Odkrywanie reguł asocjacyjnych

Tomasz Kubik

Na podstawie dokumentu: CS583-association-rules.ppt

Odkrywanie reguł asocjacyjnych

- Autor metody Agrawal et al in 1993.
- Analiza asocjacji danych w bazach danych
- Dane typu kategorycznego (dane składają się z małej liczby wartości, z których każda jest konkretną wartością kategorii lub etykietą)
- Algorytm nie pracuje dla danych numerycznych
- Początkowo użyty do analizy koszyka zakupów tj. do znalezienia powiązań pomiędzy kupowanymi artykułami

  Bread $\rightarrow$ Milk [sup = 5%, conf = 100%]

Model formalny

- $I = \{i_1, i_2, \ldots, i_m\}$: zbiór *elementów*
- t : pojedyncza transakcja,

  t jest zbiorem elementów, $t \subseteq I$.
- $T = \{t_1, t_2, \ldots, t_n\}$: zbiór transakcji,

Przykład: koszyk zakupów

- Transakcje (koszyk zakupów) :

  t1: {bread, cheese, milk}

  t2: {apple, eggs, salt, yogurt}

  … …

  tn: {biscuit, eggs, milk}

- Znaczenie:
  - i_k : produkt typu k występujący w koszyku
  - *I*: zbiór wszystkich sprzedawanych produktów
  - *transakcja*: produkty w koszyku (rozumiane jako transakcja, która może mieć własny numer ID)
  - *zbiór transakcji*: wszystkie zaobserwowane konfiguracje produktów w koszyku

Przykład: zbiór dokumentów

- **Każdy dokument rozumiany jest jako zbiór słów**

doc1: Student, Teach, School

doc2: Student, School

doc3: Teach, School, City, Game

doc4: Baseball, Basketball

doc5: Basketball, Player, Spectator

doc6: Baseball, Coach, Game, Team

doc7: Basketball, Team, City, Game

Model formalny: reguły

- X jest zbiorem elementów z *I*
  - np. X = {milk, bread, cereal}
- Transakcja *t* zawiera *X* jeśli $X \subseteq t$.
- Regułą asocjacyjną jest implikacja w postaci:

  $X \rightarrow Y$, gdzie X, $Y \subset I$, *oraz* $X \cap Y = \varnothing$
- Z_k jest zbiorem elementów z *I* o liczności *k*.
  - np.., {milk, bread, cereal} jest zbiorem Z_3

Miary ważności reguł

- Wsparcie: reguła posiada wsparcie (support) *sup* w *T* (zbiorze transakcji) jeśli sup% transakcji zawiera $X \cup Y$.
  - $sup = P(X \cup Y)$.
- Zaufanie: reguła zachodzi w *T* z zaufaniem (confidence) *conf* jeśli *conf*% transakcji zawierających *X* zawiera również *Y.*
  - $conf = P(Y \mid X)$
- Reguła asocjacyjna jest schematem, który określa, że kiedy wystąpi *X* occurs, to *Y* wystąpi z pewnym prawdopodobieństwem.

Wsparcie i zaufanie

- Wartość wsparcia X.count: jest to liczba transakcji w zbiorze transakcji *T,* które zawierają zbiór elementów *X.* Zakładając, że liczba transakcji w T wynosi n, to:

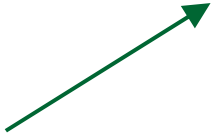
$$support = \frac{(X \cup Y).count}{n}$$

$$confidence = \frac{(X \cup Y).count}{X.count}$$

Cel i kluczowe założenia

- **Cel:** Znaleźć wszystkie reguły, które spełniają założony przez użytkownika poziom minimalnego wsparcie (minsup) oraz minimalnego zaufania (minconf).
- **Kluczowe założenia:**
  - Zupełność: znaleźć należy wszystkie reguły.
  - Niezdefiniowane elementy wynikowe: elementy po prawych stronach reguł są początkowo nieznane
  - Wykorzystanie pamięci zewnętrznej

