Aprendizado Descritivo

Aula 05 – Representações compactas

Professor Renato Vimieiro

DCC/ICEx/UFMG

Introdução

- Nessa aula, vamos discutir representações compactas para o conjunto de todos os conjuntos de itens frequentes de uma base de dados
- Representações compactas são subconjuntos a partir dos quais é possível derivar todos os conjuntos de itens frequentes
- Para motivar a necessidade dessas representações, considere uma base de dados com somente duas transações e 100 itens:
  - D = {(0, a_1, a_2, ..., a_{50}), (1, a_1, a_2, ..., a_{100})}
- Se considerarmos um minsup=1, essa base terá
  - $\binom{100}{1} + \binom{100}{2} + \cdots + \binom{100}{100} = 2^{100} - 1 \approx 1.27E^{30}$

Introdução

- Nesse caso, o problema se torna incomputável por qualquer das abordagens que vimos anteriormente
- Embora esse seja um caso extremo, não é raro que situações similares a essa ocorram na prática
  - É possível, por exemplo, que um subconjunto das transações e itens apresentem esse comportamento em uma base de dados maior
- Note que podemos particionar os itemsets frequentes em duas classes de equivalência:
  - Os que ocorrem em ambas as transações; e
  - Os que ocorrem somente na segunda

Introdução

- Os que ocorrem em ambas as transações, possuem cobertura c(X)=01, e, portanto, são equivalentes a $a_1a_2...a_{50}$
- Os que ocorrem somente na segunda transação, possuem cobertura c(X)=1, e, portanto, são equivalentes a $a_1a_2...a_{100}$
  - Além disso, se estivéssemos somente interessados nos itemsets frequentes sem a informação da frequência, todos seriam equivalentes a esse itemset
- Em outras palavras, os mais de 10^{30} itemsets que seriam retornados por qualquer dos algoritmos vistos poderiam ser representados somente por esses dois conjuntos
- Esses conjuntos formam, dessa forma, uma **representação compacta** de todo o conjunto de itemsets frequentes
- Em particular, eles estão relacionados a dois tipos de representações compactas que veremos nessa aula
  - Conjuntos frequentes máximos
  - Conjuntos frequentes fechados

Representações compactas

- O particionamento dos itemsets no exemplo anterior se deu pelos tidsets que compunham suas extensões
- De fato, o raciocínio se aplica a qualquer base de dados. Ou seja, podemos particionar os itemsets conforme sua cobertura
- Dentro de cada classe de equivalência, podemos ordenar os elementos conforme a relação de subconjunto
  - O maior elemento da classe é chamado de conjunto fechado ou **closed itemset**
  - Os menores elementos da classe são chamados de **minimal generators**
- Os maiores elementos entre todos os conjuntos fechados são chamados de conjuntos frequentes máximos (**maximal itemsets**)

Representações compactas

- Os conjuntos máximos são os maiores itemsets frequentes
- Eles definem a ‘borda’ entre o que é frequente e infrequente
- Como, por definição, não existem conjuntos frequentes maiores que eles, **todos os conjuntos frequentes podem ser derivados a partir dos conjuntos máximos**
- No entanto, o cálculo do suporte não pode ser obtido diretamente desses itemsets, sendo necessária uma nova passada na base de dados para computá-lo
- O itemset $a_1a_2...a_{100}$ no nosso exemplo inicial é um conjunto frequente máximo

