ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

Курсова работа по

Приложен изкуствен интелект

Тема:

Правила за асоцииране.

Алгоритми: Apriori, Eclat, FP-growth

Изготвили:

Вяра Паскалева Паскова - 471219036,

Дебора Борисова Данчева - 471219037

ФПМИ, ИСН, 3 курс, 77 гр.

Съдържание

Асоциативни правила

Асоциативните правила са една от основните техники в областта Data mining. Те водят началото си от анализа на **market basket** (пазарната кошница), чиято цел е да бъде изследвано потребителското поведение на клиентите. Складираните данни се сканират за незабележими на пръв поглед зависимости в множествата от елементи ***(item sets)***, представени под формата на транзакции и най-често съхранявани в релационни бази от данни или хранилища.

Асоциативното правило има две части - **antecedent** *(if)* и **consequent** *(then)*. Antecedent е елемент (или множество от елементи) намерени в данните. Consequent е елемент (или множество от елементи), които са намерени в комбинация с antecedent. Правилата се записват във вида *If A then B (A→B).* Друго използвано означение за двете множества в асоциативните правила са *л*ява *(LHS - left-hand side)* и дясна страна *(RHS- right-hand side)*. Лявата съответства на antecedent, а дясната на consequent. Асоциативното правило представя зави симостта между множествата на antecedent и consequent. Съществуват две основни мерки, описващи зависимостите в правилата - ***support*** и ***confidence***. Support-а на асоциативното правило е частта от транзакции, която съдържа antecedent и consequent.

За правилото *A→B*, support-е равна на:

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

Confidence се равнява на процента от транзакциите, които съдържат *A*, но също ще съдържат и *B*:

- $confidence\ (A \rightarrow B) = support\ (A \cup B) / support\ (A)$

Алгоритми за извличане на често срещани елементи и създаване на асоциативни правила

Ще разгледаме основните три подхода за извличане на асоциативни правила. Поради огромното приложение на асоциативния анализ са представени много оптимизации.

Apriori

Същност

Основният алгоритъм за извличане на чести елементи и генериране на асоциативни правила е Apriori. Подходът приема, че ако едно множество се среща често, то и неговите съставни подмножества ще се съдържат често. С помощта на това свойство, алгоритъмът редуцира в голяма степен обема от транзакции за претърсване. Apriori извлича асоциативните правила чрез двустъпков процес - намиране на честите множества от елементи по зададени критерии за транзакциите и използването им за създаване на асоциативни правила, които трябва да отговарят на зададените support и confidence. Процесът на извличане на асоциативни правила включва няколко стъпки. Първоначално Apriori преминава през базата данни, изчислявайки support-а на всеки единичен елемент. За целта потребителят трябва да е задал минимален праг на support. Елементите, които не удовлетворяват зададеният минимум се отстраняват. На следващата стъпка се извършва генериране на кандидат- множества.

Изчислява се техния support и се сравнява с минималния праг. Процесът се повтаря докато се извлекат всички често срещани елементи. На третата стъпка се осъществява съставянето на

асоциативните правила. Чрез критериите "support" и "confidence" се намират наборите от елементи с най-голяма зависимост помежду им.

Как работи алгоритъмът Apriori?

За да изгради правила за асоцииране между елементи, алгоритъмът отчита 3 важни фактора, които са support, confidence и lift. Всеки от тези фактори се обяснява по следния начин:

Support:

Поддръжката на елемент I се дефинира като съотношението между броя на транзакциите, съдържащи елемент I, към общия брой транзакции, изразен като:

$$support(I) = \frac{Number\ of\ transactions\ containing\ I}{Total\ number\ of\ transactions}$$

Confidence:

Определя се от пропорцията на транзакциите съдържащи елемент I1 и елемент I2. Confidence между два елемента I1 и I2 в транзакция се дефинира като общия брой транзакции, съдържащи и двата елемента I1 и I2, се разделя на общия брой транзакции, съдържащи елемент I1.

$$confidence(I1 \rightarrow I2) = \frac{Number\ of\ transactions\ containing\ items\ I1\ and\ I2}{Total\ number\ of\ transactions\ containing\ I1}$$

Lift:

Lift е съотношението между confidence и support, изразено като:

$$lift(I1 \rightarrow I2) = \frac{confidence(I1 \rightarrow I2)}{support(I2)}$$

Нека за пример използваме следната таблица - можете да видите 5 транзакции, в които участват определени хранителни продукти.

ТР	хляб (Х)	месо (М)	бира (Б)	вода (В)	яйца (Я)
ТР1	1	1	1	0	0
ТР2	1	0	1	0	1
ТР3	0	1	1	1	0
ТР4	1	1	1	0	0
ТР5	1	1	0	1	0

Ако приемем, че сме задали като минимален праг 0.3, алгоритъмът ще премахне тези продукти, които се срещат в по-малко от 30% от всички транзакции.

Стъпка 1 – преглед на честотата на срещане на индивидуалните продукти.

Х	М	Б	В	*Я*
0.8	0.8	0.8	0.4	0.2

В нашия случай яйцата попадат под праговата стойност и съответно се премахват от по-нататъшните разглеждания. Apriori работи на принципа, че ако един продукт е рядко срещан, то комбинациите с него също са.