Przykład:



t1:	Beef, Chicken, Milk
t2:	Beef, Cheese
t3:	Cheese, Boots
t4:	Beef, Chicken, Cheese
t5:	Beef, Chicken, Clothes, Cheese, Milk
t6:	Chicken, Clothes, Milk
t7:	Chicken, Milk, Clothes

- Transakcje i dane
- Założenie:

  minsup = 30%

  minconf = 80%
- Przykład częstego zbioru elementów:

  {Chicken, Clothes, Milk} [sup = 3/7]
- Reguły asocjacyjne ze zbioru elementów:

  Clothes → Milk, Chicken [sup = 3/7, conf = 3/3]

  …

  Clothes, Chicken → Milk, [sup = 3/7, conf = 3/3]

Reprezentacja danych w transakcjach

- Uproszczone spojrzenie na koszyk zakupów
- Pominięte niektóre ważne informacje, np..:
  - ilość produktów zakupionych
  - cena

Algorytmy odkrywania reguł

- Istnieje ich bardzo wiele
- Używają różnych strategii i struktur danych
- Wynikowy zbiór reguł jest taki sam
  - Dla danego zbiory transakcji *T*, oraz zdefiniowanego minimalnego wsparcia i minimalnego zaufania, zbiór reguł asocjacyjnych występujących w *T* jest jednoznacznie określony.
- Algorytmy, mimo iż produkują te same wyniki, mają różną złożoność obliczeniową i różne wymagania co do zasobów obliczeniowych.

Algorytm Apriori

- **Jest to jeden z najlepszych znanych algorytmów**
- Składa się on z dwóch kroków
  - Znalezienia (odkrycia) wszystkich zbiorów elementów, które mają zadeklarowany poziom wsparcia (*zbiory częste*).
  - Użycia zbiorów częstych do wygenerowania reguł.

- Przykład zbioru częstego:

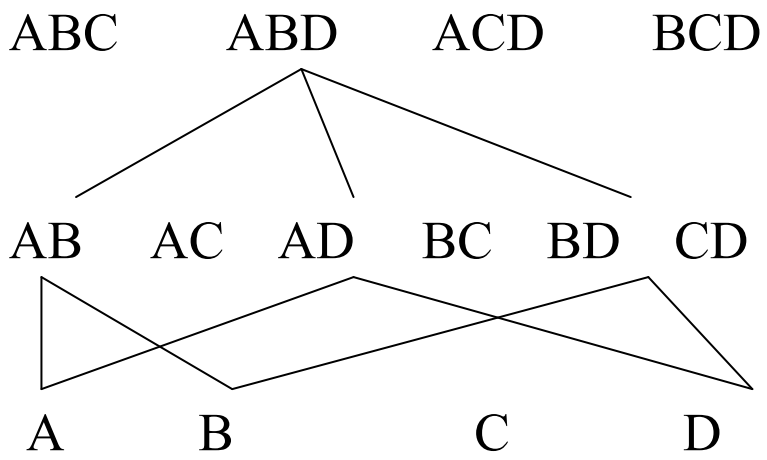
  {Chicken, Clothes, Milk} [sup = 3/7]

  oraz przykład reguły na nim opartej: Clothes $\rightarrow$ Milk,

  Chicken [sup = 3/7, conf = 3/3]

Krok 1: Odkrycie zbiorów częstych

- Zbiór częsty jest zbiorem elementów, którego wsparcie jest ≥ minsup.
- Własność apriori: dowolny podzbiór zbioru częstego jest również zbiorem częstym



Krok 1: implementacja

Jest to algorytm iteracyjny, w którym dokonuje się przeszukiwania wszerz, level-wise search. Jego idea jest następująca:

Znajdź wszystkie zbiory częste typu Z_1, następnie wszystkie zbiory częste Z_2, i tak dalej. W każdej iteracji *k*, rozważaj tylko te zbiory elementów, które zawierają zbiory częste Z_{k-1}.

