Aprendizado Descritivo

1 Definição

Aprendizado descritivo se enquadra na seguinte classificação dos métodos de aprendizado de máquina:

Aprendizado preditivo: supervisionado ou não, tem como objetivo predizer dada característica (*target*).

Aprendizado descritivo: supervisionado ou não, tem como objetivo obter uma descrição para os dados.

Contrastando, o aprendizado preditivo pretende prever dados futuros ou desconhecidos, enquando o aprendizado descritivo prentende trazer uma instrospecção sobre os dados. Tal diferença pode se apresentar de forma tênue. Um exemplo são os algorítmos de agrupamento, que podem ser utilizado em ambas formas. Particularmente, o algorítmo *K-means* se apresenta:

Preditivo: *K-means*: $P \to C$

Descritivo: *K-means*: $A \to C$

Em aprendizado preditivo, o *K-means* possui como domínio toda a população, ainda que o treinamento seja realizado apenas com uma amostra. Já no aprendizado descritivo, nos limitamos à amostra, afinal ela constitui o alvo total de análise.

2 Mineração de itens frequentes

Para um conjunto de dados:

1. Chamamos de itens os elementos do conjunto de variáveis de análise I.
2. Um conjunto $X \subseteq I$ é denominado *itemset*
3. O conjunto de todos *k-itemsets* é denotado port $I^{(k)}$.
4. A amostra de transações é denominada por T.
5. Cada transação é identificada unicamente por um `tid`.
6. Um conjunto $Y \subseteq T$ é denominado *tidset*.
7. Cada transação consiste de um identificador, e um conjunto de itens: $(\text{tid}, X),\ X \subseteq I$

Formalmente, um conjunto de dados será uma tripla (T, I, D), onde $D \subseteq T \times I$ é uma relação binária:

$$(t, i) \in D \iff \left[i \in X \text{ na transação } (t, X)\right]$$

Uma transação pode **conter** um *itemset*, e tal relação é definida da seguinte forma:

$$X \subseteq t \iff \forall\, i \in X : (t, i) \in D$$

O conjunto de transações que contém um *itemset* X é denominado **extensão** ou **cobertura** de X.
Tal conjunto é definido pela seguinte operação:

$$c : \mathcal{P}(I) \to \mathcal{P}(T)$$
$$c(X) = \big\{ t \in T \mid \forall\, i \in X : (t, i) \in D \big\}$$

Analogamente, o maior conjunto de itens comuns à um *tidset* Y é chamado de **intensão** de Y.

$$i : \mathcal{P}(T) \to \mathcal{P}(I)$$
$$i(Y) = \big\{ x \in I \mid \forall\, t \in Y : (t, x) \in D \big\}$$

Desta forma, podemos representar um conjunto de dados de duas formas:

Horizontal: o conjunto de transações e suas intensões: $\big\{ \big(t, i(t)\big) \mid t \in T \big\}$

Vertical : o conjunto de itens e suas coberturas: $\big\{ \big(x, c(x)\big) \mid x \in I \big\}$

2.1 Metodologia

A identificação de regras de associação, em geral, envolve duas etapas:

1. Mineração de conjuntos de itens frequentes
2. Descoberta de regras de associação

Devido à natureza computacionalmente intensa da primeira etapa, nossos estudos a focam.

O limiar que separa os itens frequentes dos infrequentes é chamado de **suporte mínimo**.
O suporte mínimo de um *itemset* é o tamanho de sua cobertura:

$$\sup(X) = \big|c(X)\big|$$

Admite-se também a definição de **suporte relativo**:

$$\mathrm{rsup}(X) = \frac{\big|c(X)\big|}{|T|}$$

2.2 Algoritmos

O espaço de busca do problema é o conjunto potência do conjunto de itens. Se considerarmos a relação de subconjuntos como uma relação de ordem parcial, temos que o espaço de busca é estruturado como um reticulado. Este reticulado pode ser visualizado como um grafo, onde somente as relações diretas são representadas.

$\rightarrow$ Se $A \subseteq B \wedge |A| = |B| - 1$, então existe uma aresta entre A e B.

Assim, a mineração de conjunto de itens frequentes é resolvida por uma busca neste reticulado, seja em largura ou em profundidade. De fato, existem abordagens baseadas em ambas as buscas.

No entanto, a maioria das abordagens compartilham a mesma estrutura de busca:

1. Identificam candidatos navegando o espaço de busca
2. Computam o suporte desses candidatos, descartando os infrequentes

2.2.1 Algoritmo ingênuo

Um algorítmo ingênuo é definido: enumerar cada *itemset* possível, e verificar no conjunto de dados quais transações contêm esse *itemset*.

- A computação do suporte de um *itemset* requer uma passada sobre o conjunto de dados: $\mathcal{O}(|T|)$
- Verificar se uma dada transação contém um *itemset*: $\mathcal{O}(|I|)$
- Portanto, o custo total de computação do suporte é $\mathcal{O}(|I| \cdot |T|)$
- O espaço de busca, por sua vez, é o conjunto potência de I.