Стъпка 2 – преглед на комбинациите от 2 продукта

Х-М	Х-Б	*Х-В*	М-Б	М-В	*Б-В*
0.6	0.6	0.2	0.6	0.4	0.2

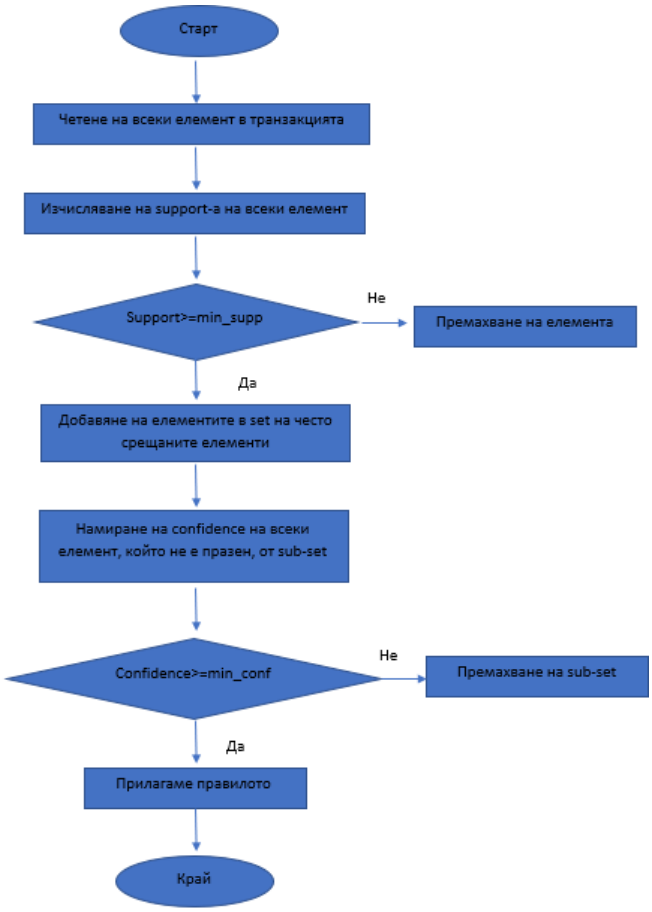
Две от комбинациите (хляб-вода и бира-вода) са под зададения праг и се премахват.

Стъпка 3 – преглед на комбинации от 3 продукта

Х-М-Б
0.4

Остава само една възможна комбинация, която е над зададения праг, след което алгоритъмът приключва работа.

Блок схема



Имплементация на алгоритъма Apriori на Python

```
import pandas as pd
from apyori import apriori
from IPython.display import display

Data = pd.read_csv('E:/Университет/AprioriAlgorithm/Market_Basket_Optimisation.csv', header=None)
# Intializing the list
transacts = []
# populating a list of transactions
for i in range(0, 7501):
    transacts.append([str(Data.values[i, j]) for j in range(0, 20)])

rule = apriori(transactions=transacts, min_support=0.003, min_confidence=0.2, min_lift=3, min_length=2,
                   max_length=2)

output = list(rule)  # returns a non-tabular output

# putting output into a pandas dataframe
def inspect(output):
    lhs = [tuple(result[2][0][0])[0] for result in output]
    rhs = [tuple(result[2][0][1])[0] for result in output]
    support = [result[1] for result in output]
    confidence = [result[2][0][2] for result in output]
    lift = [result[2][0][3] for result in output]
    return list(zip(lhs, rhs, support, confidence, lift))


output_DataFrame = pd.DataFrame(inspect(output),
                                columns=['Left_Hand_Side', 'Right_Hand_Side', 'Support', 'Confidence', 'Lift'])
display(output_DataFrame)
display(output_DataFrame.nlargest(n = 10, columns = 'Lift'))
```

Стъпка 1: Предварителна обработка на данните

Тази стъпка включва импортиране на библиотеките и по-късно трансформиране на нашите данни в подходящ формат за априорния модел. Следователно, първото нещо, което ще направим е да инсталираме пакет apyori, съдържащ всички алгоритми на априорния модел.

Инсталиране на необходимия пакет

```
pip install apyori
```

Импортиране на библиотеките

По-долу е кодът, който импортира необходимите библиотеки, с които ще взаимодействаме в рамките на тази програма.

```
import pandas as pd # lib for data analysis
from apyori import apriori
from IPython.display import display
```

Импортиране на набора от данни

Нашият набор от данни не съдържа имена на колони. Следователно, в етапа на импортиране, трябва да посочим това с помощта на аргумента заглавие; в противен случай python ще третира първото наблюдение като имена на колони, което не трябва да е така. По-долу е кодът, който импортира нашия набор от данни като dataframe на pandas.

```
Data = pd.read_csv('../Market_Basket_Optimisation.csv', header = None)
```

Преобразуване на нашия panda набор от данни в набор от данни за списък

Тъй като ще демонстрираме априорен алгоритъм, който приема входни данни в списъчен формат, трябва да трансформираме данните от CSV файла в списък с транзакции. За да създадем този списък, започваме с инициализиране на празен списък. След това го попълваме с различни транзакции от данните.

За да постигнем това, ще използваме for-loop цикъл, който ще обхожда нашия набор от данни и ще попълни празния ни списък като добавя извлечените транзакции ред по ред.

Ще използваме два for-цикла, единият за итериране на всички 7501 различни транзакции, а вторият за 20-те колони, така че да добавим всички елементи от всяка транзакция.

