093459	0.008166	1.526711	2.093459
20	(yogurt, root vegetables)	(other vegetables)	0.033090	0.239838	0.016545	0.500000	2.084737	0.008609	1.520323	2.084737
19	(whipped/sour cream, root vegetables)	(other vegetables)	0.021886	0.239838	0.010943	0.500000	2.084737	0.005694	1.520323	2.084737
5	(butter, yogurt)	(whole milk)	0.018760	0.311621	0.011985	0.638889	2.050214	0.006139	1.906281	2.050214
4	(butter, root vegetables)	(whole milk)	0.016545	0.311621	0.010552	0.637795	2.046704	0.005397	1.900526	2.046704
11	(domestic eggs, root vegetables)	(whole milk)	0.018369	0.311621	0.010943	0.595745	1.911763	0.005219	1.702833	1.911763
9	(yogurt, curd)	(whole milk)	0.022147	0.311621	0.012897	0.582353	1.868788	0.005996	1.648232	1.868788
25	(pip fruit, root vegetables)	(whole milk)	0.019932	0.311621	0.011464	0.575163	1.845717	0.005253	1.620339	1.845717
8	(curd, other vegetables)	(whole milk)	0.022017	0.311621	0.012637	0.573964	1.841869	0.005776	1.615779	1.841869
3	(butter, other vegetables)	(whole milk)	0.025664	0.311621	0.014721	0.573604	1.840713	0.006724	1.614414	1.840713
32	(whipped/sour cream, tropical fruit)	(whole milk)	0.017718	0.311621	0.010162	0.573529	1.840473	0.004640	1.614131	1.840473
29	(root vegetables, tropical fruit)	(whole milk)	0.026967	0.311621	0.015373	0.570048	1.829302	0.006969	1.601062	1.829302
31	(yogurt, root vegetables)	(whole milk)	0.033090	0.311621	0.018629	0.562992	1.806659	0.008318	1.575210	1.806659
30	(whipped/sour cream, root vegetables)	(whole milk)	0.021886	0.311621	0.012116	0.553571	1.776427	0.005295	1.541970	1.776427
10	(domestic eggs, other vegetables)	(whole milk)	0.028530	0.311621	0.015763	0.552511	1.773026	0.006873	1.538317	1.773026
12	(frozen vegetables, other vegetables)	(whole milk)	0.022798	0.311621	0.012376	0.542857	1.742045	0.005272	1.505830	1.742045
14	(rolls/buns, margarine)	(whole milk)	0.018890	0.311621	0.010162	0.537931	1.726237	0.004275	1.489776	1.726237
28	(rolls/buns, whipped/sour cream)	(whole milk)	0.018760	0.311621	0.010031	0.534722	1.715940	0.004185	1.479502	1.715940
0	(baking powder)	(whole milk)	0.022277	0.311621	0.011855	0.532164	1.707729	0.004913	1.471411	1.707729
26	(pip fruit, yogurt)	(whole milk)	0.023059	0.311621	0.012246	0.531073	1.704231	0.005060	1.467990	1.704231
34	(yogurt, whipped/sour cream)	(whole milk)	0.026576	0.311621	0.013940	0.524510	1.683168	0.005658	1.447726	1.683168

Рис. 6. Значущі АП, отримані за допомогою застосування Apriori для пошуку частих наборів

Також помітно, що параметри, що використовувалися при побудові АП, дійсно обмежують велику множину можливих АП, тому що підтримка для всіх АП не знижується до 0.01, а достовірність не падає нижче за 0.5.

Звернімо нашу увагу на другий стовпчик під назвою *consequents*. В ньому містяться лише два значення: *other vegetables* та *whole milk*. Десь вже ми бачили назви цих двох продуктів: це ж ті самі продукти, які купуються найчастіше! Отже, за допомогою такого набору параметрів ми отримали АП, що засновані на найчастіших наборах, а в нашому випадку це саме ці два товари.

Також у наведеному на рис. 6 об'єкті *DataFrame* використані метрики, що не згадувалися раніше. Дамо їм визначення та формулу для обчислення:

$leverage(X \Rightarrow Y) = Supp(X \Rightarrow Y) - Supp(X)Supp(Y)$. Ця метрика показує, наскільки набори незалежні, Якщо її значення близьке до 0, то набори майже незалежні. А можливі значення коливаються у проміжку $[-1, 1]$. Як ми бачимо, для великих обсягів даних ця метрика майже не несе змісту, тому що для всіх АП її значення буде близьке до 0, адже сильних залежностей між наборами

немає. Ця метрика може бути корисною лише для невеликої кількості транзакцій або невеликою кількістю товарів.

$Conviction(X \Rightarrow Y) = \frac{1-Supp(Y)}{1-Conf(X \Rightarrow Y)}$. Ця метрика за своєю суттю дуже схожа на покращення або ліфт, тому не має особливого змісту. Якщо поглянути на відсортований за спаданням покращення *DataFrame*, то можна помітити, що він також відсортований за метрикою *conviction*, що доводить їх спільне походження.

Отже, на даному етапі виконання роботи ми побудували множину значущих АП для початкових транзакцій та визначили метрики, які впливають на розміри результуючої множини АП.

3. Побудувати часті набори та множину АП, використовуючи алгоритм FP-росту.

Спочатку розглянемо даний алгоритм детально, щоб мати змогу при отриманні певних результатів вірно трактувати їх.

Теоретична основа та опис алгоритму FP-росту

Одним з найбільш ефективних процедур пошуку АП є алгоритм, який отримав назву **Frequent Pattern-Growth** (алгоритм **FPG**), що можна перекласти як «вирощування популярних (тих, що часто зустрічаються) предметних наборів».

В основі методу лежить первинна обробка бази транзакцій, в процесі якої ця база даних перетворюється в компактну деревоподібну структуру, яка називається **Frequent-Pattern Tree** – дерево популярних предметних наборів (звідки і походить назва алгоритму). Надалі для стислості будемо називати цю структуру FP-дерево G. Його вершини – об'єкти $i \in I$, причому різні вершини дерева можуть містити одні й ті самі об'єкти. Шлях від кореня g_0 до вершини g – набір об'єктів $F \subseteq I$. $G(i) = \{g \in G : g = i\}$ – множина вершин для об'єкта i. $Supp(i) = \sum_{g \in G(i)} Supp(g)$ – підтримка об'єкта i. Різні дерева відповідають об'єктам за спаданням $Supp(i)$, причому $Supp(i) \geq Supp_{min}$, тобто маємо порядок на множині об'єктів.

Алгоритм FPG складається з двох етапів та одного попереднього етапу.

Попередній етап. Сортування БД та створення словника елементів

Перший раз проходимо БД транзакцій і підраховуємо підтримку кожного об'єкту. Сортуємо об'єкти за спаданням величини підтримки. Сортуємо БД транзакцій. Елементи відсортованої БД можуть розглядатися як слова або словник. Тоді можна використати відомі ефективні методи збереження словників і пошуку в них.

Етап 1. Побудова FP-дерева

Дано $D = \{T_1, T_2, \dots, T_n\}$ – множина транзакцій (навчальна вибірка).

Необхідно знайти дерево $G = \{g \mid g = (Name(g), Supp(g), Child(g))\}$.

Вузол FP-дерева – це структура, яка зберігає значення вузла $Name$, значення його підтримки $Supp$, а також посилання на всі його дочірні елементи $Child$. Для кожного елемента кожної відсортованої транзакції з вхідного набору будуються вузли за таким правилом:

- якщо для чергового елемента в поточному вузлі є нащадок, що містить цей елемент, то новий вузол не створюється, а підтримка цього нащадка збільшується на 1;
- в іншому випадку створюється новий вузол-нащадок з підтримкою 1. Поточним вузлом при цьому стає знайдений або побудований вузол.

