CURSO DE DATA MINING

Sandra A. de Amo
deamo@ufu.br

Universidade Federal de Uberlândia
Faculdade de Computação
Agosto 2003

Conteúdo

Capítulo 1

Introdução - Slides

Capítulo 2

Mineração de Regras de Associação

Suponha que você seja gerente de um supermercado e esteja interessado em conhecer os hábitos de compra de seus clientes, por exemplo, "quais os produtos que os clientes costumam comprar ao mesmo tempo, a cada vez que vêm ao supermercado". Conhecer a resposta a esta questão pode ser útil : você poderá planejar melhor os catálogos do supermercado, os folhetos de promoções de produtos, as campanhas de publicidade, além de organizar melhor a localização dos produtos nas prateleiras do supermercado colocando próximos os items frequentemente comprados juntos a fim de encorajar os clientes a comprar tais produtos conjuntamente. Para isto, você dispõe de uma *mina* de dados, que é o *banco de dados de transações* efetuadas pelos clientes. A cada compra de um cliente, são registrados neste banco todos os itens comprados, como mostra a tabela da figura 2. Para facilitar a representação dos artigos na tabela, vamos utilizar associar números a cada artigo do supermercado, como ilustrado na figura 1 :

Artigo (item)	número que o representa
Pão	1
Leite	2
Açúcar	3
Papel Higiênico	4
Manteiga	5
Fralda	6
Cerveja	7
Refrigerante	8
Iogurte	9
Suco	10

Figura 2.1: Representação numérica de cada artigo do supermercado

TID	Itens comprados
101	{1,3,5}
102	{2,1,3,7,5}
103	{4,9,2,1}
104	{5,2,1,3,9}
105	{1,8,6,4,3,5}
106	{9,2,8}

Figura 2.2: Um banco de dados de transações de clientes

Cada conjunto de itens comprados pelo cliente numa única transação é chamado de *Itemset*. Um itemset com k elementos é chamado de k-itemset. Suponha que você, como gerente, decide que um itemset que aparece em *pelo menos* 50% de todas as compras registradas será considerado *frequente*. Por exemplo, se o banco de dados de que você dispõe é o ilustrado na figura 2, então o itemset {1,3} é considerado frequente, pois aparece em mais de 60% das transações. Porém, se você for muito exigente e decidir que o mínimo para ser considerado frequente é aparecer em *pelo menos* 70% das transações, então o itemset {1,3} não será considerado frequente. Definimos *suporte* de um itemset como sendo a porcentagem de transações onde este itemset aparece, isto é, onde este itemset está contido. Por exemplo, a tabela da figura 3 contabiliza os suportes de diversos itemsets com relação ao banco de dados de transações da figura 2.

Itemset	Suporte
{1,3}	0,6666
{2,3}	0,3333
{1,2,7}	0,16666
{2,9}	0,5

Figura 2.3: Suporte de alguns itemsets

Repare que o que identifica uma transação é o *identificador da transação* TID e não o *identificador do cliente*. Assim, um mesmo cliente será contado várias vezes a cada vez que fizer uma compra no supermercado. O que interessa é a transação (a compra), e não o cliente.

Caso a sua exigência mínima para um itemset ser considerado frequente seja 50%, então os seguintes itemsets da tabela da figura 3 serão considerados frequentes : {1,3}, {2,9}. Entretanto, se o teu limite de suporte mínimo for 60% então somente o itemset {1,3} será considerado frequente.

2.1 Formalização do Problema

Nesta seção vamos formalizar o problema da mineração de regras de associação, que foi introduzido em [1]. Para isso, precisamos definir precisamente o que é uma regra de associação e os diversos conceitos envolvidos. Na seção precedente, já definimos de maneira informal as noções de *itemset* e de *suporte de um itemset* com respeito a um banco de dados de transações. Agora, vamos definir todos estes conceitos de forma rigorosa.

Seja $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ um conjunto de itens (o conjunto de todos os artigos do supermercado). Seja $\mathcal{D}$ um banco de dados de transações, isto é, uma tabela de duas colunas, a primeira correspondente ao atributo TID (identificador da transação) e o segundo correspondente à transação propriamente dita, isto é, a um conjunto de itens (itemset), os itens comprados durante a transação. A figura 2 é um exemplo de banco de dados de transações. Os elementos de $\mathcal{D}$ são chamados de *transações*. Um *itemset* é um subconjunto não vazio de $\mathcal{I}$. Dizemos que uma transação T *suporta* um itemset I se I $\subseteq$ T. Por exemplo, a primeira transação do banco de dados da figura 2 suporta os itemsets {1}, {3}, {5}, {1,3}, {1,5}, {3,5}, {1,3,5}.

Repare que, embora uma transação e um itemset sejam a mesma coisa (conjunto de items), chamamos de *transação* somente aqueles itemsets que estão registrados no banco de dados como sendo a compra *total* feita por algum cliente.

Definição 2.1 Uma *regra de associação* é uma expressão da forma $A \rightarrow B$, onde A e B são itemsets (não necessariamente transações !).

Por exemplo, {pão, leite} $\rightarrow$ {café} é uma regra de associação. A idéia por trás desta regra é : pessoas que compram pão e leite *têm a tendência de* também comprar café, isto é, **se** alguém compra pão e leite **então** também compra café. Repare que esta regra é diferente da regra {café} $\rightarrow$ {pão,leite}, pois o fato de ter muita gente que quando vai ao supermercado para comprar pão e leite também acaba comprando café, não significa que quem vai ao supermercado para comprar café também acaba comprando pão e leite.

A toda regra de associação $A \rightarrow B$ associamos um *grau de confiança*, denotado por $conf(A \rightarrow B)$. Este grau de confiança é simplesmente a porcentagem das transações que suportam B dentre todas as transações que suportam A, isto é :

$$conf(A \rightarrow B) = \frac{\text{número de transações que suportam } (A \cup B)}{\text{número de transações que suportam } A}$$

Por exemplo, o grau de confiança da regra {cerveja} $\rightarrow$ {manteiga}, isto é, {7} $\rightarrow$

{5}, com relação ao banco de dados da figura 2 é 1 (100%).

Será que o fato de uma certa regra de associação ter um grau de confiança relativamente alto é suficiente para a considerarmos uma "boa" regra ? Repare que no nosso banco de dados da figura 2, os itens *cerveja, manteiga* aparecem juntos somente em uma transação entre 6, isto é, poucos clientes compram estes dois itens juntos. Entretanto, somente pelo fato de que em 100% das transações onde cerveja aparece também o item manteiga foi comprado, temos que o grau de confiança da regra {cerveja} → {manteiga} é de 100%. Nossa intuição diz que isto não é suficiente para que esta regra seja considerada "boa", já que esta confiança diz respeito somente às poucas transações que suportam {cerveja, manteiga}. A fim de garantirmos que uma regra $A \rightarrow B$ seja *boa* ou *interessante*, precisamos exigir que seu *suporte* também seja relativamente alto, além de seu grau de confiança.

A toda regra de associação $A \rightarrow B$ associamos um *suporte*, denotado por $sup(A \rightarrow B)$ definido como sendo o suporte do itemset $A \cup B$. Por exemplo, o suporte da regra {cerveja} → {manteiga} com relação ao banco de dados da figura 2 é 0.1666%

Uma regra de associação r é dita *interessante* se $conf(r) \geq \alpha$ e $sup(r) \geq \beta$, onde α e β são respectivamente um grau mínimo de confiança e um grau mínimo de suporte especificados pelo usuário. No nosso exemplo, caso $\alpha = 0.8$ e $\beta = 0.1$ então nossa regra {cerveja} → {manteiga} é interessante. Entretanto, caso sejamos mais exigentes e estabeleçamos $\beta = 0.5$, então esta regra deixa de ser interessante, mesmo que seu grau de confiança ultrapasse de longe o grau de confiança mínimo α estabelecido pelo usuário.

O problema da mineração de regras de associação

Este problema é o seguinte :

INPUT : São dados um banco de dados de transações $\mathcal{D}$, um nível mínimo de confiança α e um um nível mínimo de suporte β.

OUTPUT : Pede-se todas as regras de associação interessantes com relação a $\mathcal{D}$, α e β.

2.2 O algoritmo Apriori

Este algoritmo foi proposto em 1994 pela equipe de pesquisa do Projeto QUEST da IBM que originou o software *Intelligent Miner*. Trata-se de um algoritmo que resolve o *prob-*

lema da mineração de itemsets frequentes, isto é :

INPUT : São dados um banco de dados de transações $\mathcal{D}$ e um um nível mínimo de suporte β.

OUTPUT : Pede-se todos os itemsets frequentes com relação a $\mathcal{D}$ e β.

Detalhes deste algoritmo e análise dos resultados experimentais podem ser encontrados no artigo [1].

Suponha que tenhamos obtido todos os itemsets frequentes com relação a $\mathcal{D}$ e β. A fim de obter as regras de associação interessantes, basta considerarmos, para cada itemset frequente L, todas as regras *candidatas* $A \rightarrow (L - A)$, onde $A \subset L$ e testarmos para cada uma destas regras candidatas se o seu grau de confiança excede o nível mínimo de confiança α. Para calcular a confiança de $A \rightarrow (L - A)$ não é preciso varrer novamente o banco de dados $\mathcal{D}$. De fato, durante a execução do algoritmo Apriori já calculamos o suporte de L e A. Note que :

$$conf(A \rightarrow (L-A)) = \frac{\text{total de trans. suportando L}}{\text{total de trans. suportando A}} = \frac{\frac{\text{total de trans. suportando L}}{\text{total de trans}}}{\frac{\text{total de trans. suportando A}}{\text{total de trans}}} = \frac{sup(L)}{sup(A)}$$

Assim, para calcular a confiança de $A \rightarrow (L - A)$ basta dividir o suporte de L pelo suporte de A. Estes suportes já foram calculados antes, durante a execução do algoritmo Apriori.

2.2.1 As fases de Apriori : geração, poda, validação

O algoritmo Apriori possui três fases principais : **(1)** a fase da geração dos candidatos, **(2)** a fase da poda dos candidatos e **(3)** a fase do cálculo do suporte. As **duas primeiras fases** são realizadas na memória principal e não necessitam que o banco de dados $\mathcal{D}$ seja varrido. A memória secundária só é utilizada caso o conjunto de itemsets candidatos seja muito grande e não caiba na memória principal. Mas, mesmo neste caso é bom salientar que o banco de dados $\mathcal{D}$, que normalmente nas aplicações é extremamente grande, não é utilizado. Somente na terceira fase, a fase do cálculo do suporte dos itemsets candidatos, é que o banco de dados $\mathcal{D}$ é utilizado.

Tanto na fase de geração de candidatos (Fase 1) quanto na fase da poda dos candidatos (Fase 2) a seguinte propriedade de **Antimonotonia da relação de inclusão entre os**

itemsets é utilizada :

Propriedade Apriori - ou Antimonotonia da relação $\subseteq$: Sejam I e J dois itemsets tais que $I \subseteq J$. Se J é frequente então I também é frequente. Assim, para que um certo itemset seja frequente é necessário que todos os itemsets contidos nele sejam também frequentes. Caso um único itemset contido nele não seja frequente, seu suporte nem precisa ser calculado (o banco de dados portanto não precisa ser varrido), pois sabemos de antemão que ele nunca poderá ser frequente. **O leitor deve provar a validade desta propriedade !!**

O algoritmo Apriori é executado de forma iterativa : os itemsets frequentes de tamanho k são calculados a partir dos itemsets frequentes de tamanho $k-1$ que já foram calculados no passo anterior (a partir dos itemsets frequentes de tamanho $k-2$, etc).

No que se segue, suponhamos que estejamos no passo k e que portanto já tenhamos obtido no passo anterior o conjunto L_{k-1} dos **itemsets frequentes** de tamanho $k-1$.

A fase da geração dos candidatos de tamanho k

Nesta fase, vamos gerar os itemsets **candidatos** (não necessariamente frequentes !) de tamanho k a partir do conjunto L_{k-1}. Como estamos interessados em gerar somente itemsets que tenham alguma chance de serem frequentes, devido à propriedade Apriori sabemos que todos os itemsetsde tamanho $k-1$ contidos nos nossos candidatos de tamanho k deverão ser frequentes, portanto, deverão pertencer ao conjunto L_{k-1}. Assim, o conjunto C'_k de itemsets candidatos de tamanho k é construído *juntando-se* pares de itemsets de tamanho $k-1$ que tenham $k-2$ elementos em comum. Desta maneira temos certeza de obter um itemset de tamanho k onde *pelo menos dois* de seus subconjuntos de tamanho $k-1$ são frequentes. A figura 4 ilustra esta construção :

A função Apriori-Gen descrita abaixo é responsável pela construção do conjunto dos *pré-candidatos* C'_k :

insert into C'_k
select $p.\text{item}_1$, $p.\text{item}_2$, ..., $p.\text{item}_{k-2}$,$p.\text{item}_{k-1}$, $q.\text{item}_{k-1}$
from L_{k-1} p, L_{k-1} q
where$p.\text{item}_1 = q.\text{item}_1$, $p.\text{item}_2 = q.\text{item}_2$, ...,$p.\text{item}_{k-2} = q.\text{item}_{k-2}$, $p.\text{item}_{k-1} < q.\text{item}_{k-1}$;

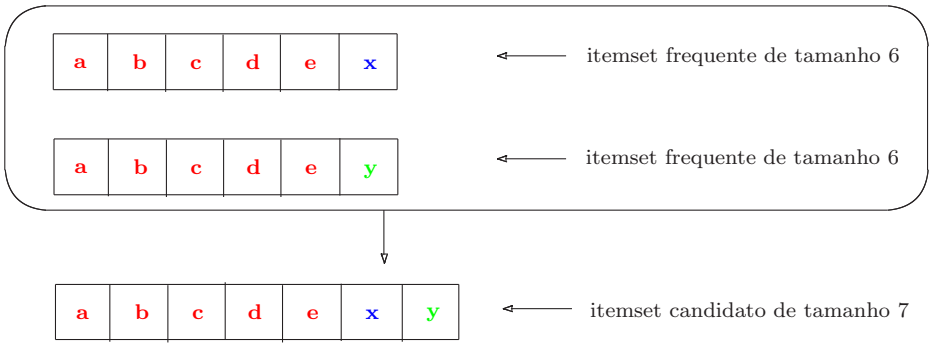


Figura 2.4: Construção de um k-itemset candidato a partir de dois frequentes de tamanho $k-1$

Exemplo 2.1 Consideremos o banco de dados de transações dado na figura 2 e suponhamos que no passo 2 da iteração tenhamos obtido o seguinte conjunto de itemsets frequentes de tamanho 2 :

$$L_2 = \{\{1,3\},\{1,5\},\{1,4\},\{2,3\},\{3,4\},\{2,4\}\}$$

Então o conjunto dos pré-candidatos C'_3 da iteração seguinte será :

$$C'_3 = \{\{1,3,5\},\{1,3,4\},\{1,4,5\},\{2,3,4\}\}$$

Fase da Poda dos Candidatos

Utilizando novamente a propriedade Apriori, sabemos que se um itemset de C'_k possuir um subconjunto de itens (um subitemset) de tamanho $k-1$ que não estiver em L_{k-1} ele poderá ser descartado, pois não terá a menor chance de ser frequente. Assim, nesta fase é calculado o conjunto C_k :

$$C_k = C'_k - \{I \mid \text{ existe } J \subseteq I \text{ tal que } \mid J \mid = k-1 \text{ e } J \notin L_{k-1}\}$$

A notação $\mid J \mid$ significa "o número de elementos do itemset J".

Exemplo 2.2 Consideremos a situação apresentada no exemplo 2.1. Neste caso :

$$C_3 = C'_3 - \{\{1,4,5\},\{1,3,5\}\} = \{\{1,3,4\},\{2,3,4\}\}$$

O itemset {1,4,5} foi podado pois não tem chance nenhuma de ser frequente : ele contém o 2-itemset {4,5} que **não é frequente**, pois não aparece em L_2. Por que o itemset {1,3,5} foi podado ?

Fase do Cálculo do Suporte

Finalmente, nesta fase é calculado o suporte de cada um dos itemsets do conjunto C_k. Isto pode ser feito varrendo-se **uma única vez o banco de dados** $\mathcal{D}$: Para cada transação t de $\mathcal{D}$ verifica-se quais são os candidatos suportados por t e para estes candidatos incrementa-se de uma unidade o contador do suporte.

Como são calculados os itemsets frequentes de tamanho 1

Os itemsets de tamanho 1 são computados considerando-se todos os conjuntos unitários possíveis, de um único item. Em seguida, varre-se uma vez o banco de dados para calcular o suporte de cada um destes conjuntos unitários, eliminando-se aqueles que não possuem suporte superior ou igual ao mínimo exigido pelo usuário.

2.3 Algumas técnicas de otimização de Apriori

Diversas técnicas de otimização foram propostas e colocadas em prática para melhorar a performance do algoritmo Apriori. Vamos descrever aqui algumas delas. Primeiramente, vamos discutir alguns detalhes de implementação referentes ao gerenciamento do buffer. Depois, passaremos à descrição de algumas técnicas importantes para melhorar a performance de Apriori.

GERENCIAMENTO DO BUFFER : problemas de memória

Caso haja espaço na memória principal para estocar os conjuntos L_{k-1} e C'_k, as fases 1 e 2 do algoritmo Apriori (geração dos pré-candidatos e poda) podem ser realizadas na memória principal. Do contrário, será preciso estocá-los em disco. Além disto, para a fase 3 (contagem do suporte), além de termos que estocar os candidatos C_k, temos que garantir uma página no buffer para as transações que serão testadas para avaliar os contadores de suporte de cada elemento de C_k. Vejamos como agir em caso de memória principal insuficiente :

- Suponhamos que L_{k-1} caiba na memória principal mas o conjunto dos pré-candidatos C'_k não caiba. Neste caso, a etapa k do algoritmo Apriori deve ser modificada : na fase de geração dos pré-candidatos C'_k, gera-se o número máximo de candidatos que é possível colocar na memória principal. Depois disto, poda-se estes candidatos. Se sobrar espaço, gera-se mais pré-candidatos, poda-se, etc, até que não haja mais espaço na memória principal. Depois disto, inicia-se a fase de contagem do suporte

(fase 3), varrendo-se uma vez a base de dados para contar o suporte destes candidatos. Aqueles candidatos que são frequentes, são armazenados em disco. Os que não são frequentes são suprimidos do buffer. Este processo é repetido até que todos os pré-candidatos C'_k sejam gerados e avaliados se são ou não frequentes.

Assim, neste caso (quando não há espaço para todos os candidatos na memória principal), somos obrigados a varrer a base de dados um número de vezes igual ao número de partições em que deve ser dividido o conjunto dos candidatos (já podados) C_k para que cada partição caiba na memória principal.

- Suponhamos agora que L_{k-1} não caiba na memória principal. Neste caso, como vamos ver a seguir, não poderemos executar a fase de podagem dos candidatos antes de varrer a base para o cálculo do suporte. Teremos que passar da fase 1 para a fase 3 diretamente, infelizmente. Como faremos isto ?

  Lembre-se que os itens são enumerados, isto é, associamos a cada item um número inteiro positivo. Ordena-se e estoca-se L_{k-1} em disco (a ordem é a lexográfica). Carrega-se na memória principal um bloco de itemsets de L_{k-1} nos quais todos os $k-2$ primeiros elementos são idênticos. Faz-se a junção dos elementos deste bloco e varre-se a base de dados para o cálculo do suporte destes elementos. Os que são frequentes são armazenados em disco, os que não são, são suprimidos. Repete-se o processo, até que todos os candidatos de tamanho k tenham sido gerados e seus respectivos suportes tenham sido calculados. Como um exemplo, suponha que $k = 4$ e L_3 é dado pela tabela abaixo :

1	2	3
1	2	4
1	2	5
1	2	6
1	2	7
1	3	4
1	3	5
1	3	6

  Suponha que na memória principal caibam no máximo 3 itemsets de tamanho 3 (além do espaço que deverá ser reservado para uma página do buffer onde serão carregadas as transações do banco de dados na fase da contagem do suporte). O primeiro bloco que será trazido para a memória será : {1,2,3}, {1,2,4} e {1,2,5}. A partir deste bloco serão obtidos os itemsets de tamanho 4 : {1,2,3,4}, {1,2,3,5} e {1,2,4,5}. O suporte dos três itemsets é avaliado. O processo se repete, e desta vez trazemos para a memória principal os itemsets {1,2,3}, {1,2,6} e {1,2,7}. A

partir deste bloco serão obtidos os itemsets de tamanho 4 : {1,2,3,6}, {1,2,3,7} e {1,2,6,7}. No próximo passo, traz-se para a memória principal {1,2,4}, {1,2,5}, {1,2,6}. E assim por diante.

Agora, vamos passar à descrição das técnicas de otimização para melhorar a performance de Apriori. As técnicas que vamos descrever atuam da seguinte maneira :

- **Categoria 1** : Diminuem o número de candidatos que são testados para cada transação (para ver quais são suportados por cada transação) na fase do cálculo do suporte. Também atuam na fase de poda, diminuindo o número de subconjuntos a serem testados para cada pré-candidato (lembre-se que um pré-candidato de tamanho k só é considerado bom se todos os seus subconjuntos de tamanho $k-1$ aparecem no L_{k-1}).

- **Categoria 2** : Diminuem o tamanho do banco de dados a cada iteração.

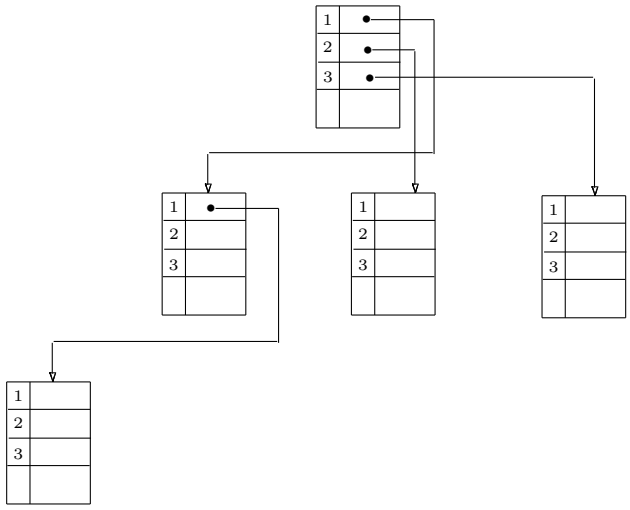
- **Categoria 3** : Diminuem o número de varridas do banco de dados. No algoritmo Apriori clássico, o banco de dados é varrido a cada iteração.

- **Categoria 4** : Diminuem o número de candidatos gerados.

2.3.1 Utilizando uma árvore-hash para armazenar C_k e L_{k-1}

O que é uma árvore-hash ?

Uma *árvore-hash* é uma árvore onde as folhas armazenam conjuntos de itemsets, e os nós intermediários (inclusive a raiz) armazenam tabelas-hash contendo pares do tipo (número, ponteiro).



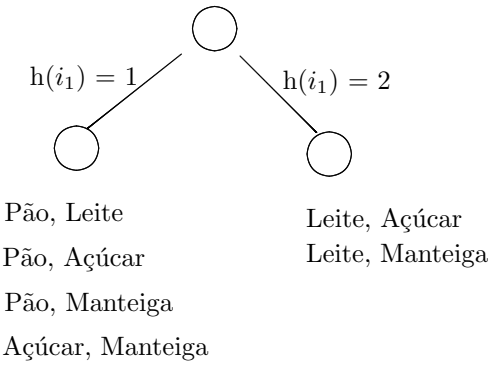
Uma árvore-hash é utilizada para estocar itemsets. Por exemplo, suponhamos que queiramos estocar o seguinte conjunto de 2-itemsets numa árvore-hash :

Pão	Leite
Pão	Açúcar
Pão	Manteiga
Leite	Açúcar
Leite	Manteiga
Açúcar	Manteiga

Vamos utilizar a enumeração dos itens descrita na figura 1 e vamos ordenar os itemsets segundo a ordem lexográfica associada a esta enumeração. Suponhamos que número máximo de itemsets numa folha é M = 3 e que cada nó tem no máximo N = 2 descendentes. Para a estocagem deste conjunto de itemsets numa árvore-hash respeitando estes parâmetros, utilizamos uma função hash h que servirá para distribuir os itemsets nas folhas da árvore. Esta função é definida como se segue : h(Pão) = 1, h(Leite) = 2, h(Açúcar) = 1, h(Manteiga) = 2. No início, todos os cinco itemsets estão na raiz.

Pão, Leite
Pão, Açúcar
Pão, Manteiga
Leite, Açúcar
Leite, Manteiga
Açúcar, Manteiga

Como o número máximo de itemsets numa folha é 3, é preciso transformar a raiz num nó intermediário e criar nós descendentes da raiz. A figura abaixo ilustra a criação dos dois filhos da raiz :



A primeira folha excede o número máximo de elementos permitido. Logo, deve se transformar num nó intermediário e nós-folha devem ser criados a partir deste primeiro nó. A figura 1 ilustra este processo.

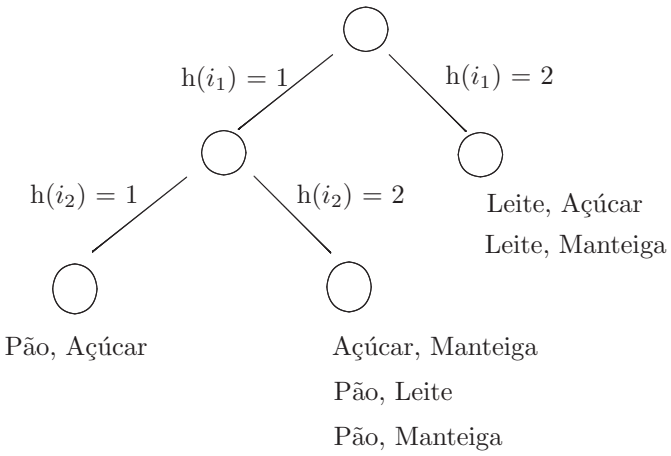


Figura 2.5: Itemsets armazenados numa árvore-hash

Repare que agora, o número de elementos das folhas não excede 3 e portanto o processo termina. Esta última figura ilustra como um conjunto de itemsets é armazenado numa árvore-hash satisfazendo os parâmetros dados.

Como utilizar uma árvore-hash na fase de poda dos candidatos

Suponhamos que tenhamos gerado os pré-candidatos C'_3 e vamos utilizar L_2 (itemsets frequentes de tamanho 2 calculados na fase anterior) para podar os pré-candidatos e obter C_3. Para isso, armazenamos o conjunto de itemsets L_2 numa árvore-hash, como descrevemos no parágrafo precedente. Para cada elemento de C'_3, temos que verificar se este contém um 2-itemset que não está em L_2. Por exemplo, suponha que I = {Pão, Açúcar, Manteiga} $\in C'_3$. Construimos todos os subconjuntos de 2 elementos de I : {Pão, Açúcar}, {Pão, Manteiga}, {Açúcar, Manteiga}. Suponha que nosso L_2 esteja armazenado na árvore-hash da figura 1. Para cada um dos 3 subconjuntos de I de tamanho 2, vamos verificar se está presente numa das folhas da árvore. O itemset {Pão, Açúcar} é procurado somente na primeira folha, já que h(Pão) = 1 e h(Açúcar) = 1. Assim, evita-se de percorrer elementos de L_2 que decididamente nunca poderiam conter {Pão, Açúcar}. Os itemsets {Pão, Manteiga} e {Açúcar, Manteiga} são procurados somente na segunda folha. A terceira folha não é varrida, economizando-se tempo com isso.

Como utilizar uma árvore-hash na fase de avaliação do suporte

Nesta fase, armazena-se os candidatos já podados C_k numa árvore-hash. Para cada transação t do banco de dados, vamos aumentar o contador de suporte de cada elemento de C_k que esteja contido em t (isto é, que seja suportado por t). Por exemplo, suponha que C_2 esteja armazenado na árvore-hash da figura 1 e que t = {Pão, Leite, Manteiga} seja uma transação do banco de dados. Quais candidatos de C_2 são suportados por t ? Para isso, fazemos uma busca recursiva dos subconjuntos de 2 elementos de t : {Pão, Leite}, {Pão, Manteiga}, {Leite, Manteiga}. O primeiro, {Pão, Leite} só pode se encontrar na folha 2 (já que h(Pão) = 1 e h(Leite) = 2). Logo, esta folha é varrida. Se o itemset é encontrado, aumentamos seu contador de 1. Analogamente, o segundo 2-itemset, {Pão, Manteiga} só pode se encontrar na folha 2 e o terceiro, {Leite, Manteiga} só pode se encontrar na folha 3. Veja que a folha 1 não é varrida, economizando-se tempo com isso. A figura 2 ilustra os itemsets que são suportados pela transação t e que, portanto, têm seus contadores aumentados de 1.

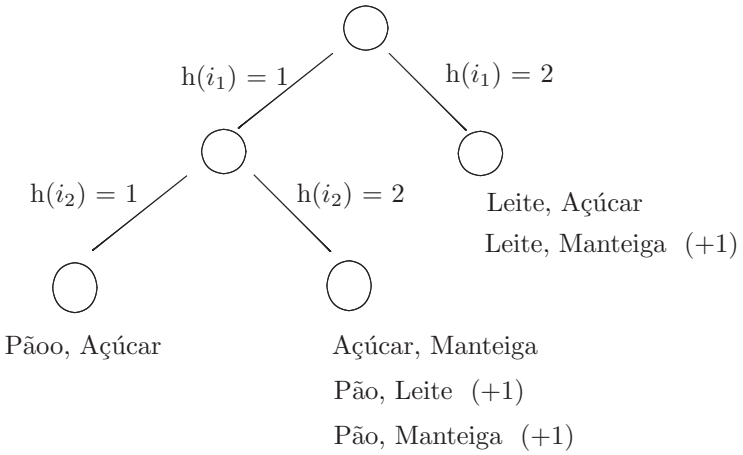


Figura 2.6: Utilização de uma árvore-hash na fase de cálculo do suporte

Recomenda-se ao leitor consultar um **livro de Estruturas de Dados** e estudar o problema de **colisões** , quando uma folha excede o número de elementos permitido mas não é possível subdividí-la pois no passo anterior já foi utilizado o elemento i_k de maior índice. Este detalhe precisa ser muito bem entendido para a implementação do projeto final do curso.

2.3.2 Diminuindo o tamanho do banco de dados a cada iteração.

Uma transação que não contém nenhum itemset frequente de tamanho k (isto é, que não participou no contador de nenhum destes k-itemsets frequentes), não conterá com certeza nenhum itemset frequente de tamanho $k+1$. Isto porque todo subconjunto de items de um $(k+1)$-itemset frequente deve ser frequente (Propriedade Apriori). Além disto, se uma transação contém um $(k+1)$-itemset deverá obviamente conter todos os subconjuntos deste itemset. Logo, se a transação contiver um $(k+1)$-itemset frequente I , deverá conter todo k-itemset frequente contido em I.

Desta maneira é possível diminuir o banco de dados de transações a cada iteração.

2.3.3 Diminuindo o número de varridas do banco de dados

Vamos descrever dois tipos de otimização : uma que reduz o número de varridas a 2 e outra que reduz a uma única varrida.

Otimização particionando o banco de dados de transações : banco de dados é varrido somente duas vezes

Esta técnica foi introduzida em [8].

Repare que se o número m de itens for pequeno, podemos calcular os itemsets frequentes de maneira direta, fazendo uma única passada no banco de dados, simplesmente testando cada um dos 2^m itemsets possíveis. Porém, o número de itens geralmente é grande : $m > 1000$, imagine 2^m !! Assim, este procedimento não é factível. Daí a necessidade de um algoritmo tal como Apriori que reduz os itemsets a serem testados, mas que em contrapartida exige diversas varridas do banco de dados, uma a cada iteração. Por outro lado, como normalmente o banco de dados é enorme, fazer várias varridas torna o processo de mineração muito demorado. A técnica que vamos descrever permite determinar todos os itemsets frequentes com apenas **duas** varridas no banco de dados. O esquema geral da técnica é o seguinte :

- Divide-se o banco de dados D em várias partes $D_1, D_2, ..., D_n$ que não se sobrepõem umas às outras, isto é, $D_i \cap D_j = \emptyset$ para todos i, j onde $i \neq j$. Isto é feito de tal modo que cada uma das partes D_i caiba na memória principal.

- Aplica-se o algoritmo Apriori para cada uma das partes D_i, todas as 3 fases do algoritmo (inclusive a do cálculo do suporte) sendo realizadas em memória principal. Desta maneira, encontra-se todos os *itemsets frequentes locais*, numa única varrida do banco de dados. Estes itemsets frequentes locais não necessariamente são itemsets frequentes globais, isto é, com relação a todo o banco de dados. Mas, a seguinte propriedade é verificada :

  Todo itemset frequente global é frequente em ao menos uma das partes D_i.

  De fato : Seja α o limite mínimo do suporte. Suponhamos que I é um itemset que é globalmente frequente. Seja x o número de transações do banco de dados D onde I aparece e N o número total de transações de D. Então, $\frac{x}{N} \geq \alpha$.

  Suponha agora, por absurdo que I não seja frequente em nenhuma das partes D_i. Seja x_i e M_i respectivamente o número de transações de D_i onde I aparece e o número total de transações de D_i. Então, $\frac{x_i}{M_i} < \alpha$ e portanto $x_i < \alpha * M_i$, para todo $i = 1, ..., n$ (n = número de partes de D). Assim :

  $$(x_1 + x_2 + ... + x_n) < \alpha * (M_1 + ... + M_n)$$

  Como $M_1 + M_2 + ... + M_n = N$ (pois $D_i \cap D_j = \emptyset$ para todos os i, j distintos) e $x_1 + x_2 + ... + x_n = x$, temos :

$$x < \alpha * N$$

o que é um absurdo, pois estamos supondo que I é frequente com relação a D.

- Faz-se uma **segunda** varrida no banco de dados onde são testados somente os itemsets frequentes locais para ver quais são frequentes em termos globais.

O algoritmo APRIORITID : banco de dados é varrido uma única vez

A idéia do algoritmo é a seguinte : a cada iteração k, calculamos os candidatos C_k a partir dos itemsets frequentes L_{k-1} utilizando a função Apriori-Gen conforme foi descrito na seção **??**. Além disto, calcula-se uma relação $\overline{C_k}$ que é constituida por duas colunas : a primeira contém o identificador de uma transação IdTrans e a segunda contém o conjunto de todos os candidatos de C_k que são suportados pela transação IdTrans. Para calcular o suporte dos candidatos C_k será utilizada a tabela $\overline{C_{k-1}}$ da fase anterior. A tabela $\overline{C_k}$ calculada na iteração k será utilizada para calcular o suporte dos candidatos C_{k+1} da iteração seguinte. Assim, vemos que a cada iteração, o cálculo do suporte é feito utilizando-se uma tabela $\overline{C_k}$ que é, em princípio, bem menor que o banco de dados. Este só é varrido na primeira iteração, no momento de calcular o suporte dos candidatos de tamanho 1.

Vamos ilustrar esta técnica com um exemplo.

Exemplo 2.3 Considere o seguinte banco de dados de transações :

TID	Itemset
100	{1,3, 4}
200	{2, 3, 5}
300	{1, 2, 3, 5}
400	{2, 5}

e consideremos um nível de suporte mínimo igual a 0,5.

Iteração 1 : $C_1 = \{ \{1\}, \{2\}, \{3\}, \{4\}, \{5\} \}$, $L_1 = \{ \{1\}, \{2\}, \{3\}, \{5\} \}$. Estes conjuntos são calculados exatamente como é feito no Algoritmo Apriori na primeira iteração. Nesta fase precisamos varrer o banco de dados para o cálculo de L_1. Esta vai ser a **única vez** em que o banco de dados será **totalmente** varrido. Vamos transformar o banco de dados na seguinte tabela :

$\overline{C_1}$

TID	Conjunto de Itemsets
100	{ {1} , {3} , {4} }
200	{ {2} , {3} , {5} }
300	{ {1} , {2} , {3} , {5} }
400	{ {1} , {2} , {3}, {5} }

Esta tabela será utilizada na iteração 2 para o cálculo do suporte. É claro que na iteração 2, a tabela $\overline{C_1}$ tem a mesma ordem de grandeza do banco de dados e portanto não é aí que se vê alguma melhora de performance. Esta aparece nas iterações posteriores, onde o conjunto $\overline{C_k}$ (que substitui o banco de dados) vai ser bem menor do que o banco de dados.

Iteração 2 : Calcula-se primeiramente o conjunto dos candidatos C_2 como é feito no algoritmo Apriori. Este conjunto é dado por :

C_2

Itemset
{1,2}
{1,3}
{1,5}
{2,3}
{2,5}
{3,5}

Para o cálculo do suporte de cada candidato em C_2, vamos varrer a tabela $\overline{C_1}$. Considere o primeiro registro desta tabela : (100, { {1} , {3} , {4} }). Quais são os elementos de C_2 que são suportados por este registro ? Considere os candidatos um a um. Para cada candidato : (1) retiramos o último elemento. O que sobra está contido no registro ? Sim (pois é {1}). (2) retiramos o antipenúltimo elemento. O que sobra está contido no registro ? Sim (pois é {2}). Se as duas respostas forem "sim", então o candidato é suportado pelo registro e aumentamos seu contador de 1. Fazemos isto para cada um dos candidatos de C_2. No final obtemos a seguinte tabela para L_2 (os itemsets frequentes de tamanho 2) :

L_2

Itemset	Suporte
{1,3}	0,5
{2,3}	0,5
{2,5}	0,75
{3,5}	0,5

A idéia é que para $\{i_1, i_2, ..., i_k\}$ ser suportado pela transação (idtrans, $\{I_1, ..., I_m\}$) é necessário que todos os items $i_1, ..., i_k$ apareçam em ao menos um dos I_i. Ora, se $\{i_1, ..., i_{k-1}\}$ e $\{i_1, ..., i_{k-2}, i_k\}$ são itemsets que pertencem a $\{I_1, ..., I_m\}$) então todos os itens $i_1, i_2, ..., i_k$ aparecem em algum itemset I_i e portanto o k-itemset $\{i_1, i_2, ..., i_k\}$ é suportado pela transação idtrans.

Para finalizar a iteração 2, precisamos calcular $\overline{C_2}$ que será utilizado em substituição ao banco de dados na próxima iteração. Isto é feito, na verdade, simultaneamente com o cálculo da tabela L_2. Para cada registro (idtrans, X) de $\overline{C_1}$ selecionamos aqueles candidatos de C_2 que são suportados pela transação idtrans. Seja Y o conjunto destes 2-itemsets. Então (idtrans, Y) será um registro de $\overline{C_2}$. A tabela assim resultante é ilustrada abaixo :

$\overline{C_2}$

TID	Conjunto de Itemsets
100	{ {1,3} }
200	{ {2,3} , {2,5} , {3,5} }
300	{ {1,2} , {1,3} , {1,5} , {2,3}, {2,5}, {3,5} }
400	{ {2,5} }

Iteração 3 : O leitor está convidado a fazer os cáculos sozinho e conferir os resultados com a tabela abaixo :

C_3

Itemset
{2,3,5}

$\overline{C_3}$

TID	Conj-Itemsets
200	{{2,3,5}}
300	{{2,3,5}}

L_3

Itemset	Suporte
{2, 3, 5}	0,5

2.3.4 Otimização reduzindo o número de candidatos gerados - introdução de restrições nos padrões

Suponha que você, como gerente do supermercado, está interessado em descobrir regras de associação onde apareçam os itens Pão **e** Manteiga **ou** apareçam os itens Café, Açúcar **e não** apareça o item Sal. Na verdade, você está impondo uma **restrição** sobre as regras a serem obtidas, o que vai diminuir em muito o número de candidatos gerados a cada iteração. Por exemplo, o itemset {Pão,Café,Sal} não será gerado, pois não satisfaz a restrição do usuário : os itens Pão e Manteiga não aparecem juntos e o item Sal aparece.

A princípio, uma maneira simples de satisfazer as restrições impostas pelo usuário, sem alterar o algoritmo de mineração, consiste em gerar todas as regras sem nenhuma restrição e depois, numa etapa de **pós-processamento**, selecionar somente aquelas que satisfaçam as restrições do usuário. Este procedimento, obviamente é muito custoso. A idéia é modificar o algoritmo de mineração a fim de **incorporar** as restrições dentro da fase de geração de candidatos. Assim, a cada iteração somente os candidatos que satisfazem as restrições impostas pelo usuário serão gerados e testados na fase de validação do suporte.

O que é uma restrição de itens

Existem vários tipos de restrições que o usuário pode impor. Nesta seção, vamos considerar somente *restrições de itens*, que foram introduzidas em [2]. Precisamos primeiro definir alguns conceitos. Considere $\mathcal{I}$ um conjunto de itens $\mathcal{I}$. Um *literal positivo* é um elemento de $\mathcal{I}$ (um item). Um *literal negativo* é uma expressão do tipo $\neg L$, onde L é um item. Uma *restrição de itens* é uma expressão do tipo $D_1 \vee D_2 \vee ... \vee D_n$ onde cada D_i é do tipo :

$$(L_1^i \wedge L_2^i \wedge ... \wedge L_{k_i}^i)$$

e onde cada L_j^i é um *literal* (positivo ou negativo). Por exemplo, a expressão :

(Pão $\wedge$ Manteiga) $\vee$ (Açúcar $\wedge$ Café $\wedge \neg$ Sal)

é uma restrição de itens que traduz o que dissemos acima : que só interessam regras onde apareçam os itens Pão **e** Manteiga **ou** apareçam os itens Café, Açúcar **e não** apareça o item Sal.

O algoritmo Direct

Suponha dados : um banco de dados $\mathcal{D}$ de transações, um nível mínimo de suporte α e uma restrição de itens $\mathcal{B}$. Queremos descobrir todos os itemsets que são frequentes E satisfazem a restrição $\mathcal{B}$.

O algoritmo Direct atua exatamente como o Apriori, somente mudam a fase de geração e poda dos candidatos. Nestas fases será incorporada a restrição $\mathcal{B}$. Assim :

Fase de Geração e Poda : serão gerados somente os candidatos que satisfazem a restrição $\mathcal{B}$ e que são potencialmente frequentes.

As modificações que serão introduzidas (com relação ao Algoritmo Apriori) são baseadas na seguinte idéia :

(*) Se um itemset I de tamanho $k+1$ satisfaz $\mathcal{B}$ então existe ao menos um k-itemset contido em I que satisfaz $\mathcal{B}$, a menos que todo D_i verificado por I tenha exatamente $k+1$ literais positivos.

O leitor está convidado a refletir sobre a validade desta propriedade.

Seja C^b_{k+1} o conjunto dos candidatos de tamanho $k+1$ que satisfazem $\mathcal{B}$. Este conjunto será gerado em 4 etapas :

1. $C^b_{k+1} := L^b_k \times F$, onde L^b_k é o conjunto dos itemsets frequentes de tamanho k **satisfazendo $\mathcal{B}$** que foi construido na iteração precedente, e F é o conjunto dos *itens* que são frequentes.

2. Suprimimos de C^b_{k+1} todos os itemsets que não satisfazem $\mathcal{B}$.

3. Agora vamos à fase da poda dos candidatos que não têm chance nenhuma de serem frequentes. Repare que dispomos agora do conjunto L^b_k (da fase anterior) e **não** do conjunto L_k. Vamos podar aqueles candidatos que possuem um subitemset de tamanho k que não seja frequente. Como fazer isto, utilizando L^b_k ? Ora, se um subitemset não aparece em L^b_k então, de duas uma : ou ele não é frequente ou ele não satisfaz $\mathcal{B}$. Assim, se podarmos aqueles itemsets que possuem um k-subitemset que satisfaz $\mathcal{B}$ e não está em L^b_k teremos certeza de que este subitemset não será frequente e portanto o itemset em questão (que está sendo podado) não terá chance nenhuma de ser frequente. Resumindo : podamos todos os elementos de C^b_{k+1} que possuem um k-subitemset que satisfaz $\mathcal{B}$ e não está em L^b_k.

4. Para cada $D_i = L^i_1 \wedge L^i_2 \wedge ... \wedge L^i_{n_i}$ que compõe a restrição $\mathcal{B}$, com exatamente $k+1$ literais positivos, inserimos em C^b_{k+1} o itemset formado por estes $k+1$ literais caso todos os literais (itens) forem frequentes. Por que isto ? Devido à propriedade (*) é possível que existam $k+1$-itemsets frequentes I satisfazendo $\mathcal{B}$, onde nenhum k-subitemset satisfaz $\mathcal{B}$. Isto acontece quando cada D_i que é satisfeita por I tem exatamente $k+1$ literais positivos. Quando for este o caso, repare que a construção que foi feita no item (1) não inclue tais $k+1$-itemsets (que poderiam ser frequentes). É por isso que precisamos inseri-los em C^b_{k+1}.