Следният код трансформира нашия набор от данни в списък с транзакции.

```
# Intializing the list
transacts = []
# populating a list of transactions
for i in range(0, 7501):
  transacts.append([str(Data.values[i,j]) for j in range(0, 20)])
```

До този момент етапът на предварителна обработка на данните за нашия априорен алгоритъм е завършен. Следва прилагането на самия алгоритъм

Стъпка 2: Извикване на априорния алгоритъм

За имплементацията на самия алгоритъм използвахме готовата библиотека apriori, като зададохме следните аргументи:

- Списъкът от транзакциите
- Minimal support
- Minimal confidence
- Minimal lift
- Minimal length
- Maximum length

```
from apyori import apriori
rule = apriori(transactions = transacts, min_support = 0.003, min_confidence = 0.2, min_lift = 3,
min_length = 2, max_length = 2)
```

Стъпка 3: Визуализация на резултатите

Тъй като apriori методът връща резултат от тип RelationRecord, който е трудно четим, за да направим резултата по-лесен за интерпретация, трансформирахме данните по следния начин:

```
output = list(rule) # returns a non-tabular output
# putting output into a pandas dataframe
def inspect(output):
   lhs         = [tuple(result[2][0][0])[0] for result in output]
   rhs         = [tuple(result[2][0][1])[0] for result in output]
   support    = [result[1] for result in output]
   confidence = [result[2][0][2] for result in output]
```

```
    lift      = [result[2][0][3] for result in output]
    return list(zip(lhs, rhs, support, confidence, lift))
output_DataFrame = pd.DataFrame(inspect(results), columns = ['Left_Hand_Side', 'Right_Hand_Side',
'Support', 'Confidence', 'Lift'])
```

Показване на резултатите, сортирани по низходящ ред спрямо Lift

```
display(output_DataFrame.nlargest(n = 10, columns = 'Lift'))
```

Анализ на резултати

От подредбата на резултатите можем да направим заключение, че мед се купува най-много, когато се купува и formage blanc. Следователно, ако искаме да продадем повече мед, то може да пуснем formage blanc на промоция.

	Left_Hand_Side	Right_Hand_Side	Support	Confidence	Lift
3	fromage blanc	honey	0.003333	0.245098	5.164271
0	light cream	chicken	0.004533	0.290598	4.843951
2	pasta	escalope	0.005866	0.372881	4.700812
8	pasta	shrimp	0.005066	0.322034	4.506672
7	whole wheat pasta	olive oil	0.007999	0.271493	4.122410
5	tomato sauce	ground beef	0.005333	0.377358	3.840659
1	mushroom cream sauce	escalope	0.005733	0.300699	3.790833
4	herb & pepper	ground beef	0.015998	0.323450	3.291994
6	light cream	olive oil	0.003200	0.205128	3.114710

Eclat

Същност

Алгоритъмът ECLAT (Equivalence Class Clustering and bottom-up Lattice Traversal) е Data mining алгоритъм за извличане на правила за асоцииране, предназначени за решаване на проблеми с анализа на market basket. Целта е да се разбере кои продукти обикновено се купуват заедно.

На всеки етап от генерираната база данни, алгоритъмът Eclat използва текущия генериран набор от данни, за да извлече често срещан набор от елементи, за разлика от Apriori, който сканира оригиналната база данни многократно. Тъй като Eclat сканира базата данни веднъж, това е много по-бързо от алгоритъма Apriori.

Това обаче не означава, че алгоритъмът Apriori е по-лош. Напротив, когато работим с по-голям набор от данни, Apriori има най-добро изпълнение. Eclat работи по-добре с малки и средни набори от данни.

Докато повечето алгоритми за извличане на асоциативни правила съдържат редица ключови показатели за техните правила, при ECLAT не е така. Например, ECLAT не предлага показателите Confidence и Lift, които са от съществено значение за интерпретацията в алтернативните модели. Тук имаме само support, който показва колко пъти даден елемент е в базата данни. От друга страна, това позволява на модела да бъде по-бърз: потребителят има избор между скорост и повече показатели.

Важно е да се отбележи, че Eclat работи добре с вертикален формат на данни. Тъй като повечето набори от данни са в хоризонтален формат, за да приложим алгоритъма на Eclat, първо трябва да ги преобразуваме във вертикален формат.

По-долу са дадени примери за хоризонтални и вертикални формати на данни.

Хоризонтален:

TID	Items
1	Bread,Butter,Jam
2	Butter,Coke
3	Butter,Milk
4	Bread,Butter,Coke
5	Bread,Milk
6	Butter,Milk
7	Bread,Milk
8	Bread,Butter,Milk,Jam
9	Bread,Butter,Milk

Вертикален:

Item Set	TID set
Bread	1,4,5,7,8,9
Butter	1,2,3,4,6,8,9
Milk	3,5,6,7,8,9
Coke	2,4
Jam	1,8

Сравнение между Eclat и Apriori

Eclat е въведен по-късно от Apriori, за да го подобри. Между тези два алгоритъма имаме значителни разлики:

- За разлика от алгоритъма Apriori, който е приложим с хоризонтален набор от данни, алгоритъмът Eclat е приложим само с набор от данни във вертикален формат.
- По-малко параметри при Eclat (само min support)

Предимства

1. Алгоритъмът Eclat има ниски изисквания за памет в сравнение с Apriori, тъй като използва подхода Depth-First Search

2. Алгоритъмът Eclat не сканира многократно данните, за да открие често срещани набори от елементи, следователно е по-бърз от алгоритъма Apriori.

3. Алгоритъмът Eclat превъзхожда алгоритъма Apriori, при условие че наборът от данни не е твърде голям.