Більш формальний алгоритм:

1. В кожній транзакції впорядкувати об'єкти $i \in I$ за спаданням $Supp(i)$. Фільтрація – вилучення об’єктів з $Supp(i) \geq Supp_{min}$;
2. $\forall\, T_k \in D$
   2.1. $g \coloneqq g_0$
   2.2. $\forall\, i \in I: i \in T_k$
      2.2.1. Якщо у g немає дочірньої вершини, рівної i, то створити дочірню вершину x, рівну i, $Name(x) \coloneqq i, Supp(x) \coloneqq 1$;
      2.2.2. $Supp(x) \coloneqq Supp(x) + 1$, $g \coloneqq x$

Візуалізацію послідовної побудови FP-дерева наведено на рис. 7:

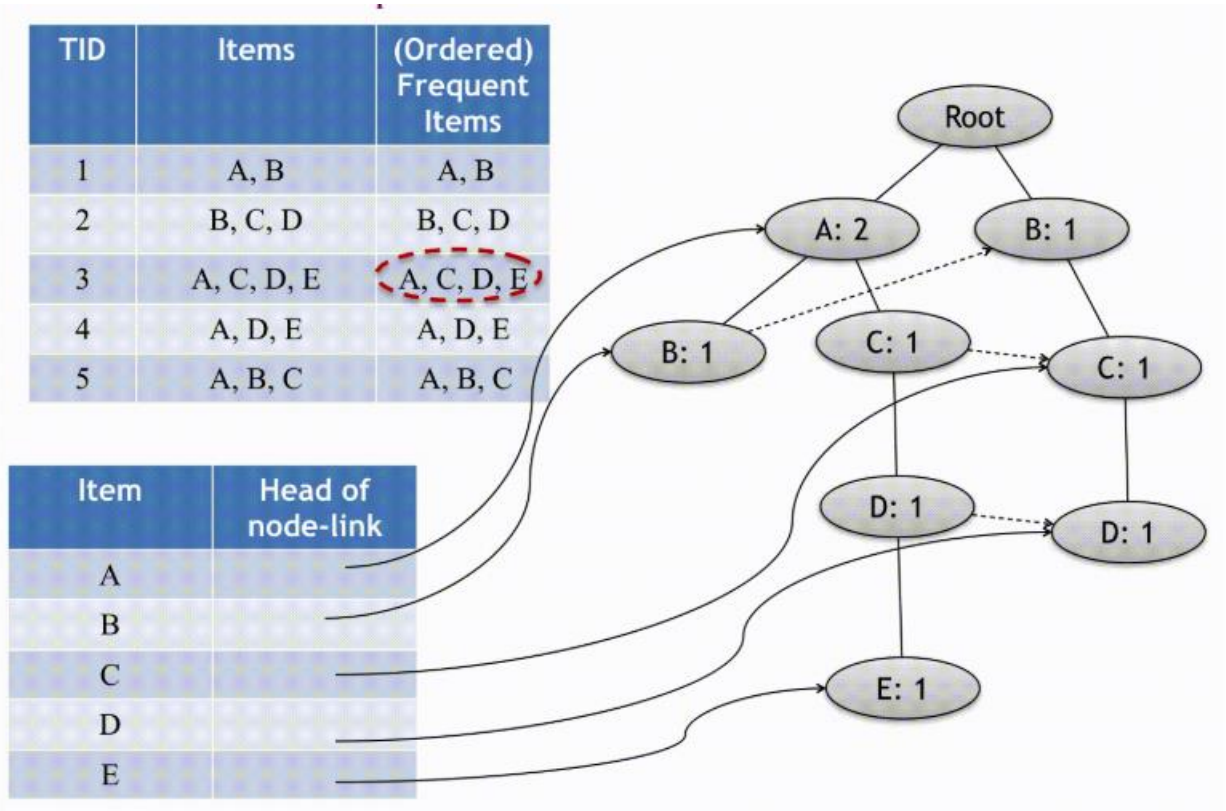

Рис. 7. Візуалізація побудови FP-дерева

Етап 2. Пошук частих наборів у FP-дереві

Дано FP-дерево G, набір об’єктів F. Необхідно знайти часті набори для F.

$F \coloneqq \emptyset$

Процедура $FP(G, F)$:

1. $\forall\, i \in I: G(i) \neq \emptyset$ по рівням **знизу вверх**. Якщо $Supp(i) \geq Supp_{min}$, то:
   1.1. $F' \coloneqq F \cup \{i\}$ – частий набір;

1.2. Побудувати ***умовне FP-дерево*** G' за об'єктом i;

1.3. $FP(G', F')$ – знайти часті набори за деревом G' для частого набору F', в якому є об'єкт i.

Алгоритм пошуку умовного FP-дерева

Умовне FP-дерево $G' \coloneqq G|i$ – це FP-дерево за підмножиною транзакцій $D_i = \{T_k \mid i \in T_k\}$, які містять заданий об'єкт $i \in I$, з якого вилучені вершини $g \in G(i)$ та всі їх потомки. БД D_i має набагато менший об'єм в порівнянні з початковою БД транзакцій D. Умовне дерево G' буде, як правило, набагато меншим порівняно з початковим FP-деревом G. Алгоритм, побудований на FP-деревах, дуже економний.

1. В дереві G вилучаємо всі шляхи, які не містять об'єкт i.
2. В дереві G вилучаємо потомки вершин, які відповідають об'єкту i.
3. Перераховуємо підтримку вузлів, що залишилися в G: $Supp(g) \coloneqq \sum_{x \in Child(g)} Supp(x)\ \forall g \in G$.
4. В дереві G вилучаємо вершини, які відповідають об'єкту i.
5. Результуюче дерево – шукане умовне FP-дерево.

На рис. 8 показано умовне дерево за елементом a:

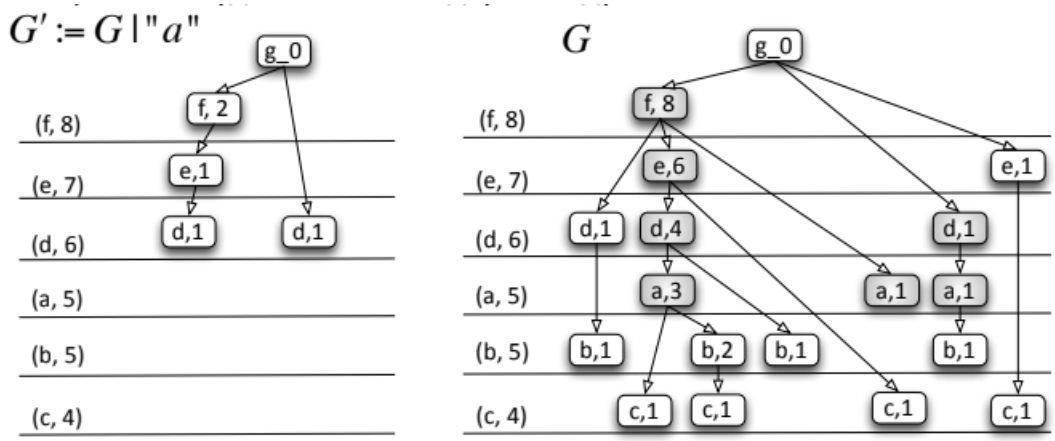

Рис. 8. Візуалізація побудови умовного FP-дерева

Побудова АП за знайденими частими наборами відбувається аналогічно до випадку, описаному на попередньому етапі роботи для алгоритму Apriori.