Vamos ilustrar esta construção no seguinte exemplo :

Exemplo 2.4 Suponha os itens {1,2,3,4,5} e $\mathcal{B} = (1 \wedge 2) \vee (4 \wedge \neg 5)$. Vamos supor que todos os itens são frequentes, isto é $F = \{1, 2, 3, 4, 5\}$. Assim, $L_1^b = \{\{4\}\}$. Para gerar C_2^b, primeiramente consideramos :

$$L_1^b \times F = \{\{1, 4\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}$$

.

Como {4,5} não satisfaz $\mathcal{B}$, ele é podado no passo 2. No passo 3, nada é mudado em C_2^b já que todos os 1-subitemsets que satisfazem $\mathcal{B}$ são frequentes. Finalmente, na fase 4, inserimos {1,2} em C_2^b, pois os itens 1, 2 aparecem exatamente no primeiro bloco D_1 de $\mathcal{B}$ como literais positivos e todos eles são itens frequentes. Obtemos no final da etapa de geração e poda (feitas conjuntamente) :

$$C_2^b = \{\{1, 2\}, \{1, 4\}, \{2, 4\}, \{3, 4\}\}$$

.

Capítulo 3

Mineração de Sequências

Suponha agora que você, como gerente do supermercado esteja interessado em conhecer a *evolução* das compras de seus clientes, por exemplo, "que sequência de produtos são comprados por um *mesmo* cliente em momentos consecutivos" ? Se você pudesse descobrir que uma sequência de produtos $< p_1, p_2, p_3 >$ é frequentemente comprada **nesta ordem** pelos clientes, você poderia enviar folhetos promocionais envolvendo os produtos p_2 ou p_3 para aqueles clientes que compraram o produto p_1. Você saberia que tais clientes têm grande chance de comprar os produtos p_2 e p_3 no futuro e portanto os recursos gastos nesta campanha de marketing (campanha dirigida a clientes potencialmente dispostos a comprar p_2 e p_3) teriam grandes chances de não estarem sendo despendiçados em vão.

Para descobrir tais sequências, você precisa dispor de um banco de dados de transações dos clientes, onde estes sejam *identificados* (veja que você não necessitava disto para minerar regras de associação), uma vez que será necessário identificar as sequências de produtos que aparecem sucessivamente nas faturas de um *mesmo* cliente. Além disto, as transações dos clientes precisam ser datadas. A figura 1 é um exemplo de um banco de dados de transações próprio para mineração de sequências.

Uma *sequência* ou *padrão sequencial* de tamanho k (ou k-sequência) é uma coleção ordenada de itemsets $< I_1, I_2, ..., I_n >$. Por exemplo, $s =$ <{TV,aparelho-de-som},{Vídeo}, {DVDPlayer}> é um padrão sequencial. Repare que este padrão comportamental se manifesta nos clientes 2 e 3. Todos eles compram num primeiro momento (não importa quando) TV e aparelho-de-som (conjuntamente), num segundo momento compram um Vídeo Cassete e tempos depois compram um DVDPlayer. Suponha que você, como gerente, decide que um padrão sequencial que se manifesta em pelo menos 50% dos clientes registrados será considerado *frequente*. Neste caso, o padrão s acima será considerado frequente. Caso você seja muito exigente e decida que o mínimo para ser considerado frequente é que pelo menos 70% dos clientes manifestem tal comportamento então o padrão s acima não será considerado frequente.

IdCl	Itemsets	Data
1	{TV, ferro-elétrico}	10/02/2002
2	{sapato, aparelho-de-som, TV}	01/03/2002
1	{sapato, lençol}	03/03/2002
3	{TV, aparelho-de-som, ventilador}	04/03/2002
2	{lençol, Vídeo}	05/03/2002
3	{Vídeo, fitas-de-vídeo}	07/03/2002
1	{biscoito, açúcar}	10/04/2002
4	{iogurte, suco}	14/04/2002
4	{telefone}	21/04/2002
2	{DVDPlayer, fax}	23/04/2002
3	{DVDPlayer, liquidificador}	28/04/2002
4	{TV, Vídeo}	30/04/2002

Figura 3.1: Um banco de dados de transações de clientes

3.1 O Problema da Mineração de Sequências

Nesta seção vamos definir todos os elementos envolvidos no problema da mineração de sequências. Este problema foi introduzido em [3]. É claro que estamos interessados em minerar sequências frequentes num banco de dados de transações de clientes no formato da figura 1.

Antes de mais nada, vamos fazer uma primeira transformação no banco de dados, eliminando deste informações inúteis para os nossos propósitos. Repare que não precisamos saber *exatamente* em que data os clientes compraram os produtos. Só precisamos saber a **ordem** em que os produtos são comprados. Assim, vamos eliminar o atributo Data e armazenarmos, para cada cliente, a sequência de suas compras no supermercado. Esta sequência é chamada *sequência do cliente*. Assim, nosso banco de dados da figura 1 transforma-se no banco de dados da figura 2.

IdCl	Sequências de Itemsets
1	< {TV, ferro-elétrico}, {sapato, lençol}, {biscoito, açúcar} >
2	< {sapato, **aparelho-de-som, TV**}, {lençol, **Vídeo**}, {**DVDPlayer**, fax} >
3	< {**aparelho-de-som, TV**, ventilador}, {**Vídeo**, fitas-de-vídeo}, {**DVDPlayer**, liquidificador}>
4	< {iogurte, suco}, {telefone}, {TV, Vídeo} >

Figura 3.2: Um banco de dados de sequências de clientes

A partir de agora, vamos utilizar a seguinte *enumeração dos itens* do supermercado :

Artigo (item)	número que o representa
TV	1
Ferro-elétrico	2
Açúcar	3
Aparelho-de-som	4
DVD-Player	5
Vídeo	6
Fax	7
Telefone	8
Iogurte	9
Suco	10
Lençol	11
Sapato	12
Ventilador	13
Liquidificador	14
Fitas-de-víeo	15
Biscoito	16

Figura 3.3: Representação numérica de cada artigo do supermercado

Definição 3.1 Sejam s e t duas sequências, $s = < i_1 i_2 ... i_k >$ e $t = < j_1 j_2 ... j_m >$. Dizemos que s *está contida* em t se existe uma subsequência de itemsets em t, $l_1, ..., l_k$ tal que $i_1 \subseteq l_1$, ... , $i_k \subseteq l_k$. Por exemplo, sejam $t =< \{1,3,4\}, \{2,4,5\}, \{1,7,8\} >$ e $s =< \{3\}, \{1,8\} >$. Então, é claro que s está contida em t, pois $\{3\}$ está contido no primeiro itemset de t e $\{1,8\}$ está contido no terceiro itemset de t. Por outro lado, a sequência $s' =< \{8\}, \{7\} >$ não está contida em t (explique por que).

De agora em diante, vamos utilizar a seguinte nomenclatura para diferenciar as sequências :

- sequências que fazem parte do banco de dados de sequências (correspondem a alguma sequência de itemsets comprados por algum cliente) : **sequência do cliente**
- sequências que são possíveis padrões que pode aparecer nos dados : **padrão sequencial**

Repare que tanto sequência do cliente quanto padrão sequencial são sequências de itemsets.

Definição 3.2 Uma sequência de cliente t *suporta* um padrão sequencial s se s está contido em t.

Definimos *suporte* de um padrão sequencial s com relação a um banco de dados de sequências de clientes $\mathcal{D}$ como sendo a porcentagem de sequências de clientes que suportam s, isto é :

$$sup(s) = \frac{\text{número de sequências de clientes que suportam } s}{\text{número total de sequências de clientes}}$$

Por exemplo, a tabela da figura 3 contabiliza os suportes de diversos padrões sequenciais com relação ao banco de dados da figura 1.

Padrão	Suporte
$< \{12\} >$	0,5
$< \{12\}, \{3\} >$	0,25
$< \{1\} >$	1
$< \{1,4\}, \{6\}, \{5\} >$	0,5

Figura 3.4: Suporte de diversos padrões sequenciais

Um padrão sequencial s é dito *frequente* com relação a um banco de dados de sequências de clientes $\mathcal{D}$ e um nível minimo de suporte α, se $\sup(s) \geq \alpha$. Por exemplo, suponhamos o banco de dados da figura 1 e $\alpha = 0,5$. Então os seguintes padrões são frequentes : $< \{12\} >, < \{1\} >, < \{1,4\}, \{6\}, \{5\} >$.

O problema da mineração de padrões sequenciais

Este problema é o seguinte :

INPUT : São dados um banco de dados de sequências de clientes $\mathcal{D}$, um nível mínimo de suporte α.

OUTPUT : Pede-se todos os padrões sequenciais frequentes com relação a $\mathcal{D}$ e α.

3.2 O algoritmo AprioriAll para mineração de padrões sequenciais

O algoritmo AprioriAll é baseado em idéias semelhantes àquelas utilizadas no algoritmo Apriori. Vamos discutir tais idéias primeiramente, antes de descrever o algoritmo. Detalhes deste algoritmo e algumas de suas variantes podem ser encontrados em [3].

3.2.1 Propriedades importantes de antimonotonidade

Dizemos que um itemset I está contido numa sequência $s = < s_1, s_2, ..., s_k >$ se existe i tal que I está contido em s_i, isto é, se a sequência unitária $< I >$ está contida em s. O itemset é dito frequente se a sequência $< I >$ é frequente. Prove as seguintes propriedades :

1. Todo subconjunto de um itemset frequente é um itemset frequente.
2. Todo itemset de uma sequência frequente é um itemset frequente.
3. Toda subsequência de uma sequência frequente é uma sequência. frequente.

3.2.2 As 3 fases do Algoritmo AprioriAll

Para ilustrar todas as etapas do Algoritmo AprioriAll, vamos utilizar o banco de dados de sequências da figura 5 :

IdCl	Sequências de Itemsets
1	< {3}, {9} >
2	< {1,2}, {3}, {4,6,7} >
3	< {3,5,7}>
4	< {3}, {4,7}, {9} >
5	< {9} >

Figura 3.5: Um banco de dados de sequências de clientes

O nível mínimo de suporte é $\alpha = 0,4$.

1. **ETAPA dos itemsets frequentes** : Encontrar todos os itemsets frequentes e os enumerar. Para encontrar os itemsets frequentes usa-se o algoritmo Apriori ou derivados com uma pequena adaptação : no caso de Apriori clássico, o suporte de um itemset é a fração das transações em que o itemset está presente. No caso dos

sequential patterns, o suporte de um itemset é a fração dos *clientes* que compraram todo o pacote representado pelo itemset, numa única transação. Na figura 6 abaixo temos a lista dos itemsets frequentes no nosso banco de dados da figura 5.

Itemsets	Suporte	Mapeado em
{3}	0,8	1
{4}	0,4	2
{7}	0,6	3
{4,7}	0,4	4
{9}	0,6	5

Figura 3.6: Itemsets frequentes

A enumeração dos itemsets é feita para agilizar as comparações entre duas sequências (no cálculo do suporte de uma sequência) : teremos que ver se todos os itemsets de uma sequência é igual a um itemset da outra sequência. Se estes itemsets são números, esta comparação é mais fácil do que comparar dois conjuntos de itens para ver se são iguais.

2. **ETAPA da Transformação** :

   Em cada sequência do cliente, cada transação (correspondente a um itemset i) é substituida pelo conjunto de todos os itemsets frequentes contidos neste itemset i. Por que ? Nosso objetivo é calcular sequências frequentes. Ora, todo itemset de uma sequência frequente deve ser frequente (veja propriedade ? acima). Portanto, nesta etap já deixamos preparados todos os itemsets frequentes que aparecem em cada transação de uma sequência do cliente. Isto vai facilitar o teste do cálculo do suporte de um padrã sequencial $s =< s_1, s_2, ..., s_k >$. Só vamos calcular o suporte de padrões sequenciais onde todos os s_i são frequentes. Para isto, se representamos cada sequência do cliente sob a forma $t = < T_1, ..., T_n >$ onde cada T_i é um conjunto de itemsets, o teste se s está contido em t será simples : vai se reduzir a testar se um número pertence a um conjunto (já que enumeramos os itemsets frequentes na fase anterior).

   Repare que as seguintes simplificações podem ser feitas no banco de dados de sequências do cliente sem prejuízo nenhum para os cálculos de suporte.

   - se um itemset de uma sequência do cliente não contém nenhum itemset frequente, ele pode ser retirado da sequência do cliente.

- se nenhum itemset de uma sequência do cliente contém um itemset frequente, eliminamos esta sequência do banco de dados transformado, mas o cliente correspondente ainda contribui no total dos clientes.

A figura 7 ilustra o banco de dados de sequências da figura 5 após a etapa de transformação.

IdCl	Sequências de Itemsets
1	$< \{1\}, \{5\} >$
2	$< \{1\}, \{2,3,4\} >$
3	$< \{1,3\} >$
4	$< \{1\}, \{2,3,4\}, \{5\} >$
5	$< \{5\} >$

Figura 3.7: Banco de dados de sequências transformado

3. **A ETAPA das Sequências** : aqui são calculadas todas as sequências frequentes. A idéia é exatamente a mesma utilizada no Algoritmo Apriori, onde agora no lugar de "itemsets" temos "sequências" e no lugar de "itens" temos "itemsets".

Algoritmo	Elemento atômico	Padrão	Padrão após enumeração dos elementos atômicos
Apriori	Itens	Itemset	Conjunto de Números (na verdade utilizamos sequência de números)
AprioriAll	Itemsets	Sequência de Itemsets	Sequência de Números

As diferenças entre os algoritmos Apriori e AprioriAll podem ser resumidas como se segue :

Fase de Geração dos Candidatos :

(a) **No Algoritmo Apriori** : Por exemplo, se {1,3} e {1,4} estão em L_2, gera-se o candidato {1,3,4}. Veja que os elementos do conjunto são ordenados.

(b) **No Algoritmo AprioriAll** : Por exemplo, se $< 1, 3 >$ e $< 1, 4 >$ estão em L_2, gera-se os candidatos $< 1, 3, 4 >$ e $< 1, 4, 3 >$. Veja que agora as **sequências** $< 1, 3, 4 >$ e $< 1, 4, 3 >$ são distintas. Já os **conjuntos** $\{1, 3, 4\}$ e { 1,4,3 } são idênticos, e portanto convenciona-se considerar somente o ordenado, no Algoritmo Apriori.

A função Apriori-Generate responsável pelo cálculo das sequências pré-candidatas C'_k a partir das sequências frequentes L_{k-1} é dada por :

insert into C'_k
select $p.\text{item}_1$, $p.\text{item}_2$, ..., $p.\text{item}_{k-2}$,$p.\text{item}_{k-1}$, $q.\text{item}_{k-1}$
from L_{k-1} p, L_{k-1} q
where$p.\text{item}_1 = q.\text{item}_1$, $p.\text{item}_2 = q.\text{item}_2$, ...,$p.\text{item}_{k-2} = q.\text{item}_{k-2}$;

Fase da Poda : idêntica nos dois algoritmos. Só muda o teste de inclusão. Testar se um itemset está contido em outro é diferente de testar se uma sequência está contida em outra (aqui a ordem é importante).

Fase do Cálculo do Suporte :

(a) **No Algoritmo Apriori** : os elementos da base de dados ãso **itemsets** e os padrões são **itemsets**. Assim, para verificar se um registro da base suporta um padrão candidato, é necessário fazer um teste de **inclusão de conjuntos**.

(b) **No Algoritmo AprioriAll** : Após a fase de transformação, os elementos da base de dados ãso **sequências de itemsets** (tais itemsets são conjuntos de números, cada número representando um conjunto de itens) e os padrões são **sequências de números** (cada número representando um conjunto de itens). Assim, para verificar se um registro da base suporta um padrão candidato, é necessário fazer um teste de **inclusão de uma sequência de números numa sequência de conjuntos de números**.

Vamos ilustrar o funcionamento do Algoritmo Apriori-All (a etapa das sequências) sobre o banco de dados transformado da figura 7, lembrando que o suporte mínimo é 0,4.

Iteração 1 : Calculamos todas as sequências de tamanho 1, $s =< I >$. Isto é o mesmo que calcular todos os itemsets de tamanho 1, onde o conjunto de itens é $\mathcal{J} = \{1,2,3,4,5\}$ representando os itemsets frequentes. Estas sequências são : $L_1 = \{< 1 >, < 2 >, < 3 >, < 4 >, < 5 >\}$.

Iteração 2 : C'_2 é o conjunto de todas as sequências $< a, b >$, onde $a, b \in \{1, 2, 3, 4, 5\}$. Após a poda temos $C_2 = C'_2$ (nenhuma sequência é podada). Após o cálculo do suporte (varrendo-se uma vez a base), temos :

$$L_2 = \{< 1, 2 >, < 1, 3 >, < 1, 4 >, < 1, 5 >\}$$

Repare que $< 2, 1 >$ não é frequente pois embora o itemset 2 apareça nas sequências dos clientes 2 e 4, o itemset 1 não aparece depois nestas mesmas sequências.

Iteração 3 : $C'_3 = \{< 1, 2, 2 >, < 1, 2, 3 >, < 1, 2, 4 >, < 1, 2, 5 >, < 1, 3, 2 >, < 1, 3, 3 >$ $, < 1, 3, 4 >, < 1, 3, 5 >, < 1, 4, 2 >, < 1, < 1, 4, 3 >, < 1, 4, 4 >, < 1, 4, 5 >, < 1, 5, 2 >, <$ $1, 5, 3 >, < 1, 5, 4 >, < 1, 5, 5 >\}$.

Após a poda temos $C_2 = \emptyset$. Por exemplo, $< 1, 4, 2 >$ é podada pois a subsequência $< 4, 2 >$ não aparece em L_2 e portanto a sequência $< 1, 4, 2 >$ não poderá ser frequente.

Neste capítulo, vamos ver um outro algoritmo para minerar padrões sequenciais, o algoritmo GSP, cuja performance é bem superior à performance do algoritmo AprioriAll. Detalhes deste algoritmo e análise comparativa de performance podem ser encontrados em [4].

Na última seção deste capítulo, discutiremos as principais razões que tornam GSP muito mais eficiente para minerar padrões frequenciais do que AprioriAll. Já podemos adiantar o seguinte : a principal razão reside no fato de que GSP poda muito mais candidatos na fase de podagem, e assim leva para a fase de validação muito menos elementos a serem testados. Sobretudo em dados reais, normalmente o nível mínimo de suporte é bem pequeno, o que acarreta muitos candidatos nas fases posteriores (obtidos fazendo-se a junção de L_{k-1} e L_{k-1}). Assim, se a fase se podagem pudesse eliminar o máximo possível de candidatos que não são potencialmente frequentes, isto poderia otimizar o processo de mineração. É justamente o que faz GSP.

No algoritmo GSP, o conceito fundamental que muda com relação a AprioriAll é o conceito de k-sequência :

Definição 3.3 Uma k-sequência é uma sequência com k *itens*. Um item que aparece em itemsets distintos é contado uma vez para cada itemset onde ele aparece.

Por exemplo, $< \{1, 2\} >, < \{1\}, \{2\} >, < \{1\}, \{1\} >$ são 2-sequências. Repare que no formalismo utilizado no algoritmo AprioriAll, uma k-sequência é uma sequência com k *itemsets*. Assim :

- No algoritmo AprioriAll, em cada iteração k os conjuntos L_k e C_k são constituídos de sequências de k itemsets.

- No algoritmo GSP, em cada iteração k os conjuntos L_k e C_k são constituídos de sequências de k itens.

A figura 1 ilustra como os elementos gerados na iteração 1 de AprioriAll são distribuídos entre as diferentes fases de GSP.

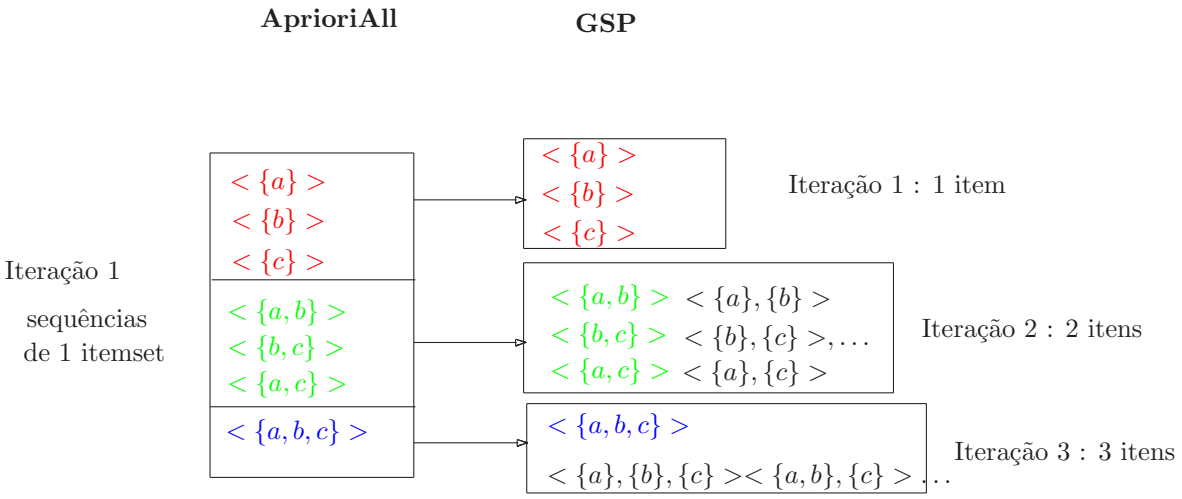


Figura 3.8: A geração de candidatos em GSP é mais refinada do que em AprioriAll

3.3 O Algoritmo GSP

Seguindo a mesma idéia dos algoritmos da família Apriori, o algoritmo GSP gera as k-sequências frequentes (sequência com k itens) na iteração k. Cada iteração é composta pelas fases de geração, de poda e de validação (cálculo do suporte).

3.3.1 Fase da Geração dos Candidatos

Caso $k \geq 3$

Suponhamos que L_{k-1} já tenha sido gerado na etapa $k-1$. Duas sequências $s = < s_1, s_2, \ldots, s_n >$ e $t =< t_1, t_2, \ldots, t_m >$ de L_{k-1} são ditas *ligáveis* se, retirando-se o primeiro item de s_1 e o último item de t_m as sequências resultantes são iguais. Neste caso, s e t podem ser ligadas e produzir a sequência v, onde :

- se t_m não é unitário : $v = < s_1, s_2, \ldots, s_n \cup t' >$, onde t' é o último item de t_m.

- se t_m é unitário : $v = < s_1, s_2, \ldots, s_n, t_m >$

Repare que estamos, como de hábito, supondo que cada itemset está ordenado segundo a ordem lexográfica de seus itens. Estamos supondo também que o conjunto dos itens foi ordenado (a cada item foi associado um número natural).

A figura 2 ilustra o processo de junção de duas sequências de $k-1$ itens a fim de produzir uma sequência de k itens, isto é, como acrescentar um item a mais na primeira sequência de modo que o resultado tenha chance de ser frequente.

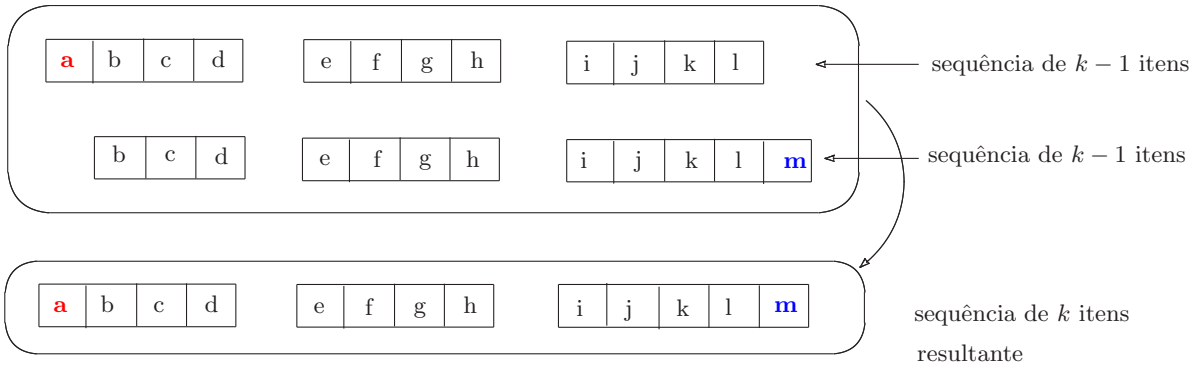


Figura 3.9: Como juntar sequências

Exemplo 3.1 Sejam $s =< \{1,2\},\{3\},\{5,7\} >$, $t =< \{2\},\{3\},\{5,7,10\} >$. Retirando-se o primeiro item de s_1 (o item 1) e o último item de t_3 (o item 10) obtemos a mesma sequência : $< \{2\},\{3\},\{5,7\} >$. Logo, s e t são ligáveis e sua junção produz a sequência :

$$s =< \{1,2\},\{3\},\{5,7,10\} >$$

Definimos o conjunto dos pré-candidatos C'_k como sendo o conjunto obtido ligando-se todos os pares ligáveis de sequências de L_{k-1}.

Exemplo 3.2 Considere L_3 representado na tabela abaixo :

$$
\begin{array}{c}
< \{1,2\},\{3\} > \\
< \{1,2\},\{4\} > \\
< \{1\},\{3,4\} > \\
< \{1,3\},\{5\} > \\
< \{2\},\{3,4\} > \\
< \{2\},\{3\},\{5\} >
\end{array}
$$

Então o conjunto dos pré-candidatos C'_4 é dado por :

$$
\begin{array}{c}
< \{1,2\},\{3,4\} > \\
< \{1,2\},\{3\},\{5\} >
\end{array}
$$

Repare que uma propriedade da sequência resultante da junção de duas sequências s_1 e s_2 é que ao eliminarmos o primeiro item do primeiro itemset da junção obtemos s_2.

Caso $k = 2$: Para juntar duas sequências $s_1 =< \{x\} >$ e $s_2 =< \{y\} >$ de 1 item a fim de produzir uma de dois itens precisamos adicionar o item y de s_2 em s_1 tanto como parte do itemset $\{x\}$ quanto como um itemset isolado. Assim a junção de s_1 com s_2 produz duas sequências de 2 elementos : $< \{x,y\} >$ e $< \{x\},\{y\} >$.

Repare que a propriedade acima mencionada se verifica para as duas sequências obtidas como resultado da junção de s_1 e s_2 : nas duas sequências, ao eliminarmos o primeiro item do primeiro itemset obtemos a sequência $s_2 =< \{y\} >$.

Caso $k = 1$: O cálculo de C_1 considerando-se todas as sequências de 1 item $< \{i\} >$ e testando-se o suporte para cada uma delas. As que são frequentes constituem o conjunto L_1.

3.3.2 Fase da Poda dos Candidatos

Seja s uma k-sequência. Se s for frequente, então, pela Propriedade Apriori, sabemos que toda subsequência de s deve ser frequente. Seja t uma subsequência qualquer obtida de s suprimindo-se um item de algum itemset. Se t não estiver em L_{k-1} então s não tem chance nenhuma de ser frequente e portanto pode ser podada.

Exemplo 3.3 Considere a mesma situação do exemplo 3.2. A sequência $< \{1,2\},\{3\},\{5\} >$ será podada, pois se retiramos o item 2 do primeiro itemset, a sequência resultante $< \{1\},\{3\},\{5\} >$ não está em L_3. Assim, após a fase da poda, o conjunto C_4 resultante é $\{ < \{1,2\},\{3,4\} > \}$.

Exercício : Mostre que o conjunto dos candidatos C_k assim construído contém todas as k-sequências realmente frequentes, isto é : se uma k-sequência é frequente ela tem que necessariamente estar presente em C_k. (Sugestão : utilize a propriedade Apriori.)

3.3.3 Fase da Contagem do Suporte

A cada iteração, cada sequência de cliente d é lida **uma** vez e incrementa-se o contador dos candidatos de C_k que estão contidos em d. Assim, dado um conjunto C_k de sequências candidatas de uma sequência de cliente d, precisamos encontrar todas as sequências em C que estão contidas em d. Duas técnicas são utilizadas para resolver este problema :

1. Usamos uma estrutura de árvore-hash para reduzir o número de candidatos de C que serão testados para d.

2. Transformamos a representação da sequência de cliente d de tal modo que possamos testar de forma eficiente se um determinado candidato de C é suportado (está contido) em d.

3.4 Detalhes de implementação

Construção da árvore hash para armazenar as sequências candidatas

Uma *árvore-hash* é uma árvore onde as folhas armazenam conjuntos de padrões sequenciais (sequências de itemsets), e os nós intermediários (inclusive a raiz) armazenam tabelas-hash contendo pares do tipo (número, ponteiro). A construção é análoga à que foi feita para armazenar conjuntos de itemsets na Aula 3. Temos dois parâmetros : M = número máximo de sequências numa folha e N = número máximo de ramos saindo de cada nó. Para armazenar uma sequência s na árvore, aplicamos uma função hash a cada **item** da sequência. Observamos que no algoritmo AprioriAll, esta função é aplicada para cada **itemset** da sequência (lembre-se que em AprioriAll, cada itemset é representado por um número).

Exemplo 3.4 Seja M = 3 e N = 2. Suponhamos que tenhamos 4 itens e considere a função hash $h(1) = 1, h(2) = 2, h(3) = 1, h(4) = 2$. Consideremos o seguinte conjunto de 2-sequências :

$$\{< \{1,3\} >, < \{1\},\{3\} >, < \{2\},\{3\} >, < \{3\},\{3\} >, < \{2,3\}, < \{1\},\{4\} >\}$$

A princípio inserimos $< \{1,3\} >, < \{1\},\{3\} >, < \{2\},\{3\} >$ na raiz. Quando vamos inserir $< \{3\},\{3\} >$ na raiz, o número de sequências é 4. Então, o nó é quebrado e obtemos uma árvore com uma raiz e dois nós descendentes : nó (1) contém as sequências $< \{1,3\} >, < \{1\},\{3\} >, < \{3\},\{3\} >$ e nó (2) contém a sequência $< \{2\},\{3\} >$. Quando vamos inserir a sequência $< \{2,3\} >$, calculamos $h(2) = 2$. Neste caso, esta sequência é inserida no nó 2 que sai da raiz. Agora, vamos inserir o último padrão sequencial, $< \{1\},\{4\} >$. Calcula-se $h(1) = 1$. Logo, este padrão é inserido no nó (1). Porém, agora este nó contém 4 sequências, excendendo assim o limite de 3 sequências permitido. Assim sendo, quebra-se o nó (1) em dois nós descendentes, aplicando-se a função h aos segundos itens de cada padrão. A distribuição final das sequências na árvore é ilustrada na figura 3 abaixo. Como todas as folhas não contém mais de 3 sequências, o processo termina.

Observação : estamos supondo, é claro, que os itens foram ordenados. Sejam i e j duas *ocorrências* de itens em uma sequência s. Dizemos que a ocorrência i é menor do que a ocorrência j em s se uma das duas condições se verificam : (1) i e j aparecem num mesmo itemset de s e os itens correspondentes (que também denotamos por i e j) satisfazem $i < j$, (2) j aparece num itemset J de s e i aparece num itemset I de s e $s = < ..., I, ..., J, ... >$. Por exemplo : seja $s =< \{1,3\},\{1\},\{2\} >$. Neste caso, a primeira ocorrência do item 1 é inferior à segunda ocorrência do item 1. A (única) ocorrência do item 3 é inferior à (única) ocorrência do item 2, pois a primeira se dá no primeiro itemset e a segunda se dá no terceiro itemset.

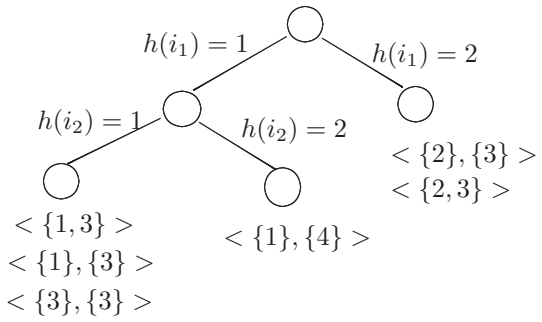


Figura 3.10: Uma árvore-hash armazenando sequências

Como determinar em que folhas buscar candidatos possivelmente suportados por uma dada sequência do cliente d

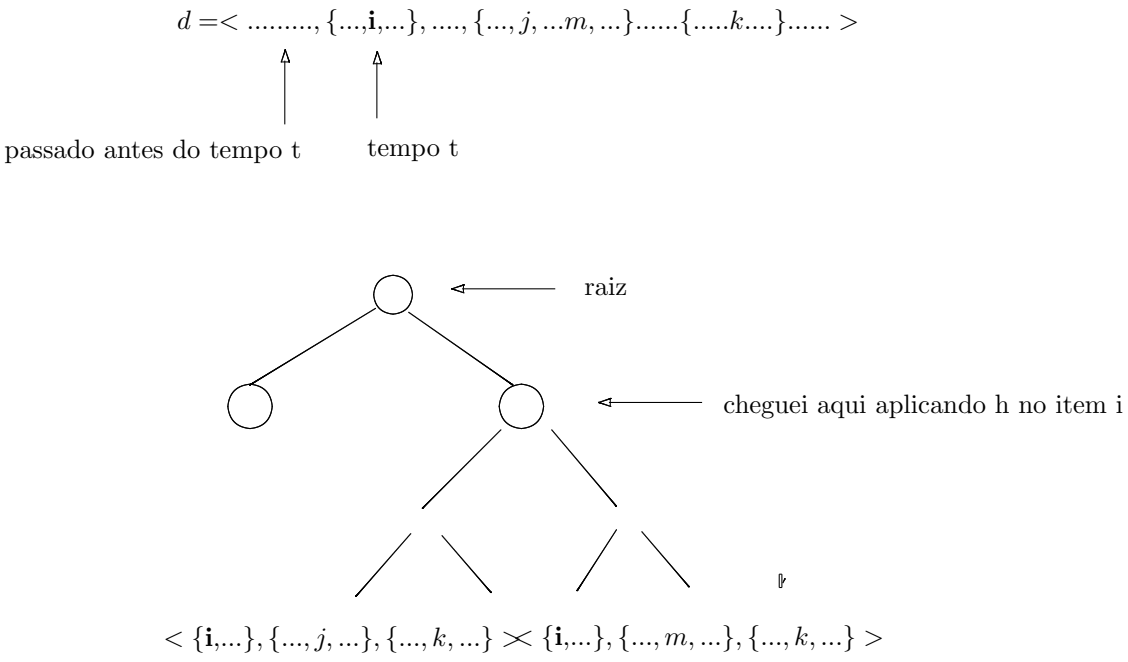
1. Calcula-se a função h para cada item de d e dirige-se para o nó correspondente indicado pelo valor da função.

2. Caso o nó a que se chegou é uma folha, aplica-se nesta folha o procedimento Include descrito abaixo, a cada elemento s da folha.

3. Caso o nó a que se chegou não é uma folha : suponha que se chegou a este nó aplicando-se a função h ao item i de d, cujo tempo-de-transação é t. Aplica-se a função h a todos os itens de d cujo tempo-de-transação seja superior ou igual a t. E dirige-se para o nó indicado pelo valor da função.

Por que não se aplica a função h a itens de d cujo tempo-de-transação seja inferior a t ?

Suponha que tenhamos chegado à um nó do nível 2 da árvore aplicando a função h a um item da sequência d que corresponde a um tempo t. A partir daí estamos à procura de candidatos possivelmente suportados por d tais que seu **primeiro** item é i e os **restantes** dos itens aparecem em d, no mesmo itemset de i (tempo t presente) ou em itemsets futuros (tempo maior do que t). Logo, não estamos interessados em itens que aparecem no **passado** de i, isto é, que aparecem em itemsets antes do tempo t. Assim, não vamos aplicar a função h a estes itens do passado de i.



Vamos descrever num exemplo, o processo esboçado acima para determinar quais as folhas deverão ser varridas, para cada sequência do cliente d. Suponhamos que o conjunto de itens é {1,2,3,4,5}, que a função h é definida como sendo $h(1) = h(3) = h(5) = 1$, $h(2) = h(4) = 2$. Suponhamos que a sequência do cliente é

$$d =< \{1, 5\}, \{1\}, \{3\} >$$

Suponhamos também que os candidatos foram armazenados na árvore-hash da figura 3.

Passo 1 : $h(1) = 1$. Vamos para o primeiro nó do segundo nível.

Passo 2 : $h(5) = 1$. Vamos para o primeiro nó do terceiro nível. Trata-se de uma folha. Aplicamos o procedimento **Include** a cada elemento s desta folha.

Passo 3 : $h(1) = 1$. Vamos para o primeiro nó do terceiro nível. Trata-se de uma folha. Já foi visitada.

Passo 4 : $h(3) = 1$. Vamos para o primeiro nó do terceiro nível. Trata-se de uma folha. Já foi visitada.

Passo 5 : $h(5) = 1$. Vamos para o primeiro nó do segundo nível.

Passo 6 : $h(1) = 1$. Vamos para o primeiro nó do terceiro nível. Trata-se de uma folha. Já foi visitada.

Passo 7 : $h(3) = 1$. Vamos para o primeiro nó do terceiro nível. Trata-se de uma folha. Já foi visitada.

Passo 8 : $h(1) = 1$. Vamos para o primeiro nó do segundo nível.

Passo 9 : $h(3) = 1$. Vamos para o primeiro nó do terceiro nível. Trata-se de uma folha. Já foi visitada.

Passo 10 : $h(3) = 1$. Vamos para o primeiro nó do segundo nível.

Passo 11 : Como chegamos a este nó aplicando-se h a uma ocorrência de item cujo tempo-de-transação é 3 e não existe nenhuma outra ocorrência de item maior ou igual a esta, o algoritmo pára.

Repare que neste exemplo, a segunda folha do nível 2 e a segunda folha do nível 3 não serão varridas.

Como testar de forma eficiente se um candidato está contido em d

Procedimento **Include**

Input : s, d (duas sequências)

Output: Responde 'Sim', se s está incluída em d. 'Não', em caso contrário.

Abaixo, descrevemos de forma informa como o procedimento **Include** opera.

Cria-se um array que tem tantos elementos quanto o número de itens do banco de dados. Para cada item de d, armazenamos no array uma lista dos tempos das transações de d que contém este item. Por exemplo : suponha que $d = < \{1,2\}, \{4,6\}, \{3\}, \{1,2\}, \{3\}, \{2,4\}, \{6\} >$ e que os tempos de cada itemset de d são dados na tabela abaixo :

Tempo	Itens
1	1, 2
2	4,6
3	3
4	1, 2
5	3
6	2, 4
7	6

Suponhamos que o número total de itens é 7. Uma representação alternativa para a sequência d é o array de 7 elementos, onde cada elemento é uma lista de tempos, como mostra a tabela abaixo :

Item	Lista de Tempos
1	[1,4]
2	[1,4,6]
3	[3,5]
4	[2,6]
5	[ ]
6	[2,7]
7	[ ]

Para testar se um padrão sequencial $s =< s_1, s_2, ..., s_n >$ é suportado por d :

1. Encontramos a primeira ocorrência de s_1. Para isto, varremos os itens de s_1 e encontramos o primeiro tempo de cada item (nas respectivas listas). Caso este tempo for o mesmo para cada item, este será o tempo t_1 da primeira ocorrência de s_1 em d e o processo termina. Caso contrário, seja t = máximo dos primeiros tempos de cada item de s_1. Repetimos o processo, mas agora tentando encontrar o primeiro tempo $\geq t$ de cada item nas respectivas listas.

2. Encontramos a primeira ocorrência de s_2 após o tempo t_1 encontrado no item anterior. O processo é o mesmo descrito acima. Caso consigamos encontrar uma tal ocorrência, esta vai corresponder a um tempo $t_2 > t_1$. Repetimos o processo para s_3, etc.

3. Caso estejamos testando a primeira ocorrência de um itemset s_i após um tempo t_i, o processo que descrevemos acima pára quando uma das listas é vazia (neste caso, a sequência s não é suportada por d) ou quando se consegue encontrar um mesmo

tempo $t_{i+1} \geq t_i$ correspondendo ao menor tempo $\geq t_i$ de cada uma das listas dos itens de s_i.

Exemplo 3.5 Vamos considerar a sequência do cliente d ilustrada acima e consideremos o padrão sequencial $s =< \{2,4\}, \{6,7\} >$.

1. Consideremos o primeiro itemset {2,4}. As listas dos tempos de 2 e 4 são respectivamente : $[1,4,6]$ e $[2,6]$. Os primeiros tempos de cada uma são : 1 e 2. Como não são iguais, repetimos o processo tentando encontrar os primeiros tempos maiores ou iguais a 2 (máximo entre 1 e 2). Estes tempos são 4 e 2. Como não são iguais, repetimos o processo tentando encontrar os primeiros tempos maiores ou iguais a 4 (máximo entre 4 e 2). Estes tempos são 4 e 6. Como não são iguais, repetimos o processo tentando encontrar os primeiros tempos maiores ou iguais a 6 (máximo entre 4 e 6). Estes tempos são 6 e 6. Como são iguais, a primeira ocorrência de $s_1 = \{2,4\}$ é no tempo $t_1 = 6$.

2. Consideramos agora o itemset {6,7}. Vamos tentar encontrar a primeira ocorrência deste itemset *depois* do tempo 6. As listas dos tempos maiores do que 6 para cada um dos itens 6 e 7 são respectivamente : [7] e [ ]. Como uma das listas é vazia, o processo pára e conclui-se que s não está contida em d.

3.5 Discussão : comparação de performances entre AprioriAll e GSP

Existem duas maneiras de implementar AprioriAll. Normalmente, a fase de transformação exige muito espaço em disco para armazenar o banco de dados de sequências sob forma de *sequências de conjuntos de itemsets*. Assim, na maioria dos casos faz-se a transformação *on-the-fly*, isto é : a cada varrida do banco de dados, para cada sequência do cliente, calcula-se sua versão transformada na memória principal e testa-se os candidatos que são suportados pela sequência transformada. A sequência transformada **não é armazenada** em disco. Na próxima iteração, no momento de calcular o suporte, todos os cálculos para obter as sequências transformadas serão refeitos. Referimo-nos a AprioriAll-Cached quando se tratar da implementação que armazena em disco o banco de dados transformado e AprioriAll simplesmente quando se tratar da implementação que calcula as sequências transformadas *on-the-fly*. É claro que AprioriAll-Cached é bem mais eficiente do que AprioriAll.

Testes sobre dados sintéticos mostraram que GSP é até 5 vezes mais rápido do que AprioriAll e até 3 vezes mais rápido do que AprioriAll-Cached. Em dados reais, com níveis de suporte de 0.01%, GSP chegou a ser 20 vezes mais eficiente do que AprioriAll e em torno de 9 vezes mais rapido do que AprioriAll-Cached.

Análise : Por que GSP é mais eficiente do que AprioriAll

Existem duas razões principais para esta melhor performance de GSP com relação a AprioriAll :

1. A cada iteração, o número de candidatos testados é menor, pois a fase de podagem de GSP elimina muito mais candidatos indesejáveis do que AprioriAll. De fato, AprioriAll poda candidatos s testando se as subsequências obtidas eliminando-se um *itemset* inteiro de s não estão em L_{k-1}. GSP é mais refinado, ele poda candidatos s testando se as subsequências obtidas eliminando-se um *item* de s não estão em L_{k-1}. Veja que para s permanecer como candidato após a fase da poda, em AprioriAll é necessário que um certo conjunto de subsequências S_1 (as sequências obtidas eliminando-se um itemset de s) esteja contido em L_{k-1}. Em GSP, é necessário que um certo conjunto de subsequências S_2 (as sequências obtidas eliminando-se um item de s) muito maior do que S_1 esteja contido em L_{k-1}. Assim, é muito mais fácil para s passar ileso pela podagem de AprioriAll do que pela podagem de GSP. GSP é bem mais refinado.

2. Na versão *on-the-fly* de AprioriAll, os cálculos que devem ser realizados a cada varrida do banco de dados para obter a sequência transformada são responsáveis em grande parte pela melhor performance de GSP com relação a AprioriAll. O procedimento utilizado na fase de cálculo de suporte de GSP tem desempenho igual ou ligeiramente inferior ao procedimento correspondente em AprioriAll-Cached. Mas vale a pena lembrar que AprioriAll-Cached tem que fazer a transformação do banco de dados uma vez.

