Algorytmy eksploracji danych: Wykład 10

Wojciech Kempa

Politechnika Śląska
Wydział Matematyki Stosowanej

Transakcja wspierająca

- Formułując problem odkrywania reguł asocjacyjnych, nie musimy w bazie danych uwzględniać wszystkich badanych produktów (atrybutów), a jedynie te, które występują w danej transakcji.

Transakcja wspierająca

- Formułując problem odkrywania reguł asocjacyjnych, nie musimy w bazie danych uwzględniać wszystkich badanych produktów (atrybutów), a jedynie te, które występują w danej transakcji.
- Definiujemy zatem pewien zbiór $L = \{l_1, ..., l_m\}$ elementów (produktów) i bazę danych $D = \{T_1, ..., T_n\}$, w której T_i oznacza i-tą transakcję, przy czym teraz $T_i \subseteq L$ dla każdego $i \in \{1, ..., n\}$.

Transakcja wspierająca

- Formułując problem odkrywania reguł asocjacyjnych, nie musimy w bazie danych uwzględniać wszystkich badanych produktów (atrybutów), a jedynie te, które występują w danej transakcji.
- Definiujemy zatem pewien zbiór $L = \{l_1, ..., l_m\}$ elementów (produktów) i bazę danych $D = \{T_1, ..., T_n\}$, w której T_i oznacza i-tą transakcję, przy czym teraz $T_i \subseteq L$ dla każdego $i \in \{1, ..., n\}$.
- Mówimy, że transakcja T_i **wspiera element** $l_j \in L$, jeżeli $l_j \in T_i$. Podobnie, mówimy, że transakcja T_i **wspiera zbiór** $X \subseteq L$, jeżeli $X \subset T_i$ (transakcja wspiera wówczas każdy element zbioru X).

Wsparcie i ufność reguły asocjacyjnej

Weźmy pod uwagę regułę asocjacyjną $X \longrightarrow Y$. Wykorzystując wprowadzone wyżej oznaczenia, możemy teraz zdefiniować wsparcie i ufność tej reguły w następujący sposób:

$$supp(X \longrightarrow Y) = \frac{|\{T_i \in D : T_i \text{ wspiera } X \cup Y\}|}{|D|} \tag{1}$$

oraz

$$conf(X \longrightarrow Y) = \frac{|\{T_i \in D : T_i \text{ wspiera } X \cup Y\}|}{|\{T_j \in D : T_j \text{ wspiera } X\}|}. \tag{2}$$

Miara wsparcia reguły jest **symetryczna** względem X i Y: innymi słowy, reguły asocjacyjne $X \longrightarrow Y$ oraz $Y \longrightarrow X$ mają w tym samym zbiorze D identyczne wsparcie. Miara ufności jest **asymetryczna** względem X i Y.

Ogólny algorytm odkrywania silnych reguł asocjacyjnych

W praktyce zawsze interesują nas przede wszystkim silne reguły asocjacyjne, spełniające warunek minimalnego wsparcia i minimalnej ufności. Pierwszy algorytm odkrywania silnych binarnych reguł asocjacyjnych został skonstruowany w 1983 roku. Ogólny algorytm wygląda w następujący sposób.

1. Znajdź wszystkie zbiory $L_i \subseteq L$ takie, że $supp(L_i) \geq min\, supp$ (zbiory takie, spełniające warunek minimalnego wsparcia, nazywamy **zbiorami częstymi**).
2. Dla każdego zbioru częstego L_i znajdź wszystkie jego niepuste podzbiory $sub\, L_i \subseteq L_i$.
3. Dla każdego zbioru $sub\, L_i$ takiego, że $\frac{supp(L_i)}{supp(sub\, L_i)} \geq min\, conf$ wygeneruj regułę asocjacyjną postaci

$$sub\, L_i \longrightarrow (L_i \setminus sub\, L_i). \tag{3}$$

Jest to szukana silna reguła asocjacyjna.

Znajdowanie zbiorów częstych

- Jak łatwo można zauważyć, kluczowym krokiem opisanego powyżej prostego algorytmu jest znalezienie zbiorów częstych.