Варіювання параметрів алгоритму FPG

Як і на попередньому етапі, будемо варіювати наступні параметри з відповідними значеннями:

```
min_support = [0.001, 0.005, 0.01]
max_len = [2, 3, 4]
min_confidence = [0.3, 0.4, 0.5]
```

`min_support` – це мінімальне порогове значення підтримки $Supp_{min}$.

`max_len` – це максимальна довжина набору при пошуку частих наборів.

`min_confidence` – це мінімальне порогове значення достовірності $Conf_{min}$.

При цьому будемо відстежувати кількість правил, які мають $Improv(R_k) > 1$, $Improv(R_k) > 2$ та $Improv(R_k) > 3$. Тільки відразу

необхідно зауважити, що висока кількість таких правил НЕ є мірою якості алгоритму з відповідними параметрами. Це лише міра, що буде використовуватися для порівняння.

Результати варіювання параметрів наведено у таблиці 2. Дані, що використовувалися для побудови цієї таблиці та отримані безпосередньо внаслідок виконання коду, наведені в додатку Б в кінці роботи.

Таблиця 2. Результати варіювання параметрів для алгоритму FPG

min_support	max_len	min_confidence	n_imp>1	n_imp>2	n_imp>3	Time, sec
0.001	2	0.3	273	41	13	0.6282
		0.4	94	16	5	0.5964
		0.5	15	8	1	0.6257
	3	0.3	6861	3393	1090	0.8916
		0.4	3625	1864	541	0.9043
		0.5	1976	1243	287	0.8941
	4	0.3	19590	13044	5689	1.0938
		0.4	11842	8216	3320	1.0556
		0.5	7584	5818	1988	1.0423
0.005	2	0.3	134	9	2	0.2538
		0.4	56	6	2	0.2858
		0.5	6	3	0	0.3158
	3	0.3	691	150	6	0.2649
		0.4	401	81	5	0.2957
		0.5	170	58	1	0.2659
	4	0.3	819	241	29	0.3242
		0.4	478	131	14	0.2988
		0.5	244	92	4	0.3079
0.01	2	0.3	91	2	0	0.1905
		0.4	33	0	0	0.1852
		0.5	2	0	0	0.1486
	3	0.3	220	30	0	0.2112
		0.4	111	12	0	0.1902
		0.5	35	9	0	0.1992
	4	0.3	225	33	1	0.2146
		0.4	113	13	0	0.1875
		0.5	37	10	0	0.2163

На перший погляд може здатися, що значення у таблиці не змінилися. Але це не так. Змінилися лише значення часу виконання пошуку АП. І змінилися вони дуже суттєво: тепер він лише трошки перевищує 1 секунду для пошуку 19,5 тис. (!) значущих АП. Отже, до зроблених висновків на попередньому етапі роботи (бо всі згадані вище закономірності та особливості зберігаються і поширюються на результати, отримані для алгоритму FPG), можна додати лише один новий висновок: для великих наборів даних (великої кількості транзакцій) алгоритм FPG значно кращий через свою швидкість.

Як і для попереднього етапу, оберемо наступні параметри для побудови АП (у таблиці 2 вони виділені кольором):

```
min_support = 0.01
max_len = 3
min_confidence = 0.5
```

Так як результати аналогічні отриманим для алгоритму Apriori, то не бачимо необхідності ще раз писати одні й ті самі речі.

4. Зробити висновки щодо впливу параметрів алгоритмів Apriori та FP-росту на знайдені множини АП

Як вже було з'ясовано на попередніх етапах роботи, варіювання параметрів вищезгаданих алгоритмів впливає лише на кількість АП, що будуть наведені у якості результату (а також, зрозуміло, і на час їх побудови). Для візуалізації цього нанесемо для побудованих АП на одній площині достовірність та покращення. Для обох алгоритмів ці графіки мають бути однаковими, але все ж наведемо обидва варіанти на рис. 9:

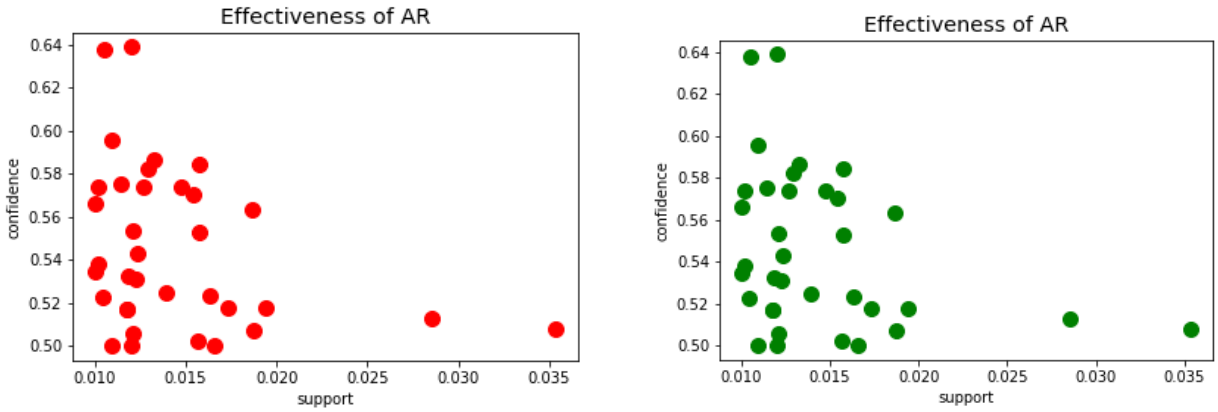

Рис. 9. Візуалізація ефективності побудованих АП для алгоритму: (зліва направо) а) Apriori; б)FPG

І дійсно, графіки виявилися однаковими, за винятком хіба що кольору. На них чіткіше видно, як саме відбувається відсіювання за обраними параметрами: за нижніми межами достовірності та підтримки, що зображені на відповідних вісях графіку.

Тепер, як і було обіцяно на етапі 2, дослідимо суттєве збільшення АП з покращенням, що перевищувало значення 3. Спочатку ще раз поглянемо на формулу:

$$Lift(X \Rightarrow Y) = Improv(X \Rightarrow Y) = \frac{Supp(X \Rightarrow Y)}{Supp(X)Supp(Y)} = \frac{Conf(X \Rightarrow Y)}{Supp(Y)}.$$

Чому ж для не дуже частих наборів (що відсіюються ще на етапі побудови частих наборів для мінімального порогового значення більше за 0.005) значення покращення таке велике? Відповідь криється у формулі: через вкрай малу підтримку, що міститься у знаменнику (при доволі незначних змінах достовірності) значення покращення зростає для таких наборів.

Саме тому для обраних у минулих пунктах параметрів ми не отримували високі значення покращення: тому що наші набори були частими (що доведено ще при первинному аналізі транзакцій). Тому варіювання параметрів має велику роль у побудові АП: ***якщо ми хочемо отримати АП з глобальними*** (для всієї множини транзакцій) ***частими наборами, то необхідно максимізувати параметри*** $\boldsymbol{Supp_{min}}$ ***та*** $\boldsymbol{Conf_{min}}$. Так як набори є глобально частими, то достовірність таких правил буде, звичайно вищою (алгоритм «впевнений» у таких АП).