Capítulo 4

Mineração de Sequências com Restrições

Os algoritmos de mineração de sequências que vimos até aqui não fornecem ao usuário mecanismos que restringem os padrões sequenciais de interesse, além da restrição imposta pelo suporte. Nesta aula, vamos ver quais outras restriçẽos interessantes pode-se impor aos padrões a fim de melhor atender às expectativas do usuário. Na primeira parte destas notas, vamos introduzir diferentes tipos de restrições com diversos exemplos. Na segunda parte, vamos introduzir uma família importante de algoritmos que permite minerar sequências que satisfazem um certo tipo de restrição importante : uma expressão regular dada pelo usuário.

4.1 Tipos de Restrições : na Fase de Geração e na Fase de Validação

Restrições são condições impostas pelo usuário, que os padrões sequenciais devem satisfazer a fim de serem minerados. Podemos classificar as restrições em duas categorias : as restrições "de geração" e as restrições de "validação". As primeiras são restrições que são impostas na Fase de Geração dos algoritmos de mineração a fim de diminuir o espaço de busca dos padrões. As segundas são restrições que só podem ser verificadas na Fase de Validação dos algoritmos de mineração. Os padrões sequenciais são gerados de maneira livre e só na fase de validação do suporte é que são eliminados os padrões que não satisfazem as restrições. A seguir vamos dar exemplos de cada uma das categorias de restrições.

4.1.1 Restrições de Validação

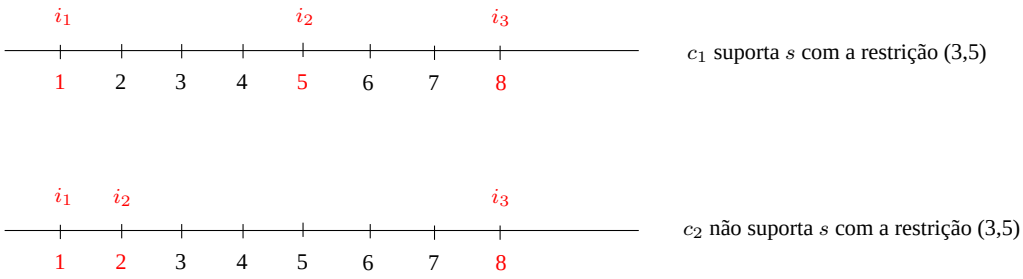
MIN-MAX

Este tipo de restrição foi introduzido por Agrawal/Srikant em [4]. Imagine que para avaliar o grau de interesse de um padrão sequencial $< s_1, s_2 >$, você não esteja interessado em considerar clientes que compram os itens de s_1 e somente depois de dois anos compram os itens de s_2. Neste caso, há um espaço de tempo demasiado grande entre as compras de s_1 e s_2. Um tal cliente não deve ser levado em conta no momento de calcular o suporte do padrão $< s_1, s_2 >$, uma vez que se está interessado apenas em clientes que compram s_1 seguido de s_2 *algum tempo* (não muito) depois. Por outro lado, também não se está interessado em contar para o suporte de $< s_1, s_2 >$, clientes que compram s_1 seguido de s_2 *pouco tempo* depois, por exemplo, com um dia de diferença. A fim de eliminar tais clientes da contagem do suporte, impõe-se dois limites de tempo para os intervalos entre as compras : um limite *mínimo* m e um limite *máximo* M.

Uma restrição de MIN-MAX é, portanto, um par de inteiros (m, M). Um cliente c suporta o padrão sequencial $s =< s_1, ..., s_n >$ com a restrição (m, M) se existem instantes $t_1, t_2, ..., t_n$ tais que $(c, s_1, t_1), ..., (c, s_n, t_n)$ está no banco de dados de transações e para todo $i = 1, ..., n-1$ tem-se : $m \leq \mid t_{i+1} - t_i \mid \leq M$. Dizemos que o padrão s satisfaz a restrição de MIN-MAX (m, M) se o número de clientes que suportam s com a restrição (m, M) dividido pelo número total de clientes é superior ou igual ao nível mínimo de suporte.

Exemplo : Considere o padrão $s = < i_1, i_2, i_3 >$, $m = 3$, $M = 5$. Considere as sequintes transações de clientes :

IdCl	Itemset	Tempo
c_1	i_1	1
c_1	i_2	5
c_1	i_3	8
c_2	i_1	1
c_2	i_2	2
c_2	i_3	8



Time-Window

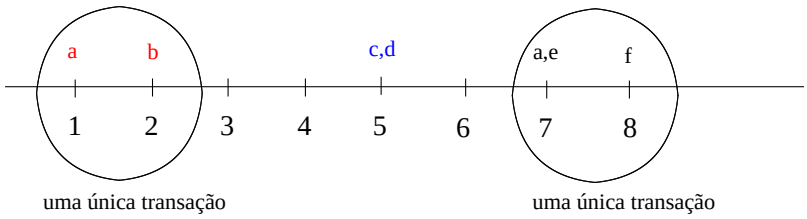
Este tipo de restrição também foi introduzido por Agrawal/Srikant em [4]. Imagine que para avaliar o grau de interesse de um padrão sequencial $< s_1, s_2 >$, você não esteja interessado em considerar *apenas* clientes que compram os itens de s_1 numa mesma transação e os itens de s_2 numa outra transação posterior. Na verdade, um cliente c que compra parte dos itens de s_1 numa transação de manhã e o restante à noite, deveria ser considerado na avaliação do suporte de $< s_1, s_2 >$. Seria como se as *duas* transações realizadas para a compra de todos os itens de s_1 fossem uma *única* transação. Para isso, impõe-se um limite W (denominado *time-window*) tal que todas as transações realizadas no intervalo $[t_0 - W, t_0 + W]$ são consideradas como tendo sido efetuadas no mesmo instante t_0.

Uma restrição de Time-Window é, portanto, um número $W \geq 0$. Um cliente c suporta o padrão sequencial $s =< s_1, ..., s_n >$ com a restrição de Time-Window W se existem instantes $t_1, ..., t_n$ tais que para todo item $i \in s_j$ existe $t_0^j \in [t_j - W, t_j + W]$ tal que (c, i, t_0^j) está no banco de dados de transações.

Exemplo

Considere o padrão sequencial $s = < \{a, b\}, \{c, d\}, \{e, f, a\} >$, $W = 2$

A figura abaixo ilustra o fato de uma sequência de cliente suportar o padrão s com a restrição de time-window W :



No artigo [4] é apresentada uma versão bem geral do algoritmo GSP que inclui restrições de MIN-MAX e de Time-Window.
Observação importante: Para integrar restrições MIN-MAX e TIME-WINDOW na fase de validação será preciso que o banco de dados armazene também os tempos de cada itemset, já que estes serão essenciais no momento de verificar as restrições. Cada sequência do cliente é armazenada da seguinte maneira : (IdCl, {(tempo, itemsets)}). Assim, considere o banco de dados da figura 4 :

IdCl	Itemsets	Tempo-de-Trans
1	{TV, ferro-elétrico}	1
2	{sapato, aparelho-de-som, TV}	2
1	{sapato, lençol}	3
3	{TV, aparelho-de-som, ventilador}	4
2	{lençol, Vídeo}	5
3	{Vídeo, fitas-de-vídeo}	6
1	{biscoito, açúcar}	7
4	{iogurte, suco}	8
4	{telefone}	9
2	{DVDPlayer, fax}	10
3	{DVDPlayer, liquidificador}	11
4	{TV, Vídeo}	12

Figura 4.1: Um banco de dados de transações de clientes

Este banco de dados será armazenado como mostra a figura 5 :

4.1.2 Restrições de Geração

Restrições de Conjuntos

Restrições de Conjuntos são restrições impostas aos padrões na fase de geração, do tipo : só se gera padrões $s =< s_1, \ldots, s_n >$ onde os *conjuntos* de itens s_i satisfazem uma determinada condição envolvendo operações entre conjuntos. Por exemplo, podemos estar interessados em padrões sequenciais $< s_1, s_2, \ldots, s_n >$ onde cada transação s_i possui um item que se repete nas outras transações. Uma tal restrição pode ser expressa pela equação $s_1 \cap s_2 \cap \ldots \cap s_n \neq 0$ que deve ser fornecida pelo usuário como input do algoritmo de mineração.

IdCl	Sequências do cliente
1	{ (1, {TV, ferro-elétrico}), (2, {sapato, lençol}), (7, {biscoito, açúcar})}
2	{ (2, {sapato, aparelho-de-som, TV}), (5, {lençol, Vídeo}), (10, {DVDPlayer, fax}) }
3	{ (4, {TV, aparelho-de-som, ventilador}), (6, {Vídeo, fitas-de-vídeo}), (11, {DVDPlayer, liquidificador}) }
4	{ (8, {iogurte, suco}), (9, {telefone}), (12, {TV, Vídeo}) }

Figura 4.2: Banco de dados de transações de clientes transformado

Restrições de Expressão Regular

Este tipo de restrição foi introduzida por Garofalakis, Rastogi, Shim em [5, 6]. Imagine que você esteja interessado somente em minerar padrões sequenciais $< s_1, ..., s_n >$ que satisfazem uma determinada expressão regular, por exemplo, que começam por {TV} e terminam em {DVD Player}. Assim, somente serão gerados padrões satisfazendo a expressão regular :

$$\{TV\}a^*\{DVDPlayer\}$$

onde a^* representa uma sequência qualquer de itemsets. Para ser mais exato : $a = (a_1 + a_2 + ... + a_n)$, onde $\{a_1, ..., a_n\}$ é o conjunto de todos os itemsets possíveis de serem formados com os itens dados.

4.2 Os algoritmos da família SPIRIT - idéia geral

Formulação do Problema

Input = um banco de dados $\mathcal{D}$, um nível mínimo de suporte α e uma expressão regular $\mathcal{R}$.

Output = todas as sequências s com $\sup(s) \geq \alpha$ e que satisfazem $\mathcal{R}$.

Uma primeira idéia para resolver este problema de mineração seria a seguinte :

Seja $L^k = k$-sequências frequentes satisfazendo $\mathcal{R}$.

Fase de Geração : usando L^k e $\mathcal{R}$, produzir um conjunto $\overline{C}^{k+1}$ de candidatos tais que :

- Os candidatos devem satisfazer $\mathcal{R}$.
- Os candidatos são $k+1$-sequências potencialmente frequentes.
- Assim, os candidatos $\overline{C}^{k+1}$ devem conter L^{k+1}.

Fase da Podagem : Suprimir de $\overline{C}^{k+1}$ aquelas sequências σ que não têm nenhuma chance de serem frequentes.

Repare que a dificuldade em utilizar esta idéia é que a fase de podagem deve ser efetuada utilizando somente o conjunto L^k calculado na fase anterior, e que é constituído de todas as sequências de tamanho k *que são frequentes e que satisfazem a expressão regular* R. Note que a restrição de ser frequente é **Antimonotônica** mas a restrição de satisfazer uma expressão regular **não é**. Por exemplo, a sequência abb satisfaz a expressão regular ab^*, mas sua subsequência bb não satisfaz ab^*. Logo, na fase de podagem, não basta simplesmente eliminar as $k+1$-sequências que possuem uma k-sequência que não está em L^k.

Seja $L = L_1 \cup L_2 \cup ... \cup L_k$. Precisamos eliminar sequências σ que não sejam frequentes. Para isto, é *suficiente* que σ possua uma subsequência $\sigma' \subseteq \sigma$ que não seja frequente. Ora, se $\sigma' \notin L$ e σ' satisfaz a expressão regular R, teremos certeza de que σ' não é frequente. Assim :

$$C^{k+1} = \overline{C}^{k+1} - \{\sigma \in \overline{C}^{k+1} \mid \exists \sigma' \subseteq \sigma, \quad \sigma' \notin L \text{ e } \sigma' \text{ satisfaz } \mathcal{R}\}$$

Problema com esta idéia :

Seja $A^{k+1} = \{\sigma \in \overline{C}^{k+1} \mid \exists \sigma' \subseteq \sigma, \quad \sigma' \notin L \text{ e } \sigma' \models \mathcal{R}\}$ o conjunto de sequências que são podadas. Repare que quanto **mais** restritiva for a expressão regular $\mathcal{R}$, **menor** será o conjunto A^{k+1}, isto é, menos sequências serão podadas. A figura abaixo ilustra este fato:

"Poder de Restrição" de $\mathcal{R}$	$\overline{C}^{k+1}$	A^{k+1}	$\overline{C}^{k+1} - A^{k+1}$
$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$

Assim, a introdução da restrição $\mathcal{R}$, por um lado, na fase de geração restringe os candidatos gerados, mas por outro lado, na fase da podagem, também restringe as sequências podadas, o que não é interessante. Precisamos encontrar uma espécie de "meio-termo" : como restringir suficientemente os candidatos na fase de geração sem diminuir muito o conjunto de sequências que serão podadas na fase de podagem ?

Observação : No exercício 5 da Lista 2, vai ficar claro por que na fase da podagem precisamos testar subsequências de tamanho qualquer e não somente de tamanho k.

Idéia : Considerar um "relaxamento" apropriado da expressão regular $\mathcal{R}$

O que é um "relaxamento" de $\mathcal{R}$? Sabemos que a expressão regular $\mathcal{R}$ especifica uma *linguagem regular*, isto é, o conjunto de todas as palavras (sequências) que satisfazem $\mathcal{R}$. Um "relaxamento" de $\mathcal{R}$ seria qualquer condição c (inclusive uma outra expressão regular $\mathcal{R}'$) mais fraca do que $\mathcal{R}$, isto é, tal que a linguagem satisfazendo c contivesse a linguagem satisfazendo $\mathcal{R}$. Assim, c é menos restritiva do que $\mathcal{R}$. Que tipo de relaxamento seria considerado "apropriado" ? Cada relaxamento $\mathcal{R}'$ de $\mathcal{R}$ corresponde a um Algoritmo SPIRIT($\mathcal{R}'$), cuja idéia geral de execução é a descrita acima, mas considerando, ao invés de $\mathcal{R}$, a condição $\mathcal{R}'$. Estuda-se as performances dos diversos algoritmos da família e chega-se à conclusão, de forma experimental, qual o relaxamento mais apropriado.

4.3 Os quatro algoritmos principais da família SPIRIT

Antes de discutirmos estes algoritmos, notamos que se $\mathcal{R}$ é **antimonotônica** então as fases de geração e podagem são exatamente como nos algoritmos da família Apriori. Neste caso, não é necessário procurar um relaxamento $\mathcal{R}'$ de $\mathcal{R}$, pois as fases de geração e podagem estarão em "sintonia".

Se σ é frequente e satisfaz $\mathcal{R}$ e $\sigma' \subseteq \sigma$ então σ' deve ser frequente e satisfazer $\mathcal{R}$.

Logo, a fase da podagem consiste simplesmente em eliminar as sequências candidatas $\sigma \in \overline{C}^{k+1}$ tais que$\exists \sigma' \subseteq \sigma, \quad \sigma' \notin L^k$.

Os quatro principais algoritmos da família SPIRIT são SPIRIT(N), SPIRIT(L), SPIRIT(V) e SPIRIT($\mathcal{R}$), cada um deles correspondente a um relaxamento da restrição $\mathcal{R}$.

1. **SPIRIT(N)** : aqui consideramos o maior de todos os relaxamentos de $\mathcal{R}$, aquele que não impõe nenhuma restrição às sequências. Assim, neste caso, uma qualquer sequência satisfaz a "restrição" N.

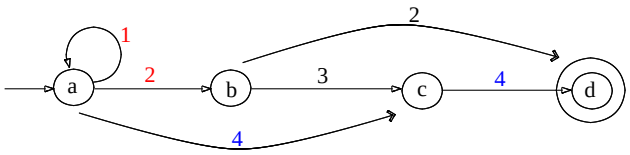
2. **SPIRIT(L)** : neste relaxamento, somente são consideradas as sequências *legais* com respeito a algum estado do autômato correspondente à expressão regular $\mathcal{R}$, que denotamos por $A_{\mathcal{R}}$. Dizemos que uma sequência $a_1a_2...a_n$ é *legal* com respeito ao estado q do autômato $\mathcal{R}$ se existe um caminho no autômato que começa no estado q e que percorre a palavra $a_1a_2...a_n$.

3. **SPIRIT(V)** : neste relaxamento, somente são consideradas as sequências *válidas* com respeito a algum estado do autômato $A_{\mathcal{R}}$. Dizemos que uma sequência $a_1a_2...a_n$

é *válida* com respeito ao estado q do autômato $\mathcal{R}$ se existe um caminho no autômato que começa no estado q e **termina num estado final** e que percorre a palavra $a_1 a_2 ... a_n$.

4. **SPIRIT($\mathcal{R}$)** : este, não é um relaxamento. Corresponde exatamente à expressão $\mathcal{R}$. Somente as sequências *válidas* (isto é, aquelas que começam no estado inicial e terminam num estado final do autômato) são aceitas.

A seguinte figura ilustra as noções de sequências *legais* com respeito a algum estado de $\mathcal{A}_\mathcal{R}$, de sequências *válidas* com respeito a algum estado de $\mathcal{A}_\mathcal{R}$ e de sequências válidas com respeito a $\mathcal{A}_\mathcal{R}$.



$< 1, 2 >$: **legal** com respeito ao estado a do autômato, pois existe um caminho no autômato percorrendo a sequência $< 1, 2 >$.

$< 2 >$: **válida** com respeito ao estado b do autômato, pois existe um caminho no autômato, saindo do estado b, percorrendo a sequência $< 2 >$ e chegando num estado final.

$< 4, 4 >$: **válida**, pois existe um caminho no autômato, saindo do estado inicial, percorrendo a sequência $< 4, 4 >$ e chegando num estado final.

A tabela abaixo resume as restrições consideradas por cada um dos algoritmos SPIRIT.

Algoritmo	**Relaxamento $\mathcal{R}'$**
SPIRIT(N)	nenhuma restrição
SPIRIT(L)	somente sequências *legais* com respeito a algum estado de $\mathcal{A}_\mathcal{R}$
SPIRIT(V)	somente sequências *válidas* com respeito a algum estado de $\mathcal{A}_\mathcal{R}$
SPIRIT(R)	somente sequências válidas ($\mathcal{R}' = \mathcal{R}$)

Um exercício para o leitor : Mostre que

$$\boxed{L^k(R) \subseteq L^k(V) \subseteq L^k(L) \subseteq L^k(N)}$$

Assim, V é um "relaxamento" de $\mathcal{R}$, L é um "relaxamento" de V e N é o maior de todos os "relaxamentos", o menos restritivo, já que não impõe restrição nenhuma às sequências.

Resumo : O esquema geral dos algoritmos SPIRIT é o seguinte:

ETAPA 1 : Etapa do relaxamento R'
Calcula-se o conjunto L' das sequências frequentes que satisfazem um relaxamento R' da expressão regular R original fornecida pelo usuário (vamos denotar A_R o autômato correspondente a R). R' pode ser : (1) o relaxamento total (algoritmo SPIRIT(N)), (2) o relaxamento correspondente às sequências legais com respeito a algum estado do autômato A_R (algoritmo SPIRIT(L)), (3) o relaxamento correspondente às sequências válidas com respeito a algum estado do autômato A_R (algoritmo SPIRIT(V)), (4) nenhum relaxamento, isto é, R' = R (algoritmo SPIRIT(R)).

ETAPA 2 : Etapa da Restrição R
Elimina-se de L' as sequências que não satisfazem R, obtendo-se assim o conjunto L das sequências frequentes e que satisfazem R. Isto se faz através de um procedimento que dado um autômato e um string, verifica se o string é ou não aceito pelo autômato.

Repare que o algoritmo SPIRIT(N) corresponde a aplicar o algoritmo GSP sem nenhuma restrição na fase de geração (ETAPA 1). A ETAPA 2 corresponden a uma etapa de pós-processamento, onde são eliminadas as sequências que não interessam. Num outro extremo está o algoritmo SPIRIT(R), onde a ETAPA 2 não realiza nada, pois a ETAPA 1 já fornece o conjunto L das sequências frequentes e que satisfazem R.

4.4 Resultados Experimentais

Estes 4 algoritmos foram testados em dados reais constituídos de logs de páginas web acessadas por usuários de um Departamento de Ciência da Computação (DCC) durante uma semana. A expressão regular R corresponde a todas as sequências de URL's que se iniciam pelo endereço "home" do DCC e terminam no endereço do curso de Mestrado do DCC. O objetivo é minerar todos os caminhos mais frequentes percorridos por usuários que acessam a página do DCC e que chegam á página do curso de Mestrado. O nível mínimo de suporte foi de 0,3 % e o banco de dados de sequências continha 12868 sequências. Os resultados são ilustrados no quadro abaixo :

Algo	Tempo de Exec (seg.)	Total de Candidatos	Iterações
SPIRIT (N)	1562,80	5896	13
SPIRIT (L)	32,77	1393	10
SPIRIT (V)	16	59	5
SPIRIT (R)	17,67	52	7

A partir deste quadro podemos concluir que :

- os algoritmos que incorporam a restrição R ou um relaxamento dela na ETAPA 1 são muito mais eficientes que o algoritmo SPIRIT(N) onde nada é feito na ETAPA 1 em termos de restringir o espaço de busca na fase de geração dos candidatos. Repare a quantidade enorme de candidatos gerados em SPIRIT(N) com relação aos outros 3 algoritmos.
- o algoritmo SPIRIT(V) é o mais eficiente dentre os quatro, embora seu desempenho não seja tão superior ao algoritmo SPIRIT(R), onde todo o processo de restrição é realizado já na ETAPA 1.

4.5 Detalhes de Implementação de SPIRIT

Maiores detalhes sobre esta seção podem ser encontrados em [6].

Já que o algoritmo SPIRIT(V) é o mais eficiente dentre os quatro da família SPIRIT, vamos detalhar as fases de geração e poda de cada iteração da ETAPA 1, **somente para este algoritmo**. Detalhes dos outros algoritmos podem ser encontrados em [6].

4.5.1 Fase de Geração

Na iteração k, dispomos do conjunto L'_{k-1} das sequências frequentes de tamanho $k-1$ e que são válidas com respeito a algum estado do autômato A_R. Sejam $\{q_0, q_1, ..., q_n\}$ o conjunto dos estados do autômato A_R. Então :

$$L'_{k-1} = L'_{k-1}(q_0) \cup ... \cup L'_{k-1}(q_n)$$

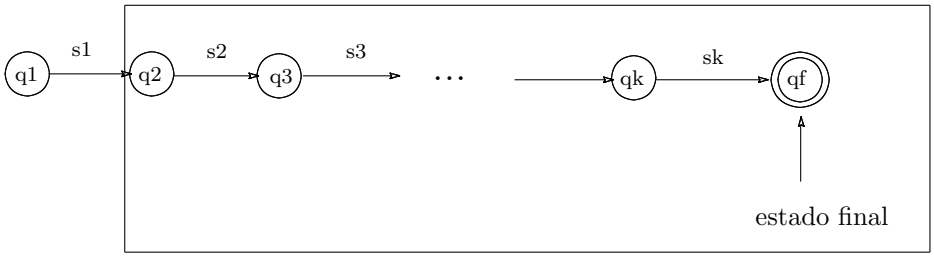
onde $L'_{k-1}(q)$ denota o conjunto das sequências frequentes de tamanho $k-1$ e que são válidas com respeito ao estado q do autômato A_R.

Para que uma sequência $< s_1, s_2, ..., s_k >$ de tamanho k seja válida com respeito a algum estado q_1 do autômato A_R é preciso que :

1. o sufixo $< s_2, ..., s_k >$ de tamanho $k-1$ seja válida com respeito a algum estado q_2 do autômato A_R e

2. exista uma transição no autômato indo de q_1 para q_2, com label s_1.

Além disto, se queremos que $< s_1, s_2, ..., s_k >$ tenha chance de ser frequente é preciso *ao menos que* o sufixo $< s_1, s_2, ..., s_k >$ seja frequente. Logo, a partir desta condição e da condição (1) acima, é preciso exigir que este sufixo esteja em L'_{k-1}.



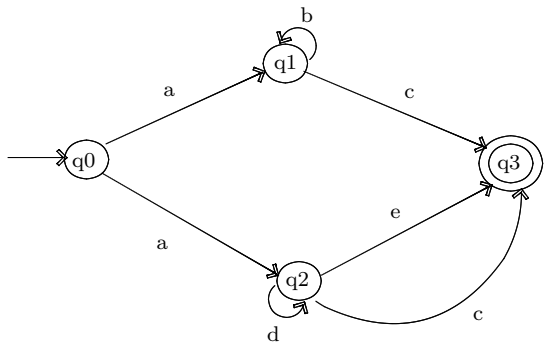
o sufixo de tamanho k-1 deve estar em L'k-1

Logo, o procedimento para calcular os **pré-candidatos** C'_k de tamanho k a partir de L'_{k-1} é o seguinte :

- Para cada estado q do autômato, identifique em L'_{k-1} qual é o conjunto $L'_{k-1}(q)$.
- Para cada transição $q \xrightarrow{a} q'$ e para cada sequência $< b_1, ..., b_{k-1} >$ de $L'_{k-1}(q')$ construa a sequência $< a, b_1, ..., b_{k-1} >$. O conjunto de todas as sequências assim obtidas é $L'_k(q)$.
- O conjunto L'_k é a união de todos os $L'_k(q)$, para cada estado q do autômato.

É claro que todos os pré-candidatos são válidos com respeito a algum estado do autômato A_R e são potencialmente frequentes. Veja que só estamos exigindo que um pré-candidato tenha seu sufixo de tamanho $k-1$ frequente. É exigir muito pouco. Logo, é de se esperar que muitas sequências não tenham chance nenhuma de serem frequentes e que serão podadas na fase de podagem.

Exemplo 4.1 Considere o seguinte autômato A_R :



Suponhamos que $L'_2 = L'_2(q_0) \cup L'_2(q_1) \cup L'_2(q_2)$ e $L'_2(q_0) = \{< a, c >, < a, e >\}$, $L'_2(q_1) = \{< b, c >\}$, $L'_2(q_2) = \{< d, c >, < d, e >\}$.

1. Consideremos q_0. Temos duas transições partindo de q_0.

   (a) $q_0 \xrightarrow{a} q_1$: neste caso consideramos a 3-sequência $< a, b, c >$, já que $L'_2(q_1) = \{< b, c >\}$.

   (b) $q_0 \xrightarrow{a} q_2$: neste caso consideramos as 3-sequências $< a, d, c >, < a, d, e >$, já que $L'_2(q_2) = \{< d, c >, < d, e >\}$.

2. Consideremos q_1. Temos duas transições partindo de q_1 :

   (a) $q_1 \xrightarrow{b} q_1$: neste caso consideramos a 3-sequência $< b, b, c >$, já que $L'_2(q_1) = \{< b, c >\}$.

   (b) $q_1 \xrightarrow{c} q_3$: neste caso não consideramos nenhuma 3-sequência já que $L'_2(q_3) = \emptyset$.

3. Consideremos q_2. Temos três transições partindo de q_2 :

   (a) $q_2 \xrightarrow{d} q_2$: neste caso consideramos as 3-sequências $< d, d, c >, < d, d, e >$, já que $L'_2(q_2) = \{< d, c >, < d, e >\}$.

   (b) $q_2 \xrightarrow{e} q_3$: neste caso não consideramos nenhuma 3-sequência já que $L'_2(q_3) = \emptyset$.

   (c) $q_2 \xrightarrow{c} q_3$: neste caso não consideramos nenhuma 3-sequência já que $L'_2(q_3) = \emptyset$.

Logo, o conjunto dos pré-candidatos é dado por $C'_3 = \{< a, b, c >, < a, d, c >, < a, d, e >, < b, b, c >, < d, d, c >, < d, d, e >\}$.

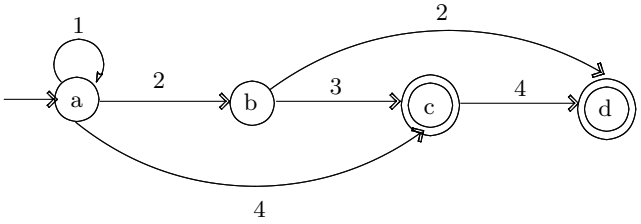
4.5.2 Fase de Poda

Vamos podar de C'_k as sequências que possuem uma subsequência com tamanho inferior a k e que não estejam em L' = união dos L'_i para $i = 1, ..., k-1$. Repare que, como a condição de satisfazer a restrição não é antimonotônica, não podemos simplesmente podar aquelas que não estão no último L_{k-1} (Veja problema 5 da segunda lista de exercícios).

Primeiramente, portanto, para cada k-sequência pré-candidata s precisamos calcular todas as subsequências de tamanho maximal que são válidas com respeito a algum estado do autômato A_R (isto é, nenhuma subsequência de s contendo estritamente uma destas subsequências será válida com relação a algum estado do autômato A_R - estas são as "maiores possíveis"). Depois, verificamos se uma destas subsequências não está em L' (=união dos L'_i para $i = 1, ..., k-1$), então s deverá ser podada.

Por que isto ? : suponha que s' é uma subsequência de s de tamanho maximal que seja válida com respeito a algum estado do autômato (todas as de tamanho superior não o são). Suponha também que s' não esteja em L'. É possível que s seja frequente ? Ora, se s' não está em L' então não estará em nenhum dos L'_i (para $i = 1, ..., k-1$). Se tamanho de s' = N ($N < k$) então obviamente s' não estará em L'_N e portanto não será frequente. Portanto, s não po derá ser frequente, já que contém uma subsequência s' que não é frequente.

Exemplo 4.2 Consideremos uma pequena variante do autômato que vimos na aula anterior :



Repare que agora o estado c também é final (logo, os estados finais são : c e d)

Suponhamos que $L' = \{< 1,4 >, < 2,2 >, < 2,3 >, < 1,2,2 >, < 2,3,4 >\}$. O conjunto dos pré-candidatos C'_3 é dado por :

$$C'_3 = \{< 1,1,2,2 >, < 1,2,3,4 >\}$$

A sequência $< 1,1,2,2 >$ não é podada pois a única subsequência maximal de tamanho inferior a 4 que é válida é {1,2,2} que está em L'_3.

Por outro lado, a sequência $< 1, 2, 3, 4 >$ será podada pois as subsequências maximais de tamanho inferior a 4 que são válidas são $\{< 1, 2, 3 >, < 2, 3, 4 >, < 1, 4 >\}$. Uma delas, a sequência $< 1, 2, 3 >$ não está em L'. Logo, $< 1, 2, 3, 4 >$ deve ser podada.

Como exercício constate que se considerarmos o autômato do exemplo 1.1, nenhuma sequência será podada de C'_3 (calculado neste exemplo).

Em [6] é desenvolvido um algoritmo FINDMAXSEQ (um tanto complexo) para encontrar todas as subsequências de uma sequência válida s dada, que são válidas com relação a um estado do autômato A_R, tenham tamanho inferior ao de s e que sejam maximais.

4.5.3 Condição de Parada

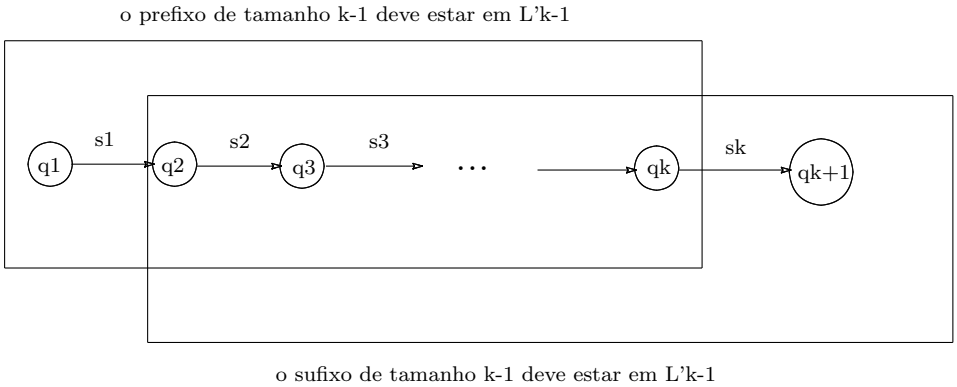
Quando $L'_k = \emptyset$ o algoritmo SPIRIT(V) pára. Veja que para isso é necessário que para cada estado q do autômato, o conjunto das k-sequências frequentes e válidas com relação a este estado q seja vazio. Por exemplo, o fato de que $L'_2(q_0)$ seja vazio não implica necessariamente que $L'_4(q_0)$ seja vazio. Basta considerar o autômato :

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{c} q_3 \xrightarrow{d} q_f$$

onde q_0 é estado inicial e q_f é estado final.

Compare esta condição de parada com a condição de parada do algoritmo SPIRIT(L) (Exerçício 10, Lista de exercícios 2).

A fase de geração do algoritmo SPIRIT(L) : a fase de geração para o algoritmo SPIRIT(L) é muito parecida com a do algoritmo SPIRIT(V). A idéia é ilustrada na figura abaixo :



Na iteração k, dispomos do conjunto L'_{k-1} das sequências frequentes de tamanho $k-1$ e que são legais com respeito a algum estado do autômato A_R. Sejam $\{q_0, q_1, ..., q_n\}$ o conjunto dos estados do autômato A_R. Então :

$$L'_{k-1} = L'_{k-1}(q_0) \cup ... \cup L'_{k-1}(q_n)$$

Para que uma sequência $< s_1, s_2, ..., s_k >$ de tamanho k seja frequente e legal com respeito a algum estado q_1 do autômato A_R é preciso que :

1. o sufixo $< s_2, ..., s_k >$ de tamanho $k-1$ seja legal em relação a q_2 e frequente,
2. o prefixo $< s_1, ..., s_{k-1} >$ de tamanho $k-1$ seja legal em relação a q_1 e frequente,
3. exista uma transição no autômato indo de q_1 para q_2, com label s_1.

Logo, a partir das condições (1) e (2) acima, é preciso exigir que o prefixo e o sufixo estejam em L'_{k-1}. Portanto, para gerar os pré-candidatos em SPIRIT(L) :

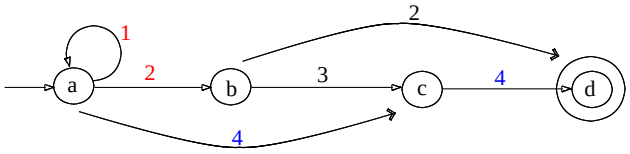
- Para cada estado q do autômato, identifique em L'_{k-1} qual é o conjunto $L'_{k-1}(q)$.
- Para cada par de estados q e q' : Junte sequências de $L_{k-1}(q)$ com sequências de $L_{k-1}(q')$ se (a) retirando o primeiro elemento de uma delas e o último elemento da outra, obtemos a mesma sequência e (b) se existe uma transição $q \xrightarrow{s_1} q'$, onde s_1 é o primeiro elemento da primeira sequência.
- O conjunto L'_k é a união de todos as junções de $L'_k(q)$ com $L'_k(q')$, onde q e q' são estados do autômato.

Vamos agora ver num exemplo, onde os quatro algoritmos são executados sobre o mesmo input.

Exemplo 4.3 Considere o banco de dados $\mathcal{D}$:

Dataset $\mathcal{D}$
$< 1, 2, 3, 2 >$
$< 1, 1, 2, 2 >$
$< 2, 4, 3, 4 >$
$< 2, 3, 4, 3 >$
$< 1, 1, 2, 3 >$

Considere o autômato da aula passada:



Suponhamos também que o nível mínimo de suporte é $\alpha = 0,4$ (40%, logo para ser frequente é preciso ser suportado por ao menos 2 sequências de $\mathcal{D}$).

Na primeira iteração, todos os algoritmos são executados da mesma maneira.

$L'1$ = sequências unitárias constituídas dos itens frequentes = $< 1 >, < 2 >, < 3 >, < 4 >$.

Na segunda iteração : o processo é idêntico nos quatro algoritmos : combina-se todos os elementos de $L'1$ obtendo-se todas as 2-sequências possíveis. Depois, elimina-se aquelas que não satisfazem a condição R'. O conjunto resultante é L'_2.

A partir da terceira iteração, cada algoritmo tem sua forma própria de atuar :

SPIRIT(N) :

L'_2
$< 1,1 >$
$< 1,2 >$
$< 1,3 >$
$< 2,2 >$
$< 2,3 >$
$< 2,4 >$
$< 3,4 >$
$< 4,3 >$

C_3	Contador
$< 1,1,1 >$	0
$< 1,1,2 >$	2
$< 1,1,3 >$	1
$< 1,2,2 >$	2
$< 1,2,3 >$	2
$< 2,2,2 >$	0
$< 2,2,3 >$	0
$< 2,2,4 >$	0
$< 2,3,4 >$	0
$< 2,4,3 >$	0

L'_3
$< 1,1,2 >$
$< 1,2,2 >$
$< 1,2,3 >$
$< 2,3,4 >$
$< 2,4,3 >$

C_4	Contador
$< 1,1,2,2 >$	1
$< 1,1,2,3 >$	1

SPIRIT(L) :

Est.	L'_2
a	$< 1, 1 >$
a	$< 1, 2 >$
a	$< 2, 2 >$
a	$< 2, 3 >$
b	$< 3, 4 >$

Est.	C_3	Cont.
a	$< 1, 1, 1 >$	0
a	$< 1, 1, 2 >$	2
a	$< 1, 2, 2 >$	2
a	$< 1, 2, 3 >$	2
a	$< 2, 3, 4 >$	2

Est.	L'_3
a	$< 1, 1, 2 >$
a	$< 1, 2, 2 >$
a	$< 1, 2, 3 >$
a	$< 2, 3, 4 >$
a	

Est.	C_4	Cont.
a	$< 1, 1, 2, 2 >$	1
a	$< 1, 1, 2, 3 >$	1

SPIRIT(V) :

Est.	L'_2
a	$< 2, 2 >$
b	$< 3, 4 >$

Est.	C_3	Cont.
a	$< 1, 2, 2 >$	2
a	$< 2, 3, 4 >$	2

Est.	L'_3
a	$< 1, 2, 2 >$
a	$< 2, 3, 4 >$

Est.	C_4	Cont.
a	$< 1, 1, 2, 2 >$	1
a	$< 1, 2, 3, 4 >$	0

SPIRIT(R) :

L'_2
$< 2, 2 >$

C_3	Contador
$< 1, 2, 2 >$	2
$< 2, 3, 4 >$	2

L'_3
$< 1, 2, 2 >$
$< 2, 3, 4 >$

C_4	Contador
$< 1, 1, 2, 2 >$	1
$< 1, 1, 3, 4 >$	0

Capítulo 5

Classificação

5.1 Introdução

Suponha que você é gerente de uma grande loja e disponha de um banco de dados de clientes, contendo informações tais como *nome, idade, renda mensal, profissão* e se comprou ou não produtos eletrônicos na loja. Você está querendo enviar um material de propaganda pelo correio a seus clientes, descrevendo novos produtos eletrônicos e preços promocionais de alguns destes produtos. Para não fazer despesas inúteis você gostaria de enviar este material publicitário apenas a clientes que sejam potenciais compradores de material eletrônico. Outro ponto importante : você gostaria de, a partir do banco de dados de clientes de que dispõe no momento, desenvolver um método que lhe permita saber que tipo de atributos de um cliente o tornam um potencial comprador de produtos eletrônicos e aplicar este método no futuro, para os novos clientes que entrarão no banco de dados. Isto é, a partir do banco de dados que você tem hoje, você quer descobrir regras que classificam os clientes em duas classes : os que compram produtos eletrônicos e os que não compram. Que tipos de atributos de clientes (idade, renda mensal, profissão) influenciam na colocação de um cliente numa ou noutra classe ? Uma vez tendo estas regras de classificação de clientes, você gostaria de utilizá-las no futuro para classificar novos clientes de sua loja.

Por exemplo, regras que você poderia descobrir seriam :
Se idade está entre 30 e 40 e a renda mensal é 'Alta' então ClasseProdEletr = 'Sim'.
Se idade está entre 60 e 70 então ClasseProdEletr = 'Não'.

Quando um novo cliente João, com idade de 25 anos e renda mensal 'Alta' e que tenha comprado discos, é catalogado no banco de dados, o seu classificador lhe diz que este cliente é um potencial comprador de aparelhos eletrônicos. Este cliente é colocado na classe ClasseProdEletr = 'Sim', mesmo que ele ainda não tenha comprado nenhum produto eletrônico.

5.1.1 O que é um classificador ?

Classificação é um processo que é realizado em três etapas :

1. Etapa da criação do modelo de classificação. Este modelo é constituído de regras que permitem classificar as tuplas do banco de dados dentro de um número de classes pré-determinado. Este modelo é criado a partir de um banco de dados de *treinamento.* cujos elementos são chamados de *amostras ou exemplos.*

   Por exemplo, vamos considerar o seguinte banco de dados de treinamento :

Nome	Idade	Renda	Profissão	ClasseProdEletr
Daniel	=< 30	Média	Estudante	Sim
João	41..50	Média-Alta	Professor	Sim
Carlos	41..50	Média-Alta	Engenheiro	Sim
Maria	41..50	Baixa	Vendedora	Não
Paulo	=< 30	Baixa	Porteiro	Não
Otávio	> 60	Média-Alta	Aposentado	Não

   Esta etapa também é chamada de **Etapa de Aprendizado** : através de técnicas especiais de aprendizado aplicadas no banco de dados de treinamento, um modelo de classificação é criado e as seguintes regras de classificação são produzidas :

   (a) Se idade = 41..50 e Renda = Média-Alta então ClasseProdEletr = Sim.

   (b) Se Renda = Baixa então ClasseProdEletr = Não.

   O atributo correspondente à classe, no caso do exemplo, o atributo CompraProdEletr, é chamado de **Atributo-Classe**. Em "Classificação", o Atributo-Classe é fornecido, bem como os possíveis valores que possa assumir. Neste caso, o processo de aprendizado (criação do modelo, isto é, das regras de classificação) é chamado de *supervisionado.* Em contrapartida, em "Clustering", o atributo-classe não é conhecido e o número de classes também não. Neste caso, o processo de aprendizado é dito *não supervisionado.*

2. Etapa da verificação do modelo ou **Etapa de Classificação** : as regras são testadas sobre um outro banco de dados, completamente independente do banco de dados de treinamento, chamado de *banco de dados de testes.* A qualidade do modelo é medida

em termos da porcentagem de tuplas do banco de dados de testes que as regras do modelo conseguem classificar de forma satisfatória. É claro que se as regras forem testadas no próprio banco de dados de treinamento, elas terão alta probabilidade de estarem corretas, uma vez que este banco foi usado para extraí-las. Por isso a necessidade de um banco de dados completamente novo.

Por exemplo, consideremos o seguinte banco de dados de testes :

Nome	Idade	Renda	Profissão	ClasseProdEletr
Pedro	41..50	Média-Alta	Ecologista	Não
José	41..50	Média-Alta	Professor	Não
Luiza	41..50	Média-Alta	Assistente Social	Não
Carla	=< 30	Baixa	Vendedora	Não
Wanda	=< 30	Baixa	Faxineira	Não
Felipe	> 60	Média-Alta	Aposentado	Não

As tuplas (1), (2), (3) não são bem classificadas pelo modelo. As tuplas (4),(5),(6) o são. Logo, o grau de acertos (accuracy) do modelo é de 50%. Caso este grau de acertos for considerado bom, pode-se passar para a etapa seguinte.

3. Etapa da utilização do modelo em novos dados : após o modelo ter sido aprovado nos testes da etapa anterior, ele é aplicado em novos dados.

5.1.2 Métodos de Classificação - Critérios de Comparação de métodos

Nas próximas aulas vamos estudar alguns métodos de classificação. Tais métodos podem ser comparados e avaliados de acordo com os seguintes critérios:

1. **O grau de acertos** (accuracy) : este critério refere-se a capacidade do modelo em classificar corretamente as tuplas do banco de dados de testes.
2. **Rapidez** : refere-se ao tempo gasto no processo de classificação.
3. **Robustez** : refere-se à abilidade do modelo em fazer uma classificação correta mesmo em presença de ruídos ou valores desconhecidos em alguns campos dos registros.
4. **Escalabilidade** : refere-se à eficiência do processo de aprendizado (construção do modelo) em presença de grandes volumes de dados de treinamento.

5. **Interpretabilidade** : refere-se ao nível de entendimento que o modelo fornece, isto é, o quanto as regras fornecidas são entendidas pelos usuários do classificador.

5.1.3 Preparando os dados para classificação

Os seguintes passos de pré-processamento podem ser aplicados aos dados a fim de aumentar a qualidade (accuracy), eficiência e escalabilidade do processo de classificação.

1. **Limpeza dos dados** : remover ruídos e resolver problemas de tuplas com valores desconhecidos, por exemplo, substituindo estes valores pelos valores mais correntes do atributo correspondente, ou o valor mais provável, baseado em estatística. Este passo ajuda a reduzir possível confusão durante o aprendizado.