Znajdowanie zbiorów częstych

- Jak łatwo można zauważyć, kluczowym krokiem opisanego powyżej prostego algorytmu jest znalezienie zbiorów częstych.
- W sposób trywialny można to zrobić, generując wszystkie podzbiory zbioru L, a następnie obliczając wartości ich wsparcia. Jest to jednak metoda nieefektywna i w praktyce nieakceptowalna w przypadku zbiorów L o dużej liczności.

Znajdowanie zbiorów częstych

- Jak łatwo można zauważyć, kluczowym krokiem opisanego powyżej prostego algorytmu jest znalezienie zbiorów częstych.
- W sposób trywialny można to zrobić, generując wszystkie podzbiory zbioru L, a następnie obliczając wartości ich wsparcia. Jest to jednak metoda nieefektywna i w praktyce nieakceptowalna w przypadku zbiorów L o dużej liczności.
- W eksploracji danych istnieje dość sporo efektywnych **algorytmów odkrywania zbiorów częstych**. Algorytmy te możemy podzielić na dwie zasadnicze grupy.

Algorytmy eksploracji poziomej i pionowej

- **Algorytmy eksploracji poziomej** (wierszowej), opierają się o bazę danych D postaci

Transakcja	Produkty
1	A, B, C
2	A, C
3	A, D
4	A, B, C, D

Algorytmy eksploracji poziomej i pionowej

- **Algorytmy eksploracji poziomej** (wierszowej), opierają się o bazę danych D postaci

Transakcja	Produkty
1	A, B, C
2	A, C
3	A, D
4	A, B, C, D

- **Algorytmy eksploracji pionowej** (kolumnowej), bazują na zbiorze D postaci

Produkt	Transakcje
A	1, 2, 3, 4
B	1, 4
C	1, 2, 4
D	3, 4

Algorytm Apriori (1)

- Analizę rozpoczniemy od algorytmów eksploracji poziomej. Omówimy dwa najważniejsze z nich: algorytm Apriori oraz algorytm FP-Growth.

Algorytm Apriori (1)

- Analizę rozpoczniemy od algorytmów eksploracji poziomej. Omówimy dwa najważniejsze z nich: algorytm Apriori oraz algorytm FP-Growth.
- **Algorytm Apriori**, zaproponowany przez R. Agrawala i R. Srikanta w 1994 roku, wykorzystuje tzw. **własność antymonotoniczności miary wsparcia**.

Algorytm Apriori (1)

- Analizę rozpoczniemy od algorytmów eksploracji poziomej. Omówimy dwa najważniejsze z nich: algorytm Apriori oraz algorytm FP-Growth.
- **Algorytm Apriori**, zaproponowany przez R. Agrawala i R. Srikanta w 1994 roku, wykorzystuje tzw. **własność antymonotoniczności miary wsparcia**.
- Własność tę można zapisać w następujący sposób: jeżeli X i Y są pewnymi podzbiorami wyjściowego zbioru L elementów (produktów), to wówczas

$$X \subseteq Y \iff supp(Y) \leq supp(X). \tag{4}$$

Algorytm Apriori (1)

- Analizę rozpoczniemy od algorytmów eksploracji poziomej. Omówimy dwa najważniejsze z nich: algorytm Apriori oraz algorytm FP-Growth.
- **Algorytm Apriori**, zaproponowany przez R. Agrawala i R. Srikanta w 1994 roku, wykorzystuje tzw. **własność antymonotoniczności miary wsparcia**.
- Własność tę można zapisać w następujący sposób: jeżeli X i Y są pewnymi podzbiorami wyjściowego zbioru L elementów (produktów), to wówczas

$$X \subseteq Y \iff supp(Y) \leq supp(X). \tag{4}$$

- Z własności antymonotoniczności wynika prosta, ale bardzo ważna zasada w praktyce ograniczająca przestrzeń poszukiwań zbiorów częstych: zbiór X jest zbiorem częstym wtedy i tylko wtedy, gdy wszystkie jego podzbiory są zbiorami częstymi.

