Aprendizado Descritivo

Aula 06 – Mineração de sequências

Professor Renato Vimieiro

DCC/ICEx/UFMG

Introdução

- Nessa aula, vamos discutir o problema de mineração de sequências em bases de dados
- Esse problema ocorre com frequência em diversas áreas
  - Identificar trajetórias dos alunos de computação
  - Identificar perfil (temporal) de compras dos clientes (celular -> capa protetora -> fone de ouvido)
  - Identificar padrões de genes e proteínas no genoma
- Enquanto itemsets são padrões intra-transações, aqui estamos buscando padrões inter-transações

Introdução

- Para ilustrar, considere a seguinte base de dados
- Os clientes fazem diversas compras na loja
  - Mais de 1 item pode ser adquirido em uma transação
- Existe algum padrão de compras?
  - Determinados itens são comprados em sequência?
- Esse problema é conhecido como **mineração de sequências (frequentes)**

ID Cliente	Data	Transação
1	25/06/19	aveia
1	30/06/19	castanha
2	10/06/19	granola, mel
2	15/06/19	aveia
2	20/06/19	banana, suco, leite
3	25/06/19	aveia, iogurte, leite
4	25/06/19	aveia
4	30/06/19	banana, leite
4	25/07/19	castanha
5	12/06/19	castanha
6	10/06/19	aveia, granola
6	11/06/19	leite
6	17/06/19	banana, leite

Definições

- A base de dados é um conjunto de transações consistindo em:
    - ID do cliente
    - Timestamp da transação
    - Itens 'comprados'
- Os itens da transação são um itemset de uma coleção de possíveis itens
- Uma **sequência** é uma **lista ordenada de itemsets**
    - Os itemsets também são chamados de elementos
- Para um conjunto de itens I, uma sequência $s = \langle s_1 s_2 \dots s_n \rangle$ em que cada $s_i \subseteq I$ é um itemset
    - Por definição, um item não pode aparecer mais de uma vez num itemset, mas pode aparecer várias vezes numa sequência

Definições

- Por exemplo a sequência $\langle (gm)a(bsl) \rangle$ representa a sequência de compras do cliente 2 na base de dados anterior
  - Itemsets são delimitados por parêntesis; itemsets unitários são representados sem parêntesis
- Uma sequência $\alpha = \langle a_1 a_2 \ldots a_n \rangle$ é uma subsequência de uma sequência $\beta = \langle b_1 b_2 \ldots b_m \rangle$, $\alpha \sqsubseteq \beta$, se existe uma função φ: $[1{:}\, n] \rightarrow [1{:}\, m]$ tal que
  - $a_i \subseteq b_{\varphi(i)}$; e
  - $\forall i, j \; [i < j \; \rightarrow \varphi(i) < \varphi(j)]$
- As sequências $\langle (gm)b \rangle$, $\langle mab \rangle$ e $\langle a \rangle$ são subsequências de $\langle (gm)a(bsl) \rangle$
- A sequência $\langle (gma)b \rangle$ **não** é uma subsequência de $\langle (gm)a(bsl) \rangle$
- Note que a ordem é definida somente entre elementos, e não dentro dos itemsets
  - Contudo, vamos assumir que os elementos são dispostos conforme alguma ordem dentro dos itemsets (em nosso caso, a ordem que forma apresentados na base original)

Definições

- Podemos redefinir a base de dados como um conjunto de pares (sid, s) em que sid é um identificador de sequência e s uma sequência
  - Cada identificador de cliente é um sid
  - As diferentes transações de um cliente ordenados pelo tempo formam a sequência
- Um cliente (sid, s) suporta uma sequência α se α ⊑ s para
- Assim, definimos o suporte de uma sequência como
  - $\mathrm{sup}(\alpha) = |\{(sid, s)|\ \alpha \sqsubseteq s\}|$
- Exemplo:
  - $\mathrm{sup}(\langle \mathrm{a} \rangle) = 5$
  - $\mathrm{sup}(\langle \mathrm{gb} \rangle) = 2$
  - $\mathrm{sup}(\langle \mathrm{l} \rangle) = 4$

sid	s
1	$\langle \mathrm{ac} \rangle$
2	$\langle (\mathrm{gm})\mathrm{a}(\mathrm{bsl}) \rangle$
3	$\langle (\mathrm{ail}) \rangle$
4	$\langle \mathrm{a}(\mathrm{bl})\mathrm{c} \rangle$
5	$\langle \mathrm{c} \rangle$
6	$\langle (\mathrm{ag})\mathrm{l}(\mathrm{bl}) \rangle$

Definições

- Uma sequência α é frequente se sup(α) ≥ minsup
- Uma sequência α tem tamanho k (é uma k-sequência) se $\sum |a_i| = k$
- Uma sequência frequente é dita máxima se não existe uma supersequência própria que seja frequente
- Ela é fechada se não existe uma supersequência própria com o mesmo suporte