- Utwórz kolekcję zbiorów częstych typu Z_1: F_1
- Startując z $k = 2$
  - Utwórz C_k - zbiory elementów typu Z_k, utworzone na podstawie zbiorów z F_{k-1}, które są kandydatami na zbiory częste
  - Na bazie C_k utwórz F_k – kolekcję zbiorów częstych, które faktycznie są częste, tj. $F_k \subseteq C_k$ (odrzuć te, które nie są częste).

Przykład

Dataset T

minsup=0.5

TID	Items
T100	1, 3, 4
T200	2, 3, 5
T300	1, 2, 3, 5
T400	2, 5

zbiór:liczność

1. przeglądaj T ➔ C_1: {1}:2, {2}:3, {3}:3, {4}:1, {5}:3

   ➔ F_1: {1}:2, {2}:3, {3}:3, {5}:3

   ➔ C_2: {1,2}, {1,3}, {1,5}, {2,3}, {2,5}, {3,5}

2. przeglądaj T ➔ C_2: {1,2}:1, {1,3}:2, {1,5}:1, {2,3}:2, {2,5}:3, {3,5}:2

   ➔ F_2: **{1,3}**:2, **{2,3}**:2, **{2,5}**:3, **{3,5}**:2

   ➔ C_3: {2, 3,5}

3. przeglądaj T ➔ C_3: **{2, 3, 5}**:2 ➔ F_3: **{2, 3, 5}**

Szczegóły: porządkowanie elementów

- Elementy w *I* posortowane są w porządku leksykalnym (jest to porządek całkowity)
- W kolejnych krokach algorytmu obowiązuje ten sam porządek sortowania
- {*w*[1], *w*[2], …, *w*[*k*]} przedstawia zbiór *w* typu Z_1 składający się z elementów *w*[1], *w*[2], …, *w*[*k*], gdzie *w*[1] < *w*[2] < … < *w*[*k*] zgodnie z porządkiem całkowitym.

Szczegóły implementacji algorytmu

Algorithm Apriori(*T*)

$C_1 \leftarrow$ zainicjuj(T);
$F_1 \leftarrow \{f \mid f \in C_1, f.\text{count}/n \geq minsup\}$; // n: liczba transakcji w T
for (k = 2; $F_{k-1} \neq \emptyset$; k++) **do**
  $C_k \leftarrow$ generuj_kandydatów(F_{k-1});
  for each $t \in T$ **do**
    for each $c \in C_k$ **do**
      if c zawiera się w t **then**
        $c.count$++;
    end
  end
  $F_k \leftarrow \{c \in C_k \mid c.count/n \geq minsup\}$
end
return $F \leftarrow \bigcup_k F_k$;

Generacja kandydatów apriori

- funkcja generuj_kandydatów bierze F_{k-1} i zwraca nadzbiór (zwany kandydatami) zbioru wszystkich częstych zbiorów k-elementowych. Składa się ona z dwóch kroków:
  - *krok złączenia*: generuje wszystkich możliwych kandydatów o długości k i wpisuje ich do C_k
  - *krok eliminacji*: usuwa tych kandydatów z C_k, którzy nie są zbiorami częstymi.