Недостатъци

- Ако списъкът с транзакции, в които участва елемент, е твърде голям, алгоритъмът Eclat може да свърши без памет.

Как работи алгоритъмът Eclat?

Стъпки

Стъпка 1 — Избройте набора ID на транзакция (TID) на всеки продукт

Първата стъпка е да направите списък, който съдържа за всеки продукт списък с id на транзакции, в които се среща продуктът. Този списък е представен в следващата таблица.

Product	Transaction ID set
Wine	1, 5, 7, 9, 15, 16, 17, 19
Cheese	1, 3, 5, 7, 15, 16, 17, 19
Beer	1, 2, 4, 8, 9, 10, 12, 14, 16, 18, 20
Potato Chips	2, 4, 6, 8, 10, 12, 14, 16, 18, 20
Eggs	3, 4, 7, 8, 11, 13, 15
Flour	3, 4, 7, 8, 13, 15
Butter	3, 4, 7, 8, 11, 15

(TID) sets за всеки продукт

Тези списъци с id на транзакции се наричат набор от id на транзакция, наричан още TID набор (TID set).

Стъпка 2 — Филтриране с минималния support

Следващата стъпка е да вземете решение относно стойността на минималния support. Минималния support ще служи за филтриране на продукти, които не се срещат достатъчно често, за да е необходимо да бъдат разгледани.

В настоящия пример ще изберем стойност 7 за min support. Както можете да видите в таблицата на стъпка 1, има два продукта, които имат TID set, който съдържа по-малко от 7 транзакции: брашно и масло. Следователно ще ги филтрираме и ще получим следната таблица:

Product	Transaction ID set
Wine	1, 5, 7, 9, 15, 16, 17, 19
Cheese	1, 3, 5, 7, 15, 16, 17, 19
Beer	1, 2, 4, 8, 9, 10, 12, 14, 16, 18, 20
Potato Chips	2, 4, 6, 8, 10, 12, 14, 16, 18, 20
Eggs	3, 4, 7, 8, 11, 13, 15

Филтриране на продукти, които не достигат min support.

Стъпка 3 — Изчислете Transaction ID set за всяка двойка продукти

Сега преминаваме към двойки продукти. Повтаря се същото като в стъпка 1, но сега за двойки продукти.

Интересното за алгоритъма ECLAT е, че тази стъпка се извършва с помощта на пресечната точка на двата оригинални набора. Това го прави различен от алгоритъма Apriori.

Алгоритъмът ECLAT е по-бърз, тъй като е много по-лесно да се идентифицира пресечната точка на transactions IDs и, отколкото да се сканира всяка отделна транзакция за наличието на двойки продукти (както прави Apriori). Можете да видите на изображението по-долу колко е лесно да филтрирате transaction IDs, които са общи между продуктовата двойка вино и сирене:

Wine	**1**, -, **5, 7**, 9, **15, 16, 17, 19**
Cheese	**1**, 3, **5, 7**, - , **15, 16, 17, 19**

Намирането на пресечната точка на transaction IDs е по-лесно от сканирането на цялата база данни

Когато правите пресечната точка за всяка двойка продукти (игнорирайки продуктите, които не са достигнали support-а поотделно), това дава следната таблица:

Product Pair	Transaction ID set
Wine, Cheese	1, 5, 7, 15, 16, 17, 19
Wine, Beer	1, 9, 16
Wine, Potato Chips	16
Wine, Eggs	7, 15
Cheese, Beer	1, 16
Cheese, Potato Chips	16
Cheese, Eggs	3, 7, 15
Beer, Potato Chips	2, 4, 8, 10, 12, 14, 16, 18, 20
Beer, Eggs	4, 8
Potato Chips, Eggs	4, 8

Transaction ID sets се задава за всички двойки продукти, които все още са останали.

Стъпка 4 — Филтрирайте двойките, които не достигат минималния support

Както и преди, трябва да филтрираме резултатите, които не достигат min support от 7. Така получаваме само две оставащи двойки продукти: вино и сирене и бира и картофен чипс.

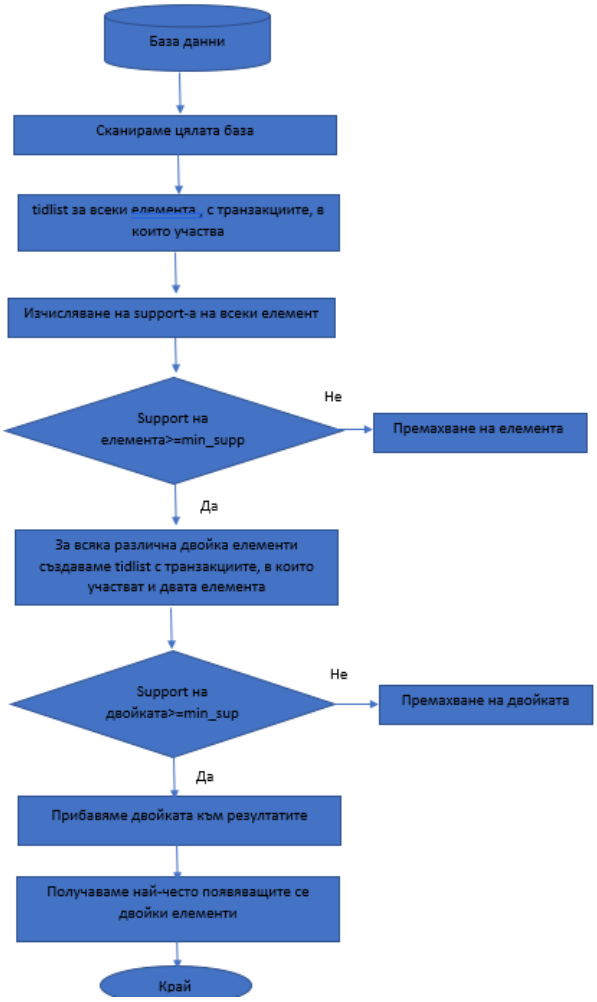
Product Pair	Transaction ID set
Wine, Cheese	1, 5, 7, 15, 16, 17, 19
Beer, Potato Chips	2, 4, 8, 10, 12, 14, 16, 18, 20

Има две двойки продукти, които отговарят на min support.

