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 лабораторної роботи № 5
з дисципліни «Інтелектуальний аналіз даних»

Виконала:
Студентка III курсу
Групи КА-76
Оркуша А. Д.

Перевірила:
Недашківська Н. І.

Київ – 2020

Варіант 11

Var_3_groceries

Хід виконання роботи:

Опис Алгоритмів

Apriori застосовується до баз даних транзакцій (наприклад, наборів товарів, куплених клієнтами, або відвідуваності вебсайту).Кожна транзакція розглядається як множина елементів. Маючи заданий поріг C, алгоритм Apriori ідентифікує множину елементів, які є підмножинами принаймні C транзакцій в базі даних.

Апріорі використовує підхід «знизу вгору», за якого список частих підмножин розширюється по одному елементу за раз (крок, відомий як *генерування кандидатів*); відтак групи кандидатів перевіряються на основі наявних даних. Алгоритм завершує роботу, коли подальших успішних розширень знайти неможливо.

Apriori використовує пошук у ширину та структуру хеш-дерева для ефективного підрахунку елементів-кандидатів множини. Він генерує множини елементів-кандидатів довжиною k з множин довжиною k-1. Потім він відсікає кандидатів, які є нечастими підмножинами. Відповідно до леми низхідного змикання (англ. *downward closure lemma*), множина кандидатів містить усі множини елементів довжини k, які часто зустрічаються. Після цього він сканує базу даних транзакцій, щоб визначити множини елементів, які часто зустрічаються серед кандидатів.

Вузьким місцем в алгоритмі a priori є процес генерації кандидатів в популярні предметні набори. Наприклад, якщо база даних (БД) транзакцій містить 100 предметів, то буде потрібно згенерувати 2100 ~ 1030 кандидатів. Таким чином, обчислювальні і тимчасові витрати, які потрібні

на їх обробку, можуть бути неприйнятними. Крім цього, алгоритм a priori вимагає багаторазового сканування бази даних транзакцій, а саме стільки разів, скільки предметів містить найдовший предметний набір. Тому було запропоновано ряд алгоритмів, які дозволяють уникнути генерації кандидатів і скоротити необхідну кількість сканувань набору даних.

Одним з найбільш ефективних процедур пошуку асоціативних правил є алгоритм, який отримав назву Frequent Pattern-Growth (алгоритм FPG), що можна перевести як «вирощування популярних (часто зустрічаються) предметних наборів». Він дозволяє не тільки уникнути витратною процедури генерації кандидатів, але зменшити необхідне число проходів БД до двох.

Алгоритм Frequent Pattern-Growth Strategy (FPG)

В основі методу лежить попередня 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 назва алгоритму). Надалі для стислості будемо називати цю структуру FP-дерево. До основних переваг даного методу відносяться:

Стиснення БД транзакцій в компактну структуру, що забезпечує дуже ефективне і повне вилучення частих предметних наборів;

При побудові FP-дерева використовується технологія поділу і захоплення (англ .: divide and conquer), яка дозволяє виконати декомпозицію однієї складної задачі на безліч більш простих;

Дозволяє уникнути витратної процедури генерації кандидатів, характерної для алгоритму apriori.

1. Взяти файл з даними у відповідності з варіантом.

Зчитуємо файл формату csv в об'єкт DataFrame бібліотеки pandas. Після завантаження і відображення датафрейму стала зрозуміла розмірність і загальний вигляд даних, для наочності додали в параметри зчитування імена стовпчиків.

In [1]:

```
import pandas as pd

df = pd.read_csv('Var_3_groceries.csv',
                 names=[f'item{i}' for i in range(32)])
df
```

Out[1]:

	item0	item1	item2	item3	item4	item5	item
0	citrus fruit	semi-finished bread	margarine	ready soups	nan	nan	nan
1	tropical fruit	yogurt	coffee	nan	nan	nan	nan
2	whole milk	nan	nan	nan	nan	nan	nan
3	pip fruit	yogurt	cream cheese	meat spreads	nan	nan	nan
4	other vegetables	whole milk	condensed milk	long life bakery product	nan	nan	nan
5	whole milk	butter	yogurt	rice	abrasive cleaner	nan	nan
6	rolls/buns	nan	nan	nan	nan	nan	nan
7	other vegetables	UHT-milk	rolls/buns	bottled beer	liquor (appetizer)	nan	nan
8	potted plants	nan	nan	nan	nan	nan	nan
9	whole milk	cereals	nan	nan	nan	nan	nan
10	tropical fruit	other vegetables	white bread	bottled water	chocolate	nan	nan
11	citrus fruit	tropical fruit	whole milk	butter	curd	yogurt	flour
12	beef	nan	nan	nan	nan	nan	nan
13	frankfurter	rolls/buns	soda	nan	nan	nan	nan
14	chicken	tropical fruit	nan	nan	nan	nan	nan

