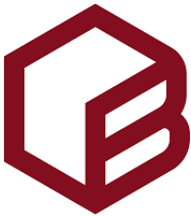



|   |  |                       |   |
|---|--|-----------------------|---|
|  | Politechnika Bydgoska im. J.J. Śniadeckich Wydział<br>Telekomunikacji, Informatyki i Elektrotechniki |                       |  |
| <b>Przedmiot</b>  | Algorytmy i eksploracja danych   |                       |   |
| <b>Prowadzący</b>   | dr inż. Michał Kruczkowski   |                       |   |
| <b>Temat</b>  | Reguły asocjacyjne   |                       |   |
| <b>Studenci</b>   | Adam Szreiber, Cezary Naskręt  |                       |   |
| <b>Nr ćw.</b>   | 6  | <b>Data wykonania</b> | 25.10.2023  |

## 1. Cel ćwiczenia

Celem tego ćwiczenia jest zapoznanie się z algorytmami eksploracji danych, w szczególności algorytmem Apriori i FP-Growth. Laboratorium ma na celu zrozumienie i praktyczne zastosowanie tych algorytmów w analizie zbiorów danych, identyfikację reguł asocjacyjnych oraz odkrywanie ukrytych wzorców. Ćwiczenie będzie również polegało na implementacji tych algorytmów oraz zapoznanie się z terminologią związaną z nimi, jak i poznanie innych algorytmów do wyszukiwania zbiorów częstych

## 2. Przebieg laboratorium

### 2.1. Zadanie 1

Dany jest transakcyjna baza danych zakupowych

| Bread | Milk | Diapers | Beer | Eggs | Cola |
|-------|------|---------|------|------|------|
| 1     | 1    |         |      |      |      |
| 1     |      | 1       | 1    | 1    |      |
|       | 1    | 1       | 1    |      | 1    |
| 1     | 1    | 1       | 1    |      | 1    |
| 1     |      |         | 1    |      | 1    |
| 1     | 1    |         |      | 1    |      |
| 1     | 1    |         |      |      |      |
| 1     |      |         |      | 1    |      |
|       |      | 1       | 1    |      | 1    |
|       | 1    | 1       | 1    |      | 1    |
| 1     | 1    | 1       | 1    | 1    | 1    |

Dane podane w instrukcji przez prowadzącego przepisuje do poniższej tabelki.

```
transactions = [
    ["Bread", "Milk"],
    ["Bread", "Diapers", "Beer", "Eggs"],
    ["Milk", "Diapers", "Beer", "Cola"],
    ["Bread", "Milk", "Diapers", "Beer"],
    ["Bread", "Milk", "Diapers", "Cola"],
    ["Bread", "Beer", "Cola"],
    ["Bread", "Milk", "Eggs"],
    ["Bread", "Milk"],
    ["Bread", "Eggs"],
    ["Diapers", "Beer", "Cola"],
    ["Milk", "Diapers", "Beer", "Cola"],
    ["Bread", "Milk", "Diapers", "Beer", "Eggs", "Cola"]
]
```

## 2.2. Zadanie 2

Wykorzystując powyższy zbiór danych zastosuj algorytmy:

- A-priori
- Frequent Pattern Growth

do wyszukania zbiorów częstych zakładając poziom 30% minimalny poziom wsparcia (MinSupp).

Korzystając z bibliotek pandas oraz mlxtend wyszukuje częste zbiory. W pierwszym kroku importuje biblioteki i inicjalizuje tablicę z transakcjami. Następnie przetwarzam dane do formatu DataFrame. Dane w takim formacie mogą już wykorzystać w metodach fpgrowth i apriori.