Стъпка 5— Продължете, докато можете да направите нови двойки над min support

От този момент нататък повтаряте стъпките възможно най-дълго. За настоящия пример, ако създадем продуктовите двойки от три продукта, ще откриете, че няма групи от три, които достигат min support. Следователно правилата за асоцииране ще бъдат тези, получени в предишната стъпка.

Блок схема



Имплементация на алгоритъма Eclat на Python

```
import pandas as pd # to deal with dataframe
from IPython.display import display
from pyECLAT import ECLAT
from pyECLAT import Example2

dataset = Example2().get()

# create an instance of eclat
my_eclat = ECLAT(data=dataset, verbose=True)

# fit the algorithm
rule_indices, rule_supports = my_eclat.fit(min_support=0.003,
                                           min_combination=2,
                                           max_combination=2)

result = pd.DataFrame(rule_supports.items(),columns=['Item', 'Support'])
display(result.nlargest(n = 10, columns = 'Support'))
```

Стъпка 1: Зареждане на данните

Инсталиране на необходимия пакет

```
pip install pyECLAT
```

Импортиране на библиотеките

По-долу е кодът, който импортира необходимите библиотеки, с които ще взаимодействаме в рамките на тази програма.

```
import pandas as pd
from IPython.display import display
from pyECLAT import ECLAT
from pyECLAT import Example2

```

Пакетът Eclat има две dataframes на panda, несъдържащи името на колоната.
За входящи данни за тази демонстрация ще използваме данните от пример 2.

```
dataset = Example2().get()
```

Стъпка 2: Извикване на Eclat метода

Зададохме следните аргументи при викането на метода:

- data
- verbose - за да видим инициатор за зареждането

```
my_eclat = ECLAT(data=dataset, verbose=True)
```

След получаване на eclat_instance, автоматично се генерира binary dataframe, наред с другите ресурси, които могат да бъдат достъпни:

- eclat_instance.df_bin - генерира binary dataframe, който може да се използва за други анализи.
- eclat_instance.uniq_ - списък с всички имена на различните елементи
- eclat_instance.support, eclat_instance.fit и eclat_instance.fit_all са функциите за извършване на изчисленията

Възможен е достъп до документацията, както и описанието на всеки метод, като се използва:

- help(eclat_instance.fit)
- help(eclat_instance.fit_all)
- help(eclat_instance.support)

Използваме метода fit(), като му поддаваме минимален support и да прави комбинации само от по два елемента.

```
# fit the algorithm
rule_indices, rule_supports = my_eclat.fit(min_support=0.03,
                                           min_combination=2,
                                           max_combination=2)
```

Стъпка 3: Визуализация на резултатите

Ще визуализираме резултатите в две колони – Item и Support, като ще сортираме по Support.

```
result = pd.DataFrame(rule_supports.items(),columns=['Item', 'Support'])
display(result.nlargest(n = 10, columns = 'Support'))
```

Анализ на резултати

Item	Support
mineral water & spaghetti	0.060646
milk & mineral water	0.048317
mineral water & eggs	0.047984
chocolate & mineral water	0.047651
milk & spaghetti	0.038654
chocolate & spaghetti	0.037321
ground beef & mineral water	0.036988
mineral water & frozen vegetables	0.036988
ground beef & spaghetti	0.035988
french fries & eggs	0.034322

Можем да направим заключение, че най-купуваната двойка продукти е минерална вода и спагети. Това означава, че ако искаме да продадем повече спагети, може например да пуснем минералната вода на промоция.

Frequent Pattern Growth Algorithm - Алгоритъм за чест растеж на модела

Същност

Този алгоритъм е подобрение на метода на Apriori. Алгоритъм Frequent Pattern Growth се използва за намиране на често срещани елементи в база данни за транзакции без генериране на set от елементи.

Frequent Pattern Growth алгоритъмът представлява често срещани елементи, представени под формата на дървовидна структура или FP-дърво

FP дърво

FP дървото е дървовидна структура, която се прави с първоначалните набори от елементи на базата данни. Целта на FP дървото е да изведе най-често срещания модел. Всеки възел на FP дървото представлява елемент от набора от елементи. Коренният възел представлява нула, докато долните възли представляват наборите елементи. Асоциацията на възлите се поддържа, докато се формира дървото.

Да разгледаме следния пример:

Transaction ID	Items
T1	{E, K, M, N, O, Y}
T2	{D, E, K, N, **O**, Y}
T3	{A, E, K, M}
T4	{C, K, M, U, Y}
T5	{C, E, I, K, O, O}

Посочените по-горе данни са хипотетичен набор от данни от транзакции, като всяка буква представлява елемент.