9835 rows × 32 columns

Зберігаємо кількість транзакцій та максимальну кількість товарів у транзакції для подальших розрахунків

In [2]:

```
n_transactions = df.shape[0]
n_items = df.shape[1]
```

Підраховуємо кількість унікальних найменувань товарів у наборі даних

In [3]:

```
import numpy as np

unique_items = np.unique(df.to_numpy(dtype=str))
n_unique = len(unique_items)

n_unique
```

Out[3]:

```
170
```

Для застосування алгоритмів апріорі та FP росту необхідно привести дані до певного вигляду. Де рядки - транзакції, стовпчики - найменування товарів, елементи таблиці - булева змінна що відповідає тому чи був товар з імені стовпчику в транзакції з імені рядка. Наступний код трансформує дані у потрібний вигляд

In [4]:

```
rows = np.ndarray((n_transactions, n_unique), dtype=bool)
counts = np.zeros((n_unique,), dtype=int)
for row, j in zip(df.itertuples(), range(n_transactions)):
    row_modified = np.ndarray((n_unique,), dtype=bool)
    for i in range(n_unique):
        row_modified[i] = unique_items[i] in row
        counts[i] += row_modified[i]
    rows[j] = row_modified

groceries = pd.DataFrame(data=rows, columns=unique_items)

groceries
```

Out[4]:

	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	baby food	b
0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.000
1	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.000
2	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.000
3	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.000
4	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.000
5	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.000
6	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.000
7	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.000
8	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.000
9	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.000
10	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.000
11	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.000
12	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.000
13	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.000
14	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.000

9835 rows × 170 columns

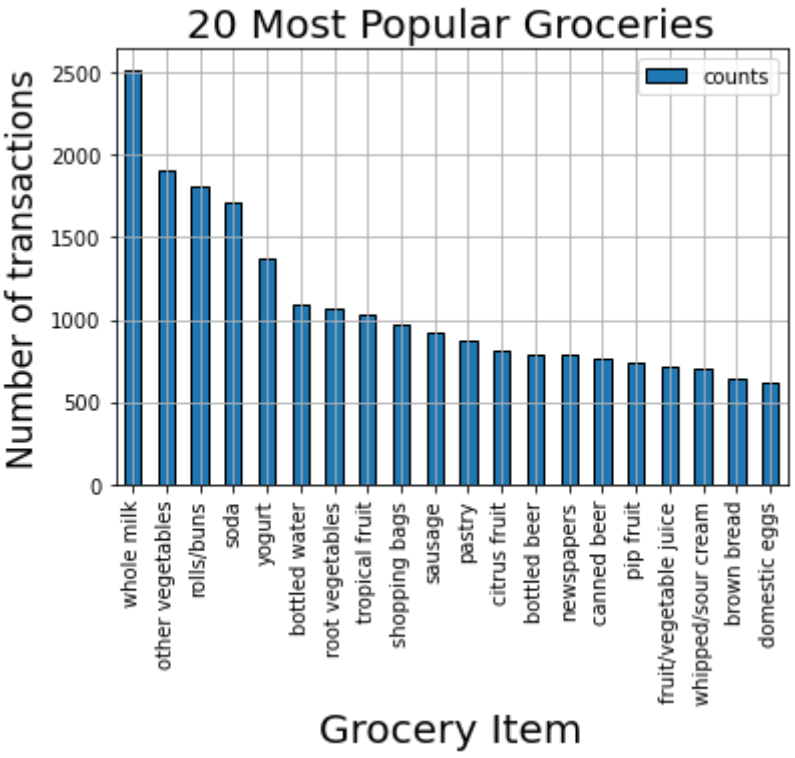
Графічно зобразимо які товари найчастіше присутні в транзакціях:

In [5]:

```
import matplotlib.pyplot as plt

counts = pd.DataFrame(data=counts, index=unique_items, columns=['counts'])

counts.sort_values(axis=0, by=['counts'], ascending=False).\
    head(20).plot.bar(width=0.5,edgecolor='k',align='center',
                      linewidth=1)
plt.xlabel('Grocery Item',fontsize=20)
plt.ylabel('Number of transactions',fontsize=17)
plt.title('20 Most Popular Groceries',fontsize=20)
plt.grid()
plt.ioff()
plt.show()
```



2. Побудувати часті набори та асоціативні правила (АП), використовуючи алгоритм Apriori. Дослідити множини АП залежно від параметрів побудови АП:

• мінімальної підтримки,

`min_support` : float (default: 0.5) = `transactions_where_item(s)_occur / total_transactions` Співвідношення між кількістю транзакцій до яких входив елемент до загальної кількості транзакцій. Нижче представлена функція побудови АП, яка приймає `min_support` як параметр. Користуючись цією функцією пробувала різні значення починаючи з дефолтного - 0.5 і поступово зменшуючи. Для 0.5 у результуючу множину не потрапило жодного правила, для 0.05 - 5 правил, зі зменшенням кількість АП збільшувалась. Для значення < 0.01 обчислювальних потужностей компютера починало не вистачати, тому зупинилася на 0.01.

• мінімальної достовірності,

`confidence(A->C) = support(A+C) / support(A), range: [0, 1]`- не є параметром побудови, проте готові правила сортуємо за дотовірністю, таким чином правила з найвищими показниками опиняються зверху, приймаємо рішення яку достовірність вважаємо достатньою аналізуючи результати.

• максимальної потужності частих наборів.

`max_len` : int (default: None) - цей параметр обмежує множину результуючих асоціативних правил за значенням підтримки зверху. Тобто якщо задати цей параметр не пустим або не достатньо великим можемо позбутися правил про найбільш часті набори , що не вигідно, тому залишимо None.

Підібрати значення параметрів алгоритму Apriori, які призводять до значущих АП. АП Rk вважати значущим, якщо покращення (improvement або ліфт (lift)): Improv(Rk) > 1

Щоб забезпечити цю умову при використанні алгоритму апріорі налаштуємо параметр `metric` : string (default: 'confidence') - lift, `min_threshold` : float (default: 0.8) - 1

Таким чином наша результуюча множина АП буде включати лише правила у яких метрика якості покращення більша за 1.

In [6]:

```
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

def tuning_rules(get_frequent_itemsets, min_support, max_len=None):

    frequent_itemsets = get_frequent_itemsets(groceries, min_support=min_support,
                                              use_colnames=True, max_len=max_len)
    if frequent_itemsets.shape[0] != 0:
        rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1) \
                        .sort_values(by=["confidence"], ascending=False)
    else:
        rules = None


    return frequent_itemsets, rules
```

In [7]:

```
import seaborn as sns
```

```
apriori_frequent_itemsets, apriori_rules = tuning_rules(apriori, 0.01)
apriori_rules.head(10)
```

Out[7]:

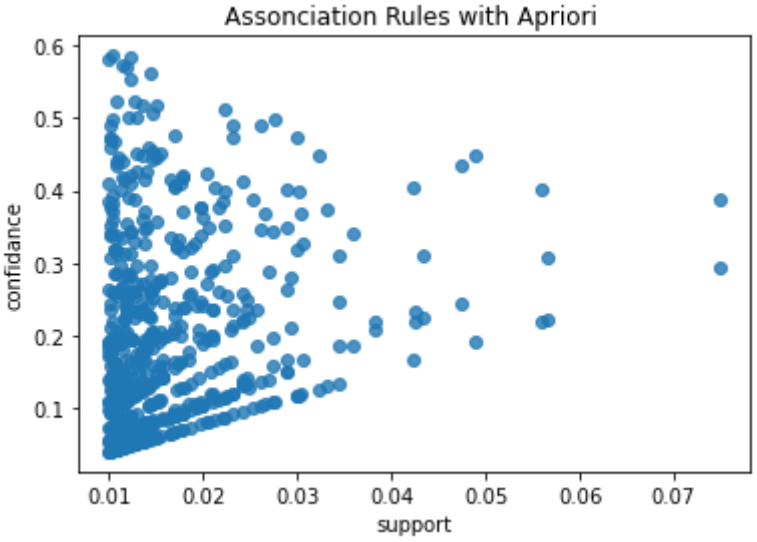
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	
418	frozenset({'root vegetables', 'citrus fruit'})	frozenset({'other vegetables'})	0.01769	0.19349	0.01037	0.58621	3.02961	0.0
490	frozenset({'root vegetables', 'tropical fruit'})	frozenset({'other vegetables'})	0.02105	0.19349	0.01230	0.58454	3.02100	0.0
436	frozenset({'curd', 'yogurt'})	frozenset({'whole milk'})	0.01729	0.25552	0.01007	0.58235	2.27913	0.0
413	frozenset({'butter', 'other vegetables'})	frozenset({'whole milk'})	0.02003	0.25552	0.01149	0.57360	2.24488	0.0
570	frozenset({'root vegetables', 'tropical fruit'})	frozenset({'whole milk'})	0.02105	0.25552	0.01200	0.57005	2.23097	0.0
575	frozenset({'yogurt', 'root vegetables'})	frozenset({'whole milk'})	0.02583	0.25552	0.01454	0.56299	2.20335	0.0
444	frozenset({'domestic eggs', 'other vegetables'})	frozenset({'whole milk'})	0.02227	0.25552	0.01230	0.55251	2.16234	0.0
592	frozenset({'whipped/sour cream', 'yogurt'})	frozenset({'whole milk'})	0.02074	0.25552	0.01088	0.52451	2.05275	0.0
552	frozenset({'rolls/buns', 'root vegetables'})	frozenset({'whole milk'})	0.02430	0.25552	0.01271	0.52301	2.04689	0.0
462	frozenset({'pip fruit', 'other vegetables'})	frozenset({'whole milk'})	0.02613	0.25552	0.01352	0.51751	2.02535	0.0
587	frozenset({'yogurt', 'tropical fruit'})	frozenset({'whole milk'})	0.02928	0.25552	0.01515	0.51736	2.02477	0.0
545	frozenset({'yogurt', 'other vegetables'})	frozenset({'whole milk'})	0.04342	0.25552	0.02227	0.51288	2.00723	0.0
533	frozenset({'whipped/sour cream', 'other vegetables'})	frozenset({'whole milk'})	0.02888	0.25552	0.01464	0.50704	1.98439	0.0
472	frozenset({'rolls/buns', 'root vegetables'})	frozenset({'other vegetables'})	0.02430	0.19349	0.01220	0.50209	2.59489	0.0
502	frozenset({'yogurt', 'root vegetables'})	frozenset({'other vegetables'})	0.02583	0.19349	0.01291	0.50000	2.58408	0.0

Покажемо усі отримані АП графічно відобразивши їх достовірність залежно від підтримки, це дозволяє краще зрозуміти загальну картину

In [8]:

```
support = np.array(apriori_rules['support'])
confidence = np.array(apriori_rules['confidence'])

plt.title(f"Assonciation Rules with Apriori ")
plt.xlabel('support')
plt.ylabel('confidance')
sns.regplot(x=support, y=confidence, fit_reg=False)
plt.show()
```



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

росту. Дослідити множини АП залежно від параметрів побудови АП, які було наведено вище для Apriori. Підібрати значення параметрів алгоритму FP-росту, які призводять до
значущих АП.

In [9]:

```
from mlxtend.frequent_patterns import fpgrowth

fp_frequent_itemsets, fp_rules = tuning_rules(fpgrowth, 0.01)
fp_rules.head(10)
```