Algorytm Apriori (2)

- Przed wykonaniem algorytmu Apriori elementy każdej transakcji ze zbioru D porządkujemy leksykograficznie.

Algorytm Apriori (2)

- Przed wykonaniem algorytmu Apriori elementy każdej transakcji ze zbioru D porządkujemy leksykograficznie.
- W kolejnych krokach algorytmu znajdować będziemy zbiory częste o coraz większej liczności.

Algorytm Apriori (2)

- Przed wykonaniem algorytmu Apriori elementy każdej transakcji ze zbioru D porządkujemy leksykograficznie.
- W kolejnych krokach algorytmu znajdować będziemy zbiory częste o coraz większej liczności.
- Przebieg algorytmu Apriori można opisać w następujący sposób.

Algorytm Apriori (3)

- Wyodrębniamy 1-elementowe podzbiory zbioru L i sprawdzamy, które z nich są zbiorami częstymi.

Algorytm Apriori (3)

- Wyodrębniamy 1-elementowe podzbiory zbioru L i sprawdzamy, które z nich są zbiorami częstymi.
- Generujemy klasę wszystkich 2-elementowych podzbiorów zbioru L takich, że każdy ich 1-elementowy podzbiór jest zbiorem częstym (są to tzw. 2-elementowe **zbiory kandydujące**). Sprawdzamy, które z nich są zbiorami częstymi i te dołączamy do listy znalezionych zbiorów częstych.

Algorytm Apriori (3)

- Wyodrębniamy 1-elementowe podzbiory zbioru L i sprawdzamy, które z nich są zbiorami częstymi.
- Generujemy klasę wszystkich 2-elementowych podzbiorów zbioru L takich, że każdy ich 1-elementowy podzbiór jest zbiorem częstym (są to tzw. 2-elementowe **zbiory kandydujące**). Sprawdzamy, które z nich są zbiorami częstymi i te dołączamy do listy znalezionych zbiorów częstych.
- Generujemy teraz 3-elementowe zbiory kandydujące: są to wszystkie 3-elementowe podzbiory zbioru L takie, że każdy ich 2-elementowy podzbiór jest zbiorem częstym itd.

Algorytm Apriori (3)

- Wyodrębniamy 1-elementowe podzbiory zbioru L i sprawdzamy, które z nich są zbiorami częstymi.
- Generujemy klasę wszystkich 2-elementowych podzbiorów zbioru L takich, że każdy ich 1-elementowy podzbiór jest zbiorem częstym (są to tzw. 2-elementowe **zbiory kandydujące**). Sprawdzamy, które z nich są zbiorami częstymi i te dołączamy do listy znalezionych zbiorów częstych.
- Generujemy teraz 3-elementowe zbiory kandydujące: są to wszystkie 3-elementowe podzbiory zbioru L takie, że każdy ich 2-elementowy podzbiór jest zbiorem częstym itd.
- Działanie algorytmu kończy się, gdy nie można już wygenerować kolejnych zbiorów kandydujących, a co za tym idzie – zbiorów częstych.

Algorytm FP-Growth (1)

- **Algorytm FP-Growth** (ang. *Frequent Pattern Growth*), zaproponowany w roku 2000 przez J. Hana, H. Pei i Y. Yina, do wyznaczenia zbiorów częstych wykorzystuje konstrukcję tzw. **FP-drzewa**.

Algorytm FP-Growth (1)

- **Algorytm FP-Growth** (ang. *Frequent Pattern Growth*), zaproponowany w roku 2000 przez J. Hana, H. Pei i Y. Yina, do wyznaczenia zbiorów częstych wykorzystuje konstrukcję tzw. **FP-drzewa**.
- Jego pierwszym, początkowym etapem jest transformacja i kompresja wyjściowej bazy danych D do postaci FP-drzewa.