2. **Análise de Relevância** : Alguns atributos podem ser totalmente *irrelevantes* para a tarefa de classificação. Por exemplo, a informação sobre telefone e e-mail dos clientes não influencia sua classificação em "Comprador de Produto Eletrônico" ou "Não comprador de Produto Eletrônico". Assim, é importante se fazer uma análise prévia de quais atributos realmente são essenciais para a tarefa de classificação.

3. **Transformação dos Dados** :

   - **Categorização** : Atributos que assumem uma grande variedade de valores, podem ser agrupados em algumas poucas categorias. Por exemplo, a renda mensal do cliente pode ser agrupada em 4 categorias : Baixa, Média, Média-Alta e Alta. A idade do cliente também pode ser agrupada em *faixas etárias* : $=< 20$, 21..30, > 60, etc.
   - **Generalização** : Certos atributos, como RUA, podem ser substituídos por um atributo mais geral CIDADE. Assim, ao invés de registrar a rua onde mora o cliente, simplesmente registra-se a cidade.
   - **Normalização** : Certos dados também podem ser normalizados, sobretudo quando se utiliza, no processo de aprendizado, técnicas de redes neurais ou métodos envolvendo medidas de distância. A normalização envolve escalonar os valores de um atributo de modo que fiquem compreendidos dentro de um intervalo pequeno, por exemplo [-1,1] ou [0,1]. Em métodos que utilizam medids de distância por exemplo, isto evitaria que atributos com valores com um grande espectro de variação (por exemplo, renda mensal) tenham preponderância sobre atributos com um baixo espectro de variação (atributos com valores binários, por exemplo).

5.2 Classificação utilizando Arvores de Decisão

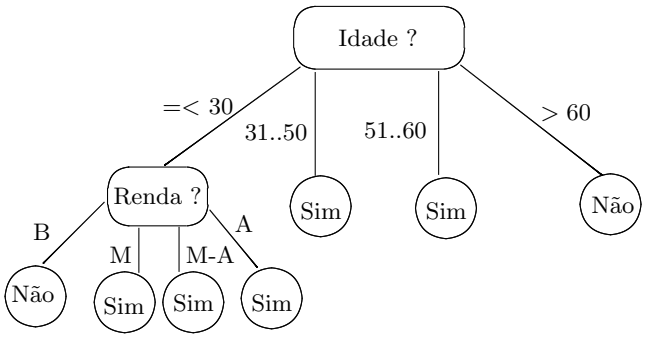
Uma *árvore de decisão* é uma estrutura de árvore onde :

- cada nó interno é um atributo do banco de dados de amostras, diferente do atributo-classe,
- as folhas são valores do atributo-classe,
- cada ramo ligando um nó-filho a um nó-pai é etiquetado com um valor do atributo contido no nó-pai. Existem tantos ramos quantos valores possíveis para este atributo.
- um atributo que aparece num nó não pode aparecer em seus nós descendentes.

Exemplo 5.1 Considere o nosso banco de dados de treinamento:

Nome	Idade	Renda	Profissão	ClasseProdEletr
Daniel	=< 30	Média	Estudante	Sim
João	31..50	Média-Alta	Professor	Sim
Carlos	31..50	Média-Alta	Engenheiro	Sim
Maria	31..50	Baixa	Vendedora	Não
Paulo	=< 30	Baixa	Porteiro	Não
Otávio	> 60	Média-Alta	Aposentado	Não

A figura abaixo ilustra uma possível árvore de decisão sobre este banco de dados:



5.2.1 Idéia geral de como criar uma árvore de decisão

A idéia geral é a que está por trás do algoritmo ID3, criado por Ross Quinlan, da Universidade de Sydney em 1986 e de seus sucessores (um deles, o algoritmo C4.5 também proposto por Quinlan em 1993). Trata-se do procedimento recursivo:*

Gera-Arvore($\mathcal{A}$,Cand-List)

Input : Um banco de dados de amostras $\mathcal{A}$ onde os valores dos atributos foram categorizados, uma lista de atributos candidatos Cand-List.

Output : Uma árvore de decisão

Método

(1) Crie um nó N; Associe a este nó o banco de dados $\mathcal{A}$

(2) Se todas as tuplas de $\mathcal{A}$ pertencem à mesma classe C então transforme o nó N numa folha etiquetada por C. **Páre.**

(3) Caso contrário : Se Cand-List = $\emptyset$ então transforme N numa folha etiquetada com o valor do atributo-Classe que mais ocorre em $\mathcal{A}$. **Páre.**

(4) Caso contrário : calcule Ganho(Cand-List). Esta função retorna o atributo com o maior ganho de informação. Será detalhada na próxima seção. Chamamos este atributo de Atributo-Teste.

(5) Etiquete N com o nome de Atributo-Teste

(6) Etapa da partição das amostras $\mathcal{A}$: para cada valor s_i do Atributo-Teste faça o seguinte :

(7) Crie um nó-filho N_i, ligado a N por um ramo com etiqueta igual ao valor s_i e associe a este nó o conjunto $\mathcal{A}_i$ das amostras tais que o valor de Atributo-Teste = s_i.

(8) Se $\mathcal{A}_i = \emptyset$: transforme o nó N_i numa folha etiquetada pelo valor do atributo-Classe que mais ocorre em $\mathcal{A}$.

(9) Caso contrário : calcule Gera-Arvore($\mathcal{A}_i$, Cand-List - {Atributo-Teste}) e "grude" no nó N_i a árvore resultante deste cálculo.

5.2.2 Como decidir qual o melhor atributo para dividir o banco de amostras ?

Agora vamos detalhar a função Ganho(Cand-List) que decide qual atributo em Cand-List é o mais apropriado para ser utilizado no particionamento das amostras.

Nesta seção vamos utilizar como exemplo o seguinte banco de dados amostral sobre condições meteorológicas. O objetivo é identificar quais as condições ideiais para se jogar um determinado jogo.

Aparência	Temperatura	Humidade	Vento	Jogo
Sol	Quente	Alta	Falso	Não
Sol	Quente	Alta	Verdade	Não
Encoberto	Quente	Alta	Falso	Sim
Chuvoso	Agradável	Alta	Falso	Sim
Chuvoso	Frio	Normal	Falso	Sim
Chuvoso	Frio	Normal	Verdade	Não
Encoberto	Frio	Normal	Verdade	Sim
Sol	Agradável	Alta	Falso	Não
Sol	Frio	Normal	Falso	Sim
Chuvoso	Agradável	Normal	Falso	Sim
Sol	Agradável	Normal	Verdade	Sim
Encoberto	Agradável	Alta	Verdade	Sim
Encoberto	Quente	Normal	Falso	Sim
Chuvoso	Agradável	Alta	Verdade	Não

Vamos considerar as 4 possibilidades para a escolha do atributo que será utilizado para dividir o banco de dados no primeiro nível da árvore. Estas possibilidades estão ilustradas na Figura 1.

Qual a melhor escolha ? Repare que se uma folha só tem 'Sim' ou só tem 'Não', ela não será mais dividida no futuro : o processo GeraArvore aplicado a esta folha pára logo no início. Gostaríamos que isto ocorresse o mais cedo possível, pois assim a árvore produzida será menor. Assim, um critério *intuitivo* para a escolha do atributo que dividirá um nó seria : "Escolha aquele que produz os nós mais puros". Por exemplo, no nosso caso, a escolha boa seria o atributo *Aparência*.

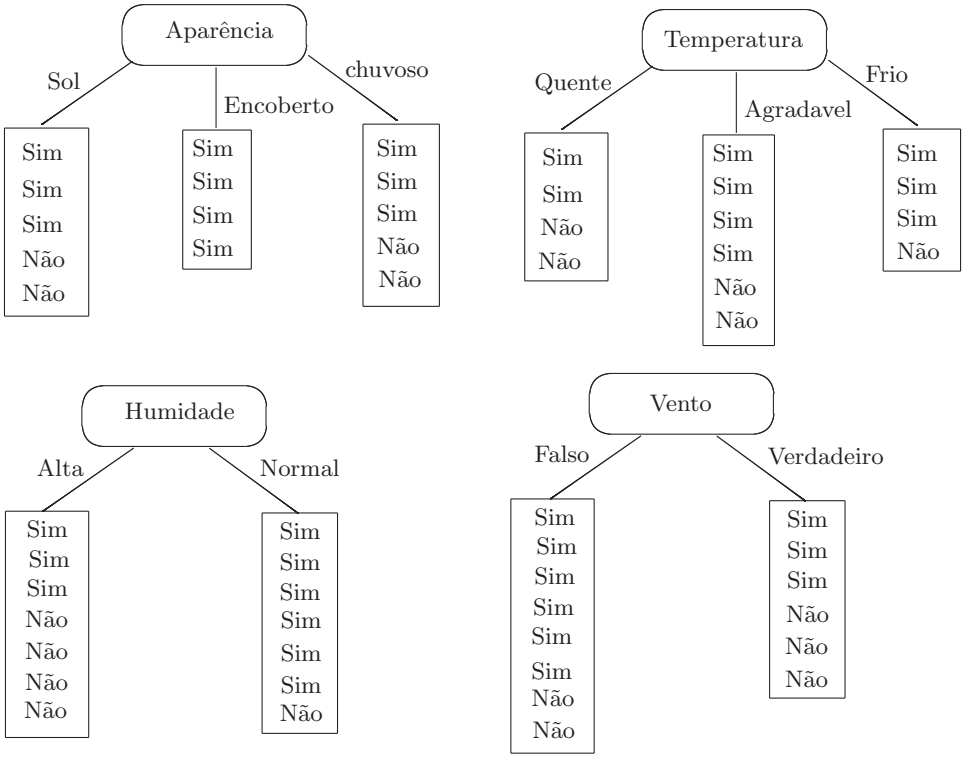


Figura 5.1: As quatro possibilidades para o atributo do nó raiz

Grau de Pureza de um atributo num nó : Entropia. Vamos definir uma função Info que calcula o *grau de pureza* de um atributo num determinado nó. Este grau de pureza representa a *a quantidade de informação esperada que seria necessária para especificar se uma nova instância seria classificada em 'Sim' ou 'Não', uma vez chegado a este nó.* A idéia é a seguinte : se A_1, A_2, ..., A_n são as folhas (tabelas) saindo deste nó, n_i = tamanho de A_i e N = total dos tamanhos das tabelas, então :

$$\mathsf{Info}(\text{Nó}) = \Sigma_{i=1}^{n} \frac{n_i}{N} \text{ Entropia}(A_i)$$

Quanto maior a entropia, maior a informação. A entropia é uma medida estatística que mede o quão "confuso" é a distribuição das tuplas entre as classes. Por exemplo, se existem 2 classes, e exatamente metade das tuplas estão numa classe e a outra metade na outra classe, então a entropia seria maximal. Por outro lado, se todas as tuplas estão numa mesma classe, então a entropia é zero.

Seja A_i uma tabela com n_i tuplas, das quais S_i estão classificadas como 'Sim' e N_i estão classificadas como 'Não'. Então a entropia de A_i é definida como :

$$\text{Entropia}(A_i) = -(\frac{S_1}{n_i} log_2 \frac{S_1}{n_i} + \frac{N_i}{n_i} log_2 \frac{N_i}{n_i})$$

Observações : esta fórmula para entropia é bem conhecida. Atente para o sinal negativo, necessário pois a entropia deve ser positiva e os logaritmos são negativos (já que são calculados sobre números entre 0 e 1). Esta fórmula é generalizada (da maneira óbvia) para um número de classes qualquer.

Exemplo 5.2 Consideremos as quatro possibilidades para o atributo do primeiro nó, conforme ilustrado na Figura 1.

- Se escolhemos o atributo Aparência :

  $$\mathsf{Info}(\text{Nó}) = \frac{5}{14} \text{ entropia(Folha 1)} + \frac{4}{14} \text{ entropia(Folha 2)} + \frac{5}{14} \text{ entropia(Folha3)}$$

  $$\text{entropia(Folha 1)} = \frac{2}{5} \, log_2 \frac{2}{5} + \frac{3}{5} \, log_2 \frac{3}{5} = 0.971$$

  $$\text{entropia(Folha 2)} = \frac{4}{4} \, log_2 \frac{5}{5} + \frac{0}{4} \, log_2 \frac{0}{4} = 0$$

  $$\text{entropia(Folha 3)} = \frac{3}{5} \, log_2 \frac{3}{5} + \frac{2}{5} \, log_2 \frac{2}{5} = 0.971$$

Logo, Info(Nó) = $\frac{5}{14}$ 0.971 + $\frac{4}{14}$ 0 + $\frac{5}{14}$ 0.971 = 0.693

- Se escolhemos o atributo Temperatura :

  Info(Nó) = $\frac{4}{14}$ entropia(Folha 1) + $\frac{6}{14}$ entropia(Folha 2) + $\frac{4}{14}$ entropia(Folha3) = 0.911

- Se escolhemos o atributo Humidade :

  Info(Nó) = $\frac{7}{14}$ entropia(Folha 1) + $\frac{7}{14}$ entropia(Folha 2) = 0.788
  item Se escolhemos o atributo Humidade :

  Info(Nó) = $\frac{8}{14}$ entropia(Folha 1) + $\frac{6}{14}$ entropia(Folha 2) = 0.892

Ganho de Informação ao escolher um Atributo. O ganho de informação ao escolher um atributo A num nó é a diferença entre a informação associada ao nó *antes*(Info-pré) da divisão e a informação associada ao nó após a divisão (Info-pós).

Info-pós = a informação do nó (Info(Nó)) que calculamos no passo anterior, ao escolhermos A como atributo divisor.

Info-pré = entropia do nó antes da divisão = $\frac{N_{Sim}}{N} log_2 \frac{N_{Sim}}{N} + \frac{N_{Nao}}{N} log_2 \frac{N_{Nao}}{N}$,

onde N_{Sim} = total de tuplas classificadas como Sim; N_{Nao} = total de tuplas classificadas como Não; N = total de tuplas no nó.

Exemplo 5.3 Consideremos a situação do exemplo 5.2. Temos que Info-pré = $\frac{9}{14} log_2 \frac{9}{14} + \frac{5}{14} log_2 \frac{5}{14} = 0.940$. Logo, os ganhos de informação de cada uma das quatro escolhas é :

ganho(Aparência) = 0.940 - 0.693 = 0.247
ganho(Temperatura) = 0.940 - 0.911 = 0.029
ganho(Humidade) = 0.940 - 0.788 = 0.152
ganho(Vento) = 0.940 - 0.892 = 0.020

Logo, o atributo ideal para dividir as amostras é o atributo Aparência, como era de se supor deste o início. Veja que é o único atributo onde uma das folhas é "arrumadinha", todas as tuplas pertencendo a uma única classe.

De posse de uma tabela de logaritmos em mãos, termine o cálculo da árvore de decisão correspondente ao banco de dados de dados meteorológicos.

5.2.3 Como transformar uma árvore de decisão em regras de classificação

Uma árvore de decisão pode ser facilmente transformada num conjunto de regras de classificação. As regras são do tipo :

IF L_1 **AND** L_2 ... **AND** L_n **THEN** Classe = Valor

onde L_i são expressões do tipo Atributo = Valor. Para cada caminho, da raiz até uma folha, tem-se uma regra de classificação. Cada par (atributo,valor) neste caminho dá origem a um L_i. Por exemplo, a árvore de decisão do exemplo 5.1 corresponde ao seguinte conjunto de regras de classificação :

- **IF** Idade =< 30 **AND** Renda = Baixa **THEN** Classe = Não
- **IF** Idade =< 30 **AND** Renda = Média **THEN** Classe = Sim
- **IF** Idade =< 30 **AND** Renda = Média-Alta **THEN** Classe = Sim
- **IF** Idade =< 30 **AND** Renda = Alta **THEN** Classe = Sim
- **IF** Idade 31..50 **THEN** Classe = Sim
- **IF** Idade 51..60 **THEN** Classe = Sim
- **IF** Idade > 60 **THEN** Classe = Não

Como exercício, considere a árvore de decisão que você calculou como exercício na seção precedente (sobre o banco de dados meteorológicos). Dê as regras de classificação associadas à árvore.

5.2.4 Discussão final

Como já dissemos anteriormente, o método de classificação baseado em árvore de decisão foi desenvolvido e refinado durante muitos anos por Ross Quinlan da Universidade de Sydney, Australia. Embora outros pesquisadores tenham investigado métodos similares, a pesquisa de Quinlan sempre esteve na dianteira do tema "Arvore de Decisão". O esquema que discutimos nesta aula, usando o critério de *ganho de informação* é essencialmente o esquema utilizado no algoritmo ID3, criado por Quinlan. Alguns melhoramentos

neste critério foram realizados ao longo dos anos e culminaram num sistema de Indução por Arvore de Decisão largamente utilizado em Classificação, chamado C4.5. Tais melhoramentos incluem métodos para se lidar com atributos numéricos com um largo espectro de valores, com atributos contendo valores desconhecidos e ruídos.

Uma boa referência para este tópico (algoritmo C4.5) é [9]

5.3 Classificação utilizando Redes Neuronais

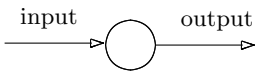
Até o momento, temos dois tipos de conceitos que podem ser produzidos em resposta a uma tarefa de classificação:

- Regras de Classificação.
- Árvore de Decisão

Uma árvore de decisão pode ser facilmente transformada num conjunto de regras de classificação e vice-versa.

Nesta aula, vamos ver um terceiro conceito que pode ser produzido em resposta a uma tarefa de classificação: uma rede neuronal. O algoritmo de classificação terá como input um banco de dados de treinamento e retornará como output uma rede neuronal. Esta rede também poderá ser transformada num conjunto de regras de classificação, como foi feito com as árvores de decisão. A única diferença é que esta transformação não é tão evidente como no caso das árvores de decisão.

As redes neuronais foram originalmente projetadas por psicólogos e neurobiologistas que procuravam desenvolver um conceito de *neurônio artificial* análogo ao neurônio natural. Intuitivamente, uma rede neuronal é um conjunto de unidades do tipo :



Tais unidades são conectadas umas às outras e cada conexão tem um *peso* associado. Cada unidade representa um *neurônio*. Os pesos associados a cada conexão entre os diversos neurônios é um número entre -1 e 1 e mede de certa forma qual a *intensidade* da conexão entre os dois neurônios. O processo de aprendizado de um certo conceito pela rede neuronal corresponde à associação de pesos adequados às diferentes conexões entre os neurônios. Por esta razão, o aprendizado utilizando Redes Neuronais também é chamado de *Aprendizado Conexionista*.

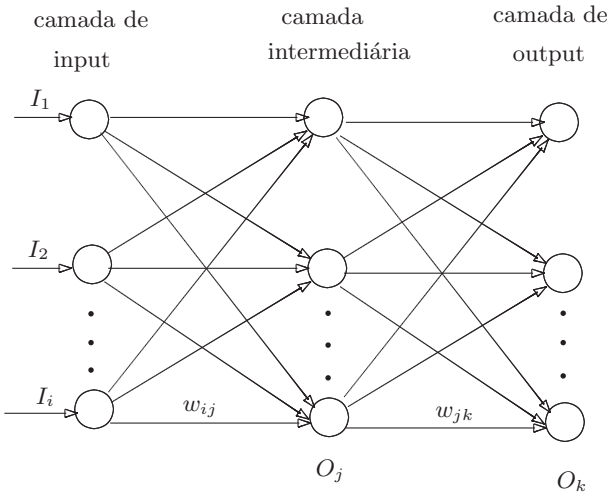


Figura 5.2: Uma rede neuronal

Mais precisamente: uma *Rede Neuronal* é um diagrama como mostra a Figura 1. A rede é composta de diversas camadas (verticais):

- **Camada de Input**: consiste dos nós da primeira coluna na Figura 1. Estes nós correspondem aos atributos (distintos do atributo classe) do banco de dados de treinamento.

- **Camada de Output**: consiste dos nós da última coluna na Figura 1. Estes nós são em número igual ao número de classes. Eles correspondem, de fato, aos possíveis valores do Atributo-Classe.

- **Camadas escondidos ou intermediários**: consiste dos nós das colunas intermediárias (indo da segunda até a penúltima). Numa rede neuronal existe pelo menos uma camada intermediária.

- **Propriedade Importante**: Cada nó de uma camada i deve estar conectado a todo nó da camada seguinte $i + 1$.

Uma rede neuronal é dita de n *camadas* se ela possui $n - 1$ camadas intermediárias. Na verdade, conta-se apenas as camadas intermediárias e a de output. Assim, se a rede tem uma camada intermediária, ela será chamada de rede de *duas camadas*, pois possui uma camada intermediário e uma de output. A camada de input não é contado, embora sempre exista.

5.3.1 Como utilizar uma rede neuronal para classificação ?

Antes de entrar nos detalhes do algoritmo de classificação, vamos descrever de forma geral, como é o processo de classificação utilizando uma rede neuronal:

- Dispõe-se de um banco de dados de treinamento composto de uma única tabela. Uma das colunas desta tabela corresponde ao Atributo-Classe. As amostras já estão classificadas. A rede será *treinada* para aprender (em cima destas amostras) como classificar corretamente novos dados.

- Para cada amostra $X = (x_1, ..., x_n)$, onde $x_1, ..., x_n$ são os valores correspondentes aos atributos não-classe, os elementos $x_1, ..., x_n$ são fornecidos a cada uma das unidades da camada de input. Cada unidade da camada de input fornece como output o mesmo x_i que recebeu como input.

- Cada unidade da próxima camada receberá como input uma combinação adequada (envolvendo os pesos das conexões) dos outputs de cada uma das unidades da camada precedente à qual está conectada. Esta unidade retornará como output o resultado da aplicação de uma certa *função de ativação* aplicada ao valor que recebeu como input.

- Este processo vai se repetindo camada por camada até chegar na última (camada de output). Seja C a classe à qual pertence a amostra X. Suponha que C corresponde à i-ésima unidade de output (lembramos que cada unidade de output corresponde a uma classe). Então o valor que deveria ter sido produzido pela rede se ela estivesse bem treinada seria (0,0,...,1,0,0,...0), onde o número 1 aparece na i-ésima coordenada. Calcula-se a diferença entre o vetor $(O_1, ..., O_n)$ (onde cada O_j é o valor de output produzido na j-ésima unidade da camada de output, para $j = 1, ..., n$) e o vetor ideal (0,0,...,1,0,0,...0) que deveria ter sido produzido se a rede estivesse treinada. Seja ϵ_j o valor da j-ésima coordenada do vetor (0,0,...,1,0,0,...0), para $j = 1, ..., n$.

- Caso a diferença $\Delta = \min \{ \mid O_j - \epsilon_j \mid, j = 1...n\}$ seja muito grande , é porque a rede ainda não está bem treinada, não aprendeu ainda a classificar uma amostra corretamente. Um processo de percurso inverso se inicia, com os pesos das conexões sendo reavaliados de acordo com uma função que depende da diferença Δ. Este processo é chamado de *Backpropagation*.

- Quando este processo de *Backpropagation* termina, estamos novamente na situação inicial, só que os pesos das conexões foram alterados. Agora, a segunda amostra do banco de dados será utilizada para o aprendizado da mesma forma como o foi a primeira.

- Quando todas as amostras do banco de dados foram escaneadas pela rede, tudo se repete a partir da primeira amostra. Cada iteração correspondendo a varrer o banco de dados de amostras uma vez é chamado de *época*.
- O algoritmo pára quando uma das condições a seguir se verifica numa determinada época:
  - Na época precedente, para cada amostra testada, todas as diferenças entre os pesos Δw_{ij} são muito pequenas, isto é, menor que um certo nível mínimo fornecido.
  - Só uma pequena porcentagem de amostras (abaixo de um nível mínimo fornecido) foram mal classificadas na época precedente.
  - Um número de épocas máximo pré-especificado já se passou.

Na prática, centenas de milhares de épocas são necessárias para que os pesos convirjam. Quando isto acontece, dizemos que a rede neuronal está *treinada*. Teoricamente, a convergência não é garantida, mas na prática em geral, ela ocorre depois de um grande número de épocas.

Antes que o processo acima descrito comece a ser executado, obviamente é preciso:

1. estabelecer como será a *topologia* da rede neuronal que será treinada para classificar corretamente as amostras. Em que consiste a *topologia* da rede ?
   - No número de camadas intermediárias,
   - Número de unidades em cada camada.
2. inicializar os parâmetros de aprendizado: os pesos entre as diferentes conexões e os parâmetros envolvidos na função que calcula o input de cada unidade j a partir dos outputs recebidos das unidades da camada precedente; além disto, é preciso especificar qual a função (de ativação) que calcula o output de cada unidade j a partir do input recebido.

5.3.2 Como definir a melhor topologia de uma rede neuronal para uma certa tarefa de classificação ?

1. **O número de unidades na camada de input**: Para cada atributo diferente do atributo classe, suponhamos que o número de valores deste atributo é n_A (é importante categorizar o domínio de cada atributo de modo que os valores assumidos sejam poucos). O número de unidades de input será igual a $n_{A_1} + ... + n_{A_k}$ onde

$A_1, ..., A_k$ são os atributos distintos do atributo classe. Por exemplo, suponhamos que Idade e RendaMensal sejam atributos não-classe e seus valores respectivos são : $\leq 30, 30..40, 40..50, \geq 50$ e {Baixa,Média,Média-Alta,Alta}. Então, teremos pelo menos 8 unidades de input, 4 para o atributo Idade (a primeira para o valor ≤ 30, a segunda para o valor 30..40, etc) e 4 para o atributo RendaMensal (a primeira para o valor 'Baixa', a segunda para o valor 'Média', etc). Uma amostra X tendo Idade = 30..40 e RendaMensal = 'Alta' vai entrar os seguintes inputs para as unidades: Unidade 1 = 1, Unidade 2 = 0, Unidade 3 = 0, Unidade 4 = 0, Unidade 5 = 0, Unidade 6 = 0, Unidade 7 = 0, Unidade 8 = 1.

Caso não seja possível categorizar os atributos, então **normaliza-se** os valores dos atributos de modo a ficarem entre 0 e 1 ou entre -1 e 1. E constrói-se tantas unidades de input quanto for o número de atributos. Para cada amostra X, os valores de seus atributos normalizados serão entrados diretamente nas unidades de input. Verifica-se na prática que a normalização dos valores dos atributos aumenta a velocidade do processo de aprendizagem.

2. **O número de unidades na camada de output**: Se existirem somente duas classes, então teremos apenas uma unidade de output (o valor 1 de output nesta unidade representa a classe 1, o valor 0 de output nesta unidade representa a classe 0). Se existirem mais de duas classes, então teremos uma unidade de output para cada classe.

3. **O número de camadas intermediárias**: Normalmente é utilizada uma única camada intermediária. Não há regra clara para determinar este número.

4. **O número de unidades nas camadas intermediárias**: Não há regra clara para determinar este número. Determinar a topologia da rede é um processo de tentativa e erro. Este número de unidades nas camadas intermediárias pode afetar o processo de aprendizagem. Uma vez que uma rede neuronal foi treinada e o grau de acertos de sua atividade de classificação não é considerado bom na fase de testes, é comum repetir todo o processo de aprendizado com uma rede neuronal com topologia diferente, isto é, com um número diferente de camadas intermediárias ou um número diferente de unidades nas camadas intermediárias.

5.3.3 Como inicializar os parâmetros da rede ?

Os pesos iniciais são inicializados por números pequenos (entre -1 e 1 ou entre -0.5 e 0.5). Outros parâmetros que determinam a função que calcula o input de cada unidade (numa camada intermediária ou de output) em função dos outputs das unidades da camada

precedente são também inicializados por valores pequenos. Tais parâmetros serão detalhados na próxima seção. Assim como a definição adequada da topologia da rede, a boa inicialização dos parâmetros da rede é um processo de tentativa e erro. Uma rede treinada que não produz resultados satisfatórios com uma base de testes pode ser re-treinada com outros valores iniciais para os pesos e demais parâmetros até que consiga bons resultados na etapa de testes.

5.4 O algoritmo de classificação por Backpropagation utilizando Redes Neuronais

Antes de descrevermos o algoritmo com detalhes, vamos discutir seus pontos cruciais:

1. ($\Longrightarrow$) **Fase de ida: Como são calculados os inputs das unidades intermediárias ?**

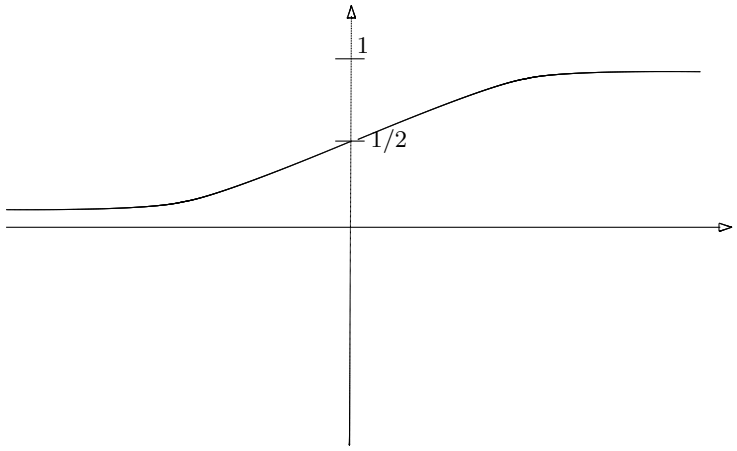
   Para calcular o input I_j na unidade j de uma certa camada intermediária, utilizamos a seguinte fórmula:

   $$I_j = (\Sigma_i w_{ij} O_i) + \theta_j$$

   onde O_i é o output de cada unidade i da camada precedente, w_{ij} é o peso da conexão entre a unidade i da camada precedente e a camada j atual, e θ_j é um parâmetro próprio da unidade j, chamado *tendência*. Trata-se de um parâmetro de ajuste ligado à unidade. Ele serve para variar a atividade da unidade.

   Depois de calculado este I_j, aplicamos uma função f, chamada *função de ativação* que tem como tarefa normalizar os valores de I_j de modo a caírem num intervalo entre 0 e 1. O resultado deste cálculo é o output O_j da unidade j. Normalmente esta função deve ser não-linear e diferenciável. Uma função que é frequentemente utilizada é a seguinte:

   $$f(x) = \frac{1}{1 + e^{-x}}$$



Um simples cálculo mostra que a derivada desta função é dada por :

$$f'(x) = f(x)(1 - f(x))$$

A Figura 2 abaixo esquematiza o cálculo do input I_j na unidade j em função dos outputs das unidades da camada precedente e o cálculo do output O_j na unidade j, aplicando a função de ativação do neurônio j em seu input I_j.

2. ($\Longleftarrow$) **Fase de volta: Como são recalculados os pesos das conexões e as tendências de cada unidade na volta para a camada de input?**

   Quando, no final da fase de ida, atinge-se a camada de output, é feito um teste para ver qual a diferença entre o output calculado em cada unidade e o valor correspondente à classe do input que está associado a esta unidade (1 ou 0). Se todas as diferenças calculadas em cada unidade de output são muito grandes (ultrapassam um certo limite fornecido), inicia-se um processo de volta para trás (Backpropagation) que vai recalcular todos os pesos e tendências em função destas diferenças que foram encontradas nas camadas de output. Isto é, a rede vai tentar "ajustar" seus pesos e tendências de modo que estes novos parâmetros reflitam o seu aprendizado sobre a amostra que acabou de ser avaliada. Como é feito este ajuste ?

   (a) Primeiramente são calculados os erros em cada unidade:

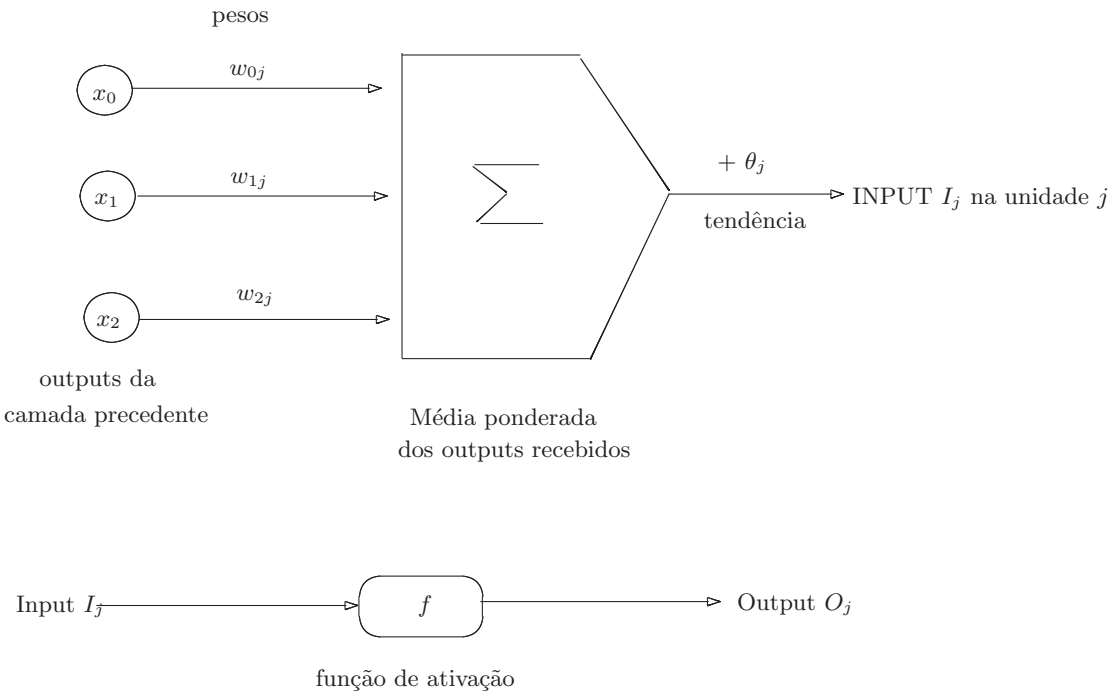


Figura 5.3: Cálculos do input e do output de uma unidade j

- Para a unidade j da camada de output é computado o erro E_j:

  $$E_j = O_j(1 - O_j)(T_j - O_j)$$

  onde O_j = valor do output da unidade j
  T_j = valor correspondente à classe da amostra X, associado à unidade j. Este valor é 1 ou 0.

  Repare que $O_j(1 - O_j)$ = derivada da função f em termos do *input* da unidade j.
- Para as unidades das camadas intermediárias: para cada unidade j de uma camada intermediária, o erro E_j é calculado como se segue:

  $$E_j = O_j(1 - O_j)(\Sigma_k E_k w_{jk})$$

  onde w_{jk} = peso da conexão ligando a unidade j à unidade k da camada seguinte, e E_k é o erro que já foi calculado nesta unidade k.

(b) Em seguida, são calculados os novos pesos de cada conexão (unidade $i \rightarrow$ unidade j)

$$\text{Novo } w_{ij} = \text{ Velho } w_{ij} + \Delta w_{ij}$$

onde $\Delta w_{ij} = \lambda E_j O_i$

A constante λ é um parâmetro do algoritmo e é chamada de *taxa de aprendizado*. É um número entre 0 e 1. Este parâmetro ajuda a evitar que o processo fique "parado" num mínimo local, isto é, num ponto onde os pesos parecem convergir (a diferença entre os pesos é muito pequena) mas que na verdade se trata de um mínimo local e não global.

(c) Em seguida são calculados as novas tendências em cada unidade:

$$\text{Nova } \theta_j = \quad \text{Velha } \theta_j + \Delta \theta_j$$

onde $\Delta \theta_j = \lambda E_j$

Vamos descrever abaixo cada passo do Algoritmo de classificação Backpropagation. O algoritmo vai varrer todas as amostras do banco de dados. Para cada amostra ele vai percorrer para frente a rede neuronal e depois para trás. Cada iteração do algoritmo corresponde a varrer todas as amostras do banco de dados.

Input: Um banco de dados de amostras, valores iniciais para os pesos, as constantex θ_i (*tendências*) para cada unidade, uma função λ que especifica a taxa de aprendizado em cada iteração t do algoritmo, limites mínimos para as condições de parada.

Output: Uma rede neuronal treinada para classificar as amostras.

1. Inicializar os pesos de cada conexão e as tendências de cada unidade. Normalmente, os valores empregados são números muito pequenos entre -1 e 1 ou entre -0.5 e 0.5. A taxa de aprendizado é normalmente especificada pela função:

$$\lambda(t) = \frac{1}{t}$$

onde t é o número correspondente à iteração em que se está. Assim, na primeira iteração, a taxa é 1, na segunda é $\frac{1}{2}$, etc.

2. **Iteração 1:** Inicia-se o processo de varrer o banco de amostras. Para cada amostra X, entra-se os valores dos atributos de X na camada de input. Propaga-se os inputs para a frente na rede neuronal até chegar na camada de output final.

3. Chegando na camada final, faz-se um teste para ver se X foi ou não bem classificada de acordo com o limite mínimo de exigência estabelecido no input do algoritmo.

4. Independentemente do resultado do teste, inicia-se o processo de volta recalculando todos os pesos das conexões e tendências das unidades até chegar na camada de input.

5. Existem duas possibilidades para os updates dos pesos e tendências:

   - Estes são atualizados a cada passada por uma amostra.
   - Para cada amostra os novos pesos e tendências são estocados em variáveis, mas não são atualizados. Assim, a próxima amostra utiliza os mesmos pesos e tendências utilizados na amostra precedente. A atualização só ocorre no final da iteração, quando todas as amostras foram varridas. Assim, na próxima iteração, para cada amostra utiliza-se os novos pesos e tendências que foram estocados para esta amostra na iteração precedente.

   Normalmente, utiliza-se a primeira política de atualização, que produz melhores resultados.

   A iteração 1 termina quando todo o banco de dados de amostras foi varrido.

6. **Iteração** k **(**$k > 1$ **)** : o processo se repete, isto é, o banco de dados é varrido novamente com os novos pesos e tendências atualizados para cada amostra.

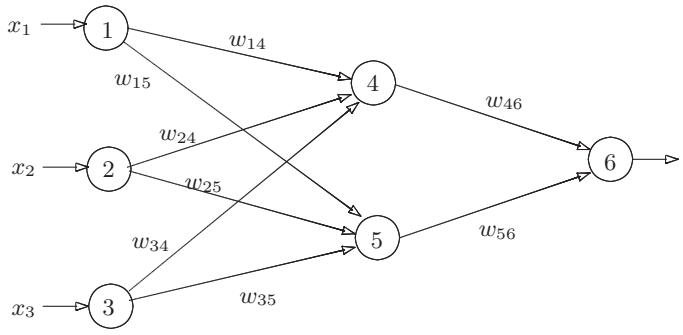
7. **Condições de Parada**: como já dissemos antes, o algoritmo pára numa certa iteração k quando uma das condições abaixo se verifica (em ordem de prioridade):

   (a) Os Δ_{ij} calculados na iteração $k-1$ são todos muito pequenos, abaixo de um limite mínimo especificado no input do algoritmo.

   (b) A porcentagem de amostras mal-classificadas na iteração precedente está abaixo de um limite mínimo especificado no input do algoritmo.

   (c) k é maior do que um limite máximo especificado no input do algoritmo.

Cada iteração do algoritmo corresponde a uma *época*. Como dissemos anteriormente, em geral são necessárias centenas de milhares de épocas até que os pesos e tendências convirjam.



5.4.1 Um exemplo

Vamos dar um exemplo das fases de ida e volta sobre uma amostra X = $(1, 0, 1)$ que é classificada na classe 1 (só temos duas classes possíveis). Suponhamos neste exemplo que temos uma unidade de input para cada um dos 3 atributos. Suporemos que a topologia da rede é a seguinte: uma única camada intermediária, com duas unidades. A Figura 3 e a tabela que vem logo abaixo ilustra a topologia da rede e os valores dos pesos e tendências iniciais:

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Fase de Ida: a tabela abaixo mostra os cálculos do input recebido por cada unidade das camadas intermediárias e final:

Unidade j	Input I_j	Output O_j
4	0.2 + 0 -0.5 - 0.4 = -0.7	$\frac{1}{1+e^{0.7}} = 0.332$
5	-0.3 + 0 0.2 + 0.2 = 0.1	$\frac{1}{1+e^{-0.1}} = 0.525$
6	(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105	$\frac{1}{1+e^{0.105}} = 0.474$

Fase de Volta: Ao chegar à unidade correspondente à classe 1 na camada final (unidade 6), é calculada a diferença (1 - 0.474) = 0.526. A fase de volta é iniciada com o cálculo dos erros e novos pesos e tendências. A tabela abaixo contém o cálculo dos erros para cada unidade:

Unidade j	E_j
6	(0.474) (1 - 0.474) (1 - 0.474) = 0.1311
5	(0.525) (1 - 0.525) (0.1311)(-0.2) = -0.0065
4	(0.332) (1 - 0.332) (0.1311)(-0.3) = -0.0087

A tabela abaixo contém os novos pesos e tendências. Supomos que a taxa de aprendizado nesta iteração é $\lambda = 0.9$.

Valor antigo	Novo Valor
w_{46}	-0.3 + (0.90)(1.3311)(0.332) = -0.261
w_{56}	-0.2 + (0.90)(1.3311)(0.525) = -1.138
w_{14}	0.2 + (0.90)(-0.0087)(1) = 0.192
w_{15}	-0.3 + (0.90)(-0.0065)(1) = -0.306
w_{24}	0.4 + (0.90)(-0.0087)(0) = 0.4
w_{25}	0.1 + (0.90)(-0.0065)(0) = 0.1
w_{34}	-0.5 + (0.90)(-0.0087)(1) = -0.508
w_{35}	0.2 + (0.90)(-0.0065)(1) = 0.194
θ_6	0.1 + (0.90)(1.1311) = 0.218
θ_5	0.2 + (0.90)(-0.0065) = 0.194
θ_4	-0.4 + (0.90) (-0.0087) = -0.408

5.4.2 Discussão Final: vantagens e desvantagens do método

O algoritmo de Backpropagation que estudamos nesta aula foi apresentado pela primeira vez no artigo [10]. De lá para cá, muitas variações deste algoritmo foram propostas, envolvendo por exemplo outras funções para cálculo de erros, ajustes dinâmicos da topologia da rede e ajustes dinâmicos da taxa de aprendizado. Muitos livros sobre "Aprendizado de Máquina" fornecem uma boa explicação deste algoritmo. Por exemplo [11].

Classificação utilizando Backpropagation envolve um tempo muito grande na fase de treinamento. O processo requer a especificação de um número de parâmetros que é tipicamente determinada de forma empírica, como por exemplo, a topologia da rede, a taxa de aprendizado e a função de ativação.

A crítica maior que se faz com relação e esta técnica é a sua interpretabilidade, isto é, não é óbvio para um usuário interpretar o resultado do algoritmo, a saber, uma rede neuronal treinada como sendo um instrumento de classificação (as regras de classificação correspondentes a uma rede treinada não são óbvias como acontece com as árvores de decisão). Além disto, é difícil para os humanos interpretar o significado simbólico que está por trás dos pesos das conexões da rede treinada.

As vantagens das redes neuronais, incluem sua alta tolerância a dados contendo ruídos assim como a grande *accuracy* (corretude) dos resultados. Quanto à desvantagem mencionada acima: atualmente diversos algoritmos já foram desenvolvidos para se extrair regras de classificação de uma rede treinada, o que minimiza de certa forma esta desvantagem do método. Na próxima aula veremos um destes algoritmos para extração de

regras de classificação de uma rede neuronal treinada. A referência para este algoritmo bem como para uma discussão mais ampla das vantagens e desvantagens de se utilizar redes neuronais em mineração de dados pode ser encontrada no artigo [12].

O artigo [13] traz uma interessante discussão sobre o aprendizado do cérebro humano e o aprendizado artificial de uma rede neuronal.

5.5 Redes Neuronais : Poda e Geração de Regras de Classificação

Nestas notas de aula, vamos apresentar um método de podagem de ramos de uma rede neuronal que permite aumentar a eficiência do processo de aprendizagem sem comprometer os resultados obtidos. Vamos apresentar também um algoritmo para extração de regras de classificação a partir de uma rede neuronal treinada. Como salientamos na aula passada, um dos aspectos negativos das redes neuronais como métodos classificatórios é sua baixa interpretabilidade, isto é, não é imediato para o usuário interpretar os pesos e tendências dos nós da rede treinada como elementos representativos do conhecimento aprendido pela rede. A bibliografia básica para esta seção é o artigo [12].

5.5.1 Poda de uma Rede Neuronal

No final da execução do algoritmo de Backpropagation obtemos uma rede neuronal totalmente conectada. Existem normalmente um número muito grande de links na rede. Com n nós na camada de input, h nós na camada intermediária (supondo uma rede com somente uma camada intermediária) e m nós na camada de output, teremos $h * (m + n)$ links. É muito difícil de manipular uma rede com tantos links. A fase de poda da rede neuronal tem como objetivo remover alguns links sem afetar a capacidade de classificação da rede.