Funkcja generująca kandydatów

Function generacja_kandydatów(F_{k-1})

$C_k \leftarrow \varnothing$;

forall $f_1, f_2 \in F_{k-1}$

with $f_1 = \{i_1, \ldots, i_{k-2}, i_{k-1}\}$

and $f_2 = \{i_1, \ldots, i_{k-2}, i'_{k-1}\}$

and $i_{k-1} < i'_{k-1}$ **do**

$c \leftarrow \{i_1, \ldots, i_{k-1}, i'_{k-1}\}$; //połącz f_1 i f_2

$C_k \leftarrow C_k \cup \{c\}$;

for each (k-1)-subset s of c **do**

if ($s \notin F_{k-1}$) **then**

delete c from C_k; // usuń

end

end

return C_k;

Przykład

- F_3 = {{1, 2, 3}, {1, 2, 4}, {1, 3, 4},
  {1, 3, 5}, {2, 3, 4}}

- Po połączeniu
  - C_4 = {{1, 2, 3, 4}, {1, 3, 4, 5}}
- Po eliminacji:
  - C_4 = {{1, 2, 3, 4}}
    ponieważ {1, 4, 5} nie jest w F_3 (tzn. {1, 3, 4, 5} jest usunięte)

Krok 2: Generowanie reguł ze zbiorów częstych

- Zbiory częste ≠ reguły asocjacyjne

Potrzeba więc dokonać kolejnych obliczeń:

- Dla każdego zbioru częstego *X*,

  dla każdego właściwego niepustego A będącego podzbiorem zbioru *X*,

  - Niech *B* = X - *A*
  - A $\rightarrow$ B jest regułą asocjacyjną, jeśli
    - zaufanie (A $\rightarrow$ B) $\geq$ minconf,

      wsparcie(A $\rightarrow$ B) = wsparcie(A$\cup$B) = wsparcie(X)

      zaufanie(A $\rightarrow$ B) = wsparcie(A $\cup$ B) / wsparcie(A)

Przykład generowania reguł

- Przypuśćmy, że {2,3,4} jest częsty, z sup=50%
  - Właściwe niepuste podzbiory: {2,3}, {2,4}, {3,4}, {2}, {3}, {4}, z sup=50%, 50%, 75%, 75%, 75%, 75% odpowiednio
  - Podzbiory te generują reguły asocjacyjne:
    - $2,3 \rightarrow 4$, zaufanie =100%
    - $2,4 \rightarrow 3$, zaufanie =100%
    - $3,4 \rightarrow 2$, zaufanie =67%
    - $2 \rightarrow 3,4$, zaufanie =67%
    - $3 \rightarrow 2,4$, zaufanie =67%
    - $4 \rightarrow 2,3$, zaufanie =67%
    - wszystkie reguły mają wsparcie = 50%

Tworzenie reguł: podsumowanie

- Aby otrzymać regułę $A \rightarrow B$, powinno być wsparcie($A \cup B$) oraz wsparcie(A)
- Wszystkie wymagane informacje dla obliczenia zaufania były już zapisane podczas generacji zbiorów elementów. Nie ma więc konieczności przeglądania zbioru transakcji *T.*
- Obliczenia w tym kroku zajmują mniej czasu niż generowanie zbiorów częstych.

Uwagi o algorytmie apriori

- przeszukiwanie wszerz
- K = rozmiar największego zbioru elementów
- wykonuje co najwyżej K przejść przez dane
- w praktyce K jest ograniczone (np. do10).
- Algorytm jest szybki, w niektórych warunkach znalezienie reguł jest liniowe w czasie.
- Można przeskalować do dużych zbiorów danych

Uwagi dodatkowe na temat odkrywania reguł asocjacyjnych

- Ilość możliwych reguł dla *I* o m elementach wyraża się liczbą wykładnicza, $O(2^m)$.
- Wykorzystując wysoki minimalny poziom wsparcia i wysoki minimalny poziom ufności można wpływać na ilość przetwarzanych danych
- Niemniej ilość produkowanych reguł może być ogromna

Różne typy danych

- Dane mogą mieć postać transakcji lub tabeli

  Postać transakcji: a, b

  a, c, d, e

  a, d, f

  Postać tabeli:

Attr1	Attr2	Attr3
a,	b,	d
b,	c,	e

- Dane w tabeli muszą być przekonwertowane do postaci transakcyjnej.