Logo, a complexidade do algoritmo ingênuo é $\mathcal{O}\big(2^{|I|} \cdot |I| \cdot |T|\big)$.

Vale notar que, devido aos seus tamanhos, a memória principal tipicamente não comporta o conjunto de dados. Tal característica agrava fortemente a ineficiência deste algorítimo, onde o componente $\mathcal{O}(|I| \cdot |T|)$ corresponde à passadas no conjunto de dados.

2.2.2 Apriori

O algoritmo Apriori é viabilizado pela propriedade de **anti-monotonicidade** da função suporte:

$$A \subseteq B \implies \sup(A) \geq \sup(B)$$

O Apriori utiliza busca em largura para minerar os padrões. A busca inicia com a identificação dos itens frequentes. Depois, os conjuntos de tamanho k são explorados antes dos imediatamente maiores. Assim como o ingênuo, ele também opera em duas etapas:

1. Geração de candidatos
2. Cálculo do suporte e eliminação dos infrequentes.

Candidatos que diferem em apenas um item são combinados para gerar os próximos candidatos, de tamanho $k + 1$. Imediatamente, os que possuirem algum subconjunto infrequente são descartados. Utilizando este método, os suportes dos candidatos são atualizados com uma única passada no conjunto de dados.

No total, o número de passadas é drasticamente reduzido: $\mathcal{O}(|I|)$

Apesar disso, o algoritmo ainda apresenta problemas:

- Nem sempre a memória primária comporta todos candidatos de um nível, demandados para busca em largura.
- As operações de poda e cálculo do suporte podem ser consideravelmente custosas, mas podem ser atenuadas com estruturas de dados apropriadas.
- A redução do suporte mínimo implica um grande impacto no custo computacional, pois quanto mais profundo o nível, seu tamanho cresce exponencialmente.
- A densidade da base de dados também decorre em um custo maior: transações com mais itens implicam *itemsets* maiores, mais subconjuntos são gerados para a contagem do suporte.

2.2.3 Equivalence Class Transformation

O Eclat tem como proposta eliminar a necessidade de passadas no conjunto de dados para computar o suporte. Para isso, utiliza-se uma representação vertical dos dados, e o fato de que a cobertura da união de dois *itemsets* é a interseção de suas coberturas.

De forma equivalente, a ideia central do algoritmo é manter os *tidsets* em memória principal para computar o suporte dos *itemsets* através de interseções desses conjuntos. Contudo, a memória principal pode não comportar todos os *tidsets*. Assim, é necessário algum mecanismo que possibilite a divisão do espaço de busca em subproblemas independentes.

Esta divisão pode ser feita conforme uma relação de equivalência estabelecida sobre os candidatos. Seja $p : \mathcal{P}(I) \times N \to \mathcal{P}(I)$ uma função prefixo. A seguinte relação é uma relação de equivalência:

$$\theta_k \subseteq \mathcal{P}(I) \times \mathcal{P}$$
$$A \, \theta_k \, B \equiv p(A, k) = p(B, k)$$

Dessa forma, induz-se uma partição dos conjuntos de itens em classes de equivalência, onde todos os elementos compartilham um certo prefixo.

Durante a busca em profundidade, o algoritmo particiona os conjuntos de itens conforme a relação de equivalência e o nível da árvore. O cálculo do suporte no algoritmo se restringe a calcular o tamanho do *tidset*.

Apesar disso, o algoritmo ainda apresenta problemas:

- O tempo de execução depende do cálculo da interseção dos *tidsets*.
- O custo computacional do algoritmo está diretamente relacionado ao tamanho dos *tidsets*.
- O custo de espaço também depende do tamanho. Quanto mais denso o conjunto de dados, mais largos serão os *tidsets*.

2.2.4 dEclat

De forma a atacar o problema de espaço do Eclat, podemos substituir os *tidsets* pela diferença entre os mesmos e os prefixos que os definem para os membros de cada classe. Tal conjunto é denomidado **diffset**. Para um prefixo P e um *itemset* PX, o *diffset* de X é

$$d(PX) = c(P) - c(X)$$

Somente os *diffsets* são armazenados, portanto o suporte não é mais obtido como a cardinalidade desse conjunto. Calculamos o suporte de um itemset PXY, obtido a partir de PX e PY, da seguinte forma:

$$\sup(PXY) = \sup(PX) - |d(PXY)|$$

Tal solução passa por computar o diffset de PXY:

$$d(PXY) = d(PY) - d(PX)$$

Em outras palavras, podemos usar os diffsets dos conjuntos base para calcular o diffset do novo candidato.

Essa abordagem se mostra muito eficiente para conjuntos densos. Porém, em conjuntos esparsos, o algoritmo original é a melhor opção.