Тепер уявимо, що ми купуємо якийсь рідкісний товар для звичайного магазину, наприклад, горщик для квітів, а до нього також додаємо ґрунт для квітучих рослин. Такий набір хоч і є логічно купувати разом, проте для високих значень $Supp_{min}$ його не буде серед АП. Тому ***якщо ми хочемо отримати АП з не дуже частими наборами, то необхідно мінімізувати параметри*** $\boldsymbol{Supp_{min}}$ ***та*** $\boldsymbol{Conf_{min}}$. Так як таких наборів може бути всього декілька серед усіх транзакцій, тому достовірність таких правил також буде низькою (алгоритм «не впевнений» чи це дійсно закономірність, а чи випадковість). І звичайно, покращення для таких наборів буде значно вищим через маленьке значення підтримки.

Отже, тепер зрозуміло, що вибір параметрів на 2 етапі роботи передбачав саме пошук закономірностей серед найчастіших наборів, задля цього і був проведений первинний аналіз найбільш популярних товарів серед покупців.

5. Знайти значення прогнозу на основі побудованої множини правил

Для побудови частих наборів будемо використовувати алгоритм FPG, так як ми на практиці довели його вищу швидкість у порівнянні з алгоритмом Apriori. Також встановимо мінімальний поріг підтримки рівний 0.001, щоб майже усі нечасті набори увійшли до множини частих наборів. Зробимо прогноз для частого й нечастого набору, а потім порівняємо отримані результати.

Виконання прогнозу для частого набору

Так як найчастішим набором є одноелементний набір, що містить товар *whole milk*, то побудуємо множину АП, яка містить цей товар. Причому будемо відбирати лише значущі правила, для яких покращення більше за 1. Отже, побудована множина значущих АП для частого набору зображена на рис. 10.

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
148081	(sliced cheese, butter, root vegetables, whipp...	(whole milk)	0.001042	0.311621	0.001042	1.000000	3.209030	0.000717	inf
143438	(onions, pip fruit, butter)	(whole milk)	0.001172	0.311621	0.001172	1.000000	3.209030	0.000807	inf
123725	(yogurt, margarine, hard cheese)	(whole milk)	0.001172	0.311621	0.001172	1.000000	3.209030	0.000807	inf
5726	(margarine, whipped/sour cream, other vegetabl...	(whole milk)	0.001042	0.311621	0.001042	1.000000	3.209030	0.000717	inf
125258	(ham, pip fruit, whipped/sour cream)	(whole milk)	0.001042	0.311621	0.001042	1.000000	3.209030	0.000717	inf
...	...	...	...	...	...	...	...	...	...
155512	(pet care)	(whole milk)	0.010813	0.311621	0.003387	0.313253	1.005238	0.000018	1.002377
58614	(newspapers, yogurt, bottled water)	(whole milk)	0.004169	0.311621	0.001303	0.312500	1.002822	0.000004	1.001279
20752	(long life bakery product, margarine)	(whole milk)	0.004169	0.311621	0.001303	0.312500	1.002822	0.000004	1.001279
49262	(rolls/buns, frankfurter)	(whole milk)	0.024622	0.311621	0.007686	0.312169	1.001761	0.000014	1.000798

Рис. 10. Значущі АП для частого набору

Таких правил вийшло побудувати аж 6380. Це відбулося внаслідок того, що мінімальний поріг достовірності також було вирішено зменшити та надати йому значення 0.01. Тому не дивно, що ми отримали таку кількість АП, які включають тільки один конкретний набір.

Як вже було сказано раніше, покращення для частих наборів не є дуже високим внаслідок порівняно великого значення підтримки. Що дійсно варто зауважити, що для деяких правил достовірність сягнула 1. Це відбулося внаслідок великої підтримки товару *whole milk* та маленького значення інших товарів, які були асоційовані з ним.

І як не дивно, прогноз дійсно відповідає реальності: з молоком в більшості випадків асоціюються молочні продукти, такі як сир, вершкове масло, сметана, йогурт та інші. Тому можна вважати даний прогноз дуже хорошим як внаслідок підтвердження з точки зору життєвого досвіду, так і з точки зору відмінної достовірності.

Тепер поглянемо на візуалізацію АП, що відповідають товару *whole milk* і тих, що також були побудовані внаслідок виконання алгоритму, на рис. 11. Цей графік виконаний у площині метрик підтримки та достовірності.

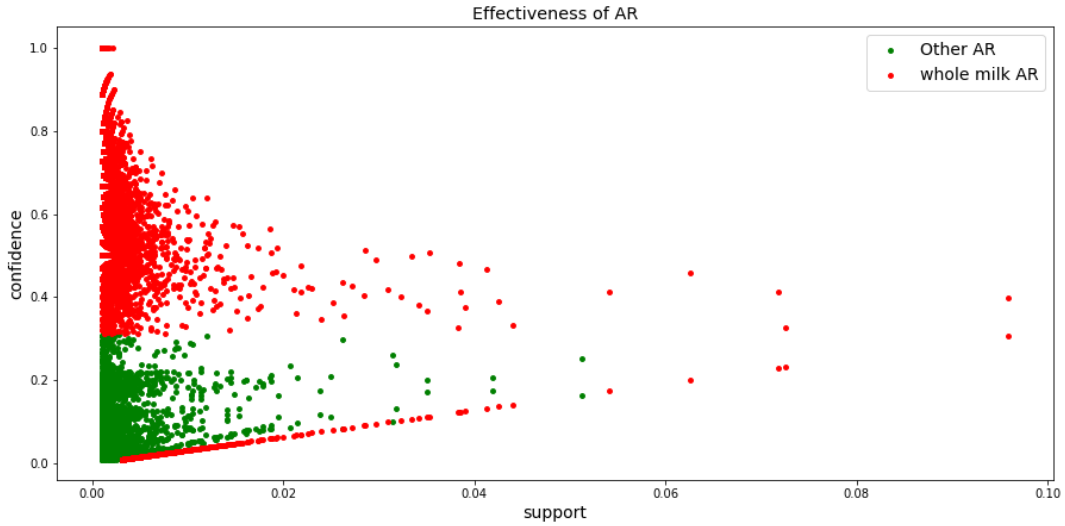

Рис. 11. Візуалізація АП для частого набору

Як видно з графіку, більшість АП для частого набору знаходяться у верхній половині площини, так як їх достовірність велика, порівняно з іншими. За підтримкою ці набори також займають перші місця серед інших. Але все ж більшість прогнозів мають підтримку меншу за 0.02.

Виконання прогнозу для нечастого набору

Для виконання прогнозу для нечастого набору спочатку визначимо його, використавши вже побудовану на минулому етапі множину частих наборів, відсортовану за зростанням. Обираємо набір (*pip fruit, curd*). Виконаємо аналогічні дії для цього набору. Отримана множина АП зображена на рис. 12, причому одразу були відібрані значущі АП.

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
46453	(pip fruit, curd)	(sausage, yogurt, whole milk)	0.010031	0.011204	0.001172	0.116883	10.432498	0.001060	1.119666
46438	(sausage, yogurt, whole milk)	(pip fruit, curd)	0.011204	0.010031	0.001172	0.104651	10.432498	0.001060	1.105679
47167	(tropical fruit, whipped/sour cream, whole milk)	(pip fruit, curd)	0.010162	0.010031	0.001042	0.102564	10.224442	0.000940	1.103108
47176	(pip fruit, curd)	(tropical fruit, whipped/sour cream, whole milk)	0.010031	0.010162	0.001042	0.103896	10.224442	0.000940	1.104602
148322	(sliced cheese, yogurt)	(pip fruit, curd)	0.010292	0.010031	0.001042	0.101266	10.095019	0.000939	1.101515
...	...	...	...	...	...	...	...	...	...
47608	(newspapers)	(pip fruit, curd)	0.095232	0.010031	0.001172	0.012312	1.227353	0.000217	1.002309
46763	(shopping bags)	(pip fruit, curd)	0.119984	0.010031	0.001303	0.010858	1.082392	0.000099	1.000836
46758	(pip fruit, curd)	(shopping bags)	0.010031	0.119984	0.001303	0.129870	1.082392	0.000099	1.011361
43806	(bottled water)	(pip fruit, curd)	0.132882	0.010031	0.001433	0.010784	1.075070	0.000100	1.000761
43801	(pip fruit, curd)	(bottled water)	0.010031	0.132882	0.001433	0.142857	1.075070	0.000100	1.011638

Рис. 12. Множина АП для нечастого набору

Отримали всього 160 правил. Тут можна сказати про підтвердження думки про зростання метрики покращення внаслідок пошуку нечастих наборів, саме тому ми НЕ використовували її як метрику якості, а тільки як показник значущості АП. Адже для побудованих правил це значення перевищує 10.