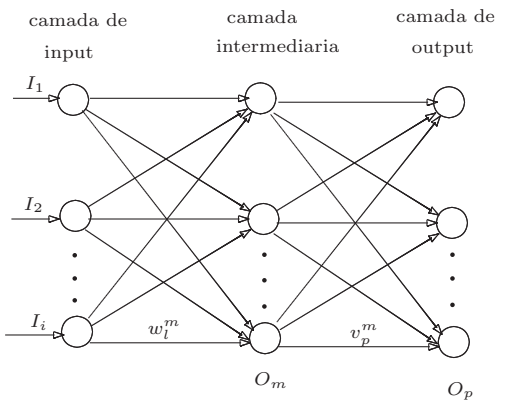
Consideremos, para simplificar a apresentação, uma rede neuronal com apenas uma camada intermediária. Vamos utilizar a notação ilustrada na figura 1 abaixo para nomear os diversos links.

Estamos supondo que a camada de output tem m nós, cada um dos quais representa uma classe A_p. Seja $X = (x_1, x_2, \ldots, x_k)$ uma tupla do banco de dados de amostras. A condição para que esta tupla seja considerada bem classificada é que:

$$max_p \mid S_p - t_p \mid \leq \eta_1$$

onde: $p \in \{1, ..., m\}$, onde m é o número de nós da camada de output

- $p \in \{1, ..., m\}$, onde m é o número de nós da camada de output,



w_l^m = link ligando nó l de input ao nó m intermediário

v_p^m = link ligando nó m intermediário ao nó p de output

Figura 5.4: Labels de uma rede neuronal treinada

- S_p é o resultado que é produzido no p-ésimo nó da camada de output,
- $t_p = 1$ caso X pertença à classe A_p e $t_p = 0$, caso contrário.
- η_1 é uma constante ≤ 0.5 denominada *coeficiente de classificação* da rede.

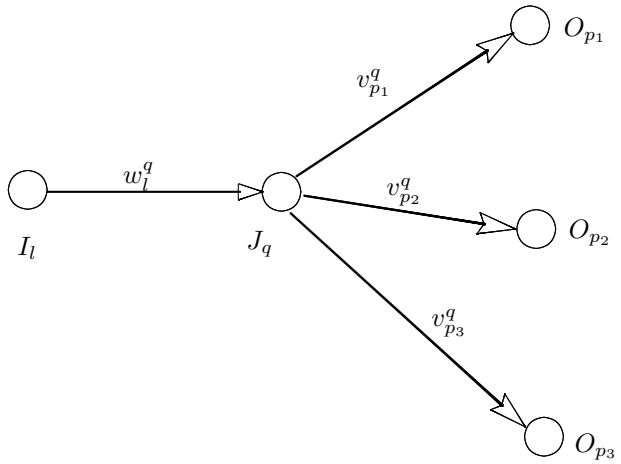
Poda de Links ligando as camadas de input e intermediária

É possível mostrar que, se a condição abaixo for verificada:

$$max_p \mid v_p^q w_l^q \mid \leq 4\eta_2$$

onde $\eta_2 < \eta_1 - 0.5$, então o link com label w_l^q pode ser eliminado sem prejuízo algum para a capacidade de classificação da rede. Isto é, uma tupla X que antes era bem classificada (de acordo com a condição acima) continua sendo bem classificada.

Repare que q e l estão fixos, q corresponde ao q-ésimo nó J_q da camada intermediária e l corresponde ao l-ésimo nó I_l da camada de input, w_l^q é o link entre estes dois nós. A condição acima exprime que cada um dos produtos envolvendo este peso w_l^q e o peso v_p^q de um link saindo do nó J_q não pode ser maior do que $4\eta_2$.



Poda de Links ligando as camadas de intermediária e de output

É possível mostrar que, se a condição abaixo for verificada:

$$\mid v^q_m \mid \leq 4\eta_2$$

então o link com label v^q_m ligando o q-ésimo nó da camada intermediária e o m-ésimo nó da camada de output, pode ser removido sem nenhum prezuízo para a rede neuronal. A demonstração detalhada deste resultado pode ser encontrada em [14][1]

O algoritmo de poda de uma rede neural treinada é o seguinte:

1. Sejam η_1 e η_2 dois números inteiros positivos tais que $\eta_1 + \eta_2 < 0.5$, onde η_1 é o coeficiente de classificação utilizado no treinamento da rede neuronal.
2. Para cada w^q_l, se

   $$(*) \quad max_p \mid v^q_p w^q_l \mid \leq 4\eta_2$$

   então remova o link w^q_l da rede.
3. Para cada v^q_m, se

   $$\mid v^q_m \mid \leq 4\eta_2$$

   então remova o link v^q_m da rede.

[1]No artigo [**?**], página 481 e página 482, este resultado é citado, mas com um erro: lá, a condição é $max_p \mid v^p_m \mid \leq 4\eta_2$, o que não tem sentido, já que p e m estão fixos !

4. Se nenhum peso satisfaz as condições acima, então remova o link w_l^q onde o número $max_p \mid v_p^q w_l^q \mid$ é o menor possível.

5. Treine novamente a rede, com os pesos obtidos no treinamento passado, com os links podados conforme as regras acima. Se a corretude dos resultados obtidos fica abaixo de um nível dado, então páre (significa que a poda não é possível sem que a corretude da rede se altere). Caso contrário (isto é, se a corretude da rede é aceitável), continue podando: vá para o passo 2 e repita todo o processo.

5.5.2 Extração de Regras de uma Rede Neuronal Treinada

O algoritmo para extrair regras de classificação de uma rede neuronal é o seguinte:

1. Poda-se a rede neuronal utilizando o algoritmo da seção anterior.

2. Discretiza-se os valores de output da camada intermediária utilizando o procedimento **Discret** dado abaixo.

3. Enumera-se todas as combinações possíveis dos valores discretizados encontrados acima que podem assumir os nós da camada intermediária. Para cada uma destas combinações, calcula-se os valores de output da camada de output.

4. Gera-se as regras "cobrindo" todos os nós intermediários e de output.

5. Para cada combinação dos valores discretizados da camada intermediária, calcula-se os valores da camada de input que levam a este output na camada intermediária.

6. Gera-se as regras "cobrindo" todos os nós de input e intermediários.

7. Combina-se os dois conjuntos de regras obtidos acima e gera-se as regras "cobrindo" todos os nós de input e de output.

Procedimento **Discret**
input : um banco de dados de amostras classificadas.

output: um conjunto de clusters que agrupam os possíveis outputs de cada nó da camada intermediária.

1. Fixe um número ϵ pequeno, $0 < \epsilon < 1$.

2. Para cada nó N, faça:

   (a) Seja X_1 a primeira tupla do banco de dados. Seja δ_1 = output do nó N quando X_1 é entrada como input da rede.

(b) Seja δ_2 o output do nó N para a segunda tupla X_2. Caso $\mid \delta_1 - \delta_2 \mid \leq \epsilon$ então δ_2 fará parte do cluster cujo representante é δ_1. Caso contrário, cria-se um outro cluster, cujo representante é δ_2.

(c) Considera-se a terceira tupla X_3 e δ_3 o output do nó N quando X_3 é entrada como input da rede. Dependendo se $\mid \delta_3 - \delta_2 \mid \leq \epsilon$ ou e $\mid \delta_3 - \delta_1 \mid \leq \epsilon$, coloco δ_3 no cluster de δ_1 ou δ_2 respectivamente. Em caso de empate, escolho um dos clusters para colocar δ_3. Caso nenhuma das duas desigualdades é satisfeita, crio um terceiro cluster para δ_3.

(d) O procedimento acima se repete até todas as tuplas do banco de dados terem sido consideradas.

(e) No final do processo, associa-se a cada cluster a média dos valores dos δ_i pertencentes ao cluster. Esta média será o representante do cluster.

3. Testa-se a qualidade dos resultados de classificação da rede com estes valores de ativação discretizados da camada intermediária. Se forem bons, retorna estes valores. Senão, repete-se todo o processo a partir de (1) considerando um novo ϵ menor do que o utilizado na fase anterior.

O seguinte exemplo ilustra o algoritmo de extração de regras de classificação que descrevemos informalmente acima.

Exemplo de uso

Consideremos a rede neuronal podada da Figura 2 (retirada do artigo [**?**]) resultante de um treinamento sobre um banco de dados amostral de 1000 tuplas, contendo 9 atributos. Os atributos com valores contínuos foram discretizados de modo que no final obteve-se 87 nós na camada de input (por exemplo, o primeiro atributo deu origem a 6 nós, uma para cada uma das 6 faixas de valores deste atributo). Assim sendo, cada tupla do banco de dados é transformada numa tupla de 87 coordenadas, tomando valores no conjunto {0,1}. Após a poda, só sobraram os nós I_1, I_3, I_4, I_5, I_{13}, I_{15}, I_{17}.

1. **Discretização dos valores da camada intermediária**.
   A camada intermediária deste exemplo tem 3 nós. O valorde ϵ inicial foi de 0.6. Para este ϵ, os clusters obtidos em cada nó foram os seguintes:

Nó	N. de Clusters	Representantes
1	3	(-1,0,1)
2	2	(0, 1)
3	3	(-1, 0.24, 1)

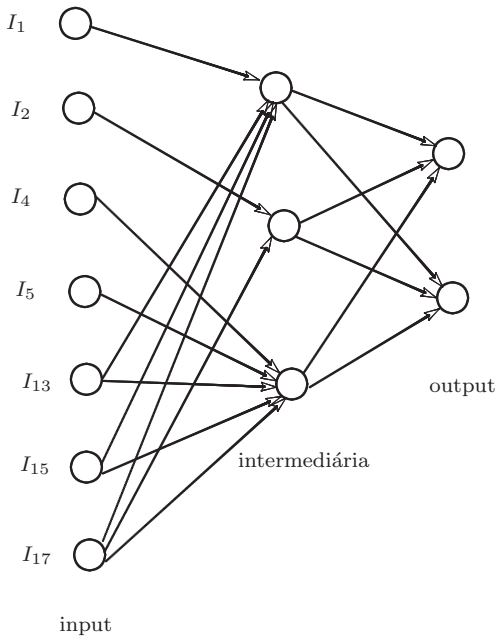


Figura 5.5: Rede Neural Podada

Assim, os valores de output do nó 1 estão agrupados em 3 clusters: o primeiro, representado por -1, o segundo representado por 0 e o terceiro por 1.

A capacidade de classificação da rede, considerando estes valores discretizados para a camada de output, foi testada. O valor de $\epsilon = 0.6$ foi suficientemente pequeno para que a capacidade de classificação não fosse muito alterada, e suficientemente grande para não produzir muitos clusters.

2. **Enumeração das combinações possíveis dos valores discretizados**
   Na tabela abaixo, enumeramos todas as combinações dos valores de cada nó intermediário e para cada uma das combinações calculamos o valor de output de cada um dos nós da camada de output.

	α_1	α_1	α_1	C_1	C_2
1	-1	1	-1	0.92	0.08
2	-1	1	0.24	0.00	1.00
3	-1	1	1	0.01	0.99
4	-1	0	-1	1.00	0.00
5	-1	0	0.24	0.11	0.89
6	-1	0	1	0.93	0.087
7	1	1	-1	0.00	1.00
8	1	1	0.24	0.00	1.00
9	1	1	1	0.00	1.00
10	1	0	-1	0.89	0.11
11	0	0.24	0.00	1.00	
12	1	0	1	0.00	1.00
13	0	1	-1	0.18	0.82
14	0	1	0.24	0.00	1.00
15	0	1	1	0.00	1.00
16	0	0	-1	1.00	0.00
17	0	0	0.24	0.00	1.00
18	0	0	1	0.18	0.82

3. **Geração das regras "cobrindo" todos os nós intermediários e de output.**

Temos cinco possibilidades para as quais $C_1 = 1$ e $C_2 = 0$: linha 1, linha 4, linha 6, linha 10, linha 16.

As linhas 4, 10 e 16 dizem que não importa o valor de α_1, uma vez que $\alpha_2 = 0$ e $\alpha_3 = -1$, teremos sempre $C_1 = 1$ e $C_2 = 0$. Isto nos dá a regra :

R_{11} : Se $\alpha_2 = 0$ e $\alpha_3 = -1$ então $C_1 = 1$ e $C_2 = 0$.

As outras 2 linhas: linha 1 e linha 6 produzem mais duas regras com o mesmo consequente:

R_{12} : Se $\alpha_1 = -1$ e $\alpha_2 = 1$ e $\alpha_3 = -1$ então $C_1 = 1$ e $C_2 = 0$.

R_{13} : Se $\alpha_1 = -1$ e $\alpha_2 = 0$ e $\alpha_3 = 0.24$ então $C_1 = 1$ e $C_2 = 0$.

As outras 13 linhas produzem 13 regras onde o consequente é $C_1 = 0$ e $C_2 = 1$.

Exercício : Dê uma única regra que seja equivalente às 13 regras acima.

4. **Cálculo dos valores da camada de input correspondendo a cada combinação de valores da camada intermediária**
   Repare que nas regras que obtivemos no item anterior, os valores que interessam para a camada intermediária são :

   $\alpha_1 = 1$, $\alpha_2 = 0$, $\alpha_2 = 1$, α_3 = -1 e $\alpha_3 = 0.24$.

   Fazendo o cálculo inverso da função que produz os outputs da camada intermediária em termos dos valores dos inputs nos nós da camada inicial (na rede podada), obtemos o seguinte:

   - inputs correspondentes ao valor 1 do nó intermediário 1: $I_{13} = 1$ ou ($I_1 = I_{13} = I_{15} = 0$) e $I_{17} = 1$.
   - inputs correspondentes ao valor 1 do nó intermediário 2: $I_2 = 1$ ou $I_{17} = 1$.
   - inputs correspondentes ao valor 0 do nó intermediário 2: $I_2 = I_{17} = 0$.
   - inputs correspondentes ao valor -1 do nó intermediário 3: $I_{13} = 0$ ou $I_{15} = I_5 = 1$.
   - inputs correspondentes ao valor 0.24 do nó intermediário 3: ($I_4 = I_{13} = 1$ e $I_{17} = 0$) ou $I_5 = I_{13} = I_{15} = 1$.

5. **Geração das regras "cobrindo" todos os nós intermediários e de input.**

   A partir do item anterior as regras cobrindo os nós intermediários e de input são óbvias. Por exemplo, a regra :

   $$R_{22} : I_1 = I_{13} = I_{15} = 0 \text{ e } I_{17} = 1 \rightarrow \alpha_1 = -1$$

   Exercício : Deduza todas as outras regras.

6. **Combinação dos dois conjuntos de regras obtidos acima para obter as regras "cobrindo" todos os nós de input e de output**

Por exemplo, sabemos que :

R_{11} : Se $\alpha_2 = 0$ e $\alpha_3 = -1$ então $C_1 = 1$ e $C_2 = 0$.

Por outro lado, sabemos que temos a seguinte regra envolvendo os nós de input e os nós intermediários:

Se $I_2 = I_{17} = 0$ então $\alpha_2 = 0$ e se ($I_{13} = 0$ ou $I_{15} = I_5 = 1$) então $\alpha_3 = -1$. Logo, combinando estas regras temos :

Se $I_2 = I_{17} = I_{13} = 0$ então $C_1 = 1$ e $C_2 = 0$.
Se $I_2 = I_{17} = 0$ e $I_{15} = I_5 = 1$ então $C_1 = 1$ e $C_2 = 0$.

Exercício : Deduza todas as regras de classificação produzidos pela rede treinada, seguindo a idéia do exemplo acima.

5.6 Classificadores Bayseanos

Classificadores Bayseanos são classificadores estatísticos que classificam um objeto numa determinada classe baseando-se na probabilidade deste objeto pertencer a esta classe. Produz resultados rapidamente, de grande correção quando aplicados a grandes volumes de dados, comparáveis aos resultados produzidos por árvores de decisão e redes neurais.

5.6.1 Classificadores Bayseanos Simples

Os classificadores Bayseanos Simples supõem como hipótese de trabalho que o efeito do valor de um atributo não-classe é independente dos valores dos outros atributos. Isto é, o valor de um atributo não influencia o valor dos outros. Esta hipótese tem como objetivo facilitar os cálculos envolvidos na tarefa de classificação.

Por exemplo, suponhamos os atributos Idade, Profissão, Renda. É claro que estes atributos não são independentes uns dos outros. Uma pessoa com profissão Médico tem maior probabilidade de ter uma renda alta do que um porteiro. Uma pessoa com idade superior a 40 tem maior probabilidade de ter uma renda alta do que alguém com menos de 25 anos. Assim, vemos que os valores dos atributos dependem uns dos outros. Por outro lado, é claro que os atributos Gênero, Cidade, Idade são independentes uns dos outros.

Um *Classificador Bayseano Ingênuo ou Simples* funciona da seguinte maneira:

Consideramos um banco de dados de amostras classificadas em m classes distintas $C_1, C_2, \ldots, C_m$.

Suponha que X é uma tupla a ser classificada (não está no banco de dados de amostras). O classificador vai classificar X numa classe C para a qual a probabilidade condicional $P[C|X]$ é a mais alta possível. Repare que os valores dos atributos de X podem ser encarados como um *evento conjunto*. Assim, se os atributos do banco de dados são Idade, Profissão e Renda e $X = (30..40, Professor, Alta)$, então X pode ser vista como o evento Idade = 30..40, Profissão = Professor e Renda = Alta. X será classificada na classe C se a probabilidade condicional de C acontecer dado que X acontece é maior do que a probabilidade de qualquer outra classe C' acontecer dado que X acontece.

Assim, a tupla X será classificada na classe C_i se

$$P[C_i|X] > P[C_j|X]$$

para todas as outras classes C_j, $C_j \neq C_i$. Esta probabilidade $P[C_i|X]$ também é chamada *probabilidade posterior*.

Como calcular as probabilidades posteriores

O Teorema de Bayes fornece uma maneira de calcular $P[C_i|X]$. Sabemos que :

$$P[X \cap C] = P[X|C] * P[C] = P[C|X] * P[X]$$

Logo:

$$P[C|X] = \frac{P[X|C] * P[C]}{P[X]} \qquad \text{(Teorema de Bayes)}$$

Como $P[X]$ é uma constante (pois X está fixo), a fim de maximizar $P[C|X]$ precisamos maximizar o numerador $P[X|C] * P[C]$. Se as probabilidades $P[C]$ não estão disponíveis para cada classe C, supõe-se que elas são idênticas, isto é, $P[C] = \frac{1}{m}$. Em seguida, resta somente maximizar $P[X|C]$. Caso as probabilidades $P[C]$ sejam conhecidas, maximiza-se o produto $P[X|C] * P[C]$.

Como é calculada a probabilidade condicional $P[X|C]$, também chamada *probabilidade a priori* ? Suponha que $X = (x_1, x_2, \ldots, x_k)$. X representa o evento conjunto $x_1 \cap x_2 \cap \ldots \cap x_k$. Logo,

$$P[X|C] = P[x_1 \cap x_2 \cap \ldots \cap x_k|C]$$

Supondo a hipótese da *independência dos atributos* discutida acima, temos que:

$$P[x_1 \cap x_2 \cap \ldots \cap x_k|C] = P[x_1|C] * P[x_2|C] * \ldots * P[x_k|C]$$

As probabilidades $P[x_i|C]$ podem ser calculadas a partir da base de amostras da seguinte maneira:

- Se o atributo A_i é categórico:

$$P[x_i|C] = \frac{\text{nº de tuplas classificadas em } C \text{ com atributo } A_i = x_i}{\text{nº de tuplas classificadas em } C}$$

- Se o atributo A_i é contínuo (não-categórico):

$$P[x_i|C] = g(x_i, \mu_C, \sigma_C) = \text{função de distribuição de Gauss}$$

onde μ_C = média, σ_C = desvio padrão.

A distribuição de Gauss é dada por :

$$g(x_i, \mu_C, \sigma_C) = \frac{1}{\sqrt{2\Pi} * \sigma_C} e^{-}(\frac{x_i - \mu_C)^2}{2\sigma_C^2}$$

Exemplo de uso

Consideremos o seguinte banco de dados (o atributo classe é Compra-Computador):

ID	Idade	Renda	Estudante	Crédito	Compra-Computador
1	≤ 30	Alta	não	bom	não
2	≤ 30	Alta	não	bom	não
3	31..40	Alta	não	bom	sim
4	> 40	Média	não	bom	sim
5	> 40	Baixa	sim	bom	sim
6	> 40	Baixa	sim	excelente	não
7	31..40	Baixa	sim	excelente	sim
8	≤ 30	Média	não	bom	não
9	≤ 30	Baixa	sim	bom	sim
10	> 40	Média	sim	bom	sim
11	≤ 30	Média	sim	excelente	sim
12	31..40	Média	não	excelente	sim
13	31..40	Alta	sim	bom	sim
14	> 40	Média	não	excelente	não

A classe C_1 corresponde a Compra-Computador = 'sim' e a classe C_2 corresponde a Compra-Computador = 'não'. A tupla desconhecida que queremos classificar é :

$$X = (Idade =\leq 30, Renda = Media, Estudante = sim, Credito = bom)$$

Precisamos maximizar $P[X|C_i]P[C_i]$ para $i = 1, 2$. As probabilidades $P[C_i]$ podem ser calculadas baseando-se no banco de dados de amostras:

$$P[C_1] = \frac{9}{14} = 0.643$$

$$P[C_2] = \frac{5}{14} = 0.357$$

Para calcular $P[X|C_i]$, para $i = 1, 2$, calculamos as seguintes probabilidades:

$$\begin{aligned}
P[Idade \leq 30|CompraComp = sim] &= \frac{2}{9} = 0.222 \\
P[Idade \leq 30|CompraComp = nao] &= \frac{3}{5} = 0.6 \\
P[Renda = Media|CompraComp = sim] &= \frac{4}{9} = 0.444 \\
P[Renda = Media|CompraComp = nao] &= \frac{2}{5} = 0.4 \\
P[Estudante = sim|CompraComp = sim] &= \frac{6}{9} = 0.667 \\
P[Estudante = sim|CompraComp = nao] &= \frac{1}{5} = 0.2 \\
P[Credito = bom|CompraComp = sim] &= \frac{6}{9} = 0.667 \\
P[Credito = bom|CompraComp = nao] &= \frac{2}{5} = 0.4
\end{aligned}$$

Utilizando as probabilidades acima, temos:

$$\begin{aligned}
P[X|CompraComp = sim] &= 0.222 * 0.444 * 0.667 * 0.667 = 0.044 \\
P[X|CompraComp = nao] &= 0.6 * 0.4 * 0.2 * 0.4 = 0.019 \\
P[X|CompraComp = sim] * P[CompraComp = sim] &= 0.044 * 0.643 = 0.028 \\
P[X|CompraComp = nao] * P[CompraComp = nao] &= 0.019 * 0.357 = 0.007
\end{aligned}$$

Desta maneira, o classificador Bayseano prediz que a tupla X é classificada na classe Compra-Computador = 'sim'.

5.6.2 Redes Bayseanas de Crença

Quando a hipótese da independência entre os atributos se verifica, então o classificador Bayseano simples (ou ingênuo) é o que tem a melhor performance em termos de resultados corretos, com relação a outros classificadores. Entretanto, na prática, é comum existir dependência entre os atributos. Neste caso, utilizamos uma *Rede Bayseana de Crença* como método classificador.

O que é uma Rede Bayseana de Crença ?

Uma Rede Bayseana de Crença é uma estrutura com duas componentes:

1. Um grafo dirigido acíclico onde cada vértice representa um atributo e os arcos ligando os vértices representam uma dependência entre estes atributos:

z depende de y

y = pai

z = filho

2. A segunda componente definindo uma Rede Bayseana de Crença consiste de uma *Tabela de Probabilidade Condicional* (CPT) **para cada atributo** Z. A probabilidade condicional associada ao atributo Z é a probabilidade $P[Z|pais(Z)]$, onde $pais(Z)$ é o conjunto dos atributos que são pais de Z. A tabela CPT para um atributo Z, tem o seguinte formato : as linhas correspondem aos possíveis valores de Z, as colunas correspondem às combinações de valores possíveis dos pais(Z). Na linha i, coluna j, temos a probabilidade condicional de Z ter o valor da linha i e seus pais terem os valores especificados na coluna j.

Classificação utilizando Rede Bayseana de Crença

O processo de classificação utilizando Redes Bayseanas de Crença tem como **input** um banco de dados de amostras e uma Rede Bayseana (com seus dois componentes especificados acima). Um dos vértices da Rede Bayseana é selecionado como sendo o

atributo classe. Podem existir diversos atributos classe numa Rede Bayseana. Caso só tenhamos um único atributo classe assumindo os valores $C_1, ..., C_m$, o **output** será a distribuição de probabilidade $P[C_1|X], ..., P[C_m|X]$. Caso existam diversos atributos classe, por exemplo $Classe_1$ e $Classe_2$, assumindo os valores $C_1^1, ..., C_m^1$ (para a $Classe_1$) e $C_1^2, ..., C_l^2$ (para a $Classe_2$), o algoritmo retornará a a distribuição de probabilidade $P[C_1^1|X], ..., P[C_m^1|X]$,$P[C_1^2|X], ..., P[C_l^2|X]$.

O processo de classificação é análogo ao classificador Bayseano simples: procura-se maximizar a probabilidade condicional posterior $P[C_i|X]$. Segundo o Teorema de Bayes:

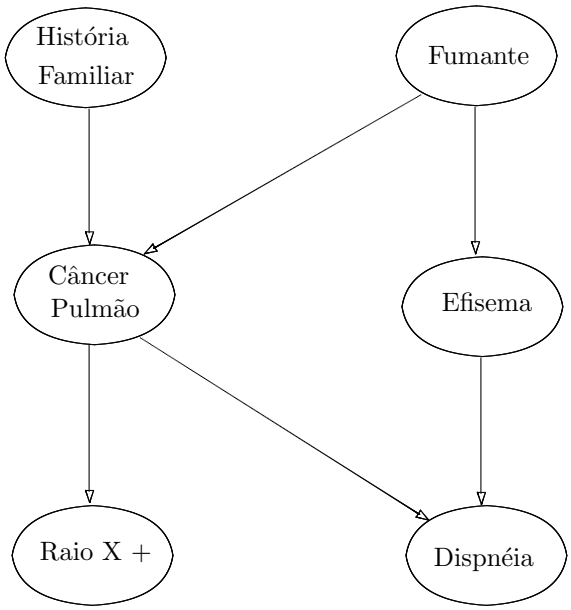
$$P[C_i|X] = \frac{P[X|C_i] * P[C_i]}{P[X]}$$

Logo, precisamos maximizar $P[X|C_i] * P[C_i]$, já que $P[X]$ é fixo. Agora, não podemos mais utilizar a hipótese simplicadora da independência dos atributos no cálculo de $P[X|C_i]$. O produto $P[X|C_i] * P[C_i]$ é a probabilidade do evento conjunto $X \cap C_i$. Supondo $X = (x_1, ..., x_k)$ (como já dissemos, a tupla X é vista como um evento conjunto $x_1 \cap x_2 \cap ...x_k$) temos que :

$$P[X|C_i]*P[C_i] = P[X \cap C_i] = P[x_1|pais(x_1)]*P[x_2|pais(x_2)]*...*P[x_k|pais(x_k)]*P[C_i|pais(C_i)]$$

Exemplo de uso

Ilustraremos o processo de classificação utilizando Redes Bayseanas de Crença através de um exemplo. Consideremos a Rede Bayseana cuja primeira componente é o seguinte grafo:



O único atributo classe é CancerPulmão. A tabela CPT para este atributo é a seguinte:

	HF=1,F=1	HF=1,F=0	HF=0,F=1	HF=0,F=0
CP = 1	0.8	0.5	0.7	0.1
CP = 0	0.2	0.5	0.3	0.9

A tabela CPT para o atributo Efisema é :

	F=1	F=0
E = 1	0.03	0.2
E = 0	0.97	0.8

A tabela CPT para o atributo RaioX+ é :

	CP=1	CP=0
RX+ = 1	0.9	0.02
RX+ = 0	0.1	0.98

A tabela CPT para o atributo Dispnéia é :

	E=1,CP=1	E=1,CP=0	E=0,CP=1	E=0,CP=0
D = 1	0.99	0.2	0.3	0.01
D = 0	0.01	0.8	0.7	0.99

Estamos utilizando as seguintes abreviações: CP = CâncerPulmão, HF = História Familiar e F = Fumante, D = Dispnéia, RX+ = RaioX+, Efisema = E.

Suponhamos a seguinte tupla $X = (HF = 1, F = 1, E = 0, RaioX+ = 1, Dispneia = 0)$ a ser classificada. Queremos maximizar $P[X|CP] * P[CP]$. Sabemos que :

$P[X|CP = 1] * P[CP = 1] = P[HF = 1] * P[F = 1] * P[CP|HF = 1, F = 1] * P[E = 1|F = 1] * P[RX+ = 0|CP] * P[D = 0|CP, E = 1]$

Para maximizar a expressão acima, precisamos maximizar

$$P[CP|HF = 1, F = 1] * P[RX+ = 0|CP] * P[D = 0|CP, E = 1]$$

já que os demais elementos envolvidos no produto acima são constantes ($P[HF = 1], P[F = 1]$).

Exercício: Utilizando as tabelas CPT para cada um dos atributos RX+, CP, E, D, determine qual o valor de CP que maximiza o produto $P[CP|HF = 1, F = 1] * P[RX+ = 0|CP] * P[D = 0|CP, E = 1]$. Esta será a classe na qual será classificada a tupla X.

Capítulo 6

Análise de Clusters

6.1 Análise de Clusters - Introdução

Análise de Clusters é o processo de agrupar um conjunto de objetos físicos ou abstratos em classes de objetos similares. Um *cluster* é uma coleção de objetos que são similares uns aos outros (de acordo com algum critério de similaridade pré-fixado) e dissimilares a objetos pertencentes a outros clusters.

As diferenças básicas entre as tarefas de Classificação e Análise de Clusters: Análise de Clusters é uma tarefa que *Aprendizado não-supervisionado*, pelo fato de que os clusters representam classes que não estão definidas no início do processo de aprendizagem, como é o caso das tarefas de Classificação (*Aprendizado Supervisionado*), onde o banco de dados de treinamento é composto de tuplas classificadas. Clusterização constitui uma tarefa de aprendizado *por observação* ao contrário da tarefa de Classificação que é um aprendizado *por exemplo*.

Clusterização Conceitual versus Clusterização Convencional: Na clusterização Conceitual o critério que determina a formação dos clusters é um determinado *conceito*. Assim, uma classe de objetos é determinada por este conceito. Por exemplo, suponhamos que os objetos são indivíduos de uma população e que o critério determinante para se agrupar indivíduos seja o risco de se contrair uma determinada doença. Já na clusterização convencional, o que determina a pertinência dos objetos a um mesmo grupo é a distância geométrica entre eles.

6.1.1 Tipos de dados em Análise de Clusters

Alguns algoritmos de análise de clusters operam com os dados organizados numa *matriz de dados* $n \times p$, conforme ilustrado na tabela abaixo:

$$\begin{vmatrix} x_{11} & \dots & x_{1f} & \dots & x_{1p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{i1} & \dots & x_{if} & \dots & x_{ip} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n1} & \dots & x_{nf} & \dots & x_{np} \end{vmatrix}$$

Esta matriz é simplesmente a tabela dos dados de treinamento. Cada linha desta tabela representa as coordenadas de um objeto i. Cada coluna representa os valores de um atributo assumidos por cada um dos n objetos.

Por outro lado, muitos algoritmos de clusterização se aplicam em dados organizados numa *matriz de dissimilaridade*, onde o elemento da coluna j e linha i da matriz é o número $d(i, j)$ representando a distância entre os objetos i e j.

$$\begin{vmatrix} 0 & \dots & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ d(n,1) & d(n,2) & d(n,3) & \dots & 0 \end{vmatrix}$$

Para que uma função d seja uma *distância* é necessário e suficiente que as seguintes condições sejam satisfeitas, para quaisquer objetos i, j, k:

1. $d(i, j) \geq 0$
2. $d(i, i) = 0$.
3. $d(i, j) = d(j, i)$ (simetria)
4. $d(i, j) \leq d(i, k) + d(k, j)$ (desigualdade triangular)

A propriedade (1) implica que todos os elementos da matriz de dissimilaridade são não-negativos, a propriedade (2) implica que a diagonal da matriz de dissimilaridade é formada por zeros. A propriedade (3), por sua vez, implica que a matriz de dissimilaridade é simétrica com relação à diagonal e por isso, só registramos nela os elementos abaixo da diagonal.

Exercício : O que implica a propriedade (4) da distância, com relação à matriz de dissimilaridade ?

Assim, qualquer função que satisfaz às quatro propriedades acima é chamada de *distância*. As mais importantes funções nesta categoria são:

- Distância Euclidiana :

$$d(i,j) = \sqrt{\mid x_{i_1} - x_{j_1} \mid^2 + \mid x_{i_2} - x_{j_2} \mid^2 + \ldots + \mid x_{i_n} - x_{j_n} \mid^2}$$

- Distância de Manhattan :

$$d(i,j) = \mid x_{i_1} - x_{j_1} \mid + \mid x_{i_2} - x_{j_2} \mid + \ldots + \mid x_{i_n} - x_{j_n} \mid$$

- Distância de Minkowski :

$$d(i,j) = \sqrt[q]{\mid x_{i_1} - x_{j_1} \mid^q + \mid x_{i_2} - x_{j_2} \mid^q + \ldots + \mid x_{i_n} - x_{j_n} \mid^q}$$

  onde $q \geq 1$. Logo, a distância de Minkowski generaliza tanto a distância euclidiana (caso especial onde $q = 2$) quanto a distância de Manhattan (caso especial onde $q = 1$).

Exercício: Sejam $X_1 = (1, 2)$ e $X_2 = (4, 6)$. Calcule cada uma das 3 distâncias acima entre os objetos X_1 e X_2 (para a de Minkowski considere $q = 3$) e ilustre no plano xy os segmentos representando cada distância e comente as diferenças entre elas.

Às vezes deseja-se ressaltar a importância de certos atributos no cálculo da distância. Para isto, considera-se uma distância *ponderada*, que consiste em se associar pesos a cada uma das coordenadas do objeto. Por exemplo, a distância euclidiana ponderada é dada por :

$$d(i,j) = \sqrt{w_1 \mid x_{i_1} - x_{j_1} \mid^2 + w_2 \mid x_{i_2} - x_{j_2} \mid^2 + \ldots + w_n \mid x_{i_n} - x_{j_n} \mid^2}$$

onde $w_1, ..., w_n$ são os pesos de cada um dos atributos envolvidos na descrição dos objetos.

6.1.2 Preparação dos Dados para Análise de Clusters

Como dissemos acima, muitos algoritmos se aplicam à matriz de dissimilaridade dos objetos (só interessam as distâncias relativas entre os objetos e não os valores dos atributos dos objetos). Assim, antes de aplicar o algoritmo é preciso transformar a matriz de dados em uma matriz de dissimilaridade. Os métodos de transformação dependem do tipo de valores que assumem os atributos dos objetos.

Atributos Contínuos em intervalos

É o caso quando todos os atributos possuem valores que ficam num intervalo contínuo $[a, b]$, como por exemplo, peso, altura, latitude, longitude, temperatura. Os valores são ditos contínuos quando não forem discretizados, isto é, o número de valores assumidos é grande. As unidades de medida utilizadas para medir estes valores (kg, g, metro, cm, ...) podem afetar a análise de clusters. Se a unidade for muito grande (muito grosseira), teremos poucos clusters, se for pequena (muito refinada), teremos muitos clusters. Assim, antes de calcular a distância entre os objetos é preciso *padronizar* os dados. O processo de *padronização* tem como objetivo dar um peso igual a cada um dos atributos. O procedimento para padronizar os dados de uma matriz de dados é o seguinte:

1. Calcula-se o *desvio médio absoluto* para cada atributo A_f :

$$s_f = \frac{1}{n}(\mid x_{1f} - m_f \mid + \mid x_{2f} - m_f \mid + \ldots + \mid x_{kf} - m_f \mid)$$

   onde m_f = valor médio do atributo A_f. Veja que s_f é um valor associado à *coluna* f da matriz de dados, onde estamos operando com os valores x_{if} da coordenada f de cada objeto X_i.

2. Calcula-se a *medida padrão* ou z-score para o atributo f de cada objeto i:

$$z_{if} = \frac{x_{if} - m_f}{s_f}$$

   Este é o valor padronizado do elemento x_{if}.

Observamos que o desvio médio absoluto s_f é mais robusto no que diz respeito a ruídos (outliers) do que o desvio médio padrão σ_f:

$$\sigma_f = \frac{1}{n}(\mid x_{1f} - m_f \mid^2 + \mid x_{2f} - m_f \mid^2 + \ldots + \mid x_{kf} - m_f \mid^2)$$

Isto é, se um dos valores aparecendo na coluna f está bem longe da média dos valores (tratando-se portanto de um outlier) seu efeito é amenizado no cálculo do desvio padrão (muito mais do que no cálculo do desvio absoluto).

Atributos binários

Atributos de tipo binário ou booleano só têm dois valores : 1 ou 0, sim ou não. Tratar valores binários como valores numéricos pode levar a análises de clusters errôneas. Para determinar a matriz de dissimilaridade para valores binários, isto é, determinar $d(i, j)$

para cada par de objetos i, j, vamos considerar primeiramente a *tabela de contingência* para i, j. Nesta tabela:

- q é o número de atributos com valor 1 para i e j
- r é o número de atributos com valor 1 para i e 0 para j
- s é o número de atributos com valor 0 para i e 1 para j
- t é o número de atributos com valor 0 para i e 0 para j
- p é o número total de atributos. Portanto $p = q + r + s + t$.

Tabela de contingência para os objetos i e j

		Objeto j		
		1	0	Soma
Objeto i	1	q	r	q + r
	0	s	t	s + t
	Soma	q + s	r + t	p

Atributos simétricos

Um atributo de tipo booleano é dito *simétrico* se ambos os valores 0 ou 1 são igualmente importantes para a análise de clusters. Por exemplo, o atributo Gênero é simétrico, pois os dois valores M e F são igualmente importantes e além disto, estes dois valores têm a mesma probabilidade de ocorrência. Neste caso, a distância entre i e j é definida como o *coeficiente de simples concordância* :

$$d(i, j) = \frac{r + s}{q + r + s + t}$$

isto é, $d(i, j)$ é a porcentagem de atributos que discordam entre os dois objetos.

Atributos assimétricos

Um atributo de tipo booleano é dito *assimétrico* se existe uma predominância de algum dos valores. Por exemplo, os resultados de um teste para detectar uma doença. Neste caso, o valor mais importante é o mais raro, isto é, teste positivo. Este será o valor 1. Logo, a concordância entre dois 1's é muito mais importante do que a concordância entre dois 0's. Neste caso, a distância entre i e j é definida como sendo o *coeficiente de Jacquard* :

$$d(i,j) = \frac{r+s}{q+r+s}$$

isto é, $d(i,j)$ é a porcentagem de atributos que discordam entre os dois objetos, onde no total de atributos foi desconsiderado aqueles atributos cujos valores concordam e são ambos iguais a 0.

Para ilustrar este cálculo, consideremos o seguinte banco de dados de treinamento:

Nome	Gênero	Febre	Tosse	Teste1	Teste2	Teste3	Teste4
Jack	M	Sim	Não	Pos	Neg	Neg	Neg
Mary	F	Sim	Não	Pos	Neg	Pos	Neg
Jim	M	Sim	Sim	Neg	Neg	Neg	Neg
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Queremos construir clusters de objetos neste banco de dados tais que objetos de um mesmo cluster correspondem a indivíduos sofrendo de uma mesma doença.

O único atributo simétrico é Gênero. Os restantes são assimétricos, pois somente o resultado positivo é importante. Para os objetivos da análise de clusters pretendida, vamos supor que a distância entre objetos é calculada tendo como base somente os atributos assimétricos referentes a resultados de testes e ocorrência de sintomas (febre, tosse). O atributo Gênero não é importante para esta análise. Neste caso, a distância é calculada utilizando o coeficiente de Jacquard. Assim,

$$d(jack, mary) = \frac{0+1}{2+0+1} = 0.33$$

$$d(jack, jim) = \frac{1+1}{1+1+1} = 0.67$$

$$d(jim, mary) = \frac{1+2}{1+1+2} = 0.75$$

Estas medidas sugerem que Jack e Mary estão mais próximos, portanto, provavelmente serão diagnosticados como sendo portadores de uma mesma doença. Por outro lado, Jim e Mary estão bem distantes. Logo, é bem provável que estejam em clusters distintos, isto é, serão diagnosticados como portadores de doenças distintas.

Atributos Nominais, Ordinais e Escalonados

Atributos Nominais

Atributo nominal é um atributo discreto, assumindo um número pequeno de valores possíveis. Trata-se de uma generalização dos atributos booleanos, onde o número de valores assumidos é 2. Um exemplo de atributo nominal é Cor, podendo assumir cinco valores: vermelho, amarelo, verde, azul e rosa. Em geral, seja M o número de valores que pode assumir um atributo nominal. Ao invés de denotar os valores por strings, podemos associar a eles números inteiros 1, 2, ... M[1] A distância entre os objetos i e j é medida de maneira análoga como foi feito no caso de atributos booleanos: considerando o coeficiente de coincidência simples:

$$d(i,j) = \frac{p-m}{p}$$

onde p é o número total de atributos e m é o número de atributos onde há coincidências. Assim, $d(i,j)$ é a porcentagem de atributos cujos valores não coincidem. Também podem ser atribuídos pesos a atributos dependendo do número de valores que pode assumir. Por exemplo, suponhamos que tenhamos dois atributos A e B, e que A assuma 5 valores e B 2 valores. Associamos a A o peso 1.5 e a B o peso 0.5. Suponhamos que somente os valores do atributo A coincidam. Então:

$$d(i,j) = \frac{2-1.5}{2} = \frac{0.5}{2} = 0.25$$

Um atributo nominal A assumindo M valores pode ser codificado por um conjunto de atributos binários (booleanos) assimétricos. Cria-se um novo atributo binário para cada um dos M valores que assume A. Se i é um objeto e o valor de A para i é n então os valores dos M atributos $(B_1, B_2, ..., B_M)$ correspondentes a A são : $B_1 = 0, B_2 = 0, ..., B_n = 1, B_{n+1} = 0, ..., B_M = 0$. Com esta transformação de um atributo nominal num atributo binário, a distância entre dois objetos i, j pode ser calculada utilizando o método para atributos binários discutido acima. Repare que esta transformação que fizemos de um atributo nominal em atributo binário é o mesmo que utilizamos nas aulas 10 e 11 para transformar o input de uma rede neural (correspondendo a uma tupla do banco de amostras) num vetor de 0's e 1's.

[1] O fato de se ter associado números inteiros aos valores do atributo, não significa que uma ordem entre estes valores foi determinada. O objetivo desta associação é simplesmente de poder tratar valores nominais como sendo números inteiros. A ordem não é considerada. Esta é a diferença fundamental entre atributos nominais e atributos ordinais que veremos mais adiante.

Atributos Ordinais

Um atributo ordinal é semelhante a um atributo nominal, exceto que os valores assumidos são ordenados, o que não acontece com os atributos nominais. Por exemplo, o atributo TipoMedalha pode assumir os valores nominais Bronze, Prata e Ouro. A estes valores são associados os números 0, 1, 2 respectivamente. A ordem entre os números estabelece uma ordem entre os valores Bronze, Prata, Ouro.

1. Seja x_{if} o valor do atributo A_f do i-ésimo objeto e suponha que estes valores podem ser mapeados numa escala crescente $0, 1, ..., M_f - 1$, onde M_f é o total de valores que pode assumir o atributo A_f. Substitua cada x_{if} pela sua correspondente posição r_{if} na escala $0, 1, ..., M_f - 1$. Por exemplo, se o atributo A_f é TipoMedalha e seus valores são {Bronze, Prata, Ouro} então a escala é : Bronze $\rightarrow$ 0, Prata $\rightarrow$ 1, Ouro $\rightarrow$ 2. Aqui, $M_f = 3$.