Algorytm FP-Growth (1)

- **Algorytm FP-Growth** (ang. *Frequent Pattern Growth*), zaproponowany w roku 2000 przez J. Hana, H. Pei i Y. Yina, do wyznaczenia zbiorów częstych wykorzystuje konstrukcję tzw. **FP-drzewa**.
- Jego pierwszym, początkowym etapem jest transformacja i kompresja wyjściowej bazy danych D do postaci FP-drzewa.
- Powstałe FP-drzewo jest następnie eksplorowane w celu znalezienia zbiorów częstych.

Algorytm FP-Growth (2)

Ogólny przebieg algorytmu konstrukcji FP-drzewa jest następujący.

- Znajdujemy 1-elementowe zbiory częste (podobnie jak w algorytmie Apriori).

Algorytm FP-Growth (2)

Ogólny przebieg algorytmu konstrukcji FP-drzewa jest następujący.

- Znajdujemy 1-elementowe zbiory częste (podobnie jak w algorytmie Apriori).
- Usuwamy z każdej transakcji te elementy, które nie są 1-elementowymi zbiorami częstymi, otrzymując zmodyfikowaną bazę danych o transakcjach $\overline{D} = \{\overline{T}_1, ..., \overline{T}_n\}$.

Algorytm FP-Growth (2)

Ogólny przebieg algorytmu konstrukcji FP-drzewa jest następujący.

- Znajdujemy 1-elementowe zbiory częste (podobnie jak w algorytmie Apriori).
- Usuwamy z każdej transakcji te elementy, które nie są 1-elementowymi zbiorami częstymi, otrzymując zmodyfikowaną bazę danych o transakcjach $\overline{D} = \{\overline{T}_1, ..., \overline{T}_n\}$.
- Sortujemy elementy każdej zmodyfikowanej transakcji $\overline{T}_i$ według malejących wartości wsparcia kolejnych jej elementów.

Algorytm FP-Growth (2)

Ogólny przebieg algorytmu konstrukcji FP-drzewa jest następujący.

- Znajdujemy 1-elementowe zbiory częste (podobnie jak w algorytmie Apriori).
- Usuwamy z każdej transakcji te elementy, które nie są 1-elementowymi zbiorami częstymi, otrzymując zmodyfikowaną bazę danych o transakcjach $\overline{D} = \{\overline{T}_1, ..., \overline{T}_n\}$.
- Sortujemy elementy każdej zmodyfikowanej transakcji $\overline{T}_i$ według malejących wartości wsparcia kolejnych jej elementów.
- Tworzymy korzeń drzewa, przypisując mu etykietę *null*.

Algorytm FP-Growth (3)

- Dla zmodyfikowanej transakcji $\overline{T}_1$ tworzymy ścieżkę wychodzącą od korzenia drzewa. Kolejność posortowanych elementów transakcji $\overline{T}_1$ na utworzonej w ten sposób ścieżce odpowiada jej kolejnym wierzchołkom (pojawiają się w kolejności malejącego wsparcia elementów transakcji $\overline{T}_1$).

Algorytm FP-Growth (3)

- Dla zmodyfikowanej transakcji $\overline{T}_1$ tworzymy ścieżkę wychodzącą od korzenia drzewa. Kolejność posortowanych elementów transakcji $\overline{T}_1$ na utworzonej w ten sposób ścieżce odpowiada jej kolejnym wierzchołkom (pojawiają się w kolejności malejącego wsparcia elementów transakcji $\overline{T}_1$).
- Dla transakcji $\overline{T}_2$ tworzymy kolejną ścieżkę. Jeżeli początkowe elementy $\overline{T}_2$ pokrywają się z początkowymi elementami transakcji $\overline{T}_1$ (mówimy, że mają **wspólny prefiks**), wówczas obie ścieżki biegną początkowo wspólnie, a następnie się rozdzielają.