Для такого набору (*pip fruit, curd*) ми отримали асоціації з такими продуктами як сосиски, йогурт, молоко, сир. Ці АП дійсно відповідають реальності. Але асоціація його з газетами чи пакетами для покупок викликає підозру. Як можна помітити, значення достовірності для таких АП лише 0.01-0.1, що не дуже хороша якість.

До того ж, варто зауважити, що для нечастих наборів ми отримали повторювані АП, що різняться лише порядком, тобто вірні обидва варіанти «якщо А, то Б» і «якщо Б, то А». Такого не спостерігалося для частих наборів. Це є однією з найбільших відмінностей між ними, окрім більшого значення покращення та меншого значення достовірності.

Тепер поглянемо на візуалізацію АП, що відповідають набору (*pip fruit, curd*) і тих, що також були побудовані внаслідок виконання алгоритму, на рис. 13. Цей графік також виконаний у площині метрик підтримки та достовірності.

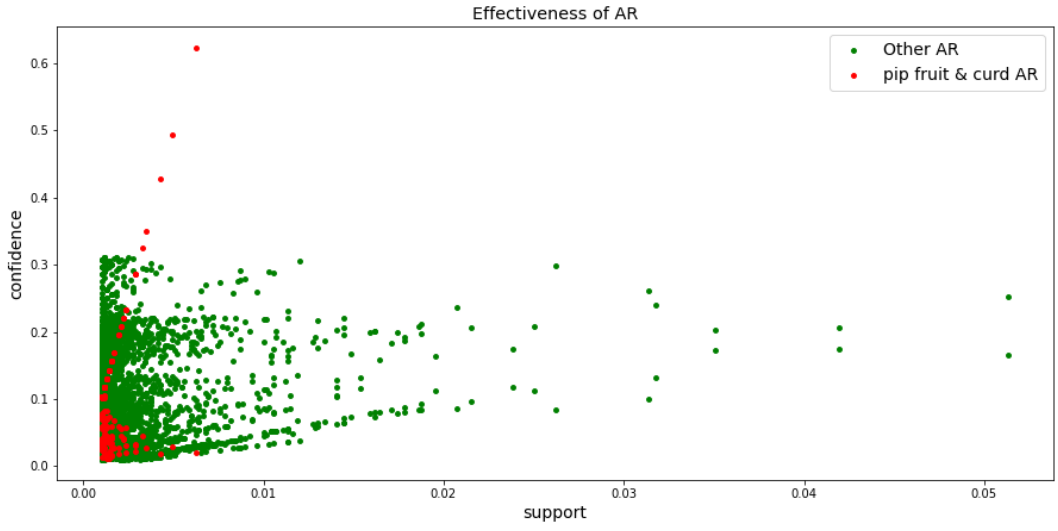

Рис. 13. Візуалізація АП для нечастого набору

Як ми бачимо, множина АП дійсно стала меншою і розташовується вона переважно у лівому нижньому кутку, що відповідає найменшим пороговим значенням, як і повинно бути для нечастого набору. Але по-справжньому цікавим є лінія на графіку, яку утворюють точки – окремі АП. І внаслідок невеликого дослідження було доведено, що ці АП містять в собі як другий компонент – товар *whole milk*. Як дивовижно! Тобто внаслідок попадання до наслідку АП частого набору збільшується його достовірність, що є цілком логічно.

Якщо придивитися до графіку для частого набору, то також можна помітити цю пряму, але, на відміну від нечастого набору, внаслідок попадання нечастого набору до висновку АП його підтримка зростає, а достовірність зростає лише незначно.

6. Порівняти результати, отримані алгоритмами Apriori та FP-росту

Отже, внаслідок виконаних досліджень, можна зробити висновок, що результати, отримані за допомогою використання обох алгоритмів, є ідентичними. Єдине, що відрізняє їх між собою – використання пам'яті та об'єм витраченого часу для формування частих наборів.

З точки зору використання пам'яті все зрозуміло: дерево значно краще та ефективніше для зберігання даних у пам'яті, ніж кожного разу шукати інформацію у БД транзакцій. До того ж, згідно з алгоритмом FPG, сканування БД відбувається всього два рази: коли ми обчислюємо підтримку усіх елементів і коли будуємо FP-дерево.

З точки зору витрат часу зробимо невелике дослідження: будемо будувати АП, використовуючи лише випадкові підмножини початкових даних. Результати зображено на рис. 14.

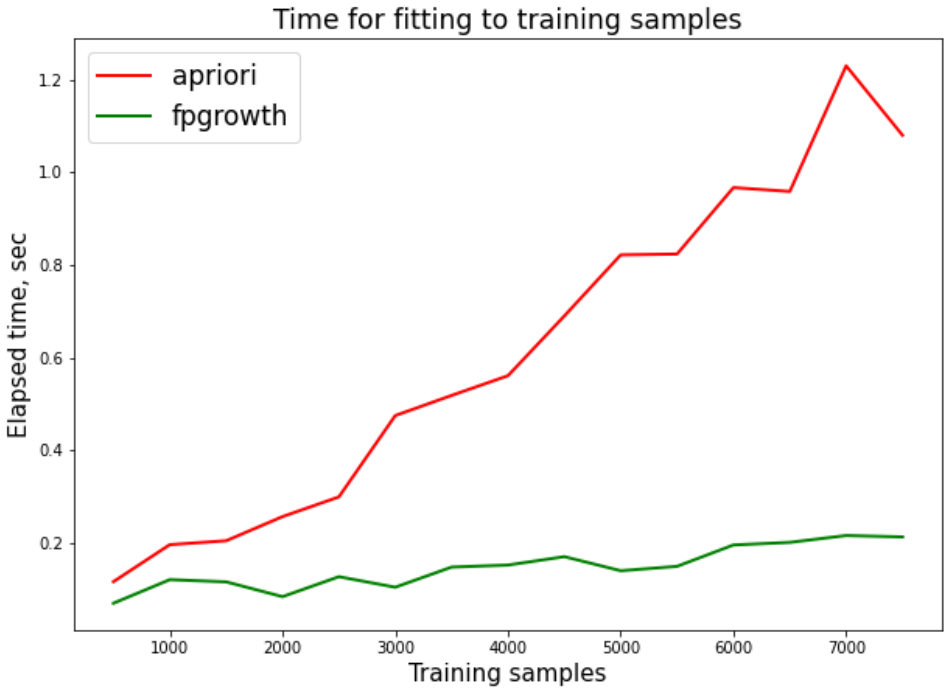

Рис. 14. Порівняння часу побудови АП з використанням досліджуваних алгоритмів

Результат виявився очікуваним: алгоритм FPG значно швидший за алгоритм Apriori. Але що дійсно є цікавим: зі збільшенням кількості транзакцій час на побудову за допомогою другого зростає значно швидше, в той час як побудова за допомогою алгоритму FP-росту зайняла трохи більше часу, ніж для меншої кількості.