Representações compactas

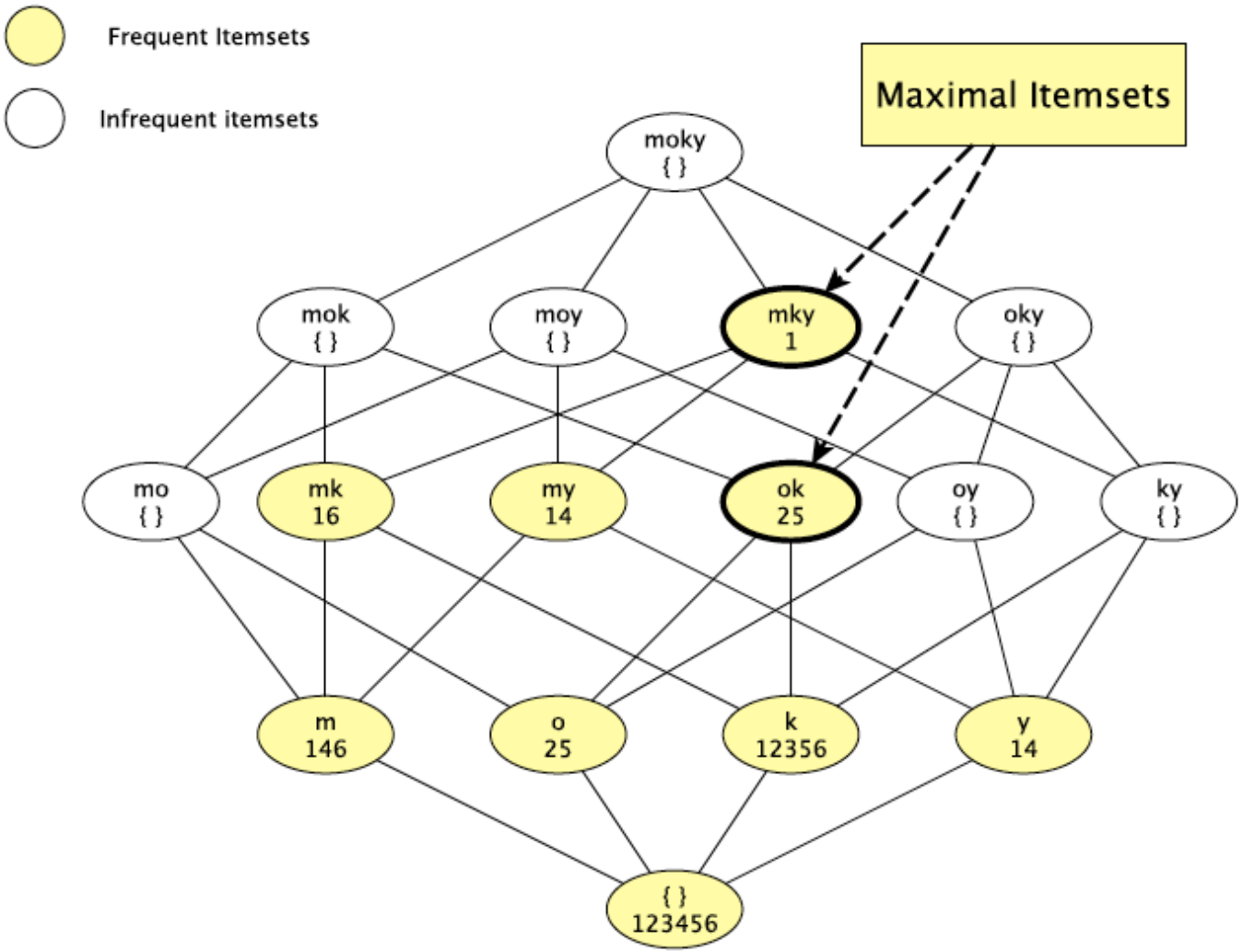
- Essa necessidade de novas passadas na base de dados para computar o suporte dos itemsets frequentes a partir dos máximos torna a representação incompleta
- Os conjuntos fechados, por outro lado, são uma representação completa, já que tanto os itemsets quanto seu suporte podem ser derivados desses conjuntos
- Como dito, todo conjunto fechado dá origem a uma classe de equivalência
  - [X] = {Y ⊆ I | c(Y) = c(X)} = {Y ⊆ I | i(c(Y)) = X}
- Assim, podemos verificar o suporte de um itemset frequente a partir dos conjuntos fechados da seguinte forma
  - $\sup(X) = \max\{\sup(Y) | \, Y \in \mathcal{C} \wedge X \subseteq Y\}$
  - Em outras palavras, basta encontrarmos a classe de equivalência à qual o itemset pertence; todo itemset frequente ou é fechado ou pertence à classe de equivalência de algum conjunto fechado, como o suporte é anti-monotônico, se ele não for fechado, ele pertence à classe do de maior suporte.