2.2.5 FP-Growth

O algoritmo FP-Growth procura atacar dois problemas presentes nas abordagens anteriores:

- Repetidas passadas sobre a base de dados.
- Geração de candidatos.

Para isso:

1. Adota-se uma estratégia de busca em profundidade.
2. Adota-se projeções dos dados para mantê-los em memória principal.
3. Utiliza-se uma árvore especial de prefixos denominada FP-Tree.
4. Busca-se os padrões inteiramente através desta árvore, sem necessidade de retorno aos dados.

A FP-Tree constitui uma árvore de prefixos tradicional, duplamente conectada, adicionada de uma tabela auxiliar de localização. Cada nó contém um item e sua frequência **naquele prefixo**. Portanto, um mesmo item pode constar em vários nós, caso ocorra em prefixos distintos. Já a tabela possui as seguintes colunas:

1. **Item:** identificador do item, índice da tabela.
2. **Frequência:** frequência individual de cada item (denota a ordem dos itens).
3. **Nós:** coleção de referências para os nós da árvore referentes ao item.

Sua construção se dá em duas fases:

1. **Computa-se a frequêcia individual dos itens:** (Uma passada nos dados)
   Itens infrequentes são descartados, pois não podem formar padrões frequentes.
2. **Insere-se cada transação, através dos seus itens ordenados:** (Outra passada)
   Itens são ordenados em ordem decrescente de frequência, sendo os infrequentes filtrados.

Com a FP-Tree construída, inicia a mineração de padrões.

Para cada item em ordem **crescente**:

1. Constrói-se uma **nova árvore de prefixos** a partir dos prefixos deste item, desconsiderando o item em questão.
2. Desta, descarta-se os itens cuja frequência no prefixo é inferior ao suporte mínimo, e agrupa-se os nós de itens iguais até que a árvore constitua apenas um ramo.
3. Considera-se este ramo um conjunto. Nesta iteração, o conjunto dos padrões extraídos é o conjunto potência do ramo, acrescendo em cada padrão o item em questão.

Exemplo: (suporte mínimo = 2)

Tabela 1: Base de dados

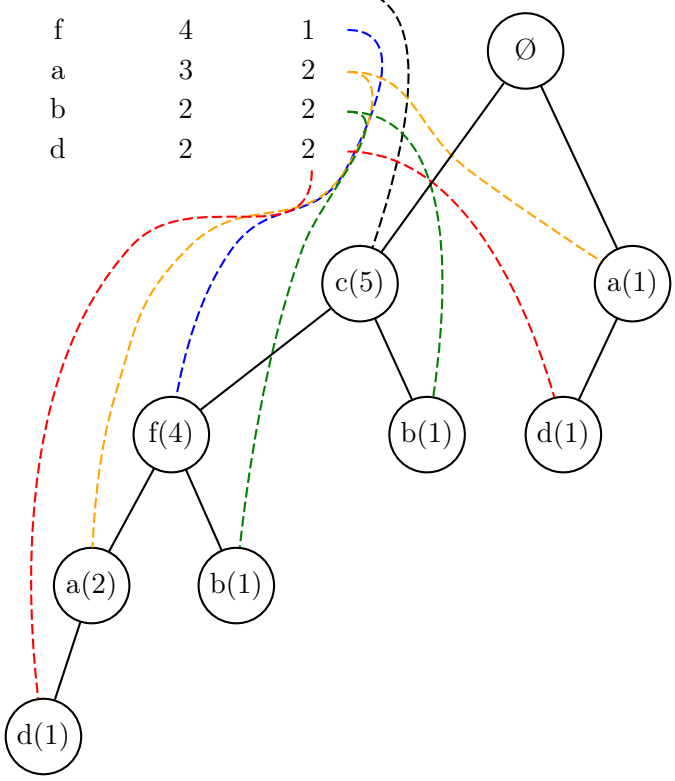
tid	a	b	c	d	e	f
1	✓		✓	✓		✓
2		✓	✓			
3			✓		✓	✓
4	✓			✓		
5		✓	✓			✓
6	✓		✓			✓

FP-Tree fase 1:

Item	Frequência	Nós
c	5	Ø
f	4	Ø
a	3	Ø
b	2	Ø
d	2	Ø

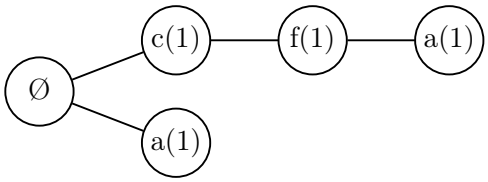
FP-Tree fase 2:

Item	Frequência	Nós
c	5	1
f	4	1
a	3	2
b	2	2
d	2	2

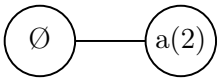


Mineração de padrões a partir do item **d**:

1. Projeção dos prefixos:



2. Filtragem e redução:



3. Padrões resultantes:

$$\{d, ad\}$$