2. Como cada atributo ordinal tem um número distinto de valores possíveis, é frequentemente necessário mapear estes valores para o intervalo $[0, 1]$ de tal maneira que cada atributo tenha um peso igual no cálculo da distância. Isto pode ser conseguido, substituindo-se o número inteiro r_{if} por:

$$z_{if} = \frac{r_{if}}{M_f - 1}$$

3. Uma vez feita esta transformação de cada valor inicial x_{if} em z_{if} procede-se ao cálculo da distância entre os objetos i e j utilizando uma das funções distâncias discutidas anteriormente (Euclidiana, Manhattan, Minkowski). Por exemplo, a distância euclidiana entre os objetos i e j é dada por:

$$d(i, j) = \sqrt{\mid z_{i1} - z_{j1} \mid^2 + \mid z_{i2} - z_{j2} \mid^2 + \ldots + \mid z_{in} - z_{jn} \mid^2}$$

Atributos escalonados não-lineares

Atributos escalonados não-lineares são como os atributos contínuos em intervalos. A diferença entre eles é que um atributo contínuo em intervalo representa uma medida segundo uma escala *linear* (temperatura, longitude, peso, altura, etc). Já um atributo escalonado não-linear representa uma medida segundo uma escala não-linear, na maioria das vezes uma escala exponencial, segundo a fórmula Ae^{Bt} ou Ae^{-Bt}, onde A e B são constantes positivas. Por exemplo, o crescimento de uma população de bactérias ou a desintegração de um elemento radioativo são conceitos medidos de acordo com uma escala exponencial.

Existem três maneiras para se calcular a dissimilaridade $d(i, j)$ entre dois objetos i e j onde todos os atributos são escalonados não-lineares:

1. Trata-se os atributos escalonados não-lineares da mesma forma como se tratou os atributos contínuos em intervalos. Esta não é uma boa maneira pois nos atributos contínuos em intervalos, a escala é linear. Logo, é bem possível que, tratando-se atributos escalonados não-lineares como se fossem lineares, a escala seja distorcida.

2. Aplica-se uma transformação logarítmica ao valor x_{if} de cada atributo A_f de um objeto i, obtendo $y_{if} = log(x_{if})$. Agora, os valores y_{if} podem ser tratados como se fossem valores contínuos em intervalos (medidos segundo uma escala linear). Repare que, dependendo de como foi escalonado o valor, outras transformações poderão ser empregadas. Nestes exemplo, utilizamos a função *log* já que é a inversa da função exponencial[2].

3. Trata-se os valores x_{if} como se fossem valores ordinais contínuos. Associa-se a cada valor um número entre 0 e $M_f - 1$, onde M_f é o número total de valores assumidos pelo atributo A_f (este número M_f, ao contrário do que acontece com os atributos ordinais, pode ser muito grande). Uma vez feita esta associação, trata-se os valores associados da mesma maneira como tratamos os atributos contínuos em intervalos.

Os dois últimos métodos são os mais eficazes. A escolha de um ou outro método depende da aplicação em questão.

Atributos Mistos

Nas seções anteriores, foi discutido como calcular a matriz de dissimilaridade entre objetos, considerando que todos os atributos são do mesmo tipo. Na realidade, um objeto possui atributos cujos tipos variam, podendo assumir tipos dos mais variados, entre todos os tipos que consideramos anteriormente.

Como calcular a dissimilaridade $d(i, j)$ entre objetos i, j, onde onde atributos são de tipos distintos ?

Enfoque de agrupamento: Neste enfoque, agrupa-se os atributos de mesmo tipo em grupos. Para cada grupo de atributos, faz-se a análise de clusters dos objetos (somente considerando os atributos do grupo, os outros são desconsiderados). Assim, teremos tantas análises de clusters quanto for o número de tipos de atributos do banco de dados de objetos. Se os resultados de cada análise são compatíveis, isto é, se objetos que são similares numa análise, continuam similares em outra análise e objetos dissimilares segundo

[2]No exemplo, foi esta a função utilizada no escalonamento dos valores do atributo.

uma análise são dissimilares segundo outra análise, então este método é factível. Entretanto, em aplicações reais, é muito improvável que os resultados de análises separadas sejam compatíveis.

Enfoque da uniformização da escala dos valores: Uma técnica que é muito utilizada consiste em transformar todos os valores dos diferentes atributos em valores de uma escala comum no intervalo [0,1]. Suponha que o banco de dados contém p atributos de tipos mistos. A dissimilaridade $d(i,j)$ entre os objetos i,j é definida por:

$$d(i,j) = \frac{\sum_{f=1}^{p} \delta_{ij}^{f} d_{ij}^{f}}{\sum_{f=1}^{p} \delta_{ij}^{f}}$$

onde:

- $\delta_{ij}^{f} = 0$ se uma das possibilidades abaixo ocorre:
  1. os valores x_{if} ou x_{jf} são incompletos (do tipo NULL, isto é, não foram fornecidos),
  2. o atributo A_f é booleano assimétrico e $x_{if} = x_{jf} = 0$.
- $\delta_{ij}^{f} = 1$, caso nenhuma das condições acima ocorrem.

Os números d_{ij}^{f} representam a contribuição do atributo A_f no cálculo da dissimilaridade entre os objetos i e j. O cálculo deste número depende do tipo do atributo A_f:

1. Se o atributo A_f é booleano ou nominal : $d_{ij}^{f} = 0$ se $x_{if} = x_{jf}$. Caso contrário, $d_{ij}^{f} = 1$.
2. Se o atributo A_f é contínuo em intervalo:

   $$d_{ij}^{f} = \frac{\mid x_{if} - x_{jf} \mid}{max_h\{x_{hf}\} - min_h\{x_{hf}\}}$$

   onde h varia entre todos os objetos onde o atributo f não é incompleto, isto é, seu valor não é NULL.
3. Se o atributo A_f é ordinal ou escalonado não-linear: calcula-se os inteiros r_{if} associados ao valor x_{if} e considera-se $z_{if} = \frac{r_{if}}{M_f - 1}$. A partir daí, trata-se os valores z_{if} como se fossem contínuos em intervalos, e calcula-se a dissimilaridade d_{ij}^{f} de acordo.

Exercício: Dê uma maneira mais refinada de se calcular a contribuição d_{ij}^{f} do atributo A_f, no caso deste atributo ser do tipo contínuo em intervalo, utilizando a padronização dos dados do atributo A_f (ver seção 2.1). Compare esta maneira com a maneira acima. Qual em sua opinião é a melhor ?

6.2 As diversas categorias de métodos de clusterização

Existe um grande número de algoritmos de clusterização na literatura. A escolha de um ou outro método depende do tipo de dados que se tem e qual o objetivo que se tem em mente. Em geral, para uma determinada tarefa de clusterização, aplica-se diferentes algoritmos sobre os mesmos dados e analisa-se os clusters obtidos. Escolhe-se o algoritmo que produz os clusters mais adequados para a aplicação a que se destinam os dados.

Os métodos de clusterização podem ser classificados nas seguintes categorias:

- **Métodos baseados em Particionamento**. A idéia geral destes métodos é a seguinte: dispõe-se de uma base de dados de n elementos e de um número $k \leq n$, representando o número de clusters que se deseja formar. Repare que neste tipo de método, o usuário precisa fornecer de antemão este número k. O método vai dividir o conjunto de dados em k partes disjuntas, satisfazendo : (1) os elementos numa mesma parte estão próximos (de acordo com um critério dado), (2) elementos de partes distintas estão longe, de acordo com este mesmo critério. Para obter tal subdivisão, os métodos por particionamento operam do seguinte modo: cria-se uma partição inicial aleatória de k partes e posteriormente, num processo iterativo, os elementos das partes vão sendo realocados para outras partes de tal modo a melhorar o particionamento a cada iteração, isto é, de tal modo que cada parte realmente contenha objetos que estão próximos e objetos em partes distintas estejam longe um do outro.

- **Métodos Hierárquicos.** Estes métodos são de dois tipos: aglomerativos e divisórios.

  - Métodos aglomerativos: No passo inicial, cada elemento do banco de dados forma um cluster. Nas próximas iterações, pares de clusters da iteração precedente que satisfazem um certo critério de distância mínima (existem diferentes funções operando sobre pares de clusters que podem ser utilizadas para este critério) são aglutinados num único cluster. O processo termina quando um número de clusters k fornecido pelo usuário é atingido. Um método conhecido deste tipo é o AGNES (AGlomerative NESting).
  - Métodos Divisórios: No passo inicial, cria-se um único cluster composto pelo banco de dados inteiro. Nas próximas iterações, os clusters são subdivididos em duas partes, de acordo com algum critério de similaridade. O processo termina quando um número de clusters k fornecido pelo usuário é atingido. Um método conhecido deste tipo é o DIANA (DIvisive ANAlysis).

Diversos algoritmos conhecidos de clusterização se classificam como hierárquicos: BIRCH, CURE, ROCK, CHAMELEON. Na próxima aula, veremos com detalhes o algoritmo CURE introduzido no artigo:

Guha, S., Rastogi, R., Shim, K.: Cure: An Efficient Clustering Algorithm for Large Databases. In ACM SIGMOD International Conference on Management of Data, 1998, Seattle, USA.

- **Métodos baseados em densidade**. Métodos adequados para descobrir clusters de formato arbitrário, e não somente de formato esférico como é o caso dos métodos por particionamento e hierárquico. Nestes métodos os clusters são regiões densas de objetos no espaço de dados que são separadas por regiões de baixa densidade (representando ruídos). Entende-se por *região densa* uma região onde uma ϵ-vizinhança de cada ponto (onde ϵ é um parâmetro dado) contém pelo menos um número dado M de pontos. Alguns algoritmos conhecidos baseados em densidade são: DBSCAN, OPTICS, DENCLUE.

- **Outros métodos.** Uma grande quantidade de outros métodos de clusterização, utilizando diferentes técnicas, existem, tais como métodos utilizando reticulados, métodos estatísticos e métodos utilizando redes neuronais.

A seguir vamos tratar dois métodos conhecidos, baseados em particionamento.

6.3 Dois métodos baseados em particionamento: k-Means e k-Medóides

Os dois métodos a seguir recebem como input um banco de dados de objetos (tuplas) e um número k (representando o número de clusters que se deseja formar entre os objetos do banco de dados). O banco de dados é entrado em forma de uma matriz de dissimilaridade entre os objetos. Conforme vimos na aula anterior, existem técnicas para construir tal matriz de dissimilaridade, dependendo do tipo de dados presentes no banco de dados.

Seja $\mathcal{C} = \{C_1, \ldots, C_k\}$ uma partição do banco de dados em k clusters e sejam $m_1, m_2, \ldots, m_k$ elementos escolhidos em cada um dos clusters, representando o centro dos mesmos (Estes elementos podem ser definidos de diversas maneiras). Definimos o *erro quadrático* da partição como sendo:

$$\text{Erro}(\mathcal{C}) = \Sigma_{i=1}^{k} \Sigma_{p \in C_i} \mid p - m_i \mid^2$$

Os métodos k-means e k-medóides que apresentamos a seguir procuram construir uma partição $\mathcal{C}$ contendo k clusters, para a qual o erro quadrático é mínimo. No método k-means os elementos representativos de cada cluster são os seus centros de gravidade respectivos. No método k-medoides, estes elementos não são necessariamente os centros de gravidade. Eles são escolhidos de modo a satisfazer certas certas condições de minimização do erro, o que torna os métodos de clusterização baseados em k-medóides menos sensíveis a *outliers*.

6.3.1 Método k-Means

A idéia geral do método k-means é a seguinte:

1. Escolhe-se arbitrariamente k objetos $\{p_1, ..., p_k\}$ do banco de dados. Estes objetos serão os centros de k clusters, cada cluster C_i formado somente pelo objeto p_i.

2. Os outros objetos do banco de dados são colocados nos clusters da seguinte maneira: para cada objeto O diferente de cada um dos p_i's, considera-se a distância entre O e cada um dos p_i's (isto é dado pela matriz de dissimilaridade). Considera-se aquele p_i para o qual esta distância é mínima. O objeto O passa a integrar o cluster representado por p_i.

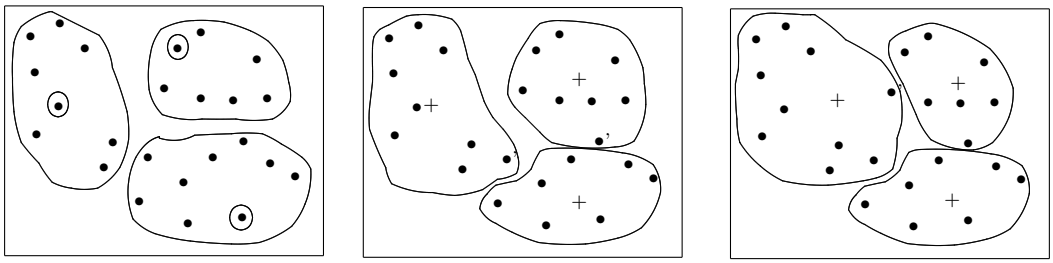
3. Após este processo, calcula-se a média dos elementos de cada cluster, isto é, o seu centro de gravidade. Este ponto será o novo representante do cluster.

4. Em seguida, volta para o passo 2 : varre-se o banco de dados inteiro e para cada objeto O calcula-se a distância entre este objeto O e os novos centros dos clusters. O objeto O será realocado para o cluster C tal que a distância entre O e o centro de C é a menor possível.

5. Quando todos os objetos forem devidamente realocados entre os clusters, calcula-se os novos centros dos clusters.

6. O processo se repete até que nenhuma mudança ocorra, isto é, os clusters se estabilizam (nenhum objeto é realocado para outro cluster distinto do cluster atual onde ele se encontra).

A figura abaixo ilustra o funcionamento do método k-means para $k = 3$. Na primeira iteração, os objetos circundados representam os que foram escolhidos aleatoriamente. Nas próximas iterações, os centros de gravidade são marcados com o sinal +.



1^a iteração 2^a iteração 3^a iteração

É possível mostrar que o método k-means produz um conjunto de clusters que minimiza o erro quadrático com relação aos centros de gravidade de cada cluster. Este método só produz bons resultados quando os clusters são "núvens" compactas de dados, bem separadas umas das outras. As **vantagens** do método são sua eficiência em tratar grandes conjuntos de dados. Suas **desvantagens** são : o fato do usuário ter que fornecer o número de clusters k, o fato de não descobrir clusters de formatos não-convexos e sobretudo o fato de ser sensível a ruídos, já que objetos com valores altos podem causar uma grande alteração no centro de gravidade dos clusters e assim, distorcer a distribuição dos dados nos mesmos.

6.3.2 Método k-Medóides

Este método é uma modificação do método k-means visando neutralizar a desvantagem deste método no que diz respeito a sua sensibilidade a ruídos. O método k-medóides também recebe como input um banco de dados D e um número k, representando o número de clusters que se deseja formar. Os dados devem ser transformados numa matriz de dissimilaridade como foi o caso do Método k-means.

Diversos algoritmos foram desenvolvidos utilizando o Método k-medóides. Entre eles, vamos descrever o PAM (Partitioning Around Medoids - 1990) que tem a desvantagem de não ser eficiente em grandes volumes de dados. Outros algoritmos recentes retomaram a mesma idéia de PAM e melhoraram a eficiência para tratar grandes volumes de dados. Entre estes melhoramentos do PAM, temos o CLARA (Clustering LARge Aplications - 1990) e o CLARANS (Clustering LARge Aplications based on RANdomized Search - 1994). Detalhes destes algoritmos podem ser encontrados em :

Ng, R.T., Han, J. : Efficient and Effective Clustering Methods for Spatial Data Mining. Proceedings of the 20th International Conference on Very Large Data Bases, 1994.

Como funciona o PAM ?

Nas três primeiras etapas do algoritmo PAM que descrevemos abaixo, o objetivo é determinar qual a melhor escolha de k objetos do banco de dados que serão os centros de k clusters. Estes objetos são chamados de medóides. O processo para encontrar estes k objetos é iterativo e utiliza a matriz de dissimilaridade dos dados. Após determinados estes k objetos "ideais", os clusters são criados em torno deles: o banco de dados é varrido (a matriz de dissimilaridade, na verdade) e cada objeto é alocado ao cluster do medóide do qual está mais próximo. Descrevemos abaixo o algoritmo PAM com algum detalhe:

1. Seleciona-se k objetos do banco de dados de forma arbitrária. Estes vão ser os medóides iniciais.

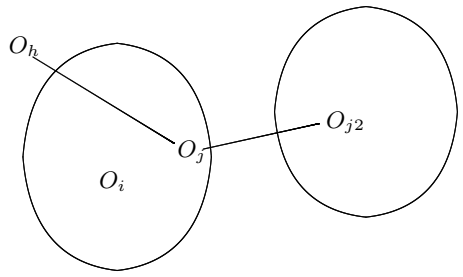
2. **Para todos** os pares de objetos O_i e O_h, onde O_i é um medóide e O_h é um não medóide, calcula-se:

   $$\mathrm{CT}_{ih} = \Sigma_j C_{jih}$$

   onde CT_{ih} = custo de substituir O_i por O_h;
   a somatória Σ_j é feita para todos os objetos O_j que não são medóides,
   e o termo C_{jih} é um número que mede o erro engendrado por esta substituição, sobre o objeto não-medóide O_j. Este número é calculado da seguinte maneira:

   - Se O_j está no cluster de O_i e com a mudança de O_i para O_h, O_j fica mais próximo de um outro medóide O_{j2}. Neste caso, O_j iria para o cluster de O_{j2} e o custo desta mudança seria:

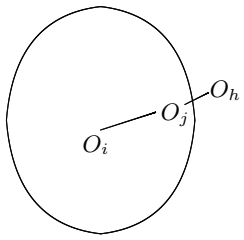
     $$C_{jih} = d(O_j, O_{j2}) - d(O_j, O_i)$$

     Repare que este número é sempre positivo. Justifique.

- Se O_j está no cluster de O_i e com a mudança de O_i para O_h, O_j fica mais próximo de O_h. Neste caso, O_j iria para o cluster de O_h e o custo desta mudança seria:

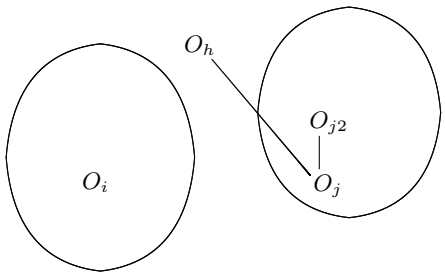
$$C_{jih} = d(O_j, O_h) - d(O_j, O_i)$$



  Repare que agora este número pode ser positivo ou negativo. Justifique.

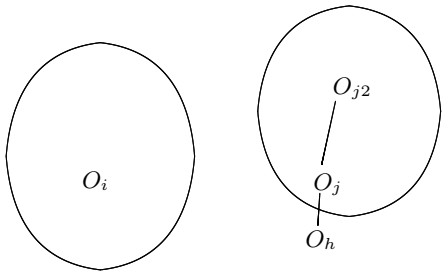
- Se O_j não está no cluster de O_i, por exemplo, está no cluster de O_{j2} e com a mudança de O_i para O_h, O_j fica mais próximo de O_{j2} do que de O_h. Neste caso, O_j permaneceria em seu próprio cluster, nenhuma mudança ocorreria e portanto:

$$C_{jih} = 0$$



- Se O_j não está no cluster de O_i, por exemplo, está no cluster de O_{j2} e com a mudança de O_i para O_h, O_j fica mais próximo de O_h do que de O_{j2}. Neste caso, O_j iria para o cluster de O_h e portanto:

$$C_{jih} = d(O_j, O_h) - d(O_j, O_{j2})$$



Repare que agora este número é sempre negativo.

3. Seleciona-se o par O_i, O_j que corresponde ao $\min_{O_i,O_h}$ TC_{ih}. Se este mínimo é negativo então substitui-se O_i por O_h e volta para o passo 2. Repare que se este mínimo é negativo então existe uma possibilidade de se substituir um medóide atual O_i por um outro objeto O_h de modo que a distribuição dos outros objetos do banco de dados em torno dos novos clusters seja mais adequada, isto é, os objetos em cada cluster estão mais próximos de seus centros do que antes.

4. Caso o mínimo seja positivo, isto significa que não há maneira de substituir os atuais medóides de modo a baixar o custo total do atual aglutinamento. Neste caso, os medóides convergiram. Agora, varre-se o banco de dados para alocar os seus elementos nos clusters adequados, isto é, cada elemento O é alocado ao cluster cujo representante medóide m está mais próximo de O.

6.3.3 DBSCAN : Um método baseado em densidade

A seguir vamos descrever a idéia geral de um algoritmo baseado em densidade. O algoritmo que descrevemos é uma versão simplificada do algoritmo DBSCAN, introduzido no artigo:

M.Ester, H.-P. Kriegel, J. Sander, X.Xu: A density-based algorithm for discovering clusters in large spatial databases with noise . In Proc. 1996 International Conference on Knowledge Discovery and Data Mining (KDD'96), pages 226-231, Portland, USA, Aug. 1996.

Os métodos baseados em densidade utilizam alguns termos próprios que definimos a seguir:

- **ϵ-vizinhança** de um objeto p é o conjunto de objetos (tuplas do banco de dados) que estão dentro de um círculo de raio ϵ com centro em p.

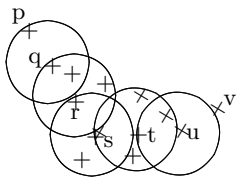
- **Um objeto core** é um objeto tal que sua ϵ-vizinhança contém ao menos um número $MinPt$ de objetos. Este número MinPt é especificado pelo usuário.

- Um objeto p é dito **diretamente alcançável por densidade** a partir do objeto q com respeito a ϵ e $MinPt$ se q é um objeto core e p está na ϵ-vizinhança de q.

- Um objeto p é dito **alcancável por densidade** a partir do objeto q com respeito a ϵ e $MinPt$ se existe um caminho de objetos $p_1, \ldots, pn$ ligando q a p, isto é, $p_1 = q$ e $p_n = p$ tal que p_{i+1} é diretamente alcançável por densidade a partir de p_i com respeito a ϵ e $MinPt$.

- Um objeto p é dito **conectável por densidade** a um objeto q com respeito a ϵ e $MinPt$ se existe um objeto o tal que p e q são alcançáveis por densidade a partir de o com respeito a ϵ e $MinPt$.

A figura abaixo ilustra estes conceitos. Neste exemplo, consideramos $MinPt = 3$. Os círculos centrados em um objeto o representam a ϵ-vizinhança deste ponto. Os objetos q, r, s, t, u são objetos core; os objetos p e q não são. Os objetos que não são objetos core, representam objetos fronteiriços, isto é, que vão se localizar na fronteira de um cluster (baseado em densidade). O objeto p é diretamente alcancável a partir do objeto q. O mesmo objeto p é alcançável a partir do objeto s. O objeto p não é alcançável a partir do objeto v, uma vez que este último é fronteiriço (não é objeto core). Entretanto, p e q são conectáveis por densidade, uma vez que p e v são alcançáveis a partir do objeto s.



Exercício: Propomos os seguintes exercício para o leitor:

1. Se p é alcançável por densidade a partir de q, isto não implica que q é alcançável por densidade a partir de p. Portanto, esta relação não é simétrica. Entretanto, se p e q são objetos core então a relação *alcançável por densidade* é simétrica.

2. Se p é conectável por densidade a q então q é conectável por densidade a p. Portanto esta relação é simétrica.

O algoritmo DBSCAN recebe como input um número real positivo ϵ, um número inteiro $MinPt$ e um banco de dados. A idéia geral do algoritmo é a seguinte:

- Calcula a ϵ-vizinhança de cada ponto.

- Detecta aqueles pontos que são objetos core (isto é, cuja ϵ-vizinhança tem pelo menos $MinPt$ objetos. Cada objeto core será o representante de um cluster formado pela sua ϵ-vizinhança. Enumeramos os clusters assim obtidos (as ϵ-vizinhanças dos objetos core) $\{C_1^1, ..., C_{k_1}^1\}$.

- Para C_1^1, procura o primeiro cluster C_j^1 tal que seu representante p_j seja diretamente alcançável a partir de p_1. Se existir, une os clusters C_1^1 e C_j^1. Os novos representantes deste cluster são p_1 e p_j. Passa para o primeiro i diferente de 1 e j e repete o processo. No final deste processo, tem-se clusters obtidos unindo-se dois clusters da etapa precedente ou clusters simples da etapa precedente.

- O processo é repetido para os clusters da etapa precedente. Enumera-se estes clusters : C_1^2,...,$C_{k_2}^2$. Para cada representante de C_1^2 procura-se o primeiro cluster C_j^2 para o qual um de seus representantes seja diretamente alcançável a partir de um representante de C_j^2. Neste caso, une-se os dois clusters. O conjunto dos representantes deste novo cluster será a união dos conjuntos de representantes de cada um dos clusters envolvidos. Repete-se o mesmo processo para o primeiro i diferente de 1 e j e assim por diante.

- O algoritmo pára quando não houver mais mudanças nos conjunto de clusters da etapa precedente, isto é, quando não houver mais possibilidade de se juntar clusters da etapa precedente.

Sugerimos ao leitor interessado a leitura da seção 4.2 do artigo acima mencionado, onde são discutidas heurísticas para se determinar valores adequados de input para ϵ e $MinPt$.

Exercício: Mostre que se $C_1, \ldots, C_k$ é o conjunto de clusters produzido pelo algoritmo DBSCAN então:

1. se p e q estão em um mesmo cluster C_i então p e q são conectáveis por densidade com respeito a ϵ e $MinPt$.

2. se p e q estão em clusters distintos então p e q não são conectáveis por densidade.

3. o que você pode dizer de um objeto p que não está em nenhum dos clusters C_i ?

6.4 CURE : Um Método Hierárquico para Análise de Clusters

O tema central destas notas de aulas gira em torno dos Métodos Hierárquicos para análise de clusters em um banco de dados. Descreveremos na primeira seção a idéia geral dos métodos hierárquicos em comparação com os métodos baseados em particionamento. Nas seções seguintes, apresentaremos o algoritmo CURE para análise de clusters, introduzido em [15], 1998. Este algoritmo apresenta bons resultados com relação a muitos algoritmos de clusterização conhecidos (BIRCH [16] e DBSCAN [17]) , tanto no que diz respeito a eficiência, à capacidade de detectar clusters de formato não-esférico, e quanto à não sensibilidade a ruídos. Os detalhes desta aula podem ser encontrados no artigo [15].

Os métodos de clusterização podem ser divididos, grosso modo, em métodos por particionamento e métodos hierárquicos. Os métodos por particionamento (k-means e k-medóides) foram objeto de estudo da aula passada. O primeiro sofre dos seguintes problemas: embora seja eficiente, é muito sensível a ruídos e não detecta clusters de formatos arbitrários. O segundo é menos sensível a ruídos mas também tem problemas para detectar clusters de formatos arbitrários. Os algoritmos PAM, CLARA e CLARANS são os principais algoritmos da classe k-medoides ([19]). O primeiro (PAM) é bem ineficiente, o último (CLARANS) é bem eficiente, mas ainda tem o problema da dificuldade em detectar clusters de formato arbitrário.

Os algoritmos de clusterização hierárquicos aglomerativos seguem a seguinte idéia geral:

1. começa com um número de clusters igual ao tamanho do banco de dados: um cluster para cada objeto.

2. pares de clusters são aglutinados a cada iteração até que um número k (dado) de clusters seja alcançado, ou que a distância mínima entre os clusters seja menor do que um limite fornecido (avaliar este parâmetro não é tarefa fácil). Geralmente, utiliza-se o critério de parada correspondente a atingir o número k de clusters desejado.

3. o critério para se aglutinar dois clusters é a distância entre eles : são aglutinados, a cada iteração, os dois clusters que estão mais próximos. Diferentes funções são utilizadas para medir a distância entre dois clusters C_i, C_j:

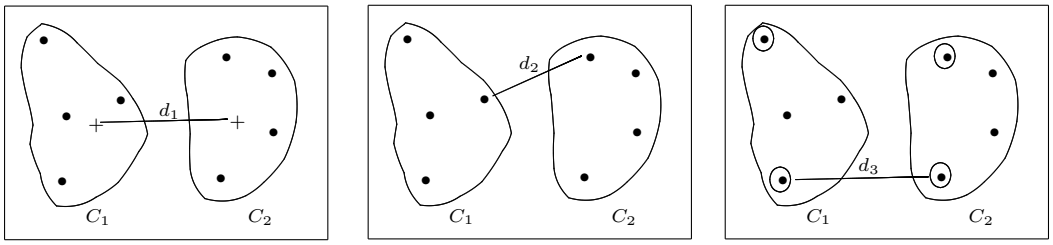
   - $d_{mean}(C_i, C_j) =\mid m_i - m_j \mid$, onde m_i é o centro de gravidade do cluster C_i. Chamamos este enfoque de *enfoque baseado em centróide*.
   - $d_{min}(C_i, C_j) = min \mid p - q \mid$, para todos $p \in C_i$ e $q \in C_j$. Chamamos este enfoque de *MST* (*Minimum Spanning Tree*).

- $d_{ave}(C_i, C_j) = \frac{1}{n_i * n_j} \Sigma_{p \in C_i} \Sigma_{q \in C_j} \mid p - q \mid$, onde n_i = tamanho do cluster C_i.
- $d_{max}(C_i, C_j) = max \mid p - q \mid$, para todos $p \in C_i$ e $q \in C_j$.

6.4.1 O algoritmo CURE

O algoritmo CURE recebe como **input** um banco de dados D e um número k que é o número de clusters que se deseja obter (assim como os métodos de particionamento, o k deve ser fornecido pelo usuário). CURE produz como resultado k clusters. Além disto, outros parâmetros de ajuste deverão ser fornecidos ao algoritmo para a sua execução. Tais parâmetros serão descritos durante a apresentação do algoritmo.

O algoritmo CURE é um algoritmo hierárquico aglomerativo que utiliza uma política mista para o cálculo da distância entre dois clusters, a cada iteração. Esta política é uma espécie de mistura entre a *política dos centróides* (onde a distância entre dois clusters é a distância entre seus centros de gravidade) e a chamada *políticca MST (Minimum Spanning Tree)* (onde a distância entre dois clusters é igual à distância mínima entre dois pontos, um em cada cluster). CURE vai considerar um número adequado de representantes de cada cluster, elementos que estão adequadamente distribuídos no cluster e que representam zonas bem distintas dentro do mesmo). A distância entre dois clusters será min ($d(c_i, c'_j)$ onde todos os representantes do primeiro cluster ($\{c_1, ..., c_n\}$) e do segundo cluster ($\{c'_1, ..., c'_m\}$) são considerados. A figura abaixo ilustra as duas políticas clássicas nos métodos hierárquicos para cálculo de distância entre dois clusters (para efeito de aglutinamento dos mesmos) e a política adotada por CURE.



$d_{mean}(C_1,C_2) = d_1$ $d_{min}(C_1,C_2) = d_2$ $d_{cure}(C_1,C_2) = d_3$

As características principais de CURE são:

1. Clusters de formas arbitrárias podem ser detectados por CURE (a exemplo dos métodos baseados em densidade). Isto normalmente não acontece com os métodos por particionamento.

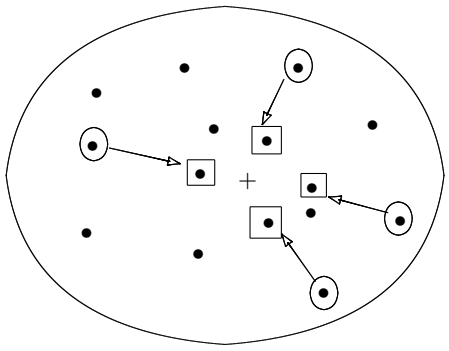
2. CURE é robusto quanto à presença de ruídos.

3. Uma desvantagem de CURE é a sua complexidade O(n^2), onde n é o tamanho do banco de dados de input. No artigo [15] é feita uma discussão sobre como escolher uma amostra dos dados de modo que o algoritmo seja executado em memória principal sobre esta amostra e a probabilidade de se perder clusters nos dados totais é pequena.

6.4.2 Idéia geral

1. Como todo algoritmo hierárquico aglomerativo, CURE inicia com cada objeto do banco de dados constituindo um cluster. Logo, temos n clusters no início, onde n = tamanho do banco de dados.

2. A fim de calcular a distância entre dois clusters, são armazenados c representantes de cada cluster. Este número c é um parâmetro de ajuste que deve ser fornecido como input ao algoritmo. Estes c pontos são escolhidos de forma a representar regiões bem distintas em cada cluster. Depois de escolhidos, é feita uma *retração* destes pontos na direção do centro do cluster, de um fator α, onde $0 \leq \alpha \leq 1$. Este número α é um dos parâmetros de ajuste que devem ser fornecidos como input. A distância entre dois clusters será a distância entre os pares de representantes mais próximos, um em cada cluster. Assim, somente os representantes são levados em consideração para o cálculo da distância entre dois clusters. Na figura abaixo, os pontos dentro de círculos são os representantes antes da retração. Após uma retração de um fator $\alpha = 1/2$ na direção do centro +, os pontos obtidos são aqueles dentro de □.



Estes c representantes tentam capturar o formato do cluster. A retração em direção ao centro de gravidade do cluster tem o efeito de diminuir a influência dos ruídos. A razão para isto é que os outliers estarão longe do centro do cluster. Caso um outlier seja escolhido como representante do cluster, após a retração ele se aproximará do centro bem mais do que os outros representantes que não são outliers.

O parâmetro α também serve para capturar o formato do cluster. Valores de α pequenos favorecem clusters de formato menos compacto, não-convexo. Valores de α grande (próximos de 1), aproximando os representantes do centro do cluster, favorecem a criação de clusters mais compactos, convexos, de forma esférica.

3. Após calculada a distância entre os clusters, aglutina-se aqueles dois primeiros que estão mais próximos. Este processo de aglutinamento envolve a cálculo dos novos representantes para o cluster aglutinado.

4. Volta para o passo anterior: calcula-se as distâncias entre os novos clusters e aglutina-se aqueles que estão mais próximos.

5. Este processo pára quando atinge-se o número k de clusters desejado (o k é parâmetro do algoritmo).

6.4.3 Descrição do algoritmo com algum detalhe

O algoritmo utiliza duas estruturas de dados, uma para armazenar os clusters a cada iteração e outra para armazenar os representantes de cada cluster, a cada iteração. Tais estruturas de dados não serão detalhadas nestas notas de aula. Só podemos adiantar que:

- uma estrutura do tipo *heap* é utilizada para armazenar os clusters numa certa ordem a ser detalhada mais adiante.

- uma estrutura de $k-d$ tree é utilizada para armazenar os representantes de cada cluster. Uma $k-d$ tree é uma estrutura de dados que é utilizada para o armazenamento e consulta eficientes de dados multi-dimensionais (como é o nosso caso: os objetos do banco de dados têm diversos atributos, cada um constituindo uma dimensão).

Tais estruturas de dados particulares otimizam a busca dos clusters e seus representantes, bem como a atualização dos novos clusters e de seus representantes a cada iteração. Detalhes das mesmas não são necessários para o entendimento do algoritmo. O leitor interessado nestes detalhes pode consultar um livro de estruturas de dados, por exemplo [18] e a seção 3.2 (Data Structures) do artigo [15].

Nestas notas de aula, vamos simplesmente imaginar que os clusters são armazenados num arquivo Q e os representantes de cada cluster são armazenados num arquivo T.

A figura (a) abaixo ilustra como os clusters são armazenados no arquivo Q. Para cada cluster u, seja $u.mp$ o seu cluster mais próximo (segundo a distância entre seus representantes). Os clusters são ordenados da seguinte maneira no arquivo Q: o primeiro

cluster será aquele tal que a distância entre u e $u.mp$ é a menor. A figura (b) abaixo ilustra o arquivo T dos representantes de cada cluster de Q. Neste exemplo, $\mathrm{d}(u_1, u_2) = 2$, $\mathrm{d}(u_1, u_3) = 5$, $\mathrm{d}(u_2, u_3) = 3$. O cluster mais próximo de u_1 é u_2, o mais próximo de u_2 é u_1 e o mais próximo de u_3 é u_2. O cluster que está mais próximo de seu mais próximo é u_1, depois vem u_2 e por último vem u_3.

Arquivo Q

id1	cluster u_1	$\mathrm{d}(u_1,u_1.mp) = 2$
id2	cluster u_2	$\mathrm{d}(u_2,u_2.mp) = 2$
id3	cluster u_3	$\mathrm{d}(u_3,u_3.mp) = 3$

(a)

Arquivo T

id1	$\{p_1^1, p_2^1, p_3^1\}$
id2	$\{p_1^2, p_2^2, p_3^3\}$
id3	$\{p_1^3, p_2^3, p_3^3\}$

(b)

O procedimento geral de criação dos clusters

1. Inicialmente, o arquivo Q contém n clusters, um para cada objeto do banco de dados. Utilizando a matriz de dissimilaridade dos dados, são calculadas as distâncias entre os objetos e os clusters são ordenados em Q segundo a ordem da menor distância.

   O arquivo T contém n elementos, cada elemento sendo formado por um conjunto unitário, correspondendo a um objeto do banco de dados.

2. Repete-se o seguinte processo até que o tamanho do arquivo Q seja maior do que um número k (atingiu-se neste ponto, o número k de clusters desejado):

   (a) Considera-se o primeiro cluster u de Q e seu cluster mais próximo $v = u.mp$ (na verdade, o cluster seguinte a u). Retira-se u e v de Q.

   (b) Calcula-se o cluster $w = merge(u, v)$. Veremos este procedimento com detalhes mais adiante. Este procedimento retorna o cluster aglutinado w e também seus representantes $w.rep$.

   (c) Retira-se de T os representantes de u e os representantes de v. Insere-se os representantes de w em T.

   (d) Calcula o cluster mais próximo de w, isto é, o cluster $w.mp$. Este é um dos clusters que estão em Q. Realoca-se $w.mp$ em Q. Veja que uma vez que foram retirados de Q os clusters u e v, é possível que a ordem dos clusters restantes tenha que ser alterada. O procedimento de cálculo do cluster mais próximo de w será visto com detalhes mais adiante. Este procedimento, além de calcular o cluster mais próximo de w também, simultaneamente, recalcula os clusters mais próximos daqueles clusters de Q para os quais os clusters mais próximos mudarão após a inserção de w e a retirada de u e v.

(e) Insere-se w em Q na posição adequada. Como calculamos no passo anterior o cluster mais próximo de w, e temos os clusters mais próximos para cada um dos outros clusters de Q, é fácil saber exatamente qual a posição em que w deverá ser inserido em Q.

O procedimento de cálculo do cluster mais próximo ao novo cluster aglutinado w

Lembramos que este procedimento, além de calcular o cluster mais próximo de w também, simultaneamente, recalcula os clusters mais próximos daqueles clusters de Q para os quais os clusters mais próximos mudarão após a inserção de w e a retirada de u e v.

1. Escolhe-se um cluster qualquer x de Q. Faz-se $w.mp := x$.

2. Para cada cluster y de Q :

   se $d(w, y) < d(w, w.mp)$ então fazemos:

   - $w.mp := y$
   - testamos se $y.mp = u$ ou v (os que foram retirados de Q).
     - **Se for o caso** : testamos também se $d(y, y.mp) < d(y, w)$.
       * Se isto acontecer, testamos, se existe cluster z_0 tal que $d(y, z_0) < d(y, w)$. Se houver, então $y.mp := z_0$. Se não houver, $y.mp := w$.
       * Se isto não acontecer, isto é, $d(y, w) \leq d(y, y.mp)$ então fazemos $y.mp := w$.

       Depois disto, realoca-se y no arquivo Q, de forma adequada, uma vez que foi alterado seu cluster mais próximo.
     - **Se não for o caso** : testamos simplesmente se $d(y, y.mp) > d(y, w)$. Neste caso, modificamos o $y.mp$ fazendo $y.mp := w$. Como houve mudança do $y.mp$ é preciso realocar o y em Q de forma adequada.

O procedimento de aglutinação de dois clusters u e v

O procedimento $merge(u, v)$ produz o cluster aglutinado w e seus representantes $w.rep$.

1. Fazemos a união dos dois clusters u e v, isto é, $w := u \cup v$.

2. Calcula-se o centro de w:

$$w.mean := \frac{n_u u.mean + n_v v.mean}{n_u + n_v}$$

onde n_u = número de elementos de u e n_v = número de elementos de v.

3. Inicializa-se uma variável $tmpSet := \emptyset$. Esta variável vai armazenar os c pontos representantes do cluster w que serão selecionados seguindo os passos descritos abaixo. Estes representantes que serão armazenados em $tmpSet$ são os pontos **antes da retração**.

4. Para cada $i = 1, ..., c$ faça : (o c é parâmetro de ajuste do algoritmo e vai indicar o número de representantes que queremos detectar em w, isto é, o tamanho máximo de $w.rep$)

   (a) Inicializa-se maxDist:= 0

   (b) para cada objeto p do cluster w faça:

      i. testa se $i = 1$ ou não:
         - **se i = 1** : $minDist := d(p, w.mean)$
         - **se i > 1** : $minDist := min\{d(p, q)$: $q \in tmpSet\}$

      ii. se $minDist \geq maxDist$ então

      $maxDist := minDist$
      $maxPoint := p$ (este será um possível representante)

   (c) $tmpSet := tmpSet \cup \{$maxPoint$\}$

   Segundo este procedimento, o primeiro ponto p_1 escolhido como representante será aquele que estiver mais afastado do centro de w. O segundo ponto p_2 será aquele que estiver mais afastado de p_1. O terceiro ponto p_3 será aquele tal que a sua distância a $\{p_1, p_2\}$ é a maior possível. Aqui, distância de p_3 a $\{p_1, p_2\}$ é o mínimo entre $d(p_3, p_1)$ e $d(p_3, p_2)$.

5. Os elementos de $tmpSet$ são os representantes de w **antes da retração**. Para obter os representantes de w faz-se:

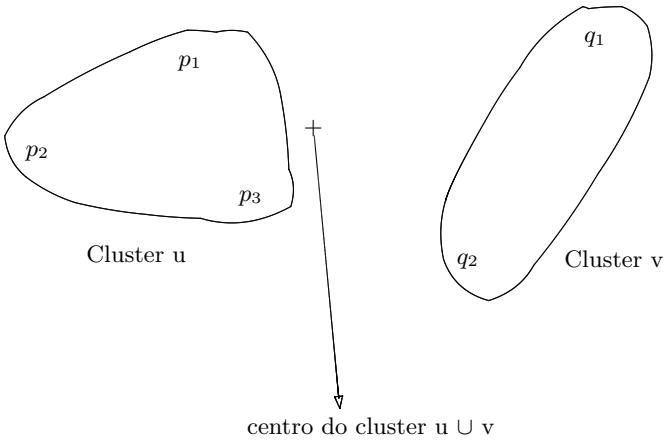
   - $w.rep := \emptyset$

- para cada elemento p de $tmpSet$ faz-se:

$$w.rep := w.rep \cup \{p + \alpha * (w.mean - p)\}$$

Para entender melhor como funciona o procedimento merge, consideramos o seguinte exemplo:

Exemplo 6.1 Suponhamos u e v dois clusters representados na figura abaixo:



Suponhamos que $c = 3$. O ponto "+" representa o centro do cluster aglutinado $w = u \cup v$.

tmpSet = $\emptyset$ e maxDist = 0;

1. **Iteração 1** : $i = 1$. Vamos calcular o primeiro representante de w. Como $i = 1$, pegamos ponto por ponto em w e calculamos sua distância até "+". Em maxDist teremos a maior destas distâncias. O ponto correspondente será denominado maxPoint e será inserido no conjunto tmpSet. No nosso exemplo, este primeiro ponto será q_1.

2. **Iteração 2** : $i = 2$. Vamos calcular o segundo representante de w. Como $i \neq 1$, vamos considerar cada ponto de w e calcular sua distância a q_1. Em maxDist teremos a maior destas distâncias. O ponto correspondente será denominado maxPoint e será inserido no conjunto tmpSet. No nosso exemplo, este segundo ponto será p_2.

3. **Iteração 3** : $i = 3$. Vamos calcular o terceiro representante de w. Como $i \neq 1$, vamos considerar cada ponto de w e calcular sua distância a $\{q_1,p_2\}$.

   - para p_1 temos que sua distância a $\{q_1,p_2\}$ = d(p_1, p_2).

- para p_2 temos que sua distância a $\{q_1,p_2\} = 0$
- para p_3 temos que sua distância a $\{q_1,p_2\} = \mathrm{d}(p_3, p_2)$.
- para q_1 temos que sua distância a $\{q_1,p_2\} = 0$.
- para q_2 temos que sua distância a $\{q_1,p_2\} = \mathrm{d}(q_2, q_1)$.

Em maxDist teremos a maior destas distâncias que é $\mathrm{d}(q_2, q_1)$, que corresponde ao ponto q_2. Logo, este será o terceiro ponto a ser inserido em tmpSet.

6.5 Análise comparativa de performance

A performance de CURE foi comparada com a performance dos seguintes algoritmos:

1. BIRCH : algoritmo de clusterização hierárquico [16]. Tem problemas para identificar clusters com formato não esférico ou que tenham grande variação de tamanho entre eles.