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import fpgrowth

transactions = [
    ["Bread", "Milk"],
    ["Bread", "Diapers", "Beer", "Eggs"],
    ["Milk", "Diapers", "Beer", "Cola"],
    ["Bread", "Milk", "Diapers", "Beer"],
    ["Bread", "Milk", "Diapers", "Cola"],
    ["Bread", "Beer", "Cola"],
    ["Bread", "Milk", "Eggs"],
    ["Bread", "Milk"],
    ["Bread", "Eggs"],
    ["Diapers", "Beer", "Cola"],
    ["Milk", "Diapers", "Beer", "Cola"],
    ["Bread", "Milk", "Diapers", "Beer", "Eggs", "Cola"]
]

te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)

print(fpgrowth(df, min_support=0.3, use_colnames=True))
print(apriori(df, min_support=0.3, use_colnames=True))
```

W wyniku na konsoli otrzymuje następujący wydruk.

|    | support  | itemsets              |
|----|----------|-----------------------|
| 0  | 0.750000 | (Bread)               |
| 1  | 0.666667 | (Milk)                |
| 2  | 0.583333 | (Diapers)             |
| 3  | 0.583333 | (Beer)                |
| 4  | 0.333333 | (Eggs)                |
| 5  | 0.500000 | (Cola)                |
| 6  | 0.500000 | (Milk, Bread)         |
| 7  | 0.333333 | (Diapers, Bread)      |
| 8  | 0.416667 | (Diapers, Milk)       |
| 9  | 0.500000 | (Diapers, Beer)       |
| 10 | 0.333333 | (Bread, Beer)         |
| 11 | 0.333333 | (Milk, Beer)          |
| 12 | 0.333333 | (Diapers, Milk, Beer) |
| 13 | 0.333333 | (Eggs, Bread)         |
| 14 | 0.416667 | (Cola, Beer)          |
| 15 | 0.416667 | (Diapers, Cola)       |
| 16 | 0.333333 | (Milk, Cola)          |
| 17 | 0.333333 | (Diapers, Cola, Beer) |
| 18 | 0.333333 | (Diapers, Cola, Milk) |
|    | support  | itemsets              |
| 0  | 0.583333 | (Beer)                |
| 1  | 0.750000 | (Bread)               |
| 2  | 0.500000 | (Cola)                |
| 3  | 0.583333 | (Diapers)             |
| 4  | 0.333333 | (Eggs)                |
| 5  | 0.666667 | (Milk)                |
| 6  | 0.333333 | (Bread, Beer)         |
| 7  | 0.416667 | (Cola, Beer)          |
| 8  | 0.500000 | (Diapers, Beer)       |
| 9  | 0.333333 | (Milk, Beer)          |
| 10 | 0.333333 | (Diapers, Bread)      |
| 11 | 0.333333 | (Eggs, Bread)         |
| 12 | 0.500000 | (Milk, Bread)         |
| 13 | 0.416667 | (Diapers, Cola)       |
| 14 | 0.333333 | (Milk, Cola)          |
| 15 | 0.416667 | (Diapers, Milk)       |
| 16 | 0.333333 | (Diapers, Cola, Beer) |
| 17 | 0.333333 | (Diapers, Milk, Beer) |
| 18 | 0.333333 | (Diapers, Cola, Milk) |

## 2.3. Zadanie 3

Na bazie zbiorów częstych wytypowanych w poprzednim zadaniu przez algorytm Frequent Pattern Growth zaprezentuj graficznie wytypowanych kandydatów w postaci drzewa FP-Tree.

Obiekt zwracany przez metodę `fpgrowth` przechowuje wiele dodatkowych informacji generowanych w trakcie wyliczania częstych zbiorów. Niestety nie daje on nam tych, które potrzebujemy do wykonania tego zadania. Dlatego postanowiłem napisać kod, który sam wykona kilka pierwszych kroków algorytmu FP-Tree..

Na początku importuje elementy z biblioteki `anytree` niezbędne do narysowania drzewa. Następnie inicjalizuję listę z transakcjami. Wykonuje pierwszy krok algorytmu FP-Tree, jakim jest policzenie ilości wystąpień każdego elementu we wszystkich transakcjach.