Mineração de sequências frequentes

- O problema de mineração de sequências frequentes consiste em encontrar todas as sequências cujo suporte esteja acima de um limiar mínimo definido pelo usuário
- Existem abordagens tanto para minerar todo o conjunto de sequências frequentes quanto para representações compactas desse conjunto
- Nessa aula veremos apenas as abordagens para minerar todo o conjunto de sequências frequentes
- Essas abordagens são extensões dos principais algoritmos para mineração de conjuntos de itens frequentes

Generalized Sequential Patterns (GSP)

- O GSP é um algoritmo baseado no Apriori para minerar sequências frequentes
- Ele também foi proposto por Agrawal e Srikant em 1996
- Por ser baseado no Apriori, o algoritmo adota a estratégia de busca em largura
- O algoritmo usa sequências frequentes de tamanho k-1 para gerar candidatas de tamanho k e avaliar o suporte,
- Ele também emprega a propriedade de antimonotonicidade do suporte para podar o espaço de busca

Generalized Sequential Patterns (GSP)

- Assim como o Apriori, o algoritmo faz diversas passadas sobre a base de dados
- Na primeira passada, o algoritmo verifica o suporte de cada item
  - Os itens individualmente são sequências simples de tamanho 1
- Pela propriedade do Apriori, somente os itens frequentes são mantidos e servem de base para a geração dos candidatos de tamanho 2
- Cada par $\langle x \rangle$ e $\langle y \rangle$ de sequências de tamanho 1 dá origem a duas sequências de tamanho 2
  - $\langle xy \rangle$
  - $\langle (xy) \rangle$
- A exceção se dá quando x=y, nesse caso somente a primeira é gerada
- Logo, o conjunto de candidatos de tamanho 2 é:
  - $C^{(2)} = \left\{ \langle xy \rangle \mid (x, y) \in F^{(1)} \times F^{(1)} \right\} \cup \left\{ \langle (xy) \rangle \mid (x, y) \in F^{(1)} \times F^{(1)} \wedge x \neq y \right\}$

Generalized Sequential Patterns (GSP)

- Os candidatos que possuam subsequências de tamanho k-1 infrequentes são removidos do conjunto
- Então, o suporte dos candidatos é computado com uma passada na base, e os infrequentes são descartados
- A partir de k=3, os candidatos são gerados da seguinte forma:
  - Sejam s1 e s2 duas sequências frequentes de tamanho k-1 tais que ambas sejam idênticas após a remoção do primeiro item de s1 e do último item de s2
  - As sequências são unidas para gerar uma candidata de tamanha k
    - A nova candidata será a sequência s1 estendida com o último item de s2
  - O último item será um elemento separado se ele era um elemento separado em s2, ou será agregado ao último elemento de s1 caso contrário
- O algoritmo repete o processo enquanto houverem candidatos no próximo nível

Sequential Pattern Discovery using Equivalence classes (Spade)

- Outra abordagem para mineração de sequências frequentes foi proposta por Zaki em 2001
- O algoritmo Spade é baseado no Eclat
- Assim como o Eclat, ele utiliza uma representação da base de dados similar à vertical e divide o espaço de busca de acordo com o prefixo das sequências

Sequential Pattern Discovery using Equivalence classes (Spade)

- Para obter a representação vertical, vamos associar a cada item *i* uma tupla (sid,pos(i))
  - pos(i) é a lista de todos os elementos em que i ocorre na sequência referente ao cliente sid
- A lista de todos os pares sequência-posição de um item é chamada de **poslist** e é denotada por $\mathcal{L}(i)$
- Exemplos:
  - $\mathcal{L}(l) = \{(2,\{3\}),(3,\{1\}),(4,\{2\}),(6,\{2,3\})\}$
  - $\mathcal{L}(g) = \{(2,\{1\}),(6,\{1\})\}$
- A representação vertical da base pode ser obtida pelas poslists de todos os itens
- Note que sup(i) = $|\mathcal{L}(i)|$

sid	s
1	$\langle ac \rangle$
2	$\langle (gm)a(bsl) \rangle$
3	$\langle (ail) \rangle$
4	$\langle a(bl)c \rangle$
5	$\langle c \rangle$
6	$\langle (ag)l(bl) \rangle$

item	pos(item)
a	$\{(1,1),(2,2),(3,1),(4,1),(6,1)\}$
b	$\{(2,3),(4,2),(6,3)\}$
c	$\{(1,2),(4,3),(5,1)\}$
g	$\{(2,1),(6,1)\}$
i	$\{(3,1)\}$
l	$\{(2,3),(3,1),(4,2),(6,23)\}$
s	$\{(2,3)\}$

Sequential Pattern Discovery using Equivalence classes (Spade)