Przekształcenie tabeli na transakcje

Tabela:

Attr1	Attr2	Attr3
a,	b,	d
b,	c,	e

Transakcje:

(Attr1, a), (Attr2, b), (Attr3, d)

(Attr1, b), (Attr2, c), (Attr3, e)

Problemy z generacją reguł asocjacyjnych

- Pojedyncze minsup: Zakłada się, że wszystkie elementy w zbiorze danych są tej samej natury lub też mają podobą częstotliwość wystąpień.
- Nie jest to prawda: W wielu zastosowaniach zdarza się, że elementy mają różną naturę, np. patelnie dużo rzadziej są kupowane od bochenków chleba.
- Jeśli mamy więc elementy o różnej naturze, pojawiają się dwa problemy:
  - Zbyt wysokie minsup nie pozwoli utworzyć reguł dla elementów rzadkich
  - Zbyt niskie minsup wygeneruje zbyt dużo reguł

Model z wieloma minsup

- Zmodyfikowana wersja algorytmu zakłada istnienie indywidualnego poziomu wsparcia dla każdego z elementów, *minimum item supports* (MIS).
- Definiując różne MIS dla różnych elementów można uzyskiwać reguły odpowiadające bardziej naturze elementów, niż w przypadku jednakowego wsparcia dla wszystkich elementów.

Minsup reguły

- Niech MIS(*i*) będzie wartością MIS dla elementu *i*. Wtedy *minsup* reguły *R* obliczane jest jako najniższa wartość MIS elementów w niej występujących

dla reguły *R*: $a_1, a_2, \ldots, a_k \rightarrow a_{k+1}, \ldots, a_r$

minimalne wsparcie obliczane jest jako:

$\min(\text{MIS}(a_1), \text{MIS}(a_2), \ldots, \text{MIS}(a_r))$.

Przykład

- Rozważmy następujące elementy:

bread, *shoes*, *clothes*

Niech wartości MIS będą zdefiniowane przez użytkownika jak niżej:

MIS(*bread*) = 2% MIS(*shoes*) = 0.1%

MIS(*clothes*) = 0.2%

Niech poziom wsparcia i zaufania wynosi

[sup=0.15%,conf =70%]

Wtedy następująca reguła spełnia założenia na minimalne wsparcie:

clothes $\rightarrow$ *bread*

bo min(MIS(*bread*), MIS(*clothes*)) = 0.2%.

Zaś następująca reguła nie spełnia założenie na minimalne wsparcie:

clothes $\rightarrow$ *shoes* [sup=0.15%,conf =70%]

bo min(MIS(*clothes*), MIS(*shoes*)) = 0.1%.

Domknięcie w dół (Downward closure property)

- Czy w nowym modelu właściwości nie są zachowane ?
- Jeśli zbiór elementów spełnia minsup, wtedy wszystkie jego podzbiory również spełniają minsup.

Rozważmy cztery elementy 1, 2, 3, 4 w bazie danych. Ich minimalne wsparcie dane jest przez:

MIS(1) = 10% MIS(2) = 20% MIS(3) = 5% MIS(4) = 6%

Dla tego przykładu podzbiór {1, 2} mógłby mieć wsparcie 9% (czyli byłby to zbiór nieczęsty).

Podzbiory {1, 2, 3} oraz {1, 2, 4} to podzbiory częste. Jednak ze względu na odrzucenie {1, 2} nie byłyby one wygenerowane! Nieodrzucenie zaś podzbioru {1,2} przeczyłoby zachowaniu właściwości domknięcia w dół.