Representações compactas

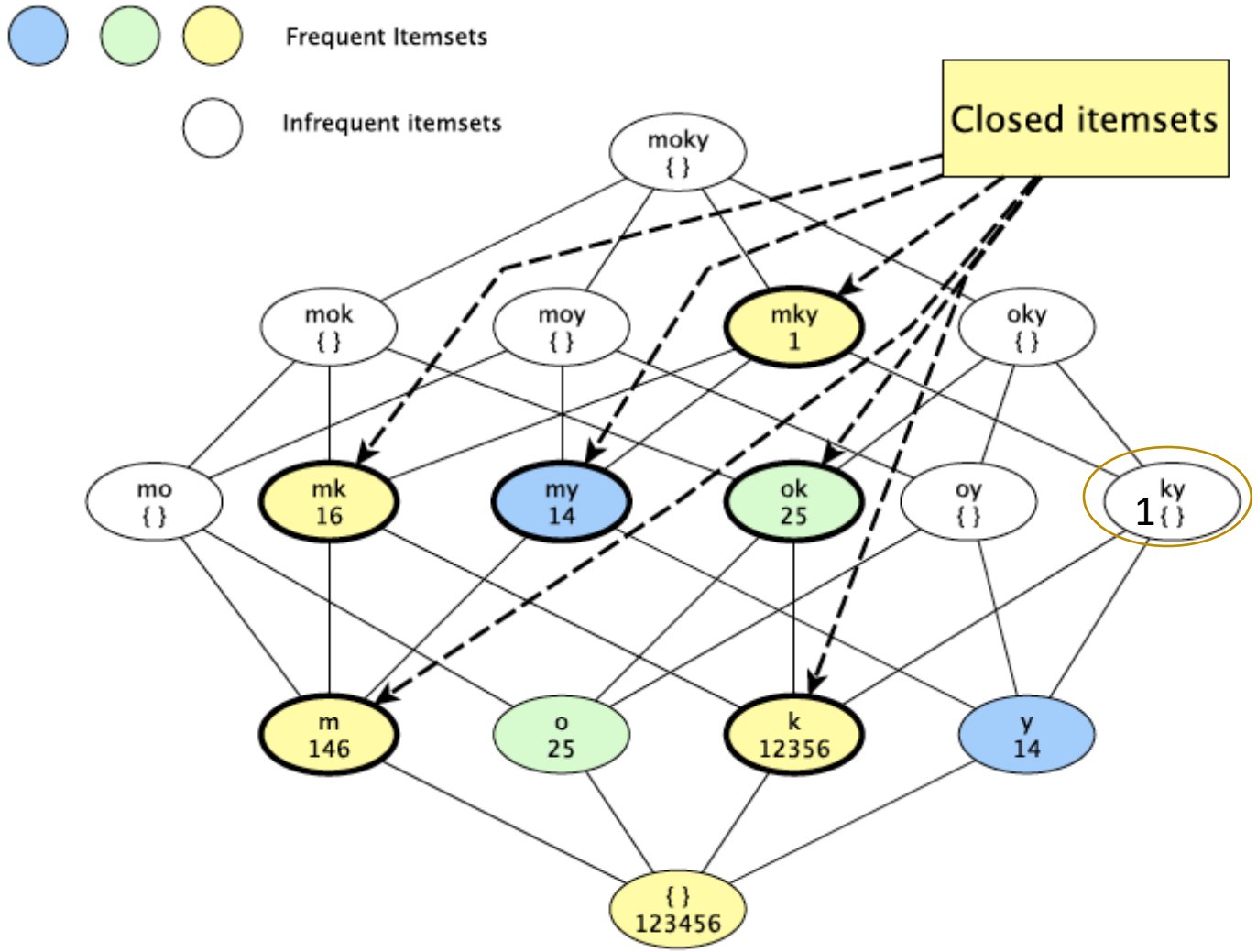
- Exemplo
  - minsup=1

TID	Muesli (m)	Oats (o)	Milk (k)	Yoghurt (y)
1	1	0	1	1
2	0	1	1	0
3	0	0	1	0
4	1	0	0	1
5	0	1	1	0
6	1	0	1	0