```
from anytree import Node, RenderTree

# transactions
transactions = [
    ["Bread", "Milk"],
    ["Bread", "Diapers", "Beer", "Eggs"],
    ["Milk", "Diapers", "Beer", "Cola"],
    ["Bread", "Milk", "Diapers", "Beer"],
    ["Bread", "Milk", "Diapers", "Cola"],
    ["Bread", "Beer", "Cola"],
    ["Bread", "Milk", "Eggs"],
    ["Bread", "Milk"],
    ["Bread", "Eggs"],
    ["Diapers", "Beer", "Cola"],
    ["Milk", "Diapers", "Beer", "Cola"],
    ["Bread", "Milk", "Diapers", "Beer", "Eggs", "Cola"]
]

# Initialize a dictionary to count occurrences
element_counts = {}

# Iterate through transactions and count occurrences of items
for transaction in transactions:
    for item in transaction:
        element_counts[item] = element_counts.get(item, 0) + 1

# Frequency of each individual item
print(element_counts)
```

Drugi krok algorytmu polega na posortowaniu elementów w każdej z transakcji zgodnie z wcześniej policzoną ilością wystąpień elementów.

```
# Sort transactions based on the sum of item occurrences
sorted_transactions = sorted(transactions, key=lambda transaction:
sum(element_counts[item] for item in transaction), reverse=True)

# Ordered-item set
print(sorted_transactions)
```

W trzecim kroku rysuje drzewo binarne zgodnie z założeniami tego algorytmu.

```
# Iterate through frequent patterns and add them to the tree
for itemset in sorted_transactions:
    current_node = root
    for item in itemset:
        # Check if the node exists, if not, create it
        child_node = next((child for child in current_node.children if
child.name == item), None)
        if child_node is None:
            child_node = Node(item, parent=current_node, count=1)
        else:
            current_node.count = current_node.count + 1
            current_node = child_node

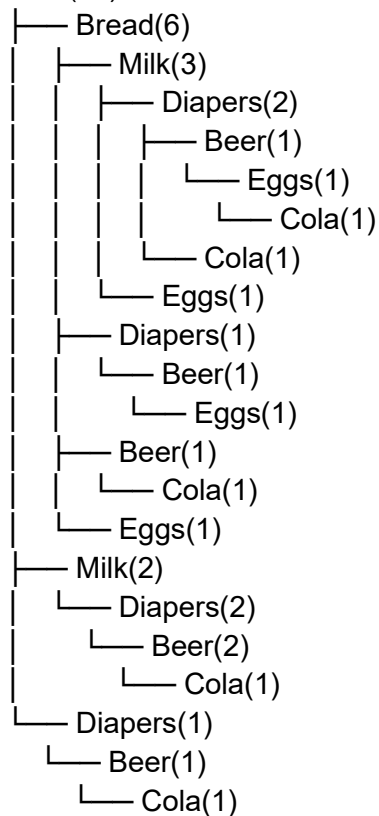
# Display the tree
for pre, fill, node in RenderTree(root):
    print(f"{pre}{node.name} ({node.count})")
```

Wynik otrzymany na konsoli. W pierwszej linijce wypisałem listę wszystkich produktów wraz z ilością ich wystąpień we wszystkich transakcjach. W drugiej linijce wyświetla się lista posortowanych transakcji. Na końcu wyświetlam drzewo zbudowane w wyniku działania algorytmu FP-Tree.

```
{'Bread': 9, 'Milk': 8, 'Diapers': 7, 'Beer': 7, 'Eggs': 4, 'Cola': 6}
```

```
[['Bread', 'Milk', 'Diapers', 'Beer', 'Eggs', 'Cola'], ['Bread', 'Milk', 'Diapers', 'Beer'], ['Bread', 'Milk', 'Diapers', 'Cola'], ['Milk', 'Diapers', 'Beer', 'Cola'], ['Milk', 'Diapers', 'Beer', 'Cola'], ['Bread', 'Diapers', 'Beer', 'Eggs'], ['Bread', 'Beer', 'Cola'], ['Bread', 'Milk', 'Eggs'], ['Diapers', 'Beer', 'Cola'], ['Bread', 'Milk'], ['Bread', 'Milk'], ['Bread', 'Eggs']]
```