Отже, можна зробити висновок, що для невеликої кількості даних (або для великих значень мінімальних порогових значень підтримки та достовірності), коли частих наборів буде не дуже багато, можна використовувати обидва алгоритми, але FPG має значно вищий пріоритет. Отже, значні переваги цього алгоритму:

1. Стиснення БД транзакцій в компактну структуру, що забезпечує дуже ефективний і повний пошук частих наборів;
2. При побудові FP-дерева використовується метод «поділяй та володарюй», який дозволяє виконати декомпозицію однієї складної задачі на безліч більш простих;
3. Дозволяє уникнути витратною процедури генерації кандидатів, характерною для алгоритму Apriori.

Тепер стисло опишемо основні переваги та недоліки двох алгоритмів у таблиці 3.

Таблиця 3. Переваги та недоліки розглянутих алгоритмів

Apriori	**FPG**
Переваги	
• простота; • швидке зменшення кількості згенерованих кандидатів при встановленні високого значення порогу мінімальної підтримки.	• дозволяє уникнути затратної процедури генерації кандидатів, характерної для Apriori; • стиснення БД в компактну структуру для швидкого отримання частих наборів; • число сканування БД зменшено до двох разів; • розмір дерева зазвичай є меншим за розмір вхідного набору даних.
Недоліки	
• багаторазове сканування БД транзакцій; • велика кількість згенерованих кандидатів при великому розмірі БД або при низькому значенні порогу мінімальної підтримки	• побудова дерева – витратна за часом операція.

Додаток А. Результати варіювання параметрів алгоритму Apriori

```
[({'min_support': 0.001, 'max_len': 2, 'min_confidence': 0.3},
  {'n_lift>1': 273, 'n_lift>2': 41, 'n_lift>3': 13},
  0.4282709760000003),
 ({'min_support': 0.001, 'max_len': 2, 'min_confidence': 0.4},
  {'n_lift>1': 94, 'n_lift>2': 16, 'n_lift>3': 5},
  0.39142996699999966),
 ({'min_support': 0.001, 'max_len': 2, 'min_confidence': 0.5},
  {'n_lift>1': 15, 'n_lift>2': 8, 'n_lift>3': 1},
  0.4061488879999997),
 ({'min_support': 0.001, 'max_len': 3, 'min_confidence': 0.3},
  {'n_lift>1': 6861, 'n_lift>2': 3393, 'n_lift>3': 1090},
  7.599606851000001),
 ({'min_support': 0.001, 'max_len': 3, 'min_confidence': 0.4},
  {'n_lift>1': 3625, 'n_lift>2': 1864, 'n_lift>3': 541},
  7.3155202070000005),
 ({'min_support': 0.001, 'max_len': 3, 'min_confidence': 0.5},
  {'n_lift>1': 1976, 'n_lift>2': 1243, 'n_lift>3': 287},
  7.239386018999998),
 ({'min_support': 0.001, 'max_len': 4, 'min_confidence': 0.3},
  {'n_lift>1': 19590, 'n_lift>2': 13044, 'n_lift>3': 5689},
  16.048515173),
 ({'min_support': 0.001, 'max_len': 4, 'min_confidence': 0.4},
  {'n_lift>1': 11842, 'n_lift>2': 8216, 'n_lift>3': 3320},
  15.929122616),
 ({'min_support': 0.001, 'max_len': 4, 'min_confidence': 0.5},
  {'n_lift>1': 7584, 'n_lift>2': 5818, 'n_lift>3': 1988},
  15.493298261999996),
 ({'min_support': 0.005, 'max_len': 2, 'min_confidence': 0.3},
  {'n_lift>1': 134, 'n_lift>2': 9, 'n_lift>3': 2},
  0.23196116299999403),
 ({'min_support': 0.005, 'max_len': 2, 'min_confidence': 0.4},
  {'n_lift>1': 56, 'n_lift>2': 6, 'n_lift>3': 2},
  0.22806576899999698),
 ({'min_support': 0.005, 'max_len': 2, 'min_confidence': 0.5},
  {'n_lift>1': 6, 'n_lift>2': 3, 'n_lift>3': 0},
  0.22781461999998953),
 ({'min_support': 0.005, 'max_len': 3, 'min_confidence': 0.3},
  {'n_lift>1': 691, 'n_lift>2': 150, 'n_lift>3': 6},
  0.9604731390000012),
 ({'min_support': 0.005, 'max_len': 3, 'min_confidence': 0.4},
  {'n_lift>1': 401, 'n_lift>2': 81, 'n_lift>3': 5},
  0.9600921020000044),
 ({'min_support': 0.005, 'max_len': 3, 'min_confidence': 0.5},
  {'n_lift>1': 170, 'n_lift>2': 58, 'n_lift>3': 1},
  0.9615172160000043),
 ({'min_support': 0.005, 'max_len': 4, 'min_confidence': 0.3},
  {'n_lift>1': 819, 'n_lift>2': 241, 'n_lift>3': 29},
  1.0395975160000006),
 ({'min_support': 0.005, 'max_len': 4, 'min_confidence': 0.4},
  {'n_lift>1': 478, 'n_lift>2': 131, 'n_lift>3': 14},
  1.0299962419999957),
 ({'min_support': 0.005, 'max_len': 4, 'min_confidence': 0.5},
  {'n_lift>1': 224, 'n_lift>2': 92, 'n_lift>3': 4},
  1.0386726569999922),
 ({'min_support': 0.01, 'max_len': 2, 'min_confidence': 0.3},
  {'n_lift>1': 91, 'n_lift>2': 2, 'n_lift>3': 0},
  0.14523479499999326),
 ({'min_support': 0.01, 'max_len': 2, 'min_confidence': 0.4},
```

```
  {'n_lift>1': 33, 'n_lift>2': 0, 'n_lift>3': 0},
  0.14557575400000644),
 ({'min_support': 0.01, 'max_len': 2, 'min_confidence': 0.5},
  {'n_lift>1': 2, 'n_lift>2': 0, 'n_lift>3': 0},
  0.1453706620000048),
 ({'min_support': 0.01, 'max_len': 3, 'min_confidence': 0.3},
  {'n_lift>1': 220, 'n_lift>2': 30, 'n_lift>3': 0},
  0.27165683099998716),
 ({'min_support': 0.01, 'max_len': 3, 'min_confidence': 0.4},
  {'n_lift>1': 111, 'n_lift>2': 12, 'n_lift>3': 0},
  0.2711887150000081),
 ({'min_support': 0.01, 'max_len': 3, 'min_confidence': 0.5},
  {'n_lift>1': 35, 'n_lift>2': 9, 'n_lift>3': 0},
  0.27039950600000395),
 ({'min_support': 0.01, 'max_len': 4, 'min_confidence': 0.3},
  {'n_lift>1': 225, 'n_lift>2': 33, 'n_lift>3': 1},
  0.27719921999999997),
 ({'min_support': 0.01, 'max_len': 4, 'min_confidence': 0.4},
  {'n_lift>1': 113, 'n_lift>2': 13, 'n_lift>3': 0},
  0.2783582139999936),
 ({'min_support': 0.01, 'max_len': 4, 'min_confidence': 0.5},
  {'n_lift>1': 37, 'n_lift>2': 10, 'n_lift>3': 0},
  0.276583665000004)]
```