Representações compactas



Representações compactas



Algoritmos para encontrar representações compactas

- Os exemplos mostram que as representações compactas apresentam vantagens sobre o conjunto de todos os itemsets frequentes
- No entanto, se usarmos os algoritmos vistos anteriormente para encontrar essas representações, não temos ganho computacional algum em relação ao problema anterior
  - Pode ser que a mineração desses padrões continue inviável
- Existem diversos algoritmos específicos para mineração de itemsets máximos e fechados
- Veremos um representante de cada desses algoritmos

DCI_Closed

- O algoritmo DCI_Closed foi proposto em 2004 por C. Lucchese, S. Orlando e R. Perego
- Ele também adota uma representação vertical da base de dados para facilitar a verificação dos conjuntos fechados
- O algoritmo explora o espaço de busca usando uma estratégia dividir-e-conquistar
- Os autores demonstraram que o problema pode ser decomposto em subproblemas independentes, permitindo inclusive uma solução paralela

DCI_Closed

- A ideia central do algoritmo é ‘escalar’ o reticulado de itemsets, percorrendo cada classe de equivalência uma única vez
- Somente um candidato de cada classe é avaliado para computar o seu conjunto fechado
- Novamente, assume-se uma ordem lexicográfica sobre os itens da base, e sua extensão sobre os itemsets
  - Qualquer ordem serve, inclusive a ordem sobre os rótulos dos itens
  - Essa ordem será representado por $\prec$
- Novos candidatos são gerados a partir dos conjuntos fechados obtidos, estendendo-os com itens ainda não investigados
  - Esses candidatos são chamados de **geradores**
  - Formalmente, um gerador é um conjunto X = Yi, para um conjunto fechado Y e um item i

DCI_Closed

- Um gerador X = Yi é dito **ordem-conservante** sse $i \prec (i(c(X)) - X)$
  - Em palavras, X é ordem-conservante se todo item que tiver que ser adicionado a X para obter o conjunto fechado for maior que i
- Teorema 1: Para todo conjunto fechado $Y \neq i(c(\emptyset))$, existe uma sequência de $n \geq 1$ extensões (items) $i_0 \prec i_1 \prec ... \prec i_{n-1}$ tais que $gen_0 = Y_0 i_0$, $gen_1 = Y_1 i_1$, $gen_{n-1} = Y_{n-1} i_{n-1}$, em que todos os gen_k são ordem-conservantes, $Y_0 = i(c(\emptyset))$, $Y_{j+1} = i(c(Y_j i_j))$ e $Y_n = Y$.
- Corolário: Essa sequência é única.

DCI_Closed

- O problema agora é verificar se um gerador é ordem-conservante
- Lema 1: Seja gen = Yi, para um conjunto fechado Y e item i. Se $\exists j \prec i\ [j \notin gen \wedge c(gen) \subseteq c(j)]$, então gen não é ordem-conservante.
  - Intuitivamente, $c(gen) \subseteq c(j)$ implica em $j \in i(c(gen))$, e como $j \notin gen$, $j \in i(c(gen)) - gen$; ou seja, $i \nprec i(c(gen)) - gen$
- Sendo assim, basta mantermos uma lista de elementos menores que i não pertencentes a gen para verificarmos se ele é ordem-conservante durante a execução do algoritmo
  - Essa lista é chamada de **pre-set**
  - Não há necessidade de manter os conjuntos fechados em memória!
- O espaço de busca pode ser percorrido a partir de $i(c(\emptyset))$ e todos os itens frequentes como possíveis extensões
  - Os geradores são avaliados conforme a ordem lexicográfica
  - Se encontrarmos um gerador não ordem-conservante, podamos o ramo
  - Após explorar o ramo com um item i, ele é colocado no **pre-set**