Rozwiązanie problemu

- Sortujemy wszystkie elementy w *I* zgodnie z ich wartościami MIS *(sorted closure property)*
- Porządek ten używamy w algorytmie dla każdego zbioru elementów.
- Posortowane podzbiory elementów mają następującą postać:

$$\{c[1], c[2], \ldots, c[k]\},$$

gdzie

$$\text{MIS}(c[1]) \leq \text{MIS}(c[2]) \leq \ldots \leq \text{MIS}(c[k]).$$

Algorytm MSapriori

Algorytm MSapriori(*T, MS*)

$M \leftarrow$ sort(I, MS);
$L \leftarrow$ init-pass(M, T);
$F_1 \leftarrow \{\{i\} \mid i \in L, i.count/n \geq \text{MIS}(i)\}$;
for (k = 2; $F_{k-1} \neq \emptyset$; k++) **do**
  if k=2 **then**
    $C_k \leftarrow$ level2-candidate-gen(L)
  else $C_k \leftarrow$ MScandidate-gen(F_{k-1});
  end;
  for each transaction $t \in T$ **do**
    for each candidate $c \in C_k$ **do**
      if c is contained in t **then**
        $c.count$++;
      if $c - \{c[1]\}$ is contained in t **then**
        $c.tailCount$++
    end
  end
  $F_k \leftarrow \{c \in C_k \mid c.count/n \geq MIS(c[1])\}$
end
return $F \leftarrow \bigcup_k F_k$;

Pierwszy przebieg poprzez dane

- W pierwszym przebiegu wyznaczane są wartości wsparcia (support count) dla każdego elementu.
- Następnie elementy są ustawiane w kolejności odpowiadającej kolejności wartości MIS(*i*) uszeregowanych rosnąco
- Mając listę elementów M pierwszy z nich (indeksowany przez *i*) wstawiany jest do L
- Teraz kolejno
  - Dla każdego elementu *j* występującego w *M* po elemencie *i*, jeśli $j.count/n \geq$ MIS(*i*) wtedy *j* jest również wstawiane w *L*, gdzie *j.count* jest licznikiem wsparcia (support count) elementu *j* oraz *n* jest liczbą wszystkich transakcji w *T*.
- *L* jest wykorzystane w funkcji level2-candidate-gen

Przykład pierwszego przebiegu przez dane

- Niech z zbiorze danych będą element 1, 2, 3, 4. Ich minimalne wsparcie będą odpowiednio:

  MIS(1) = 10% MIS(2) = 20%

  MIS(3) = 5% MIS(4) = 6%

- Załóżmy, że nasz zbiór danych zawiera 100 transakcji. Pierwsze przejście poprzez dane dostarczy nam następujących wartości liczników wsparcia:

  {3}.*count* = 6, {4}.*count* = 3,

  {1}.*count* = 9, {2}.*count* = 25.

- **Wtedy** L = {3, 1, 2} oraz F_1 = {{3}, {2}}

Element 4 nie jest w L ponieważ 4.*count*/*n* < MIS(3) (= 5%),

{1} nie jest w F_1 ponieważ 1.*count*/*n* < MIS(1) (= 10%).

Generacja reguł

- Poniższe dwie linie algorytmu MSapriori są szczególnie ważne dla tego algorytmu, nie są ważne dla algorytmu Apriori:

  if *c* – {*c*[1]} is contained in *t* **then**

  c.tailCount++

- Bez uwzględnienia tych dwóch linii wielu reguł nie dałoby się wygenerować

Wielokrotne minsup

- Model z wielokrotnymi wartościami minimalnego wsparcia podsumowuje model z pojedynczą wartością wsparcia
- Jest to model bardziej zbliżony do rzeczywistości i praktycznych aplikacji.
- Model ten pozwala znaleźć reguły dla elementów rzadkich jeszcze bez generowania wielkiej ilości nieprzydatnych reguł dla elementów częstych.
- Ustanawiając MIS dla kilku elementów na wartość 100% (or more), można efektywnie poinstruować algorytm, aby w nie generował on reguł zawierające tylko te elementy.