Додаток Б. Результати варіювання параметрів алгоритму FPG

```
[({'min_support': 0.001, 'max_len': 2, 'min_confidence': 0.3},
  {'n_lift>1': 273, 'n_lift>2': 41, 'n_lift>3': 13},
  0.6209496970000004),
 ({'min_support': 0.001, 'max_len': 2, 'min_confidence': 0.4},
  {'n_lift>1': 94, 'n_lift>2': 16, 'n_lift>3': 5},
  0.5957750379999993),
 ({'min_support': 0.001, 'max_len': 2, 'min_confidence': 0.5},
  {'n_lift>1': 15, 'n_lift>2': 8, 'n_lift>3': 1},
  0.6249206180000044),
 ({'min_support': 0.001, 'max_len': 3, 'min_confidence': 0.3},
  {'n_lift>1': 6861, 'n_lift>2': 3393, 'n_lift>3': 1090},
  0.8687539470000019),
 ({'min_support': 0.001, 'max_len': 3, 'min_confidence': 0.4},
  {'n_lift>1': 3625, 'n_lift>2': 1864, 'n_lift>3': 541},
  0.8815263290000104),
 ({'min_support': 0.001, 'max_len': 3, 'min_confidence': 0.5},
  {'n_lift>1': 1976, 'n_lift>2': 1243, 'n_lift>3': 287},
  0.8711390500000107),
 ({'min_support': 0.001, 'max_len': 4, 'min_confidence': 0.3},
  {'n_lift>1': 19590, 'n_lift>2': 13044, 'n_lift>3': 5689},
  1.0595207319999957),
 ({'min_support': 0.001, 'max_len': 4, 'min_confidence': 0.4},
  {'n_lift>1': 11842, 'n_lift>2': 8216, 'n_lift>3': 3320},
  1.028403114999989),
 ({'min_support': 0.001, 'max_len': 4, 'min_confidence': 0.5},
  {'n_lift>1': 7584, 'n_lift>2': 5818, 'n_lift>3': 1988},
  1.0087291459999932),
 ({'min_support': 0.005, 'max_len': 2, 'min_confidence': 0.3},
  {'n_lift>1': 134, 'n_lift>2': 9, 'n_lift>3': 2},
  0.25399068399998725),
 ({'min_support': 0.005, 'max_len': 2, 'min_confidence': 0.4},
  {'n_lift>1': 56, 'n_lift>2': 6, 'n_lift>3': 2},
  0.2549520099999967),
 ({'min_support': 0.005, 'max_len': 2, 'min_confidence': 0.5},
```

```
 {'n_lift>1': 6, 'n_lift>2': 3, 'n_lift>3': 0},
 0.2845047550000004),
({'min_support': 0.005, 'max_len': 3, 'min_confidence': 0.3},
 {'n_lift>1': 691, 'n_lift>2': 150, 'n_lift>3': 6},
 0.26654108099999974),
({'min_support': 0.005, 'max_len': 3, 'min_confidence': 0.4},
 {'n_lift>1': 401, 'n_lift>2': 81, 'n_lift>3': 5},
 0.3042196960000041),
({'min_support': 0.005, 'max_len': 3, 'min_confidence': 0.5},
 {'n_lift>1': 170, 'n_lift>2': 58, 'n_lift>3': 1},
 0.265647242),
({'min_support': 0.005, 'max_len': 4, 'min_confidence': 0.3},
 {'n_lift>1': 819, 'n_lift>2': 241, 'n_lift>3': 29},
 0.2959472269999992),
({'min_support': 0.005, 'max_len': 4, 'min_confidence': 0.4},
 {'n_lift>1': 478, 'n_lift>2': 131, 'n_lift>3': 14},
 0.2684282629999899),
({'min_support': 0.005, 'max_len': 4, 'min_confidence': 0.5},
 {'n_lift>1': 224, 'n_lift>2': 92, 'n_lift>3': 4},
 0.2953378100000066),
({'min_support': 0.01, 'max_len': 2, 'min_confidence': 0.3},
 {'n_lift>1': 91, 'n_lift>2': 2, 'n_lift>3': 0},
 0.18694126200000483),
({'min_support': 0.01, 'max_len': 2, 'min_confidence': 0.4},
 {'n_lift>1': 33, 'n_lift>2': 0, 'n_lift>3': 0},
 0.1823545299999978),
({'min_support': 0.01, 'max_len': 2, 'min_confidence': 0.5},
 {'n_lift>1': 2, 'n_lift>2': 0, 'n_lift>3': 0},
 0.1499203239999929),
({'min_support': 0.01, 'max_len': 3, 'min_confidence': 0.3},
 {'n_lift>1': 220, 'n_lift>2': 30, 'n_lift>3': 0},
 0.1894081669999963),
({'min_support': 0.01, 'max_len': 3, 'min_confidence': 0.4},
 {'n_lift>1': 111, 'n_lift>2': 12, 'n_lift>3': 0},
 0.18549402800000792),
({'min_support': 0.01, 'max_len': 3, 'min_confidence': 0.5},
 {'n_lift>1': 35, 'n_lift>2': 9, 'n_lift>3': 0},
 0.1807091059999948),
({'min_support': 0.01, 'max_len': 4, 'min_confidence': 0.3},
 {'n_lift>1': 225, 'n_lift>2': 33, 'n_lift>3': 1},
 0.18417362700000695),
({'min_support': 0.01, 'max_len': 4, 'min_confidence': 0.4},
 {'n_lift>1': 113, 'n_lift>2': 13, 'n_lift>3': 0},
 0.1832367089999991),
({'min_support': 0.01, 'max_len': 4, 'min_confidence': 0.5},
 {'n_lift>1': 37, 'n_lift>2': 10, 'n_lift>3': 0},
 0.1850121009999981)]
```

Лістинг

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import time
import warnings

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from mlxtend.frequent_patterns import apriori, association_rules, fpgrowth
from mlxtend.preprocessing import TransactionEncoder

warnings.filterwarnings("ignore", category=FutureWarning)


# # 1. Завантаження та первинна обробка даних

# In[2]:


dataset = pd.read_csv('Var_2_groceries - groceries.csv')
dataset.head()


# ## Видалення транзакцій, що мають лише 1 товар

# In[3]:


np.array(dataset['Item(s)'] == 1).sum()


# In[4]:


dataset = dataset.loc[dataset['Item(s)'] != 1]
dataset.head()


# ## Перетворення даних у необхідний вигляд для застосування алгоритмів побудови частих наборів

# In[5]:


def get_encoded_transactions(dataset):
    transaction_list = []
    for i in range(dataset.shape[0]):
        transaction_list.append(list(dataset.iloc[i, 1:(dataset.iloc[i, 0] +
1)]))
    encoder = TransactionEncoder()
    dataset = pd.DataFrame(encoder.fit_transform(transaction_list),
columns=encoder.columns_)
```

```
    return dataset


# In[6]:


data = get_encoded_transactions(dataset)
data.head()


# ## Візуалізація 20-ти найбільш популярних товарів

# In[7]:


quantities = [np.array(data.iloc[:, i] == True).sum() for i in
range(data.shape[1])]
best_sell_prod = list(zip(data.columns, quantities))
best_sell_prod.sort(key=lambda x: x[1], reverse=True)
best_sell_prod = np.array(best_sell_prod)

plt.figure(figsize=(17, 7))
plt.xlabel('Food Item', size='x-large')
plt.ylabel('Number of transactions', size='x-large')
plt.tick_params(labelsize=15)
plt.title('20 Most Sold Items',size='xx-large')
best_sell_prod = pd.Series(best_sell_prod[:, 1], index=best_sell_prod[:, 0],
name='Products quantities').astype('int64')
best_sell_prod.head(20).plot.bar(width=0.5,edgecolor='k',align='center',linewidt
h=1)
plt.savefig('Most_sold_items.png')