Algorytm FP-Growth (3)

- Dla zmodyfikowanej transakcji $\overline{T}_1$ tworzymy ścieżkę wychodzącą od korzenia drzewa. Kolejność posortowanych elementów transakcji $\overline{T}_1$ na utworzonej w ten sposób ścieżce odpowiada jej kolejnym wierzchołkom (pojawiają się w kolejności malejącego wsparcia elementów transakcji $\overline{T}_1$).
- Dla transakcji $\overline{T}_2$ tworzymy kolejną ścieżkę. Jeżeli początkowe elementy $\overline{T}_2$ pokrywają się z początkowymi elementami transakcji $\overline{T}_1$ (mówimy, że mają **wspólny prefiks**), wówczas obie ścieżki biegną początkowo wspólnie, a następnie się rozdzielają.
- Podobnie postępujemy z kolejnymi posortowanymi transakcjami. W każdym wierzchołku (odpowiadającym elementowi transakcji) znajduje się także **licznik**, który wskazuje na liczbę ścieżek przechodzących przez ten wierzchołek.

Algorytm FP-Growth (4)

- Otrzymane za pomocą powyższego algorytmu FP-drzewo ma postać pewnego grafu acyklicznego, w którym wierzchołki odpowiadają elementom (produktom) transakcji, a krawędzie łączą elementy wspólnej transakcji.

Algorytm FP-Growth (4)

- Otrzymane za pomocą powyższego algorytmu FP-drzewo ma postać pewnego grafu acyklicznego, w którym wierzchołki odpowiadają elementom (produktom) transakcji, a krawędzie łączą elementy wspólnej transakcji.
- Dysponując FP-drzewem, eksplorujemy teraz tzw. **ścieżki prefiksowe** kolejnych **wzorców** α tego drzewa. Ścieżka prefiksowa wzorca α to ścieżka FP-drzewa, której końcem jest wierzchołek α.

Algorytm FP-Growth (4)

- Otrzymane za pomocą powyższego algorytmu FP-drzewo ma postać pewnego grafu acyklicznego, w którym wierzchołki odpowiadają elementom (produktom) transakcji, a krawędzie łączą elementy wspólnej transakcji.
- Dysponując FP-drzewem, eksplorujemy teraz tzw. **ścieżki prefiksowe** kolejnych **wzorców** α tego drzewa. Ścieżka prefiksowa wzorca α to ścieżka FP-drzewa, której końcem jest wierzchołek α.

- Zbiór wszystkich ścieżek prefiksowych danego wzorca α (może ich być więcej) tworzy tzw. **warunkową bazę wzorca** α.

Algorytm FP-Growth (4)

- Otrzymane za pomocą powyższego algorytmu FP-drzewo ma postać pewnego grafu acyklicznego, w którym wierzchołki odpowiadają elementom (produktom) transakcji, a krawędzie łączą elementy wspólnej transakcji.
- Dysponując FP-drzewem, eksplorujemy teraz tzw. **ścieżki prefiksowe** kolejnych **wzorców** α tego drzewa. Ścieżka prefiksowa wzorca α to ścieżka FP-drzewa, której końcem jest wierzchołek α.

- Zbiór wszystkich ścieżek prefiksowych danego wzorca α (może ich być więcej) tworzy tzw. **warunkową bazę wzorca** α.
- Na podstawie bazy konstruujemy **warunkowe FP-drzewo wzorca** α.

Algorytm FP-Growth (4)

- Otrzymane za pomocą powyższego algorytmu FP-drzewo ma postać pewnego grafu acyklicznego, w którym wierzchołki odpowiadają elementom (produktom) transakcji, a krawędzie łączą elementy wspólnej transakcji.
- Dysponując FP-drzewem, eksplorujemy teraz tzw. **ścieżki prefiksowe** kolejnych **wzorców** α tego drzewa. Ścieżka prefiksowa wzorca α to ścieżka FP-drzewa, której końcem jest wierzchołek α.

- Zbiór wszystkich ścieżek prefiksowych danego wzorca α (może ich być więcej) tworzy tzw. **warunkową bazę wzorca** α.
- Na podstawie bazy konstruujemy **warunkowe FP-drzewo wzorca** α.
- Praktyczną konstrukcję FP-drzewa oraz jego eksplorację w celu znalezienia zbiorów częstych, czyli pełną postać algorytmu FP-Growth prześledzimy na przykładzie.