DCI_Closed

```
1: procedure DCI_CLOSED_d (CLOSED_SET, PRE_SET, POST_SET)
2:   while POST_SET ≠ ∅ do
3:     i ← min_≺(POST_SET)
4:     POST_SET ← POST_SET \ i
5:     new_gen ← CLOSED_SET ∪ i                ▷ Build a new generator
6:     if supp(new_gen) ≥ min_supp and
  :       ¬is_dup(new_gen, PRE_SET) then
  :                  ▷ if new_gen is both frequent and order preserving
7:       CLOSED_SET_New ← new_gen
8:       POST_SET_New ← ∅
9:       for all j ∈ POST_SET do          ▷ Compute closure of new_gen
10:        if g(new_gen) ⊆ g(j) then
11:          CLOSED_SET_New ← CLOSED_SET_New ∪ j
12:        else
13:          POST_SET_New ← POST_SET_New ∪ j
14:        end if
15:      end for
16:      Write Out CLOSED_SET_New and its support
17:      DCI_CLOSED_d(CLOSED_SET_New, PRE_SET, POST_SET_New)
18:      PRE_SET ← PRE_SET ∪ i
19:    end if
20:  end while
21: end procedure
```

```
22: function is_dup(new_gen, PRE_SET)              ▷ Duplicate check
23:   for all j ∈PRE_SET do
24:     if g(new_gen) ⊆ g(j) then
25:       return TRUE                  ▷ new_gen is not order preserving
26:     end if
27:   end for
28:   return FALSE
29: end function
```

DCI_Closed

- Exemplo: minsup=2

TID	Muesli (a)	Oats (b)	Milk (c)	Yoghurt (d)	Biscuits (e)	Tea (f)
1	1	0	1	1	0	1
2	0	1	1	0	0	0
3	0	0	1	0	1	1
4	1	0	0	1	0	0
5	0	1	1	0	0	1
6	1	0	1	0	0	1

MAFIA: Maximal Frequent Itemset Algorithm

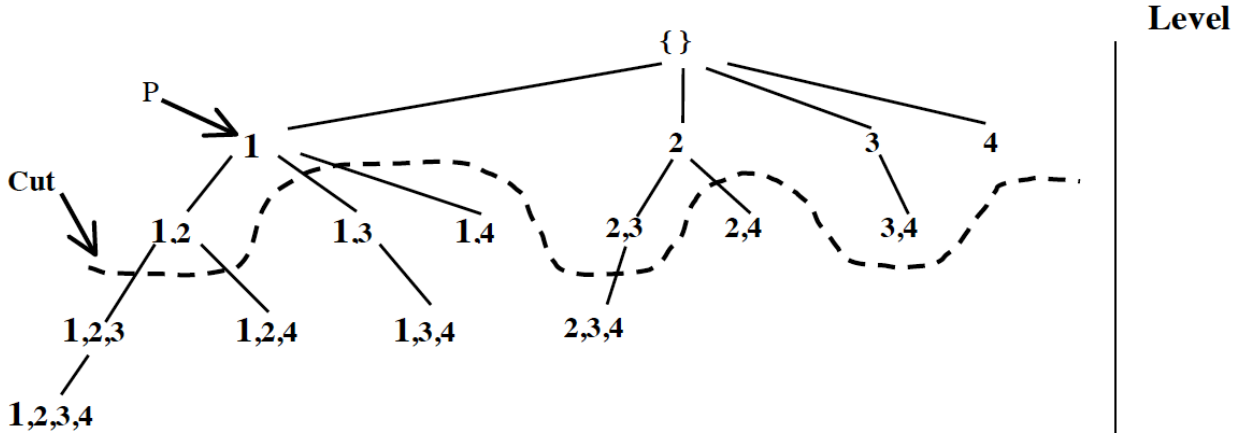
- O algoritmo MAFIA foi proposto em 2001 por D. Burdick, M. Calimlim e J. Gehrke
- A ideia geral do algoritmo é a de explorar o reticulado de itemsets com uma abordagem best-first/branch-and-bound
- Assim como o Eclat, o MAFIA também assume uma representação vertical dos dados usando vetores de bits
- O algoritmo também assume uma ordem lexicográfica sobre os itens e a ordem parcial de subconjuntos entre os itemsets durante a exploração