# ### Додаткові функції для отримання асоціативних правил та варіювання
параметрів алгоритмів

# In[8]:


def extend_zip(arrays, names=('min_support', 'max_len', 'min_confidence')):
    return [{names[0]: i, names[1]: j, names[2]: k}
            for i in arrays[0] for j in arrays[1] for k in arrays[2]]


# In[9]:


def get_association_rules(min_support, max_len, min_confidence,
use_apriori=True, dataset=data):
    t = time.process_time()
    if use_apriori:
        frequent_itemsets = apriori(dataset, min_support=min_support,
use_colnames=True, max_len=max_len)
    else:
        frequent_itemsets = fpgrowth(dataset, min_support=min_support,
use_colnames=True, max_len=max_len)
```

```
    rules = association_rules(frequent_itemsets, metric='confidence',
min_threshold=min_confidence)
    return frequent_itemsets, rules, time.process_time() - t


# In[10]:


def variate_params(min_supports, max_lens, min_confidences, use_apriori=True):
    params = extend_zip((min_supports, max_lens, min_confidences))
    results, time_hist = [], []
    for param in params:
        _, rules, el_time = get_association_rules(param['min_support'],
param['max_len'], param['min_confidence'], use_apriori)

        time_hist.append(el_time)
        results.append({'n_lift>1': np.where(np.array(rules.lift) > 1, 1,
0).sum(),
                        'n_lift>2': np.where(np.array(rules.lift) > 2, 1,
0).sum(),
                        'n_lift>3': np.where(np.array(rules.lift) > 3, 1,
0).sum()})
    return list(zip(params, results, time_hist))


# # 2. Варіювання параметрів алгоритму Apriori

# In[11]:


min_support = [0.001, 0.005, 0.01]
max_len = [2, 3, 4]
min_confidence = [0.3, 0.4, 0.5]
variate_params(min_support, max_len, min_confidence)


# ## Побудова значущих асоціативних правил за допомогою алгоритму Apriori

# In[12]:


frequent_itemsets, rules, el_time = get_association_rules(min_support=0.01,
max_len=3, min_confidence=0.5)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x:
len(x))
frequent_itemsets


# In[13]:


rules['improvement'] = rules.support / (rules['antecedent support'] *
rules['consequent support'])
rules.sort_values(by=['lift'], ascending=False)


# ## Візуалізація підтримки та достовірності побудованих правил
```

```
# In[14]:


plt.scatter(rules.support, rules.confidence, s=100, c='r')
plt.xlabel('support')
plt.ylabel('confidence')
plt.title('Effectiveness of AR', size='x-large')
plt.savefig('rules_apriori.png')


# # 3. Варіювання параметрів алгоритму FPGrowth

# In[15]:


variate_params(min_support, max_len, min_confidence, False)


# ## Побудова значущих асоціативних правил за допомогою алгоритму FPGrowth

# In[16]:


frequent_itemsets, rules, el_time = get_association_rules(min_support=0.01,
max_len=3, min_confidence=0.5, use_apriori=False)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x:
len(x))
frequent_itemsets


# In[17]:


rules['improvement'] = rules.support / (rules['antecedent support'] *
rules['consequent support'])
rules.sort_values(by=['lift'], ascending=False)


# ## Візуалізація підтримки та достовірності побудованих правил

# In[18]:


plt.scatter(rules.support, rules.confidence, s=100, c='g')
plt.xlabel('support')
plt.ylabel('confidence')
plt.title('Effectiveness of AR', size='x-large')
plt.savefig('rules_fpgrowth.png')


# # 4. Знаходження прогнозу на основі побудованої множини правил

# In[19]:


def make_predictions(predicted_item):
    _, rules, _ = get_association_rules(min_support=0.001, max_len=None,
min_confidence=0.01, use_apriori=False)
```

```
    result_rules = rules.loc[((rules.antecedents == frozenset(predicted_item)) |
(rules.consequents == frozenset(predicted_item)))
                             & (rules.lift > 1)]
    other_rules = rules.loc[((rules.antecedents != frozenset(predicted_item)) |
(rules.consequents != frozenset(predicted_item)))
                            & (rules.lift < 1)]
    return result_rules.sort_values(by=['lift'], ascending=False), other_rules


# In[20]:


def visualize_predictions(result_rules, other_rules, predicted_item):
    plt.figure(figsize=(15, 7))
    plt.scatter(other_rules.support, other_rules.confidence, label='Other AR',
s=15, c='g')
    plt.scatter(result_rules.support, result_rules.confidence,
label=f'{predicted_item} AR', s=15, c='r')

    plt.legend(fontsize=14)
    plt.xlabel('support', fontsize=14)
    plt.ylabel('confidence', fontsize=14)
    plt.title('Effectiveness of AR', size='x-large')


# ## Виконання передбачення для частого набору

# In[21]:


rules, other_rules = make_predictions({'whole milk'})
rules


# In[22]:


visualize_predictions(rules, other_rules, 'whole milk')
plt.savefig('common_product_effectiveness.png')


# ## Виконання прогнозу для рідкого набору

# In[23]:


frequent_itemsets.sort_values(by=['support']).head()


# In[24]:


rules, other_rules = make_predictions({'pip fruit', 'curd'})
rules


# In[25]:
```

```
visualize_predictions(rules, other_rules, 'pip fruit & curd')
plt.savefig('rare_product_effectiveness.png')


# In[26]:


rules.loc[rules.confidence > 0.6]


# # 5. Порівняння часу побудови асоціативних правил різними алгоритмами

# In[27]:


train_samples = range(500, dataset.shape[0], 500)
apriori_time, fpgrowth_time = [], []
for samples in train_samples:
    part_dataset = dataset.iloc[np.random.randint(0, dataset.shape[0], samples)]
    part_data = get_encoded_transactions(part_dataset)
    _, _, a_time = get_association_rules(min_support=0.005, max_len=None,
min_confidence=0.5, dataset=part_data)
    apriori_time.append(a_time)
    _, _, fp_time = get_association_rules(min_support=0.005, max_len=None,
min_confidence=0.5, use_apriori=False, dataset=part_data)
    fpgrowth_time.append(fp_time)
plt.figure(figsize=(10, 7))
plt.plot(train_samples, apriori_time, c='r', label='apriori', linewidth=2)
plt.plot(train_samples, fpgrowth_time, c='g', label='fpgrowth', linewidth=2)
plt.legend(fontsize=17)
plt.xlabel('Training samples', fontsize=15)
plt.ylabel('Elapsed time, sec', fontsize=15)
plt.title('Time for fitting to training samples', size='xx-large')
plt.savefig('train_time.png')
```

Висновок

У процесі виконання даного практикуму я дослідив два нових для себе алгоритми: Apriori та FP-Growth, які використовуються для побудови частих наборів товарів за заданими транзакціями, щоб, використавши ці дані, побудувати асоціативні правила, які дозволяють зрозуміти, які товари купуються разом, або ніколи не купуються разом. Ця процедура виконується, щоб:

- оптимізувати розміщення товарів на полицях;
- формувати персональні рекомендації;
- планувати рекламні кампанії;
- ефективніше керувати цінами й асортиментом.

Також я навчився будувати значущі (ті, що мають зміст) асоціативні правила та дослідив вплив різних параметрів алгоритмів на множину результатів.

Ця задача є цікавою не тільки з точки зору дата майнінгу, але й з чисто користувацького погляду на ці дослідження. Від цього виконання цієї роботи було дуже пізнавальним та корисним.