2. MST : método hierárquico simples que utiliza a distância entre os clusters como sendo a distância entre os pontos mais próximos entre os clusters. Este algoritmo é melhor do que CURE para detectar clusters de formato arbitrário. Mas é muito sensível a ruídos.

3. DBSCAN : algoritmo baseado em densidade [17], apresentado na aula passada. Ineficiente e difícil de produzir bons resultados caso se escolha uma amostra pequena dos dados, como é o caso de CURE. Também é bem mais sensível a ruídos do que CURE.

4. CURE : pode descobrir clusters de formatos interessantes e é menos sensível a ruídos do que MST. É possível escolher-se uma amostra adequada do banco de dados de modo que CURE, executado sobre esta amostra na memória principal, produza clusters corretos, isto é, a probabilidade de se perder clusters nos dados totais é bem pequena.

Capítulo 7

Análise de Outliers

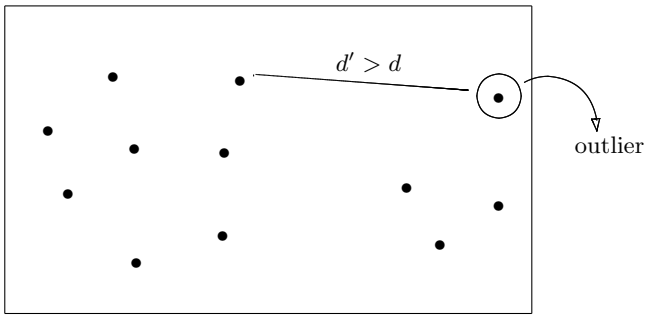
Estas notas tratam do problema de deteção de *outliers*, isto é, *exceções*, em grandes volumes de dados. A identificação de *outliers* pode levar à descoberta de conhecimento inesperado em áreas tais como comércio eletrônico, detecção de fraudes em cartões de crédito e mesmo a análise de performance de atletas profissionais (casos de dopagem, por exemplo). Apresentaremos dois algoritmos importantes para detecção de *outliers*. O primeiro (algoritmo NL) tem complexidade linear com respeito à dimensão, mas N^2 com respeito ao tamanho do banco de dados. O segundo (algoritmo **FindAllOuts**), tem complexidade linear com respeito ao tamanho do banco de dados mas exponencial com respeito à dimensão. Portanto, este último só é praticável para dimensões baixas.

7.1 Uma definição de Outlier baseada em distância

A maioria dos trabalhos sobre detecção de *outliers* utilizam técnicas estatísticas. Não existe uma definição formal de *outlier* aceita unanimemente por todos que trabalham nesta área, mas sim, uma noção informal proposta por D. Hawkins [21] : *"Um outlier é uma fato que desvia tanto de outros fatos a ponto de gerar suspeitas de que foi gerado por um mecanismo diferente"*. Esta noção informal captura o espírito do que se entende por *outlier*. Nestas notas de aula, vamos utilizar uma noção mais precisa de *outlier*, chamada de *DB-outlier*, que captura o mesmo espírito da noção informal de Hawkins.

Definição 7.1 Um objeto O de um banco de dados D é dito um *DB(p,d)-outlier* se pelo menos uma fração p $(0 < p < 1)$ dos objetos de D estão a uma distância maior do que d de O.

Por exemplo, na figura abaixo, o objeto circundado é um *DB(p,d)-outlier*, para $p = \frac{2}{3}$ e $d' > d$ (8 dos 12 objetos de D estão a uma distância maior do que d deste objeto circundado).



Banco de Dados D

A escolha dos parâmetros p e d, assim como o teste de validade (isto é, decidir se realmente um DB(p, d)-outlier é um outlier "real" de alguma significância) é tarefa de um especialista humano.

Para o leitor que deseja se aprofundar neste assunto, recomendamos a leitura do artigo [20], onde é feita uma discussão mais ampla sobre as justificativas desta definição de DB-outlier e em que situações uma tal definição pode ser utilizada com bons resultados para a detecção de outliers. Podemos adiantar que uma tal definição só é aplicável para situações onde se pode definir uma noção "razoável" de distância entre os objetos de um banco de dados, isto é, a distância entre dois objetos é um número que tem algum significado dentro da aplicação.

7.2 Algoritmo NL

Sejam D um banco de dados, p um número entre 0 e 1 e $d > 0$. Seja N o número de elementos do banco de dados D e seja dist a função distância considerada. Lembramos a definição que foi dada na aula 14 sobre d-vizinhança[1] : a d-vizinhança de um objeto O é o conjunto de pontos cuja distância a O é no máximo d (i.e. $\{Q \in D \mid \text{dist}(O, Q) \leq d\}$). A fração p é a fração mínima de objetos de D que devem ficar fora de uma d-vizinhança de um outlier. Seja M o número máximo de objetos dentro da vizinhança de um outlier, isto é $M = N(1 - p)$.

Um algoritmo óbvio para detecção de outliers

Pela formulação acima é claro que o problema de encontrar todos os $DB(p, d) - outliers$ se reduz a um problema de se encontrar, para cada objeto O de D, uma vizinhança contendo no máximo M elementos. Isto pode ser resolvido por um algoritmo simples que utiliza uma estrutura de índice multidimensional para armazenar D. Executa-se

[1] utilizamos o termo ϵ-vizinhança naquela aula

uma busca dentro de um raio d para cada objeto O de D. No momento em que $M + 1$ elementos são encontrados nesta vizinhança, a busca termina e O é declarado um *não-outlier*. Os objetos que sobram são os *outliers*. O custo da construção da estrutura de índice, mesmo não levando em consideração o custo da busca, tornam este tipo de algoritmo não competitivo com relação a outros algoritmos de deteção de outliers.

Um algoritmo mais eficiente que minimiza o custo de I/0

Para evitar o custo da construção da estrutura de índice a fim de encontrar todos os $DB(p, d)-outliers$, o algoritmo NL que vamos descrever agora, utiliza uma idéia diferente, baseada em blocos e num "loop entrelaçado" (Nested Loop, daí o nome do algoritmo NL). Este algoritmo se preocupa sobretudo em minimizar o custo de I/O. Embora sua complexidade seja O($k * N^2$) (onde k = número de atributos do banco de dados), ele é mais eficiente quando comparado a algoritmos que não se preocupam com o custo de I/O, como é o caso dos algoritmos tipo "força bruta" descritos no parágrafo anterior.

Suponhamos que a capacidade do buffer disponível é de $B\%$ do tamanho do banco de dados. Dividimos o buffer em duas partes, cada uma delas contendo um array que será preenchido com objetos do banco de dados, no decorrer da execução do algoritmo.

Algoritmo NL

1. Preencha o primeiro array do buffer com um bloco de tuplas de D de tamanho $B/2\%$ do banco de dados.

   O próximo passo vai detectar outliers nos dados do primeiro array.

2. Para cada tupla t do primeiro array, faça:

   (a) Contador := 0

   (b) Para cada tupla s do primeiro array, se dist$(t, s) \leq d$:

   - Contador := Contador + 1
   - Se Contador $> M$, marque t com um não-outlier e vá para a próxima tupla do primeiro array.

3. Para cada um dos outros blocos X restantes, faça:

   (a) Prencha o segundo array com o bloco X ;

   (b) Para cada tupla t não marcada do primeiro array (um candidato a outlier) faça:

- Para cada tupla s do segundo array, se $\text{dist}(t, s) \leq d$:
  - Contador := Contador + 1;
  - Se Contador $> M$, marque t com um não-outlier e vá para a próxima tupla não marcada do primeiro array.

4. Retorne todas as tuplas não marcadas do primeiro array, como sendo outliers.
5. Se o bloco do segundo array já foi bloco do primeiro array, páre. Senão, troque os conteúdos do primeiro e segundo array e volte para o passo 2.

Vamos ilustrar a execução do algoritmo no exemplo abaixo:

Exemplo 7.1 Suponha que a capacidade do buffer seja de 50% do tamanho do banco de dados. Assim, vamos dividir o banco de dados em 4 blocos (já que cada metade do buffer será preenchida com um bloco). Vamos denominar tais blocos A,B,C,D;

1. Primeiro o bloco A vai para o primeiro array e seus elementos são comparados dois a dois. Alguns deles são marcados como não-outliers. Em seguida, compara-se A com B, depois com C, depois com D. Estes blocos B,C,D sucessivamente vão preencher o segundo array. Em cada uma destas comparações, o contador dos elementos de A continua a ser incrementado, e eventualmente elementos não-marcados de A ficam marcados como não-outliers.

   Número total de acessos ao banco de dados = 4 (um para cada bloco)

2. Depois que o último bloco D é comparado com A, troca-se o conteúdo dos dois arrays. No primeiro array teremos o bloco D e no segundo o bloco A. Todos os passos acima são repetidos: D é comparado com D, depois com A, depois com B e depois com C.

   Número total de acessos ao banco de dados = 2 (um para o bloco B e outro para o bloco C)

3. Depois que o último bloco C é comparado com D, troca-se o conteúdo dos dois arrays. No primeiro array teremos o bloco C e no segundo o bloco D. Todos os passos acima são repetidos: C é comparado com C, depois com D, depois com A e depois com B.

   Número total de acessos ao banco de dados = 2 (um para o bloco A e outro para o bloco B)

4. Depois que o último bloco B é comparado com C, troca-se o conteúdo dos dois arrays. No primeiro array teremos o bloco B e no segundo o bloco C. Todos os passos acima são repetidos: B é comparado com B, depois com C, depois com A e depois com D.

   Número total de acessos ao banco de dados = 2 (um para o bloco A e outro para o bloco D)

5. O próximo passo seria trocar o conteúdo dos dois arrays: no primeiro iria o bloco D e no segundo o bloco B. Mas o bloco D já apareceu no primeiro array. Logo, a condição de parada é verifica e o algoritmo pára.

Neste exemplo, o total de acessos ao banco de dados foi de 10. Em cada acesso um bloco foi lido. Como o banco de dados tem 4 blocos, o número de varridas ao banco de dados foi de $10/4 = 2{,}5$.

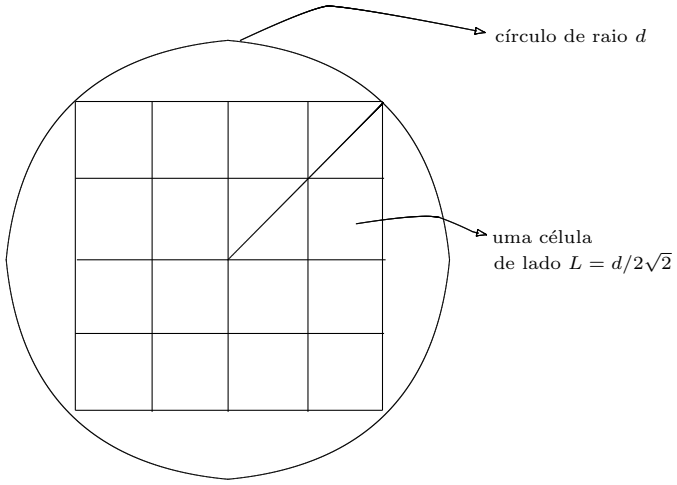
7.3 Enfoque baseado em células : o algoritmo FindAllOutsM

Nestas notas de aula, veremos apenas uma versão simplificada do algoritmo **FindAllOutsM** que assume que tanto a estrutura multidimensional de células quanto o banco de dados cabem na memória principal. Uma extensão deste algoritmo que não faz esta hipótese simplificadora pode ser encontrada no artigo [20]. A idéia é semelhante nos dois algoritmos, a única diferença sendo a maneira como o segundo algoritmo manipula os dados residentes no disco.

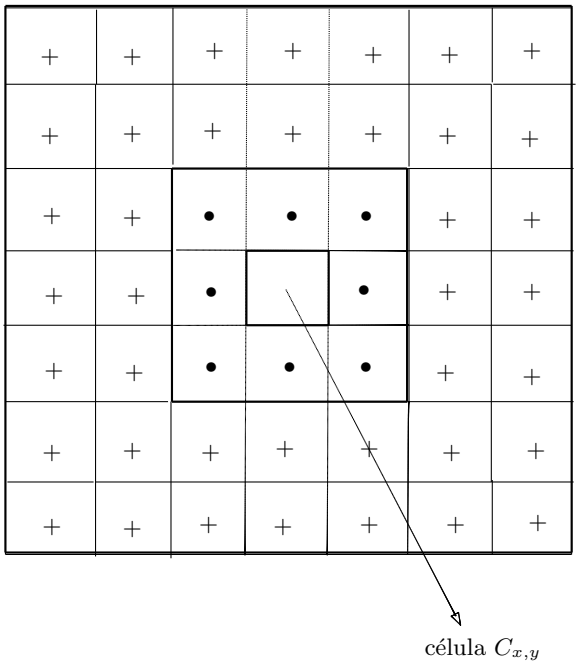
Primeiramente, a fim de melhor expor a idéia do algoritmo, suporemos que os dados são bidimensionais, isto é, o banco de dados só tem 2 atributos. Depois, faremos uma breve explanação de como a idéia pode ser generalizada para o caso k-dimensional, onde $k > 2$.

7.3.1 A estrutura de células e propriedades da mesma no caso 2-dimensional

Vamos dividir o plano num reticulado, onde cada célula (ou quadrado) tem lado $L = \frac{d}{2*\sqrt{2}}$. Para se ter uma idéia da ordem de grandeza de L comparado a d, observe a figura abaixo:



Uma célula centrada no ponto (x, y) é denotada $C_{x,y}$. Para cada célula $C_{x,y}$, consideramos sua vizinhança de nível 1, que denotamos $L_1(C_{x,y})$, e sua vizinhança de nível 2, que denotamos $L_2(C_{x,y})$, como mostra a figura abaixo. A vizinhança $L_1(C_{x,y})$ tem 8 células marcadas com $\bullet$. A vizinhança $L_2(C_{x,y})$ tem 40 células marcadas com $+$. Repare que a vizinhança $L_1(C_{x,y})$ tem "espessura" 1 e a vizinhança $L_2(C_{x,y})$ tem "espessura" 2.



As seguintes propriedades são importantes para a compreensão do algoritmo **FindAllOutsM**:

- **Propriedade 1** : a máxima distância entre 2 elementos numa mesma célula é de $d/2$.

  Isto é claro, pois a diagonal mede $L * \sqrt{2} = \frac{d}{2\sqrt{2}} * \sqrt{2} = \frac{d}{2}$.

- **Propriedade 2** : Sejam p um objeto de $C_{x,y}$ e q um objeto de sua vizinhança de nível 1. Então $\text{dist}(p, q) \leq d$.

  Isto é claro, pois a máxima distância é o dobro da diagonal da célula, que vale $2L\sqrt{2} = 2 * \frac{d}{2*\sqrt{2}}\sqrt{2} = d$.

- **Propriedade 3** : Seja um objeto qualquer fora de $C_{x,y}$, fora da vizinhança de nível 1 de $C_{x,y}$ e fora da vizinhança de nível 2 de $C_{x,y}$. Então, $\text{dist}(p, q) > d$, para todo objeto q de $C_{x,y}$.

  De fato, a distância entre os dois pontos deve ser maior do que a soma das "espessuras" das camadas L_1 e L_2 que é $3 * L = \frac{3d}{2\sqrt{2}} > d$.

- **Propriedade 4** :

  1. Se existir um número maior do que M objetos em $C_{x,y}$ então nenhum dos objetos em $C_{x,y}$ é um outlier.

     De fato, devido à propriedade 1, toda d-vizinhança de um ponto de $C_{x,y}$ contém todo a célula $C_{x,y}$ e portanto contém mais do que M objetos.

  2. Se existir um número maior do que M objetos em $C_{x,y} \cup L_1(C_{x,y})$ então nenhum dos objetos em $C_{x,y}$ é um outlier.

     De fato, devido à propriedade 2, toda d-vizinhança de um ponto de $C_{x,y} \cup L_1(C_{x,y})$ contém todo o conjunto $C_{x,y} \cup L_1(C_{x,y})$ e portanto contém mais do que M objetos.

  3. Se o número de objetos em $C_{x,y} \cup L_1(C_{x,y}) \cup L_2(C_{x,y})$ é $\leq M$ então **todo** objeto da célula $C_{x,y}$ é um outlier.

     De fato, seja p um objeto de $C_{x,y}$ e consideremos uma d-vizinhança de p. Devido à propriedade 3, todo objeto desta vizinhança deve estar em $C_{x,y} \cup L_1(C_{x,y}) \cup L_2(C_{x,y})$. Mas em $C_{x,y} \cup L_1(C_{x,y}) \cup L_2(C_{x,y})$ só existem no máximo M objetos, portanto $M - 1$ objetos distintos de p. Logo, p é um outlier.

7.3.2 O algoritmo FindAllOutsM no caso 2-dimensional

Seja m o número de células. Este número é facilmente calculado em função de d e do tamanho do banco de dados.

1. Para cada $q = 1, \ldots, m$, faça $\text{Contador}_q = 0$ (vamos contar quantos elementos existem em cada célula).

2. Para cada objeto P do banco de dados, associe a P uma célula C_q (na qual está inserido) e aumente o Contador_q de 1. Depois deste passo, todos os objetos do banco de dados foram distribuídos nas células correspondentes e o contador de cada célula diz quantos objetos existem nas mesmas. Observe que é possível que P esteja na fronteira entre duas células. Neste caso, P é associado apenas a uma das células.

   Os passos seguintes vão determinar quais objetos p são outliers, verificando apenas o número de elementos de sua célula C_p, da vizinhança L_1 e da vizinhança L_2 de C_p. Alguns (poucos) objetos restarão no final para serem testados individualmente, isto é, para os quais será necessário calcular o número de objetos que estão em sua d-vizinhança.

   A idéia é a seguinte: (1) células que tiverem mais de M elementos serão etiquetadas **Vermelha** e (2) células na vizinhança de uma célula vermelha serão etiquetadas de **Azul**. Células coloridas não têm chance nenhuma de conter *outliers* (devido respectivamente às propriedades 4(1) e 4(2)). Resta analisar os elementos de células **Brancas**. (3) Para cada célula branca serão primeiramente contabilizados os elementos da célula junto com o de sua vizinhança L_1. Caso este número de elementos for maior do que M, esta célula **Branca** será etiquetada de **Azul**, não tendo nenhuma chance de conter *outliers* (devido à propriedade 4(2)). (4) Para as demais células brancas, contabilizamos o total de elementos da célula, junto com os elementos de suas vizinhanças L_1 e L_2. Caso este número de elementos for $\leq M$, com certeza todos os elementos desta célula branca serão *outliers* (devido à propriedade 4(3)) .(5) Somente para aquelas células brancas para as quais esta soma total de elementos (junto com as duas vizinhanças) for $> M$ faremos a checagem ponto por ponto da célula.

3. Para cada $q = 1, \ldots, m$, se $\text{Contador}_q > M$, etiquete a célula C_q de **Vermelha**. As células não coloridas são etiquetadas **Branca**.

4. Para cada célula **Vermelha**, verifique as células em sua vizinhança L_1: aquelas que não foram etiquetadas de **Vermelha** ainda, serão etiquetadas de **Azul**.

5. Para cada célula **Branca** C_w faça:

(a) Contador$_{w1}$:= Contador$_w$ + $\Sigma_{i \in L_1(C_w)}$ Contador$_i$;

Conta-se o número de objetos de C_w e o número dos objetos de sua vizinhança L_1;

(b) Se Contador$_{w1} > M$ então a célula C_w é etiquetada de **Azul**;

(c) Se Contador$_{w1} \leq M$, faça:

i. Contador$_{w2}$:= Contador$_{w1}$ + $\Sigma_{i \in L_2(C_w)}$ Contador$_i$;

ii. Se Contador$_{w2} \leq$ M, marque todos os objetos de C_w como *outliers*;

iii. Se Contador$_{w2} >$ M : para cada objeto $P \in C_w$ faça:

- Contador$_P$:= Contador$_{w1}$;
- Para cada objeto $Q \in L_2(C_w)$:
  se dist(P,Q) $\leq d$: Contador$_P$:= Contador$_P$ + 1;
  Se Contador$_P > M$, P não pode ser um outlier. Vá para o passo 5(c)(iii), isto é, passe a analisar outro objeto P'.
- Marque P como sendo um outlier.

7.3.3 Caso k-dimensional

Quando passamos do caso bidimensional para o caso k-dimensional, para $k > 2$, somente algumas mudanças devem ser feitas nas dimensões da estrutura de células k-dimensionais de modo que as propriedades 1 a 4 continuem valendo. Veja que o algoritmo **FindAllOutsM** é baseado nestas propriedades. A primeira mudança envolve o tamanho das células k-dimensionais. A segunda mudança envolve a "espessura" da vizinhança L_2.

Exercício 1 Lembre-se que no caso bidimensional, $L = \frac{d}{2\sqrt{2}}$. Como a diagonal de um hipercubo k-dimensional de lado L é $L\sqrt{k}$, mostre que o comprimento L, no contexto k-dimensional, deverá ser $\frac{d}{2\sqrt{k}}$ a fim de preservar as propriedades (1) e (2).

Exercício 2 Lembre-se que no caso bidimensional, a espessura da vizinhança L_2 é 2. Mostre que, a fim de que todas as propriedades 1 a 4 continuem válidas no caso k-dimensional, a "espessura" da vizinhança L_2 deve ser o primeiro inteiro maior ou igual a $2\sqrt{k} - 1$.

Veja artigo [20], página 397-398, sugestões para resolver os exercícios acima.

7.4 Discussão Final : análise de complexidade e comparação dos dois algoritmos

A complexidade do algoritmo **FindAllOutsM** é O(c^k + N), onde N é o tamanho do banco de dados e k é a dimensão. Portanto, para dimensões pequenas (≤ 4), esta complexidade é razoável. Além disto, fixada a dimensão do banco de dados, a complexidade é linear no tamanho do mesmo. A versão do algoritmo que trata o caso em que os dados são armazenados em disco (**FindAllOutsD**) também apresenta bons resultados de performance: cada página de dados não é acessada mais do que 3 vezes. Os resultados empíricos do algoritmo mostram que:

1. os algoritmos baseados em estruturas de célula são bem superiores aos outros algoritmos para dimensões $k \leq 4$.

2. o algoritmo NL é a melhor escolha para dimensões $k > 4$.

No artigo [20], seção 6, o leitor pode encontrar uma análise comparativa completa de performance dos algoritmos NL e **FindAllOutsM** e **FindAllOutsD**, quando se varia diversos parâmetros : o tamanho do banco de dados, a dimensão k, o número p e d.

Capítulo 8

Web Mining

Devido à grande quantidade de informação disponível na Internet, a Web é um campo fértil para a pesquisa de mineração de dados. Podemos entender Web Mining como uma *extensão* de Data Mining aplicado a dados da Internet. A pesquisa em Web Mining envolve diversos campos de pesquisa em computação tais como: bancos de dados, recuperação de informação e inteligência artificial (aprendizado de máquina e linguagem natural sobretudo). Nestas notas de aula pretendemos falar de forma bem geral sobre o que é Web Mining, suas principais tarefas e as três principais categorias em que se subdivide esta extensa área de pesquisa. Alguns artigos de carácter geral que constituem leitura complementar para esta aula são [22], [26], [23], [24], [25], [27], [28].

8.1 O que é Web Mining

Web Mining é o uso de técnicas de data mining para descobrir e extrair automaticamente informações relevantes dos documentos e serviços ligados a Internet. Este campo de pesquisa é extremamente extenso e por isso, muita confusão surge quando se discute sobre as tarefas centrais de Web Mining. Web Mining é frequentemente associado com "Recuperação de Informação", mas na verdade trata-se de um processo mais amplo, interdisciplinar, envolvendo técnicas de Recuperação de Informação, estatística, inteligência artificial e mineração de dados. Em geral, concorda-se que as tarefas principais de Web Mining são as seguintes:

1. **Busca de documentos:** consiste em se encontrar sites Web contendo documentos especificados por palavras-chave. É o processo de se extrair dados a partir de fontes de textos disponíveis na Internet, tais como conteúdos de textos de documentos HTML obtidos removendo-se os tags HTML, textos extraídos de grupos de

discussão, newsletters, etc. Esta tarefa envolve a utilização de técnicas de Recuperação de Informação.

2. **Seleção e pré-processamento da informação:** consiste em selecionar e pré-processar automaticamente informações obtidas na Internet. O pré-processamento envolve qualquer tipo de transformação da informação obtida na busca, como por exemplo, podagem de textos, transformação da representação da informação em outros formalismos, tais como fórmulas da Lógica de Primeira Ordem.

3. **Generalização:** consiste em descobrir automaticamente padrões gerais em sites Web ou entre vários sites Web. Esta tarefa envolve a utilização de técnicas de inteligência artificial e de mineração de dados.

4. **Análise:** validação e interpretação dos padrões minerados.

Dados estruturados, não-estruturados e semi-estruturados

Dados *semi-estruturados* são um ponto de convergência entre as *comunidades* da Web e de Banco de Dados: os primeiros lidam com *documentos*, os segundos com *dados estruturados*. Documentos são dados não estruturados, em oposição aos dados rigidamente estruturados que são manipulados em bancos de dados. Entende-se por dados estruturados, aqueles que são armazenados em estruturas bem definidas (esquemas), como é o caso dos bancos de dados relacionais, onde um esquema composto por nomes de tabelas e atributos (em cada tabela) é especificado e armazenado logo na criação de qualquer banco de dados. Esta estrutura rígida está evoluindo para os chamados *dados semi-estruturados* (documentos XML), a fim de permitir uma representação mais natural de objetos complexos do mundo real, tais como livros, artigos, filmes, componentes de aviões, projetos de chips, etc, sem que o projetista da aplicação em questão tenha que se contorcer para poder encontrar uma representação adequada para os dados(veja [26]).

Esta forma de estruturas irregulares de dados naturalmente encoraja a adaptação ou extensão (e posterior aplicação) das técnicas conhecidas de mineração de dados estruturados, a fim de se descobrir padrões úteis e interessantes em dados semi-estruturados.

8.2 As categorias de Web Mining

Web Mining normalmente é subdividido em três categorias principais, que constituem as áreas de interesse onde minerar informação:

- Web Content Mining ou Mineração do Conteúdo de Documentos na Web.

- Web Log Mining (Web Usage Mining) ou Mineração do Uso da Web.
- Web Structure Mining ou Mineração da Estrutura de Documentos na Web.

8.2.1 Minerando o Conteúdo da Web

O conteúdo da Web consiste basicamente de diversos tipos de dados tais como textos, imagens, sons, vídeo, hiperlinks. Assim, a *Mineração do Conteúdo da Web* envolve a descoberta de conhecimento em diferentes tipos de dados, o que normalmente é chamado de *Mineração de Dados Multimídia*. Logo, podemos considerar a mineração de dados multimídia como uma subárea da Mineração do Conteúdo da Web. Entretanto, o que recebe mais atenção dentro deste vasto campo de pesquisa é a mineração do conteúdo de textos e hipertextos. Os dados que compõem o conteúdo da Web consistem de dados não-estruturados do tipo textos, de dados semi-estruturados do tipo documentos HTML e dados estruturados tais como dados contidos em bancos de dados acessados pelas páginas. Entretanto, boa parte do material na Web é constituído de textos (dados não-estruturados). A pesquisa que consiste em aplicar técnicas de mineração para descobrir conhecimento escondido em textos é chamada *Text Mining*. Assim, *text mining* é uma subárea de *Mineração do Conteúdo da Web*.

Existem dois pontos de vistas principais quando se fala de *Mineração do Conteúdo da Web*: o ponto de vista da "Recuperação de Informação" (RI) e o ponto de vista de "Banco de Dados" (BD).

O objetivo, **sob o ponto de vista de RI**, é auxiliar o usuário no processo de busca ou filtragem de informação. É o que realiza os principais mecanismos de busca na Internet ao procurar atender da melhor maneira possível as solicitações feitas por usuários através de palavras-chave.

O objetivo, sob o **ponto de vista de BD**, é modelar os dados da Web e integrá-los de tal modo que consultas mais sofisticadas do que simplesmente consultas baseadas em palavras-chave possam ser feitas. Isto pode ser realizado descobrindo-se os esquemas dos documentos na Web, construindo-se *Web Warehouses* ou uma base de conhecimento de documentos (ver [28]). A pesquisa nesta área lida sobretudo com dados semi-estruturados (XML). Dados semi-estruturados se referem a dados que possuem alguma estrutura mas não esquemas rígidos como é o caso dos bancos de dados. Um artigo interessante e introdutório nesta área é [27].

8.2.2 Minerando o uso da Web

A Mineração do uso da Web tenta descobrir regularidades nos caminhos percorridos pelos usuários quando estão navegando pela Web. Enquanto a Mineração do Conteúdo e a Mineração da Estrutura utilizam os dados reais presentes nos documentos da Internet,

a Mineração do Uso utiliza dados secundários derivados da interação do usuário com a Web. Tais dados secundários incluem registros de log de servidores de acesso a Web (daí o nome alternativo "Web Log Mining"), registros de log de servidores proxy, perfis de usuários, transações do usuário, consultas do usuário, dados de arquivos "Bookmarks" (Favoritos), etc.

8.2.3 Minerando a Estrutura de Documentos na Web

A Mineração da Estrutura de Documentos na Web tenta descobrir o modelo subjacente à estrutura de links da Web. O modelo é baseado na topologia dos hiperlinks. Este modelo pode ser utilizado para classificar páginas Web e é útil para gerar informações tais como *a similaridade ou relacionamentos* entre diferentes sites Web. Esta categoria de mineração na Web pode ser utilizada para se descobrir por exemplo os sites *de autoridade* (authority sites), isto é, sites cujos links aparecem frequentemente em outros sites (veja [25] para um tutorial no assunto).

É bom ressaltar que a distinção entre estas três categorias não é totalmente clara, isto é, uma das categorias (por exemplo, Mineração do Conteúdo) pode utilizar links (objeto principal da Mineração de Estrutura) e mesmo perfis de usuário (um dos objetos centrais da Mineração do Uso).

Neste curso vamos nos interessar apenas pela segunda categoria, isto é, Mineração do uso da Web, já que as outras duas envolvem áreas de pesquisa em computação que estão fora do escopo deste curso (recuperação de informação, extração da informação, XML, mineração de textos, mineração de dados multimídia). Para isto, vamos adaptar técnicas de mineração de sequências que vimos no curso a fim de minerar caminhos frequentemente utilizados pelos usuários ao navegar num certo portal, por exemplo.

8.3 Mineração de padrões em caminhos percorridos por usuários da Internet

Nesta aula vamos tratar do problema de mineração de caminhos percorridos por usuários que navegam num site da internet. A referência principal para esta aula é [29].

Descobrir padrões de comportamento frequentes na navegação dos usuários de um site não somente pode ajudar a melhorar a arquitetura do site (isto é, fornecer acessos eficientes entre objetos altamente correlacionados) mas também ajudar a tomar decisões apropriadas no que diz respeito à distribuição de material publicitário no site.

Os caminhos percorridos pelos usuários são armazenados num arquivo de logs, num servidor especial da organização que gerencia o site em questão. Os dados armazenados

são registros da forma :

$$\begin{array}{ccc} u_1 & (s_1, d_1) & t_1 \\ u_2 & (s_2, d_2) & t_2 \\ \vdots & \vdots & \vdots \end{array}$$

onde u_i é um identificador do usuário (seu IP, por exemplo), o par (s_i, d_i) (chamado *refer log*) é constituído de dois endereços : (1) o endereço *fonte* s_i que é o endereço da página anterior visitada pelo usuário, e (2) o endereço *destino* que é o endereço da página atualmente visitada pelo usuário. t_i é o tempo correspondente à operação de passagem de uma página para outra.

Os usuários podem caminhar através das diferentes páginas para frente e para trás de acordo com os links e ícones presentes no interior de cada página. Como consequência, uma determinada página pode ser revisitada somente por causa de sua localização e não pelo seu conteúdo. Por exemplo, quando um usuário está navegando na Web (não necessariamente dentro de um site, mas entre sites diversos), frequentemente ele utiliza o botão "para trás (Back)" e depois para frente a fim de atingir uma determinada página, ao invés de abrir a página diretamente (teclando seu endereço no campo de entrada). Assim, a fim de extrair padrões de acesso significantes, vamos primeiramente eliminar estes *caminhos de volta* de modo a descobrir somente os padrões em caminhos que realmente interessam. Desta maneira, vamos assumir que todos os acessos "para trás" são feitos somente por uma questão de facilidade de navegação e não por razões de interesse do usuário pelo conteúdo das páginas revisitadas. Vamos nos concentrar portanto somente em descobrir *caminhos para frente*.

8.3.1 Formalização do Problema

Cada vez que um usuário inicia sua visita num site, um registro do tipo $(null, d)$ é inserido no arquivo de logs, onde d é o endereço da página principal (Home) do site. No momento em que o usuário entra num outro site, um novo registro do tipo $(null, d')$ é armazenado, indicando que a sessão de visita do site anterior se encerrou e uma nova sessão está se iniciando no site d'.

Vamos considerar o seguinte exemplo ilustrado na figura abaixo:

Esta figura corresponde à navegação de um usuário através do site cujo endereço principal (home) é A. No arquivo de log, teremos o seguinte caminho correspondente a cada uma das páginas visitadas pelo usuário dentro do site (deste sua entrada no site até sua saída):

$$< A, B, C, D, C, B, E, G, H, G, W, A, O, U, O, V >$$

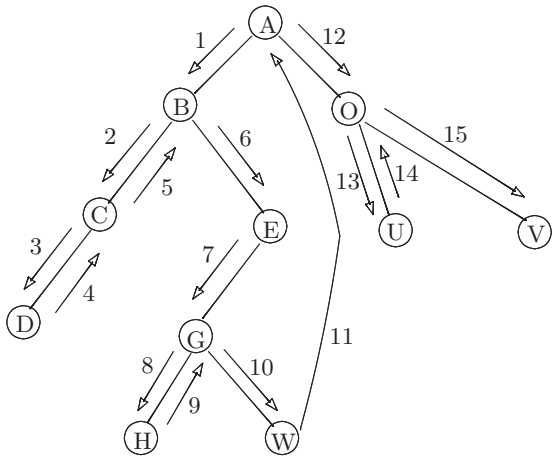


Figura 8.1: Uma sequência de páginas visitadas por um usuário

É suposto que após a visita da página V, o usuário sai da Internet, ou entra num outro site, iniciando uma nova série de registros de log com o primeiro registro $(null, d')$ correspondendo ao endereço d' do novo site.

Primeiramente, vamos transformar esta sequência de páginas visitadas num conjunto de sequências correspondendo somente às visitas *para frente*. As voltas para páginas anteriormente visitadas são eliminadas. O resultado da transformação é o seguinte conjunto de sequências:

$$\{< A, B, C, D >, < A, B, E, G, H >, < A, B, E, G, W >, < A, O, U >, < A, O, V >\}$$

Tais sequências *para frente*, correspondentes a uma única sequência de acessos do usuário é chamada de *sequências maximais para frente*.

Assim, o banco de dados de registros de log, pode ser visto como um banco de dados de sequências maximais para frente. Repare que, cada usuário possui diversas sequências maximais para frente (o identificador do usuário não é chave).

Uma *sequência de referências* é uma sequência $s =< s_1, s_2, ..., s_n >$ onde os s_i são símbolos que mapeiam endereços de páginas dentro de um site. Uma sequência de referências $s =< s_1, s_2, ..., s_n >$ é suportada por uma sequência maximal para frente $t =< t_1, t_2, ..., t_m >$ se existe $i = \{1, ..., m\}$ tal que $s_1 = t_i, s_2 = t_{i+1}, ..., s_m = t_{i+n-1} >$. O *suporte* de uma sequência de referências s é dado por:

$$sup(s) = \frac{\text{número de seq. max. para frente que suportam } s}{\text{total de seq. max para frente}}$$

Uma sequência de referências é dita *frequente* se seu suporte é maior ou igual a um nível mínimo de suporte dado.
O nosso problema é o seguinte: dado um banco de dados D de sequências maximais para frente e um nível de suporte mínimo α entre 0 e 1, descobrir todas as sequências de referências frequentes com relação a D e a α.

Observamos as seguintes diferenças entre este problema e o problema clássico de mineração de sequências:

- um usuário pode ser contado diversas vezes no cálculo do suporte de uma sequência de referências. Isto não acontece no cálculo do suporte de sequências de itemsets, pois o identificador do usuário é chave do banco de dados de sequências.
- para que uma sequência s esteja contida numa sequência t é necessário que os símbolos de s ocorram consecutivamente em t. Isto não é exigido na definição de inclusão de duas sequências de itemsets.
- as sequências de referências não possuem elementos repetidos, o que não acontece em geral para sequências de itemsets: um mesmo itemset pode ser comprado em diferentes momentos pelo mesmo usuário.

8.3.2 Esquema geral do processo de mineração das sequências de referências frequentes

O processo para determinar as sequências de referências frequentes possui as seguintes fases:

1. **Transformação do BD**: Nesta fase, as sequências de visitas de cada usuário são transformadas num conjunto de sequências maximais para frente. Esta tarefa será executada pelo algoritmo MF que descreveremos na seção seguinte.
2. **Cálculo das sequências de referências frequentes**: Nesta fase, serão mineradas as sequências de referências frequentes sobre o banco de dados transformado. Serão fornecidos dois algoritmos para esta tarefa, o algoritmo FS (Full Scan) e o algoritmo SS (Selective Scan). Estes algoritmos serão o tema principal da próxima aula.
3. **Cálculo das Sequências de referências Maximais**: Uma sequência de referências maximal é uma sequência de referências frequente que não está contida em nenhuma outra sequência maximal de referência. Por exemplo, suponha que :

   $\{< A, B >, < B, E >, < A, D >, < C, G >, < G, H >, < B, G >\}$ é o conjunto das 2-sequências de referêcias frequentes e

$\{< A, B, E >, < C, G, H >\}$ é o conjunto das 3-sequências de referências frequentes.

Então, as sequências maximais de referências são :

$\{< A, D >, < B, G >, < A, B, E >, < C, G, H >\}$

A sequência $< A, B >$ não é maximal, pois está contida na sequência maximal $< A, B, E >$. As sequências maximais de referências são os caminhos importantes de acesso frequentes.

Obter as sequências maximais dentre o conjunto das sequências frequentes é tarefa fácil e é proposto como exercício para o usuário:

Exercício 1: Dado um conjunto de sequências, desenvolver um algoritmo para determinar o conjunto de sequências maximais.

8.3.3 Algoritmo MF : encontrando referências maximais para frente

Vamos descrever o algoritmo MF (Maximal Forward) responsável pela fase de transformação do banco de dados, onde cada sequência é transformada num conjunto de sequências maximais para frente. Lembramos que cada registro referente ao usuário u é representado por um par (s, d), onde s é a página anterior, e d á página atual. Assim, cada sequência do usuário u é uma sequência do tipo:

$$< (s_1, d_1), ..., (s_n, d_n) >$$

O algoritmo MF vai escanear este arquivo de sequências e para cada uma delas construir um conjunto de sequências maximais para frente. Estas sequências maximais para frente serão armazenadas num banco de dados D_F.

Algoritmo MF

- **Passo 1**: Inicializamos $i := 1$ e $Y := \epsilon$ (string vazio). A variável Y é utilizada para armazenar os caminhos maximais para frente. Também inicializamos $F := 1$, uma bandeira que diz se o caminho está sendo percorrido para frente (1) ou para trás (0).

- **Passo 2**:
  $A := s_i$;
  $B := d_i$;
  Se $A =$ null (estamos entrando na página principal de um novo site e começando a navegação)

  - Se o conteúdo de Y não for nulo, escreva o conteúdo de Y no banco de dados D_F;
  - $Y := B$;
  - Vá para o **Passo 5**.

- **Passo 3**: Se B é igual a algum símbolo do string Y (o j-ésimo) então: (esta é uma página que já foi visitada)

  - Se $F = 1$ escreva o conteúdo de Y no banco de dados D_F;
  - Suprima todas as referências após j em Y;
  - $F := 0$;
  - Vá para o **Passo 5**.

- **Passo 4**:

  - Caso contrário, acrescente B no final de Y (estamos continuando o percurso para a frente);
  - Se $F = 0$ então faça $F := 1$;

- **Passo 5**:
  $i := i + 1$;
  Se a sequência ainda não terminou de ser escaneada, vá para o **Passo 2**.

Consideremos a sequência ilustrada na figura 1. Podemos verificar que a primeira referência para trás é encontrada no quarto movimento (indo de D para C). Neste ponto, a variável Y contém $ABCD$. Este string é escrito no banco de dados D_F (Passo 3). E na variável Y teremos apenas o string AB, já que o string CD será suprimido de Y. No próximo movimento (isto é, de C para B), embora a primeira condição do passo 3 se verifique, nada é escrito em D_F, pois a bandeira $F = 0$, significando que estamos caminhando para trás. Os próximos passos para frente vão colocar o string $ABEGH$ em Y, que é então escrito em D_F quando um movimento para trás (de H para G) faz com que uma referência já visitada seja encontrada (G).

A tabela 1 abaixo, descreve a execução do algoritmo MF sobre a sequência da figura 1:

movimento	string Y	output para D_F
1	AB	–
2	ABC	–
3	$ABCD$	–
4	ABC	$ABCD$
5	AB	–
6	ABE	–
7	$ABEG$	–
8	$ABEGH$	–
9	$ABEG$	$ABEGH$
10	$ABEGW$	–
11	A	$ABEGW$
12	$A0$	–
13	AOU	–
14	AO	AOU
15	AOV	AOV

Adaptação para visitas na Web em geral

Os registros de log de visitas em diferentes sites da Internet só contém a referência destino, ao invés do par (fonte,destino). A sequência de páginas visitadas terá a forma $< d_1, d_2, ..., d_n >$ para cada usuário. Mesmo com um input desta forma, podemos convertê-lo para um conjunto de sequências maximais para a frente. A única diferença é que, neste caso, não podemos identificar o ponto de "quebra", quando o usuário passa de um site para outro. Por exemplo, $ABEH$ e $WXYZ$ podem ser tratadas como um único caminho $ABEHWXYZ$, embora $ABEH$ e $WXYZ$ representem visitas em sites distintos. Certamente, o fato de não se ter a referência "fonte" para indicar a passagem de um site para outro, pode aumentar a complexidade computacional do algoritmo, pois os caminhos podem ser muito longos. Entretanto, este fato tem pouco efeito sobre a identificação de sequências frequentes. De fato, uma vez que não há nenhum link lógico entre H e W, uma subsequência contendo HW tem pouca chance de ocorrer com frequência.

Exercício 2 : Adapte o algoritmo MF para operar sobre sequências do tipo $< d_1, d_2, d_3, ..., d_n >$, onde não há como saber os pontos onde o usuário passou de um site para outro.

8.4 Os algoritmos FS e SS

Nesta aula, vamos dar os detalhes dos algoritmos FS e SS para mineração de padrões em caminhos percorridos por usuários da Internet. Tais algoritmos utilizam uma versão otimizada do algoritmo Apriori (o algoritmo DHP [30]), para mineração de sequências de endereços. Repare que uma sequência $< a_1, a_2, ..., a_n >$, onde cada a_i é um símbolo representando o endereço de uma página, pode ser visto como uma generalização de um *itemset*: a única diferença é que aqui os *itens* (endereços) são *ordenados* e a inclusão de uma sequência s em uma sequência t é diferente da inclusão de um itemset I num itemset J (veja esta definição na aula anterior). As referências para a aula de hoje são [30] e [29].

8.4.1 O algoritmo DHP (Direct Hashing and Pruning)

Vamos apresentar aqui o algoritmo DHP para minerar itemsets. O algoritmo DHP deve ser adaptado para ser utilizado no caso de sequências de referências. Você deve fazer isto como exercício (ver exercício 3 no final destas notas).

Idéia geral do algoritmo

Iteração 1: É idêntica à iteração 1 de Apriori, onde o conjunto de candidatos C_1 é gerado e depois testado no banco de dados D, para calcular L_1. A único ponto que muda é que ao mesmo tempo que se varre o banco de dados para contar o suporte dos candidatos C_1, vai-se construindo uma tabela hash para indexar os 2-itemsets contidos em transações de D.

Por exemplo, consideremos a situação ilustrada na figura 1 abaixo. A seguinte função hash é utilizada para indexar os 2-itemsets:

$h(\{a,b\}) = h(\{a,c\}) = h(\{b,c\}) = 1, h(\{c,e\}) = 2, h(\{a,d\}) = h(\{c,d\}) = h(\{b,e\}) = 3$. A função $N(m)$ conta o número de transações do banco de dados que suporta itemsets x tais que $h(x) = m$. A figura 1 abaixo ilustra esta construção.