Odkrywanie reguły asocjacyjnych dla klas

- Odkrywanie reguł zazwyczaj nie ma dobrze sprecyzowanego celu.
- Znajduje on wszystkie możliwe reguły wynikające z danych, w szczególności reguły, w których dane elementy występują jako przesłanki jak również jako wnioski.
- W niektórych zastosowaniach użytkownik jest zainteresowany osiągnięciem szczególnego celu
  - np. użytkownik może chcieć znaleźć w zbiorze posiadanych dokumentów na określony temat istniejących związków pomiędzy używanymi w dokumentach słowami a ich tematami.

Definicja problemu

- Niech T będzie zbiorem danych z n transakcji.
- Każda transakcja jest etykietowana oznaczeniem klasy y.
- Niech I będzie zbiorem wszystkich elementów w T, zaś Y zbiorem wszystkich etykiet oraz $I \cap Y = \varnothing$.
- Regułą asocjacyjną klasy (*class association rule - CAR*) jest implikacja w postaci:

  $X \rightarrow y$, gdzie $X \subseteq I$, oraz $y \in Y$.

- Definicje wsparcia i zaufania są identyczne jak w przypadku normalnych reguł asocjacyjnych.

Przykład

- **Zbiór dokumentów tekstowych**

  doc 1: Student, Teach, School : Education
  doc 2: Student, School : Education
  doc 3: Teach, School, City, Game : Education
  doc 4: Baseball, Basketball : Sport
  doc 5: Basketball, Player, Spectator : Sport
  doc 6: Baseball, Coach, Game, Team : Sport
  doc 7: Basketball, Team, City, Game : Sport

- Niech *minsup* = 20% oraz *minconf* = 60%. Poniżej przedstawione są dwie przykładowe reguły asocjacyjne:

  Student, School $\rightarrow$ Education [sup= 2/7, conf = 2/2]
  game $\rightarrow$ Sport [sup= 2/7, conf = 2/3]

Algorytm odkrywania reguł

- W odróżnieniu od „normalnego” odkrywania reguł, CAR może być wyznaczona wprost w jednym kroku.
- Kluczową operacją jest znalezienie wszystkich **elementów regułowych**, które mają wsparcie przewyższające *minsup*. Element regułowy ma postać:

  (*condset*, *y*)

  gdzie **condset** jest zbiorem elementów z *I* (*i.e., condset* $\subseteq$ *I*), oraz $y \in Y$ jest etykietą klasy.
- Zasadniczo każdy element regułowy reprezentuje regułę:

  $condset \rightarrow y$,
- Algorytm Apriori może być zmodyfikowany, aby generował reguły asocjacyjne klas CAR.

Wielokrotne minimalne wsparcie dla klas

- W szukaniu reguł asocjacyjnych klas można skorzystać z idei wielokrotnego minimalnego wsparcia.
- Użytkownik może określić różne minimalne wsparcie dla różnych klas, w efekcie efektywnie przypisać różny poziom minimalnego wsparcia dla regół każdej z klas.
- Na przykład, niech będzie dany zbiór z dwiema klasami: Yes oraz No. W takim przypadku można określić:
  - reguły klasy Yes powinny mieć minimalne wsparcie na poziomie 5%
  - reguły klasy Yes powinny mieć minimalne wsparcie na poziomie 5%
- Ustalając minimalne wsparcia klas na 100% (lub więcej dla wybranych klas) można poinstruować algorytm, aby nie generował reguł dla tych klas.

Podsumowanie

- Odkrywanie reguł asocjacyjnych jest popularnym tematem w drążeniu danych.
- Istnieje wiele efektywnych algorytmów do tego problemu.
- Tematy związane z odkrywaniem reguł:
  - wielopoziomowe lub zgeneralizowane odkrywanie reguł
  - Odkrywanie reguł z ograniczeniami
  - Stopniowe budowanie reguł
  - Odkrywanie maksymalnego zbioru częstego
  - Numeryczne odkrywanie reguł asocjacyjnych
  - Wizualizacja reguł
  - Algorytmy równoległe w odkrywaniu reguł
  - …