1. *Честотата на всеки отделен елемент се изчислява:*

Item	Frequency
A	1
C	2
D	1
E	4
I	1
K	5
M	3
N	2
0	3
U	1
Y	3

2. *Създаване на Frequent Pattern set*

Нека минималния support е 3. Изгражда се **Frequent Pattern set** , който ще съдържа всички елементи, чийто support е по-голям или равен на min support. Тези елементи се съхраняват в низходящ ред на съответните им честоти. След вмъкване на съответните елементи, **Frequent Pattern set** изглежда така:

L = {K : 5, E : 4, M : 3, O : 3, Y : 3}

3. *Сортиране на транзакциите*

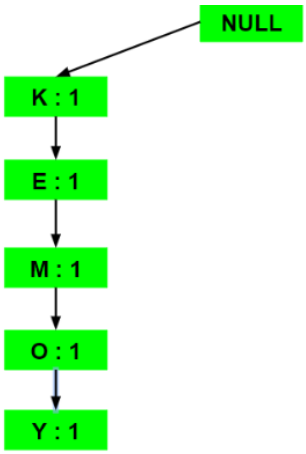
Сега за всяка транзакция се изгражда съответният Ordered-Item set. Това се прави чрез итериране Frequent Pattern set-а и проверка дали текущият артикул се съдържа във въпросната транзакция. Ако текущият артикул се съдържа, артикулът се вмъква в Ordered-Item за текущата транзакция. Следната таблица е изградена за всички транзакции:

Transaction ID	Items	Ordered-Item Set
T1	{E, K, M, N, O, Y}	{K, E, M, O, Y}
T2	{D, E, K, N, O, Y}	{K, E, O, Y}
T3	{ A, E, K, M}	{K, E, M}
T4	{C, K, M, U, Y}	{K, M, Y}
T5	{C, E, I, K, O, O}	{K, E, O}

4. *Изграждане на дървовидна структура*

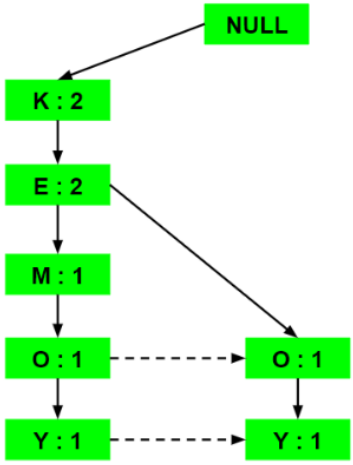
- Вмъкване на набора {K, E, M, O, Y}:

Дървото винаги започва с нулев възел. За всеки елемент от набора добавяме нов възел, където записваме съответния елемент и неговия support, като в случая започваме от едно

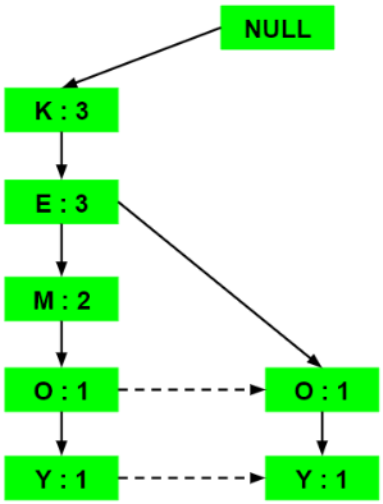


- Вмъкване на набора {K, E, O, Y}:

Тъй като вече имаме последователните възли за К и Е, създаваме разклонение от Е и добавяме нов възел за елемента О с support 1 и увеличаваме support-а на К и Е с единица

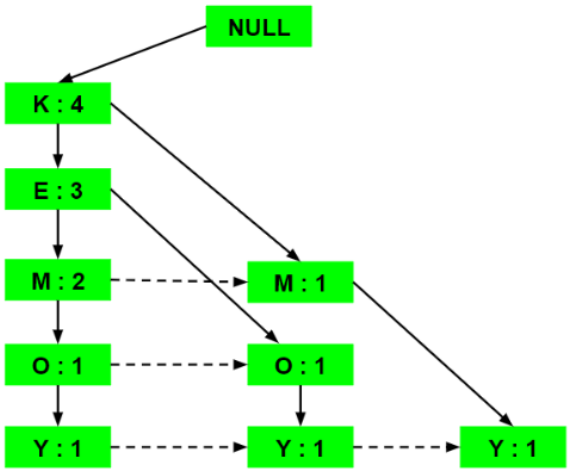


- Вмъкване на набора {K, E, M}:
  Понеже дървото разполага с последователни възли К Е и М няма нужда от добавяне на нови възли, само увеличаваме support-а на всеки един от тях с единица

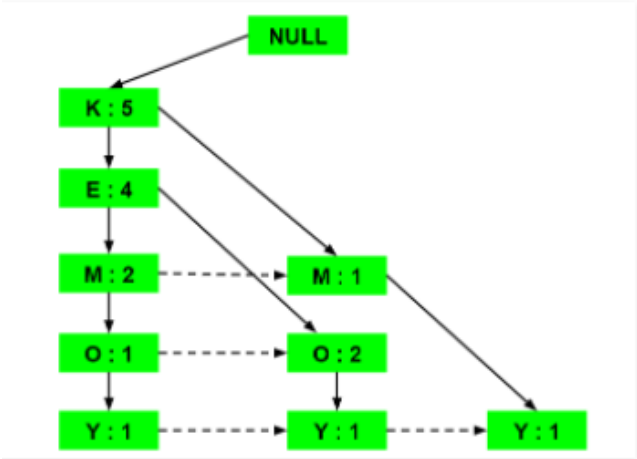


- Вмъкване на набора {K, M, Y}:

Аналогично добавяме нов възел Y с support 1



- Вмъкване на набора **{K, E, O}:**
  Увеличаваме support на всеки от елементите в набора с единица



5. *Изчисляване на Conditional Pattern Base*

За всеки елемент калкулираме пътищата, които водят до негови възли като обърнете внимание, че елементите са сортирани по възходящ ред спрямо support-а им.