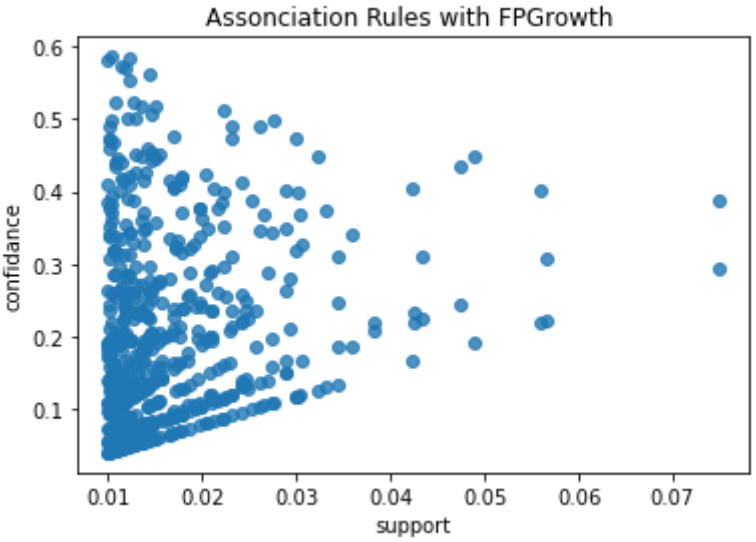
Out[9]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	
28	frozenset({'root vegetables', 'citrus fruit'})	frozenset({'other vegetables'})	0.01769	0.19349	0.01037	0.58621	3.02961	0.0
116	frozenset({'root vegetables', 'tropical fruit'})	frozenset({'other vegetables'})	0.02105	0.19349	0.01230	0.58454	3.02100	0.0
252	frozenset({'curd', 'yogurt'})	frozenset({'whole milk'})	0.01729	0.25552	0.01007	0.58235	2.27913	0.0
187	frozenset({'butter', 'other vegetables'})	frozenset({'whole milk'})	0.02003	0.25552	0.01149	0.57360	2.24488	0.0
124	frozenset({'root vegetables', 'tropical fruit'})	frozenset({'whole milk'})	0.02105	0.25552	0.01200	0.57005	2.23097	0.0
401	frozenset({'yogurt', 'root vegetables'})	frozenset({'whole milk'})	0.02583	0.25552	0.01454	0.56299	2.20335	0.0
572	frozenset({'domestic eggs', 'other vegetables'})	frozenset({'whole milk'})	0.02227	0.25552	0.01230	0.55251	2.16234	0.0
516	frozenset({'whipped/sour cream', 'yogurt'})	frozenset({'whole milk'})	0.02074	0.25552	0.01088	0.52451	2.05275	0.0
396	frozenset({'rolls/buns', 'root vegetables'})	frozenset({'whole milk'})	0.02430	0.25552	0.01271	0.52301	2.04689	0.0
158	frozenset({'pip fruit', 'other vegetables'})	frozenset({'whole milk'})	0.02613	0.25552	0.01352	0.51751	2.02535	0.0
93	frozenset({'yogurt', 'tropical fruit'})	frozenset({'whole milk'})	0.02928	0.25552	0.01515	0.51736	2.02477	0.0
73	frozenset({'yogurt', 'other vegetables'})	frozenset({'whole milk'})	0.04342	0.25552	0.02227	0.51288	2.00723	0.0
511	frozenset({'whipped/sour cream', 'other vegetables'})	frozenset({'whole milk'})	0.02888	0.25552	0.01464	0.50704	1.98439	0.0
388	frozenset({'rolls/buns', 'root vegetables'})	frozenset({'other vegetables'})	0.02430	0.19349	0.01220	0.50209	2.59489	0.0
406	frozenset({'yogurt', 'root vegetables'})	frozenset({'other vegetables'})	0.02583	0.19349	0.01291	0.50000	2.58408	0.0

In [10]:

```
support = np.array(fp_rules['support'])
confidence = np.array(fp_rules['confidence'])

plt.title(f"Assonciation Rules with FPGrowth ")
plt.xlabel('support')
plt.ylabel('confidance')
sns.regplot(x=support, y=confidence, fit_reg=False)
plt.show()
```



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

Детально вплив кожного з параметрів було розглянуто раніше. Мінімальну підтримку вказуємо в залежності від того яку частоту входження елементів до транзакцій вважаємо суттєвою. На графіку видно що вже 3% це доволі жорстке обмеження, що свідчить про те що дані розріджені. Вказувати максимальну підтримку можна тоді коли цікавлять правила з підтримкою лише з чітко заданого проміжку, якщо ж цікавлять “найкращі” з точки зору підтримки правила, цей параметр задавати не потрібно.

Достовірність вказує на те на скільки ми можемо бути впевненими що дане правило виконається. За цією метрикою множину АП відсортували, проте можна також у коді програми задати умову щоб додавати до результуючої множини АП лише правила з достовірністю більше бажаного значення. Тут цього не робили, тому що було цікаво подивитися на загальну картину.

5. Знайти значення прогнозу на основі побудованої множини правил.

Використовуючи множину правил `apriori_rules` або `fp_rules` прогноз можна записати як

(antecendents) => (consequents) імплікація не строга, з метриками якості відповідно до наступних стовпчиків.

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

Так як набір даних не дуже великий особливих відмінностей в роботі двох алгоритмів не було. Лише при заданні дуже маленьких значень мінімальної підтримки FP вилітав трохи пізніше. А результуючі множини АП однакові, оскільки це просто різні способи підрахунку одного і того самого.