MAFIA: Maximal Frequent Itemset Algorithm

- Durante a exploração do espaço de busca, o algoritmo divide os itens em dois grupos
  - **Head**: contendo o rótulo do nó corrente (itemset) na exploração
  - **Tail**: os itens que são maiores que o maior elemento do Head (possíveis extensões para o itemset)
- O conjunto de todos os itens que podem aparecer numa dada subárvore é a união entre o head e o tail (chamado de **HUT** – head union tail – pelos autores)
- Ao invés de adotar uma exploração puramente em profundidade, em cada nó, o algoritmo avalia os filhos imediatos para remover possíveis extensões do tail
  - Eles chamam essa estratégia de **reordenamento dinâmico** (**dynamic reordering**)

MAFIA: Maximal Frequent Itemset Algorithm

- Ao explorar o nó P na figura ao lado
  - Head = 1
  - Tail = 234
  - HUT = 1234
- Antes de explorar em profundidade o nó, o algoritmo avalia os filhos 12, 13 e 14
- Como só 12 é frequente, 3 e 4 podem ser removidos do tail porque qualquer candidato dessa subárvore que inclua esses itens será infrequente
  - O ramo de 12 é podado



MAFIA: Maximal Frequent Itemset Algorithm

- Note que, sempre que uma folha é visitada, um candidato a itemset máximo é encontrado
  - Ele será incluído na solução final somente se não possuir um superconjunto já incluído
  - A visitação em ordem lexicográfica em profundidade garante que conjuntos não tenham que ser removidos da solução final
- Outra poda vem do fato de que itemsets máximos são também fechados e que, portanto, para um itemset X (head) e y ∈ X.tail:
  - Se $c(X) \subseteq c(y)$, então $Xy \subseteq i(c(X))$ (isto é, y pertence ao conjunto fechado da classe à qual X pertence)
  - Nesse caso, y pode ser incorporado ao head e removido do tail
- Essa poda é chamada de Parent Equivalence Pruning (PEP)

MAFIA: Maximal Frequent Itemset Algorithm

- Finalmente, podemos usar a propriedade do Apriori para descartar um ramo baseado no HUT
  - Lembrando que HUT é a união do head com o tail e representa o maior itemset que pode ser obtido a partir dessa subárvore
- Se o HUT for subconjunto de algum itemset máximo já descoberto anteriormente, sabemos que ele e todos os seus subconjuntos são frequentes, e, além disso, não podem ser máximos. Portanto, podemos podar a subárvore.

MAFIA: Maximal Frequent Itemset Algorithm

```
Pseudocode: MAFIA (C, MFI, Boolean IsHUT) {
   name HUT = C.head ⋃ C.tail;
   if HUT is in MFI
      Stop generation of children and return
Count all children, use PEP to trim the tail, and reorder
   by increasing support.
   For each item i in C.trimmed_tail {
      IsHUT = whether i is the first item in the tail
      newNode = C ⋃ I
      MAFIA(newNode, MFI, IsHUT) }
   if (IsHUT and all extensions are frequent)
      Stop search and go back up subtree
   if (C is a leaf and C.head is not in MFI)
      Add C.head to MFI
}
```

MAFIA: Maximal Frequent Itemset Algorithm

- Exemplo: minsup=2

TID	Muesli (a)	Oats (b)	Milk (c)	Yoghurt (d)	Biscuits (e)	Tea (f)
1	1	0	1	1	0	1
2	0	1	1	0	0	0
3	0	0	1	0	1	1
4	1	0	0	1	0	0
5	0	1	1	0	0	1
6	1	0	1	0	0	1

Leitura

- Seção 9.1 Zaki e Meira
- Seção 6.2.6 Han et al.
- Burdick, D., Calimlim, M., & Gehrke, J. (2001, April). Mafia: A maximal frequent itemset algorithm for transactional databases. In *Proceedings 17th international conference on data engineering* (pp. 443-452). IEEE.
- Lucchese, C., Orlando, S., & Perego, R. (2004, November). DCI Closed: A Fast and Memory Efficient Algorithm to Mine Frequent Closed Itemsets. In *FIMI*.

Aprendizado Descritivo

Aula 05 – Representações compactas

Professor Renato Vimieiro

DCC/ICEx/UFMG