Items	Conditional Pattern Base
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}
O	{{K,E,M : 1}, {K,E : 2}}
M	{{K,E : 2}, {K : 1}}
E	{K : 4}
K	

6. *Изчисляване на Conditional Frequent Pattern Tree*

За всеки елемент определяме набора от елементи, който се съдържа във всички негови пътища в Confitional Pattern Base и калкулираме support-а на съответния набор от елементи като сумираме support-а му във всички пътища.

Items	Conditional Pattern Base	Conditional Frequent Pattern Tree
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}	{K : 3}
O	{{K,E,M : 1}, {K,E : 2}}	{K,E : 3}
M	{{K,E : 2}, {K : 1}}	{K : 3}
E	{K : 4}	{K : 4}
K		

7. *Генериране на Frequent Pattern rules*

Frequent Pattern rules се генерират чрез сдвояване на елементите от Conditional Frequent Pattern Tree със съответния им елемент, както е дадено в таблицата по-долу.

Items	Frequent Pattern Generated
Y	{<K,Y : 3>}
O	{<K,O : 3>, <E,O : 3>, <E,K,O : 3>}
M	{<K,M : 3>}
E	{<E,K : 4>}
K	

За всеки ред могат да бъдат изведени два типа правила за асоцииране, например за първия ред, могат да бъдат изведени правилата K -> Y и Y -> K. За да се определи валидното правило, се изчислява confidence-а и на двете правила и се запазва това с confidence, по-голямо или равно на минималната стойност на confidence.

Пример:

Ако разглеждаме първия Frequent Pattern rule за елемен Y, виждаме, че можем да създадем две асоциативни правила Y -> K и K -> Y. За да сравним техните confidence-и, първо трябва да ги изчислим:

За Y -> K Confidence = 3/5 = 0.6

За K -> Y Confidence = 3/3 = 1

Ако минималния ни праг на Confidence е 1, валидното асоциативно правила ще бъде K -> Y

Предимства и недостатъци

Предимства на алгоритъма за растеж на FP

- Този алгоритъм трябва да сканира базата данни само два пъти в сравнение с Apriori, който сканира транзакциите за всяка итерация.
- Сдвояването на елементи не се извършва в този алгоритъм и това го прави по-бързо.
- Той е ефективен и мащабируем в Data Mining както на дълги, така и на къси честотни модели.

Недостатъци на FP-алгоритъм за растеж

- FP Tree е по-тромаво и трудно за изграждане от Apriori.
- Може да е скъпо.
- Когато базата данни е голяма, алгоритъмът може да не се побере в споделената памет.

FP Growth в сравнение с Apriori

FP Growth	Apriori
генерира модел чрез конструиране на FP дърво	генерира шаблон чрез сдвояване на елементите в единично, в двойки и тройки.
Няма генериране на кандидати	Има генериране на кандидати
Процесът е по-бърз в сравнение с Apriori. Времето на изпълнение на процеса нараства линейно с увеличаване на броя на наборите от елементи.	Процесът е сравнително по-бавен от растежа на FP, времето за изпълнение нараства експоненциално с увеличаване на броя на наборите от елементи
Записва се компактна версия на базата данни	Комбинациите от кандидати се записват в паметта

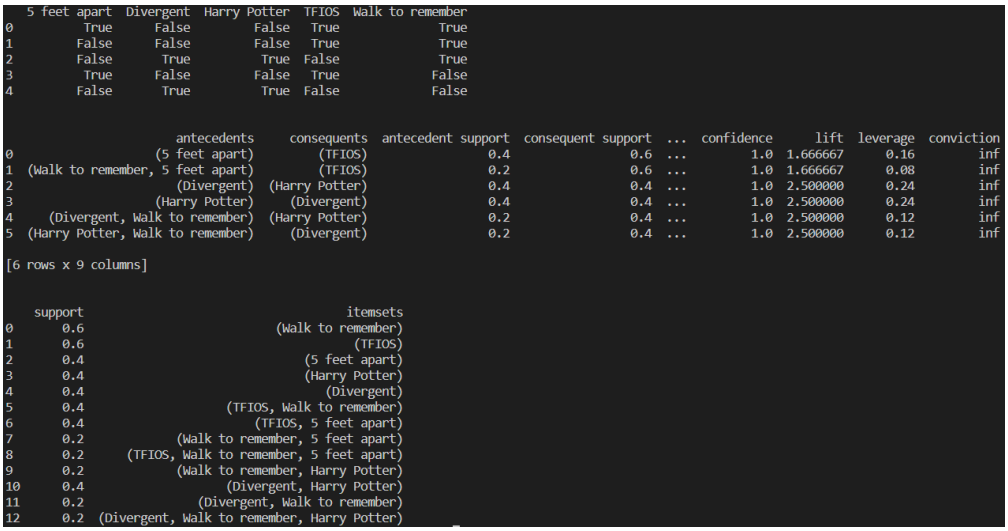
Имплементация на алгоритъма FP Growth на Python