```
Root(10)
```



## 2.4. Zadanie 4

Zaproponuj i zastosuj do rozwiązania problemu wyszukiwania reguł asocjacyjnych dowolny inny algorytm odkrywania reguł asocjacyjnych

Biblioteka mlxtend ma wiele algorytmów do wyszukiwania reguł asocjacyjnych takich jak `hmin`, `association_rules`, `fpcommon` oraz `fpmax`. Ja skorzystałem z tej ostatniej.

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

transactions = [
    ["Bread", "Milk"],
    ["Bread", "Diapers", "Beer", "Eggs"],
    ["Milk", "Diapers", "Beer", "Cola"],
    ["Bread", "Milk", "Diapers", "Beer"],
    ["Bread", "Milk", "Diapers", "Cola"],
    ["Bread", "Beer", "Cola"],
    ["Bread", "Milk", "Eggs"],
    ["Bread", "Milk"],
    ["Bread", "Eggs"],
    ["Diapers", "Beer", "Cola"],
    ["Milk", "Diapers", "Beer", "Cola"],
    ["Bread", "Milk", "Diapers", "Beer", "Eggs", "Cola"]
]

te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)

print(fpmax(df, min_support=0.3, use_colnames=True).size)
```

## 1. Wnioski

Reguły asocjacyjne to narzędzie analizy danych używane w dziedzinie eksploracji danych i uczenia maszynowego. Służą do identyfikowania interesujących zależności między danymi w zbiorze transakcyjnym lub bazie danych. Głównym celem reguł asocjacyjnych jest odkrycie, jakie elementy lub produkty często występują razem lub są kupowane razem przez klientów.

Reguły asocjacyjne są stosowane do zbiorów transakcyjnych, które zawierają informacje o różnych transakcjach, lub wydarzeniach. Przykłady to historie zakupów klientów w sklepie, odwiedzenia witryn internetowych lub jakiegokolwiek inne dane zawierające powiązania między elementami.

Wsparcie (Support) to miara częstości występowania zbioru elementów w transakcjach. Określa, jak często dana kombinacja produktów lub elementów pojawia się w zestawieniach. Wsparcie jest używane do identyfikacji zbiorów częstych.

Wiarygodność lub pewność (Confidence) to miara tego, jak często produkt lub element B jest kupowany w przypadku zakupu produktu, lub elementu A. Oznacza to, ile razy reguła asocjacyjna jest prawdziwa.

Lift to miara, która mierzy, jak bardzo pewność danej reguły asocjacyjnej przewyższa przypadkową współwystępującą zależność. Lift większy od 1 oznacza, że reguła jest bardziej wartościowa niż przypadkowe zdarzenia.

Algorytm FP-Tree wykorzystuje specjalne drzewo do przechowywania częstych wzorców, podczas gdy algorytm Apriori korzysta z kombinacji elementów. Oznacza to, że algorytm FP-Tree potrzebuje tylko dwóch przebiegów przez wszystkie dane, pierwszy do policzenia wystąpień, a drugi do stworzenia drzewa, podczas gdy algorytm Apriori tworzy kombinacje elementów i następnie wyszukuje ich wystąpienia, co oznacza, że potrzebuje on wiele przebiegów przez dane..

Algorytm FP-Tree może być bardziej wydajny pod względem potrzebnej pamięci od algorytmu Apriori, ponieważ przechowuje on wszystkie zbiory w drzewie, a algorytm Apriori przechowuje je w osobnych zbiorach.

Algorytm FP-Tree może działać szybciej dla dużych zbiorów danych, ponieważ potrzebuje tylko dwóch przebiegów, by stworzyć drzewo binarne, które zawiera już odpowiedź, której szukamy, natomiast algorytm Apriori wymaga wielu przebiegów i ilość niezbędnych operacji rośnie wraz ze wzrostem ilości i długości nowych zbiorów, które chcemy wyszukać.