- Zaki demonstrou que novas sequências podem ser geradas a partir da **junção temporal** de duas sequências que pertençam a uma mesma classe de equivalência (compartilham um prefixo de tamanho k-1)
- O suporte pode ser computado pela interseção das poslists
- Supondo que as sequências a serem unidas compartilham um prefixo P, três situações podem ocorrer:
  - Juntar duas sequências (Px) e (Py): resulta em (Pxy)
  - Juntar duas sequências (Px) e Py: resulta em (Px)y
  - Juntar duas sequências Px e Py: pode resultar em Pxy, Pyx e P(xy) dependendo da ordem temporal de x e y; um caso particular ocorre quando x=y, nesse caso, a junção só pode resultar em Pxx

Sequential Pattern Discovery using Equivalence classes (Spade)

- Juntar duas sequências (Px) e (Py): resulta em (Pxy)
  - Sejam $\mathcal{L}(Px)$ e $\mathcal{L}(Py)$ as poslists de (Px) e (Py). A poslist de (Pxy) é gerada da seguinte forma: $\mathcal{L}((Pxy)) = \{(i, pos((Px)) \cap pos((Py))) \mid (i, pos((Px))) \in \mathcal{L}((Px)) \wedge (i, pos((Py))) \in \mathcal{L}((Py)) \wedge pos((Px)) \cap pos((Py)) \neq \emptyset\}$;
  - Ou seja é o conjunto de sequência-posições em que ambos ocorrem ao mesmo tempo
  - O caso P(xy) é análogo a esse
- Juntar duas sequências (Px) e Py: resulta em (Px)y
  - $\mathcal{L}((Px)y) = \{(i, \{v \in pos(Py) \mid \exists u \in pos((Px))\ u < v \wedge (i, pos(Py)) \in \mathcal{L}(Py) \wedge (i, pos((Px))) \in \mathcal{L}((Px))\})\}$;
  - Ou seja, são todas as sequências em que ambas acontecem, porém agora somente as posições em que y ocorre temporalmente após x são mantidas
  - O caso é equivalente a Pxy e Pyx

Sequential Pattern Discovery using Equivalence classes (Spade)

- Sejam as sequências $\langle gb \rangle$ e $\langle gl \rangle$ pertencentes à classe de equivalência de g, e suas respectivas poslists $\mathcal{L}(gb) = \{(2,3), (6,3)\}$ e $\mathcal{L}(gl) = \{(2,3), (6,3)\}$
  - $\mathcal{L}(g(bl)) = \{(2,3), (6,3)\}$
  - $\mathcal{L}(gbl) = \emptyset$
  - $\mathcal{L}(glb) = \emptyset$
- O algoritmo segue explorando o espaço de busca enquanto as classes de equivalência não forem vazias

Sequential Pattern Discovery using Equivalence classes (Spade)

ALGORITHM 10.2. Algorithm SPADE

`// Initial Call:` $\mathcal{F} \leftarrow \emptyset,\ k \leftarrow 0,$

$P \leftarrow \{\langle s, \mathcal{L}(s)\rangle \mid s \in \Sigma, sup(s) \geq minsup\}$

SPADE (P, $minsup$, $\mathcal{F}$, k):

foreach $\mathbf{r}_a \in P$ **do**

  $\mathcal{F} \leftarrow \mathcal{F} \cup \{(\mathbf{r}_a, sup(\mathbf{r}_a))\}$

  $P_a \leftarrow \emptyset$

  foreach $\mathbf{r}_b \in P$ **do**

    $\mathbf{r}_{ab} = \mathbf{r}_a + \mathbf{r}_b$

    $\mathcal{L}(\mathbf{r}_{ab}) = \mathcal{L}(\mathbf{r}_a) \cap \mathcal{L}(\mathbf{r}_b)$

    if $sup(\mathbf{r}_{ab}) \geq minsup$ **then**

      $P_a \leftarrow P_a \cup \{\langle \mathbf{r}_{ab}, \mathcal{L}(\mathbf{r}_{ab})\rangle\}$

  if $P_a \neq \emptyset$ **then** SPADE ($\mathbf{P}$, $minsup$, $\mathcal{F}$, $k+1$)

Leitura

- Seções 10.1 e 10.2 Zaki e Meira
- Capítulo 11 Aggarwal e Han
- Zaki, M.J. SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning* **42,** 31–60 (2001). https://doi.org/10.1023/A:1007652502315
- Srikant R., Agrawal R. (1996) Mining sequential patterns: Generalizations and performance improvements. In: Apers P., Bouzeghoub M., Gardarin G. (eds) Advances in Database Technology — EDBT '96. EDBT 1996. Lecture Notes in Computer Science, vol 1057. Springer, Berlin, Heidelberg. https://doi.org/10.1007/BFb0014140

Aprendizado Descritivo

Aula 06 – Mineração de sequências

Professor Renato Vimieiro

DCC/ICEx/UFMG