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import association_rules

dataset = [['TFIOS', '5 feet apart', 'Walk to remember'],
['TFIOS', 'Walk to remember'],
['Divergent', 'Harry Potter','Walk to remember'],
['TFIOS', '5 feet apart'],
['Divergent','Harry Potter']]
te = TransactionEncoder()
te_array = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_array, columns=te.columns_)

from mlxtend.frequent_patterns import fpgrowth
frequent_itemsets_fp=fpgrowth(df, min_support=0.2, use_colnames=True)
rules_fp = association_rules(frequent_itemsets_fp, metric="confidence", min_threshold=0.8)
print(df)
print("\n")
print(rules_fp)
print("\n")
print(frequent_itemsets_fp)
```

Отново използваме библиотеката pandas за анализ на данни, както и mlxtend, която е разширена библиотека и има вградени методи за FPG алгоритъма.

Наборът от данни се зарежда в променлива. Това включва таблица с транзакции, съдържаща набор от елементи. Върху таблицата се създава Data Frame.

Анализ на резултати



```
   5 feet apart  Divergent  Harry Potter  TFIOS  Walk to remember
0          True      False         False   True              True
1         False      False         False   True              True
2         False       True          True  False              True
3          True      False         False   True             False
4         False       True          True  False             False


                     antecedents     consequents  antecedent support  consequent support  ...  confidence      lift  leverage  conviction
0                    (5 feet apart)        (TFIOS)                 0.4                 0.6  ...         1.0  1.666667      0.16         inf
1  (Walk to remember, 5 feet apart)        (TFIOS)                 0.2                 0.6  ...         1.0  1.666667      0.08         inf
2                      (Divergent)  (Harry Potter)                 0.4                 0.4  ...         1.0  2.500000      0.24         inf
3                    (Harry Potter)     (Divergent)                 0.4                 0.4  ...         1.0  2.500000      0.24         inf
4    (Divergent, Walk to remember)  (Harry Potter)                 0.2                 0.4  ...         1.0  2.500000      0.12         inf
5  (Harry Potter, Walk to remember)     (Divergent)                 0.2                 0.4  ...         1.0  2.500000      0.12         inf

[6 rows x 9 columns]


    support                                itemsets
0       0.6                  (Walk to remember)
1       0.6                              (TFIOS)
2       0.4                         (5 feet apart)
3       0.4                         (Harry Potter)
4       0.4                           (Divergent)
5       0.4                (TFIOS, Walk to remember)
6       0.4                   (TFIOS, 5 feet apart)
7       0.2           (Walk to remember, 5 feet apart)
8       0.2     (TFIOS, Walk to remember, 5 feet apart)
9       0.2           (Walk to remember, Harry Potter)
10      0.4                  (Divergent, Harry Potter)
11      0.2             (Divergent, Walk to remember)
12      0.2  (Divergent, Walk to remember, Harry Potter)
```

Първата таблица е визуализацията на данните. Редът представлява елементите, а колоната представлява транзакциите. „True“ означава, че артикулът присъства в тази транзакция. „False“ означава, че артикулът не присъства в тази транзакция.

Следващата таблица изобразява често срещания набор от елементи заедно с техния support, правила за асоцииране заедно с техните стойности за support и lift.

Накрая е визуализирана таблицата с генерираните често закупувани артикули с support>0.02

Можем да направим заключение, че например колкото повече хора гледат 5 feet apart, толкова повече се покачва и гледаемостта и на TFIOS.

Приложение на асоциативните правила

Анализ на пазарната кошница

Това е най-типичният пример за приложението на асоциативните правила. Данните се събират с помощта на скенери за баркод в повечето супермаркети. Тази база данни, известна като база данни „пазарна кошница“, се състои от голям брой записи за минали транзакции. Един запис изброява всички артикули, закупени от клиент в една продажба. Знаейки кои групи са склонни към конкретен набор от артикули, дава на тези магазини свободата да коригират оформлението на магазина и каталога на магазина, за да поставят оптимално един спрямо друг.

Медицинска диагноза

Асоциативните правила могат да бъдат полезни в медицинската диагностика за подпомагане на лекарите за лечение на пациенти. Диагностиката не е лесен процес и има редица грешки, които могат да доведат до ненадеждни крайни резултати. Използвайки асоциативните правила, можем да идентифицираме вероятността от възникване на заболяване по отношение на различни фактори и симптоми.

Данни от преброяването

Всяко правителство разполага с тонове данни от преброяването. Тези данни могат да се използват за планиране на ефективни обществени услуги (образование, здравеопазване, транспорт), както и за подпомагане на обществения бизнес (за създаване на нови фабрики, търговски центрове и дори маркетинг на определени продукти). Това приложение на извличане на асоциативни правила и извличане на данни има огромен потенциал за подкрепа на стабилна обществена политика и за ефективно функциониране на демократичното общество.

Протеинова последователност

Протеините са последователности, съставени от двадесет вида аминокиселини. Всеки протеин има уникална 3D структура, която зависи от последователността на тези аминокиселини. Лека промяна в последователността може да причини промяна в структурата, която може да промени функционирането на протеина. Тази зависимост на функционирането на протеина от неговата аминокиселинна последователност е обект на голямо изследване. По-рано се смяташе, че тези последователности са случайни, но сега се смята, че не са. Познаването и разбирането на асоциативните правила е изключително полезно по време на синтеза на изкуствени протеини.