A idéia central de utilizar a tabela H_2 para uma podagem extra nos candidatos C_2 da próxima iteração é a seguinte :

(*) Se $N(m)$ for inferior ao suporte mínimo para algum m então nenhum itemset indexado por m poderá ser frequente. Logo, será podado de C_2 na iteração 2. Isto é, além da poda habitual que se faz em C_2 utilizando a propriedade Apriori, faremos uma poda a mais, utilizando a tabela hash H_2, construída na iteração precedente (iteração 1) no momento de se varrer o banco de dados para o cálculo de L_1.

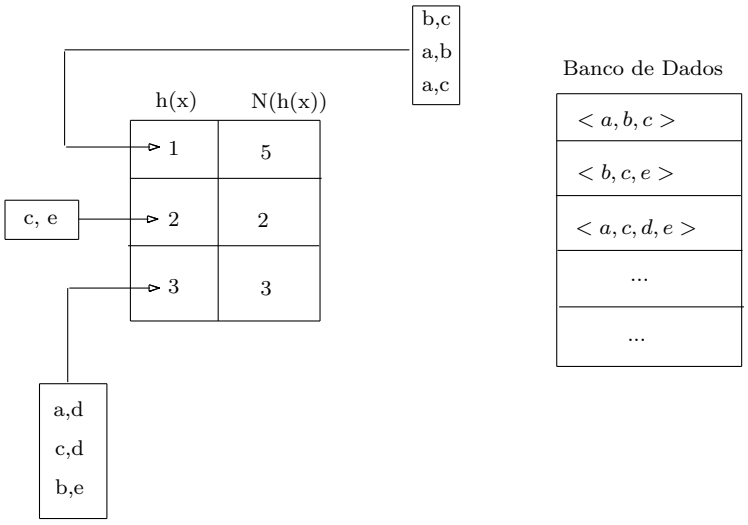


FIGURA 1

De aqui em diante, para cada iteração $k \geq 2$, vamos testar a seguinte condição :

Cond$_k$: Seja X = o número de grupos x para os quais $N(x) \geq \alpha$, onde α = suporte mínimo. $X \geq \beta$, onde β é um limite dado ?

Caso Cond$_k$ seja satisfeita, utilizamos **Rotina 1** para calcular L_k, caso contrário utilizamos **Rotina 2**.

Rotina 1

Iteração 2: Nesta iteração o banco de dados é $D_2 = D$ (banco de dados original). Temos duas fases distintas :

1. **Fase de Geração e Poda** : nesta fase vamos calcular os candidatos C_2 : para isto, vamos calcular os candidatos a partir de L_1 e podá-los utilizando a tabela hash H_2

construída na iteração anterior. Veja que na iteração 2, a poda usual de Apriori não é efetuada, pois não tem nenhum efeito.

A construção de C_2 é feita da mesma maneira como era feita em Apriori. Só que depois da poda usual, poda-se também os itemsets c tais que $H_2(h_2(c)) < \alpha$. Veja que aqui, estamos utilizando a propriedade (*) acima mencionada : itemsets c para os quais $H_2(h_2(c)) < \alpha$ não têm chance nenhuma de serem frequentes !

2. **Fase de Cálculo do Suporte** : nesta fase vamos :

   - calcular o suporte dos candidatos C_2 que sobraram da fase anterior,
   - podar o banco de dados, transformando-o num banco de dados D_3 que será utilizado na próxima iteração, para o cálculo do suporte dos candidatos C_3,
   - construir a tabela hash H_3 que servirá para podar os candidatos C_3 na próxima iteração.

   Todos estes 3 passos são realizados simultaneamente, numa única varrida do banco de dados. Construimos um array A com tantas casas quantos forem os itens que aparecem no banco de dados. Vamos supor que os items estão enumerados : $a_0, a_1, ...$ O elemento $A[i]$ vai conter um contador correspondente ao item a_i. No início, tais contadores são inicializados como zero.

   O procedimento para realizar os 3 passos acima é o seguinte : **para cada transação** t **de** D_2 :

   (a) Incrementamos os contadores dos candidatos $c \in C_2$ (que estão armazenados numa árvore-hash como de hábito) para os quais $c \subseteq t$.

   (b) Ao mesmo tempo que um candidato c está sendo avaliado, aumentamos os contadores individuais $A[i]$ de cada item a_i de c.

   (c) Itens a_i tais que $A[i]$ é inferior a 2 podem ser podados de t, pois com certeza não serão utilizados no cálculo do suporte na próxima iteração. Por que ? Ora, se t contém um 3-itemset frequente, então deverá conter exatamente três 2-itemsets frequentes. Logo, cada item de um 3-itemset frequente deverá aparecer em pelo menos dois 2-itemsets frequentes. Logo, se, no total, um item aparece em menos de dois itemsets de C_2, ele não aparecerá em nenhum 3-itemset frequente no futuro. Logo, pode ser retirado de t sem nenhum prejuízo.

   (d) Após esta *limpeza* preliminar da transação t, vamos utilizar a tabela hash H_2 a fim de construir a tabela hash H_3 para a iteração seguinte e ao mesmo tempo

podar mais a transação t.

A idéia da podagem extra é a seguinte : se um item a de t só aparece em 3-itemsets de t tais que algum 2-subitemset não está em C_2 então este item a com certeza nunca poderá aparecer num 3-itemset frequente suportado por t. Logo, pode ser podado de t sem prejuízo algum para a contagem do suporte na próxima iteração. Logo, só vão ficar em t aqueles itens que estão contidos em algum 3-itemset para o qual todos os três 2-subitemsets contidos em C_2.

Assim, construímos H_3, ao mesmo tempo que aumentamos os contadores dos itens de t: para cada 3-itemset $x = \{a, b, c\}$ de t, verificamos se todos os três subitemsets $\{a, b\}$, $\{a, c\}$ e $\{b, c\}$ estão em grupos de H_2 com suporte $\geq \alpha$. Se for o caso: (1) aumentamos o contador de $H_3(h_3(x))$, onde h_3 é a função hash que vai distribuir os 3-itemsets em grupos e (2) aumentamos os contadores de a, b e c. Aqueles itens para os quais os contadores são nulos serão podados de t.

(e) O novo banco de dados, depois que cada transação é podada, é denotado D_3. Será este banco de dados podado que será utilizado na iteração seguinte para o cálculo do suporte de C_3.

As próximas iterações são análogas à iteração 2 :

1. Constrói-se os candidatos C_k utilizando o conjunto L_{k-1}, faz-se a podagem costumeira e depois a podagem extra de C_k utilizando a tabela hash H_k construída na fase anterior.

2. Para cada transação t de D_k, aumenta-se o contador de cada candidato $c \in C_k$ que é suportado por t e ao mesmo tempo aumenta-se os contadores dos itens de t que aparecem em c. São podados aqueles itens cujos contadores são inferiores a k, isto é, que aparecem no máximo em $k - 1$ candidatos.

3. Para cada transação t, analisa-se cada $(k + 1)$-itemset x de t. Para cada um destes $(k + 1)$-itemsets, verifica-se se todos os seus k-itemsets aparecem em grupos m de H_k com $H_k(m) \geq \alpha$. Caso isto aconteça para um $(k + 1)$-itemset x, incrementa-se os contadores individuais de todos os items de x e ao mesmo tempo aumenta-se o contador de $H_{k+1}(h_{k+1}(x)$. Elimina-se de t aqueles itens a tais que seu contador $A[a]$ é nulo. Estes itens não pertencem a nenhum $(k + 1)$-itemset potencialmente frequente e portanto são inúteis em t.

Rotina 2

A primeira iteração k desta rotina :

1. **Fase de Geração e Poda** : nesta fase vamos calcular os candidatos C_k utilizando L_{k-1} e H_k da etapa anterior, exatamente como na Rotina 1.

2. **Fase de Cálculo do Suporte** : nesta fase vamos calcular o suporte dos candidatos C_k que sobraram da fase anterior e ao mesmo tempo podar o banco de dados (a primeira podagem somente). Não se constrói a tabela hash H_{k+1} e nem se realiza a segunda podagem (que normalmente é feita no momento da construção desta tabela). A partir da próxima iteração $k+1$, os candidatos C_{k+1} serão construídos como no algoritmo Apriori, sem a utilização da tabela hash H_{k+1}.

As próximas iterações da Rotina 2 :

1. **Fase de Geração e Poda** : nesta fase vamos calcular os candidatos C_k utilizando L_{k-1} como em Apriori. Não há mais a tabela H_k nesta fase.

2. **Fase de Cálculo do Suporte** : esta fase é idêntica à primeira iteração da rotina 2 : calcula-se o suporte dos candidatos C_k que sobraram da fase anterior e ao mesmo tempo poda-se o banco de dados (a primeira podagem somente). Não se constrói a tabela hash H_{k+1} e nem se realiza a segunda podagem (que normalmente é feita no momento da construção desta tabela).

Repare que :
DHP não reduz o número de varridas no banco de dados. A otimização consiste em podar mais candidatos a cada iteração (mais do que a poda usual de Apriori) e eventualmente diminuir o número de transações do banco de dados que serão testadas a cada varrida.

Resultados experimentais indicam que a Rotina 1 deve ser utilizada somente nas primeiras iterações, onde o número de candidatos é realmente muito grande e que a podagem obtida utilizando a tabela hash seja substancial, que compense o tempo gasto construindo-se a tabela hash. Normalmente, isto só vale a pena até $k = 2$, depois é melhor utilizar a Rotina 2. Sugerimos fortemente ao leitor de ver a análise detalhada de performance deste algoritmo, apresentada no artigo [30].

8.4.2 Exemplo de uso

Consideremos o seguinte banco de dados D inicial :

TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

Vamos supor uma ordem nos items : o(A) = 0, o(B) = 1, etc. Suponhamos a seguinte função hash :

$$h_2(\{x, y\}) = (o(x) * 10 + o(y)) \bmod 7$$

Temos a seguinte tabela hash H_2 :

0	3	itemsets contados : {C,E}, {C,E}, {A,D}
1	1	itemsets contados : {A,E}
2	2	itemsets contados : {B,C}, {B,C}
3	0	itemsets contados :
4	3	itemsets contados : {B,E}, {B,E}, {B,E}
5	1	itemsets contados : {A,B}
6	3	itemsets contados : {A,C}, {C,D}, {A,C}

Suponhamos que o suporte mínimo seja 50%. Uma análise premiliminar nos dá : $L_1 = \{\{A\}, \{B\}, \{C\}, \{D\}\}$.

Vamos analisar o que acontece na **fase da geração dos candidatos** C_2 :

Os pré-candidatos C_2' (como calculado por Apriori) são : {A,B}, {A,C}, {A,D}, {B,C}, {B,D}, {C,D}. Com a ajuda da tabela H_2, podemos fazer uma podagem extra : (1) $h_2(\{A, B\}) = 5$, cujo contador $H_2(5)$ é 1. Logo, deve ser descartado. (2) $h_2(\{A, E\}) = 5$, cujo contador $H_2(1)$ é 1. Logo, deve ser descartado. Assim, utilizando somente Apriori não podamos nada. Utilizando DHP, podamos 2 pré-candidatos.

Vamos agora ver o que acontece na **Fase do Cálculo do Suporte** : a transação 100 contém somente o candidato $\{A, C\}$. Assim, $A[0] = 1, A[1] = 0, A[2] = 1$. Como todos os valores de $A[i]$ são menores do que 2, esta transação é eliminada do banco de dados para a próxima iteração, isto é, todos os seus itens são eliminados. Por outro lado, a transação 300 contém 4 candidatos de tamanho 2 ({A,C}, {B,C}, {B,E}, {C,E}) e a correspondente ocorrência dos itens são :

$$A[0] = 0, A[1] = 2, A[2] = 2, A[3] = 0, A[4] = 2$$

O item A será descartado desta transação após a primeira podagem das transações. Repare que não tem sentido falar em podar o item D correpondente a $A[3]$, pois este não

aparece em t.

Consideremos uma outra situação : suponhamos que tenhamos uma transação $t = \{A, B, C, D, E, F\}$ e C_2 contém cinco 2-itemsets {AC, AE, AF, CD, EF}. Neste caso :

$$A[0] = 3, A[1] = 0, A[2] = 2, A[3] = 1, A[4] = 2, A[5] = 2$$

Logo, na primeira podagem de t são eliminados os itens B e D.

Na segunda podagem : repare que temos três possíveis 3-itemsets de {A,C,E,F} contendo C : {A,C,E}, {A,C,F} e {C,E,F}. O primeiro contém {C,E} que não está em C_2, o segundo contém {C,F} que não está em C_2 e o terceiro também contém {C,F} que não está em C_2. Logo, C é eliminado de t na segunda podagem.

Exercicio 1 : Analisar quais os itens que são eliminados de $t = \{A, B, C, D, E, F\}$ no exemplo anterior, na iteração 2.

Exercicio 2 : Execute com detalhes o algoritmo DHP no banco de dados D do exemplo anterior, supondo $\alpha = 0,5$ e $\beta = 4$.

Exercicio 3 : Adaptar DHP para ser utilizado no problema da mineração de sequências frequentes de páginas visitadas por usuários, como parte preliminar dos algoritmos FS e SS.

8.4.3 Algoritmo FS : determinando as sequências frequentes de referências

8.4.4 Algoritmo SS : um outro algoritmo para determinar as sequências frequentes de referências

8.4.5 Comparação dos dois algoritmos

Capítulo 9

Data Mining em Bioinformática

Problemas de descoberta de padrões aparecem em diferentes áreas de biologia, por exemplo, padrões que regulam a função de genes em sequências de DNA, ou padrões que são comuns em membros de uma mesma família de proteínas. Nestas notas de aula, vamos nos concentrar somente nos aspectos computacionais destes problemas. No primeiro capítulo de [32], o leitor pode encontrar um resumo bem compreensível dos aspectos biológicos envolvidos nestes problemas.

O banco de dados de sequências onde os padrões serão descobertos consiste de sequências sobre o alfabeto $\Sigma = \{A, C, G, T\}$ (no caso de sequências de DNA, onde os símbolos A, C, G, T representam os 4 nucleotídeos que compõem uma sequência de DNA) ou sobre o alfabeto dos 20 amino-ácidos que compõem as proteínas (no caso de sequências especificando proteínas).

9.1 Tipos de Problemas

1. **Problemas de descobrir padrões significantes**. O problema da descoberta de padrões geralmente é formulado da seguinte maneira: define-se uma classe de padrões de interesse que se quer encontrar e tenta-se descobrir quais os padrões desta classe que aparecem no banco de dados, com um suporte maior. O suporte de um padrão normalmente é o número de sequências nas quais o padrão ocorre. Muitas variações neste conceito ocorrem: pode-se exigir que o padrão ocorra em todas as sequências, ou num número mínimo de sequências especificado pelo usuário. Em alguns casos, o número de ocorrências não é especificado, mas é simplesmente um parâmetro de uma função que relaciona características da sequência (tamanho, por exemplo), com o número de ocorrências. Por exemplo, às vezes sequências longas com poucas ocorrências podem ser mais interessantes que sequências curtas com muitas ocorrências.

2. **Descoberta de Padrões versus Reconhecimento de Padrões**. O problema discutido acima se refere a Descoberta de Padrões. Um outro problema importante em biologia, relacionado a sequências, é o problema do *Reconhecimento de Padrões* : Em biologia, muitos padrões significantes são conhecidos a priori e é importante desenvolver ferramentas para descobrir ocorrências destes padrões conhecidos em novas sequências. Programas para reconhecimento de padrões podem ser bem gerais no sentido de que podem ser projetados para reconhecer padrões diversos (o padrão é um parâmetro de input do problema) ou podem ser bem específicos, no sentido de que são projetados para reconhecer um padrão particular. Do ponto de vista de ciência da computação, estes últimos não apresentam muito interesse, porém, são muito importantes para biólogos.

3. **Problemas de Classificação**. Tais problemas ocorrem, por exemplo, em famílias de proteínas. Um dos objetivos de se encontrar motivos comuns em famílias de proteínas é utilizar estes motivos como *classificadores*. Assim, dada uma proteína desconhecida, podemos classificá-la como membro ou não-membro de uma família, simplesmente baseando-se no fato de que ela contém os padrões que caracterizam esta família. Neste caso, temos um típico problema de aprendizado de máquina : dado um conjunto de sequências que pertencem à família (exemplos positivos) e um conjunto de sequências que não pertencem à família (exemplos negativos), queremos descobrir uma função F que associa a cada proteína um elemento de $\{0,1\}$, dizendo que a proteína pertence ($F(proteina) = 1$) ou não pertence ($F(proteina) = 0$) à família. Isto é, queremos, num primeiro momento, descobrir quais os padrões que caracterizam a família a partir dos exemplos positivos e negativos. Estes padrões especificam um *modelo de classificação* para a família. E num segundo momento, queremos utilizar o modelo para classificar proteínas desconhecidas, isto é, calcular a função F sobre a proteína. Este cálculo envolve algoritmos de *reconhecimento de padrões*: tenta-se verificar se a proteína desconhecida apresenta um número suficiente de padrões que caracterizam a família. Amostras positivas e negativas são retiradas de bancos de dados de proteínas conhecidas, tais como o SWISS-PROT, do *Swiss Institute of Bioinformatics*. Em [31], o leitor vai encontrar endereços de diversos bancos de dados de proteínas e instruções para carregá-los. Em geral, os modelos de classificação são criados a partir das amostras positivas. As amostras negativas são utilizadas para validar o modelo. Uma discussão detalhada deste tipo de problema pode ser encontrada em [33].

9.2 Tipos de Padrões

Os algoritmos para descoberta de padrões são projetados para descobrir padrões de determinados tipos.

1. **Padrões Determinísticos**. O padrão mais simples deste tipo é simplesmente uma sequência de símbolos do alfabeto Σ, tais como TATA. Outros padrões mais complexos são permitidos:

   - **Padrões com carácteres ambíguos**. Um carácter ambíguo é um carácter que corresponde a um subconjunto de Σ, podendo assumir qualquer valor deste subconjunto. Tais subconjuntos são denotados por uma lista de caracteres, por exemplo $[LF]$. Assim, o padrão $A-[LF]-G$ corresponde tanto à sequência ALG quanto à sequência AFG. Assim, $\sigma = EFFE\mathbf{ALG}A$ é uma sequência do banco de dados que contém uma ocorrência do padrão $A-[LF]-G$. No caso de sequências de nucleotídeos (componentes do DNA), utiliza-se letras especiais para cada subconjunto de nucleotídeos:
     $R=[AG], Y=[CT], W=[AT], S=[GC], B=[CGT], D=[AGT]$,
     $H=[ACT], V=[ACG], N=[ACGT]$
   - **Padrões com carácteres "curinga"**. Um *curinga* é um tipo especial de carácter ambíguo que na verdade corresponde ao alfabeto todo. Em sequências de nucleotídeos, o curinga é denotado por N, em sequências de proteínas, é denotado por X. Frequentemente, também são denotados por '.'. Uma sequência com um ou vários curingas consecutivos é chamada uma sequência "gap" (falha).
   - **Padrões com gaps flexíveis**. Um gap flexível é um gap com tamanho variável. No banco de dados de proteínas PROSITE, tais gaps são denotados por $x(i,j)$, onde i é o limite inferior do comprimento do gap e j é o limite superior. Assim, por exemplo, $x(4,6)$ representa um gap de tamanho entre 4 e 6. $x(3)$ representa um gap de tamanho fixo igual a 3. Finalmente, '$*$' representa um gap de tamanho qualquer inclusive 0. Por exemplo, o padrão $F-x(5)-G-x(2,4)-G-*-H$ ocorre nas sequências :
     $A\mathbf{F}BHBEF\mathbf{G}AA\mathbf{GH}ABB$
     $BA\mathbf{F}HHBEE\mathbf{G}AAB\mathbf{G}BB\mathbf{H}A$
     $BB\mathbf{F}BBBEE\mathbf{G}BABB\mathbf{G}BBEEEE\mathbf{H}$

   Alguns algoritmos não suportam todos estes tipos de padrões, por exemplo, não suportam gaps flexíveis, ou permitem todos os tipos de gaps mas não permitem caracteres ambíguos diferentes do curinga.

2. **Padrões permitindo um certo número de não coincidências**. Padrões determinísticos ocorrem numa sequência S se existe uma subsequência S' que coincide *exatamente* com o padrão. Por exemplo, o padrão determinístico $F - x(5) - G - x(2,4) - G - * - H$ ocorre na sequência
$S = A\mathbf{F}BHBEF\mathbf{G}AA\mathbf{GH}ABB$ pois S contém a subsequência $S' = \mathbf{F}BHBEF\mathbf{G}AA\mathbf{GH}$ que coincide exatamente com o padrão. Podemos estender o conceito de padrão deterministico, permitindo um certo número de não-coincidências no momento de definir a ocorrência de uma padrão numa sequência. Dizemos que o padrão P ocorre na sequência S com no máximo k não-coincidências se existe uma subsequência S' de S que difere de uma sequência S'' em no máximo k posições e tal que S'' contém exatamente o padrão P.

   Por exemplo, o padrão $F - x(5) - G - x(2,4) - G - * - H$ ocorre em
$S = A\mathbf{F}BHBEF\mathbf{A}AA\mathbf{GH}ABB$ com no máximo 1 não coincidência, pois ocorre exatamente em $S'' = \mathbf{F}BHBEF\mathbf{G}AA\mathbf{GH}$ e esta sequência difere da subsequência S' de S, $S' = \mathbf{F}BHBEF\mathbf{A}AA\mathbf{GH}$ somente na posição 7.

   Às vezes também se permite inserções e supressões de caracteres na sequência S': assim, o número de não-coincidências incluiria as inserções ou supressões (não somente as modificações de caracteres) que deveriam ser feitas em S' para obter S.

9.3 O problema genérico da descoberta de padrões

A fim de bem definirmos o problema genérico da descoberta de padrões, os seguintes elementos precisam estar bem claros:

1. **Os padrões**.

   A partir do exposto acima, está claro que um padrão, na verdade é um conjunto de strings. Por exemplo, o padrão $A.CG.$ sobre o alfabeto $\Sigma = \{A, C, G, T\}$ é o conjunto de strings

   $\{AACGA, AACGC, AACGT, AACGG, ACCGA, ACCGC,$
$ACCGT, ACCGG, AGCGA, AGCGC, AGCGT, AGCGG, ATCGA,$
$ATCGC, ATCGT, ATCGG\}$

   É claro também que cada um dos padrões que definimos nada mais é do que uma expressão regular sobre o alfabeto Σ.

2. **O espaço de busca dos padrões**. Assim como no caso da mineração de sequências de itemsets frequentes, estabelecíamos um espaço de busca dos padrões que estávamos

procurando (todos as sequências de itemsets sobre um conjunto de items dado), no caso de padrões moleculares, o espaço de busca estabelecido é uma *linguagem de padrões*. Repare que no caso das sequências de itemsets, cada padrão é um único string, e o espaço de busca é um conjunto de strings satisfazendo certas condições (cada elemento do string é um conjunto de items pertencentes a um conjunto de items pré-fixado). No caso das sequências biomoleculares, cada padrão P é uma expressão regular (portanto, um conjunto de strings $L(P)$) e o espaço de busca é um conjunto de expressões regulares satisfazendo certas condições.

Uma *linguagem de padrões* $\mathcal{C}$ é um conjunto de expressões regulares sobre o alfabeto Σ, isto é, um conjunto de padrões. Esta linguagem deve ser suficientemente expressiva para descrever todas as características biológicas das sequências consideradas. Esta linguagem é o espaço de busca dos padrões.

3. **O banco de dados de sequências onde serão minerados os padrões**. Um banco de dados de sequências é um conjunto finito de sequências sobre um alfabeto que contém o alfabeto dos padrões.

4. **Quando uma sequência S suporta um padrão P**. Como já definimos acima, um padrão P é suportado por uma sequência S se existe S' contida em S tal que $S' \in L(P)$ (não se esqueça de que um padrão é um conjunto de strings e que $L(P)$ denota este conjunto de strings !).

5. **Suporte de um padrão P com relação a um banco de dados de sequências**. É o número de sequências do banco de dados que suportam P.

Podemos agora enunciar o problema genérico da descoberta de padrões em dados moleculares (PGDP):

Input: Um banco de dados $S= \{s_1, s_2, \ldots, s_n\}$ de sequêcias sobre um alfabeto Σ, um inteiro $k \leq n$ e uma linguagem de padrões $\mathcal{C}$.

Output: Todos os padrões de $\mathcal{C}$ com suporte $\leq k$ com relação a S.

Como dissemos acima, a linguagem de padrões $\mathcal{C}$ deve ser suficientemente expressiva. Infelizmente, quanto maior for a expressividade de $\mathcal{C}$, maior será o custo computacional do problema. Quando $\mathcal{C} = \Sigma^*$, isto é, cada padrão de $\mathcal{C}$ é uma expressão regular composta por um único string (o padrão não contém curingas, símbolos ambíguos ou gaps), o problema PGPD pode ser resolvido em tempo linear utilizando *árvores de sufixos generalizadas* ([34]). Na maioria dos outros casos, o problema PGPD é NP-hard. Outras

possibilidades para a linguagem de padrões $\mathcal{C}$ são : (1) $\mathcal{C}$ = conjunto dos padrões contendo o curinga; (2) $\mathcal{C}$ = conjunto dos padrões contendo o símbolo $*$; (3) $\mathcal{C}$ = conjunto dos strings contendo símbolos do tipo $x(i,j)$, representando gaps flexíveis; (4) $\mathcal{C}$ = conjunto dos strings contendo símbolos do tipo $R_1, ..., R_n$, onde cada R_i é uma letra representando um conjunto de aminoácidos. Exemplo de um tal string é $A - R - F$, onde R representa o conjunto $[A, F, L]$.

Alguns critérios para se avaliar os algoritmos que resolvem o problema PGDP

1. A linguagem de padrões $\mathcal{C}$ utilizada: Em geral, procura-se por uma linguagem o mais expressiva possível. O preço é normalmente pago em termos de complexidade tempo/espaço.

2. A habilidade do algoritmo em gerar todos os padrões qualificados: Alguns algoritmos aproximativos podem ter uma boa performance em termos de complexidade, em detrimento dos resultados produzidos (não são capazes de descobrir todos os padrões da linguagem de padrões).

3. A *maximalidade* dos padrões descobertos. Um padrão P é *mais específico* do que um padrão Q se $L(P) \subseteq L(Q)$. É claro que toda sequência que suporta P suporta também Q. Logo, se P é frequente, Q também o será. Assim, um bom algoritmo, segundo o critério de maximalidade, só produziria o padrão P como resposta, sendo subentendido que todo padrão menos específico do que P também é resposta. Considere por exemplo, $k = 2$ e o seguinte banco de dados

   $$S = \{LARGE, LINGER, AGE\}$$

   O padrão $L...E$ tem suporte 2, portanto é frequente. Este padrão, entretanto, não é maximal, pois o padrão $L..GE$ também tem suporte 2 e é mais específico. Produzir padrões não-maximais pode afetar substancialmente a eficiência do algoritmo, além de dificultar a identificação de padrões que realmente importam.

9.4 As principais categorias de algoritmos para o problema PGDP

Os algoritmos que resolvem o problema PGDP podem ser classificados em dois tipos: (1) algoritmos de enumeração de padrões e (2) algoritmos de alinhamento de strings.

- **Algoritmos de enumeração de padrões**. Esta classe contém algoritmos que enumeram todos (ou parte de) os padrões pertencentes à linguagem de padrões $\mathcal{C}$ e então verifica quais dos padrões gerados durante a enumeração têm o suporte maior que o mínimo. Uma vez que tais algoritmos exploram o espaço de busca, tendem a ser exponenciais em função do tamanho do maior padrão gerado. A fim de serem eficientes, eles geralmente impõem restrições nos padrões a serem descobertos, diminuindo assim o espaço de busca.

  A idéia básica utilizada em todos os algoritmos desta classe é a seguinte: No primeiro passo $k = 0$, inicia-se com o padrão vazio e prossegue-se recursivamente gerando padrões cada vez maiores. No passo k ($k > 0$), enumera-se todos os (ou alguns) padrões pertencentes à classe $\mathcal{C}$ que possuem um padrão produzido na fase anterior $k - 1$ como prefixo. Veja que a idéia é muito parecida com a idéia de Apriori : se um padrão P for frequente, seus prefixos devem ser frequentes também. Testa-se os suportes destes novos padrões gerados. Continua-se a expansão para aqueles padrões que têm suporte acima do mínimo.

  O que diferencia os vários algoritmos desta classe é a linguagem de padrões $\mathcal{C}$ que eles reconhecem, a eficiência com a qual eles implementam o processo de expansão dos padrões a cada passo e sua habilidade de rapidamente detectar e descartar padrões que não são maximais.

- **Algoritmos de alinhamento de strings**. Os algoritmos desta classe utilizam o *múltiplo alinhamento* das sequências do banco de dados como ferramenta básica para a descoberta dos padrões frequentes. Existem diversas maneiras de definir o que é este *múltiplo alinhamento* das sequências. Uma delas, a mais natural, é através do conceito de *sequência de consenso associada a um conjunto de sequências*. Vamos ilustrar este conceito através de um exemplo:

  Considere o seguinte conjunto de sequências $S = \{s_1, s_2, s_3\}$, onde $s_1 = ABFGRTP$, $s_2 = BDFLRP$, $s_3 = AFRP$. Os símbolos das sequências são letras maiusculas. Vamos alinhar estas 3 sequências de modo a obter o maior número de coincidências entre elas, mesmo que para isto seja necessário inserir gaps nestas sequências:

A	B	-	F	G	R	T	P
-	B	D	F	L	R	-	P
A	-	-	F	-	R	-	P

  Para cada posição, vamos associar um símbolo x. No final, a sequência destes símbolos será a *sequência de consenso do banco de dados* S. O símbolo x é definido

da seguinte maneira: (1) caso nesta posição exista um símbolo que ocorre em todas as sequências, então x será igual a este símbolo. (2) caso nesta posição só existam símbolos que não ocorrem com frequência nas sequências, escolhemos aquele que ocorre com maior frequência, e x será igual à letra minúscula correspondente ao símbolo escolhido, caso este for uma letra, ou x será igual a $-$, caso este for $-$. No nosso exemplo, a sequência de consenso c é :

a b - F l R - P

Dada a sequência de consenso c, é possível transformar cada sequência de S em c utilizando um certo número de operações do tipo inserção, supressão ou modificação de símbolos em cada posição. Estas operações estão bem definidas, uma vez que c for descoberta. Por exemplo, para a primeira sequência s_1, estas operações são : (1) insere $-$ na posição 3, troca G por L na posição 5, suprime T da posição 7. A regra geral é : se x é um caracter de s_i alinhado a um caracter y de c, então (1) se ambos são letras, troca-se x por y em s_i; (2) se $x = -$, então y é inserido no lugar de x nesta posição e (3) se $y = -$, então x é suprimido de s_i.

O problema do Múltiplo Alinhamento de Sequências consiste em descobrir as sequências consenso (repare que podem existir muitas) a partir das sequências de input bem como as operações de edição que transformam cada sequência de input nas sequências de consenso. No caso de sequências de dados biológicos, é preciso, além disto, que as sequências de consenso tenham alguma relevância em termos biológicos, o que acarreta acrescentar certas restrições às sequências de consenso. O problema do Múltiplo Alinhamento de Sequências é NP-hard.

9.5 TEIRESIAS - um algoritmo para descoberta de padrões em biosequências

O algoritmo TEIRESIAS que vamos ver nesta aula e na próxima foi desenvolvido e implementado por A. Floratos [32] em 1998. Este algoritmo pertence à classe dos algoritmos de *enumeração de padrões* que descrevemos de forma geral na aula passada. As características principais do enfoque seguido no algoritmo são:

1. O método é de natureza combinatória e produz **todos** os padrões maximais tendo um suporte mínimo *sem* enumerar todo o espaço de busca dos padrões. Isto torna o método eficiente.

2. Os padrões são gerados *em ordem de maximalidade*, isto é, os padrões maximais são gerados primeiro. Desta forma, padrões redundantes (não-maximais), podem ser detectados facilmente, comparando-os com os padrões já gerados.

3. Resultados experimentais sugerem que o algoritmo é *quase linear* com respeito ao tamanho do output produzido (repare que não é com respeito ao tamanho do input).

4. Ao contrário de outros métodos, o algoritmo pode manipular de forma eficiente padrões de tamanho arbitrário.

O algoritmo TEIRESIAS é composto de duas fases: a fase de *varredura* e a fase de *convolução*. Durante a fase de varredura, são identificados *padrões elementares* com suporte superior ou igual ao mínimo. Estes padrões elementares constituem os *blocos de construção* da fase de convolução. Eles são combinados progressivamente para produzir padrões cada vez mais longos, até que todos os padrões maximais sejam gerados. Além disto, a ordem na qual a convolução é executada torna fácil identificar e descartar padrões não-maximais.

9.5.1 Terminologia e Definição do Problema

Seja Σ um alfabeto (isto é, o conjunto dos 20 amino-ácidos que compõem as proteínas ou o conjunto dos 4 nucleotídeos que compõem o DNA). A classe de padrões tratada por TEIRESIAS é :

$$\mathcal{C} = \Sigma(\Sigma \cup \{\text{`.'}\})^*\Sigma$$

Isto é, o espaço de busca dos padrões manipulados por TEIRESIAS é constituído de padrões que iniciam e terminam por um símbolo não-ambíguo. Símbolos não-ambíguos também são chamados de *resíduos*. Por exemplo, o padrão $P = A.CH..E$ é um padrão de $\mathcal{C}$. As seguintes sequências são elementos de $L(P)$:

$$\mathbf{A}D\mathbf{CH}FF\mathbf{E}, \mathbf{A}L\mathbf{CH}ES\mathbf{E}, \mathbf{A}G\mathbf{CH}AD\mathbf{E}$$

Os seguintes conceitos serão utilizados posteriormente na descrição do algoritmo TEIRESIAS. Alguns deles já foram introduzidos na aula passada, mas vamos repetir sua definição aqui a fim de facilitar o entendimento do algoritmo posteriormente.

1. **Subpadrão**: Seja P um padrão. Qualquer substring de P que é um padrão é chamado de um *subpadrão*. Por exemplo : H..E é um subpadrão do padrão A.CH..E, pois :

- H..E é um padrão de $\mathcal{C}$ (começa e termina por um símbolo não-ambíguo).
- H..E é um substring de A.C**H..E**

2. Sejam L e W dois números inteiros, com $L \leq W$. Um padrão P é dito um $< L, W >$-padrão se *todo* subpadrão de P com comprimento $\geq W$ contém pelo menos L símbolos não-ambíguos (resíduos). Os números L, W medem de certa forma, a *densidade* do padrão P: um padrão onde os resíduos são muito esparsos (muitos curingas consecutivos) não serão considerados por TEIRESIAS.

   Por exemplo, seja $L = 3$ e $W = 5$. O padrão $P = AF..CH..E$ é um $< 3, 5 >$-padrão, pois qualquer subpadrão de P com comprimento ≥ 5 tem pelo menos 3 símbolos não-ambíguos. Veja que os possíveis subpadrões nestas condições são :

   $$AF..C, F..CH, CH..E$$

   e todos eles têm pelo menos 3 resíduos.

   O padrão $AF..C.H..E$ é um $< 3, 5 >$ padrão mas não é um $< 4, 6 >$-padrão, pois $C.H..E$ é um subpadrão de comprimento 6 que não tem 4 resíduos.

3. Dado um padrão P e um conjunto $S = \{s_1, s_2, \ldots, s_n\}$ de sequências, definimos a *lista de offsets* de P com relação a S como sendo o conjunto :

   $L_S(P) = \{(i, j) \mid$ o padrão P está contido na sequência s_i a partir da posição j de $s_i\}$

   Por exemplo, seja $P = L...E$ e $S = \{APLARGEED, BLINGE, AGE\}$. Então :

   $$L_S(P) = \{(1, 3), (2, 2)\}$$

4. Um padrão P' é dito *mais específico* do que um padrão P se P' pode ser obtido de P substituindo-se um ou mais símbolos curinga por resíduos, ou concatenando-se à direita ou à esquerda de P um string com resíduos e símbolos ambíguos. Por exemplo, os padrões :

   $$AFCH..E, A.CHL.E.K, SA.CH..E$$

   são todos mais específicos do que o padrão $A.CH..E$.

5. **Propriedade Importante**: se P' é mais específico do que P então para qualquer conjunto de sequências S tem-se :

$$\mid L_S(P') \mid \leq \mid L_S(P) \mid$$

Isto é, o número de vezes que P' se "encaixa" nas sequências de S é menor ou igual ao número de vezes que P se "encaixa" nas sequências de S. Repare que nesta conta, uma mesma sequência s é contada tantas vezes quanto for o número de posições a partir das quais P se "encaixa" em s

Repare que um padrão P' pode ser mais específico do que P e $\mid L_S(P') \mid < \mid L_S(P) \mid$. Por exemplo, $P = L..GE$, $P' = L.RGE$ e $S = \{APLARGEED, BLINGE, AGE\}$. Mas podemos também ter a igualdade $\mid L_S(P') \mid = \mid L_S(P) \mid$ como no seguinte caso : $P = L..GE$, $P' = L.RGE$ e $S = \{APLARGEED, AGE\}$.

6. Dado um conjunto de sequências S, um padrão P é dito *maximal* com relação a S se não existe padrão P' mais específico do que P tal que $\mid L_S(P) \mid = \mid L_S(P') \mid$. Veja que no último exemplo acima, $P = L..GE$ não é maximal, pois $P' = L.RGE$ é mais específico mas mesmo assim $\mid L_S(P') \mid = \mid L_S(P) \mid$.

Entretanto, se considerarmos o exemplo, $P = L..GE$ e $S = \{APLARGEED, BLINGE, AGE\}$, teremos que P é maximal. Com efeito, qualquer padrão P' mais específico do que P terá $\mid L_S(P') \mid < \mid L_S(P) \mid$.

Uma outra maneira de definir padrão maximal é dizer quando é que P não é maximal : quando existir um padrão P' mais específico do que P com $\mid L_S(P') \mid = \mid L_S(P) \mid$. Dizemos neste caso que P é *subentendido* por P'.

Estamos prontos agora para enunciar o problema de descoberta de padrões que será o objetivo do algoritmo TEIRESIAS:

Input: Um conjunto $S = \{s_1, ..., s_n\}$ de sequências sobre o alfabeto Σ, inteiros L, W, k com $L \leq W$ e $2 \leq k \leq n$.

Output: Todos os $< L, W >$-padrões de $\mathcal{C}$, maximais com relação a S e com suporte $\geq k$ no conjunto S.

Daqui em diante, sempre que nos referirmos a um padrão, queremos dizer um $< L, W >$-padrão.

Discussão sobre a complexidade do problema tratado por TEIRESIAS

Na aula passada, vimos que o problema genérico da descoberta de padrões (PGDP) é NP-hard quando a linguagem de padrões inclui símbolos ambíguos. Assim, a menos que tornemos mais específicos os elementos do problema, há pouquíssima chance de se encontrar um algoritmo eficiente para resolver este problema. O problema que TEIRESIAS se propõe a resolver é mais específico do que o problema genérico de descoberta de padrões, pois se propõe a encontrar somente os $< L, W >$-padrões frequentes. A introdução do parâmetro W torna o problema tratável. Na ausência deste parâmetro de *densidade*, o problema seria NP-hard.

9.5.2 O algoritmo TEIRESIAS

Um *padrão elementar* é um $< L, W > -$padrão que contém exatamente L resíduos.

A Fase de Varredura

A fase de varredura consiste em varrer o banco de dados de sequências para encontrar os padrões elementares que têm suporte $\geq k$. A figura 1 abaixo ilustra a fase de varredura para $L = 3, W = 4$ e $k = 3$, onde o banco de dados de sequências é :

$$S = \{s_1 = SDFBQSTS, s_2 = LFCASTS, s_3 = FDASTSNP\}$$

O processo consiste em encontrar as sequências de tamanho no máximo 4 contendo exatamente 3 resíduos. Estas são as sequências mais curtas satisfazendo a condição de que *todo* subpadrão de tamanho ≥ 4 contém ao menos 3 resíduos.

O resultado da fase da varredura é um conjunto EP contendo todos os padrões elementares (e seus respectivos offsets) que são frequentes. Este conjunto EP será o input da fase de convolução. No exemplo acima, o conjunto EP é dado por :

(**F.AS**,{(1,3),(2,2),(3,1)}),
(**AST**,{(1,5),(2,4),(3,3)}),
(**AS.S**,{(1,5),(2,4),(3,3)}),
(**STS**,{(1,6),(2,5),(3,4)}),
(**A.TS**,{(1,5),(2,4),(3,3)})

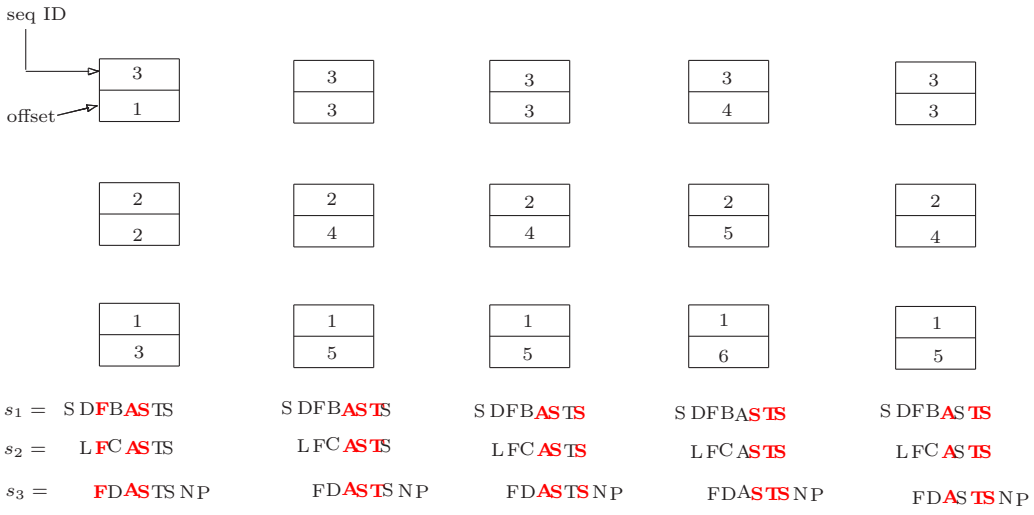


Figura 9.1: A fase de varredura de TEIRESIAS

A Fase de Convolução

A fase de convolução consiste em sucessivamente combinar padrões já gerados a fim de obter padrões mais longos. A idéia para obter um padrão de tamanho m contendo no mínimo $L+1$ resíduos é combinar dois padrões P e Q com no mínimo L resíduos tais o prefixo de P contendo exatamente $L-1$ resíduos coincide com o sufixo de Q contendo exatamente $L-1$ resíduos. Por exemplo, consideremos os padrões elementares $P = F.AS$ e $Q = AST$ gerados na fase de varredura. Observamos que o prefixo AS de Q coincide com o sufixo AS de P. Logo, podemos gerar o padrão $F.AST$ com exatamente 4 resíduos e de tamanho 5.

Veja que é praticamente a mesma idéia utilizada por Apriori: Se o padrão $F.AST$ for frequente, com certeza todo os seus subpadrões de tamanho 4 ou 3 deverão ser frequentes. Logo, $F.AS$ e AST devem ser elementos de EP.

A fim de tornar esta operação mais precisa, definimos o prefixo e sufixo de um padrão.

Definição 9.1 Seja P um padrão com exatamente L resíduos. Definimos o prefixo(P) como sendo o único subpadrão com exatamente $L-1$ resíduos e que é um prefixo de P. Por exemplo, para $L = 3$:

$$prefixo(F.ASTS) = F.A \text{ e } prefixo(AST) = AS$$

Definimos o sufixo(P) como sendo o único subpadrão com exatamente $L-1$ resíduos e que é um sufixo de P. Por exemplo, para para $L = 3$:

$$sufixo(F.A...S) = A...S \text{ e } sufixo(ASTS) = TS$$

Definimos agora a operação de *convolução* (denotada $\oplus$) entre dois padrões. Trata-se do equivalente à junção de duas sequências no algoritmo GSP, só que agora precisamos ficar atentos ao número de resíduos minimal que precisamos obter no padrão gerado.

Sejam P e Q dois padrões com no mínimo L resíduos cada. A *convolução* de P e Q, é um novo padrão $P \oplus Q$, definido como a seguir:

- $P \oplus Q = PQ'$ se $sufixo(P) = prefixo(Q)$ e Q' é um string tal que $Q = prefixo(Q)Q'$.
- $P \oplus Q =$ sequência vazia (ϵ), caso contrário.

Exemplo 9.1 Seguem alguns exemplos de convolução, no caso de $L = 3$:

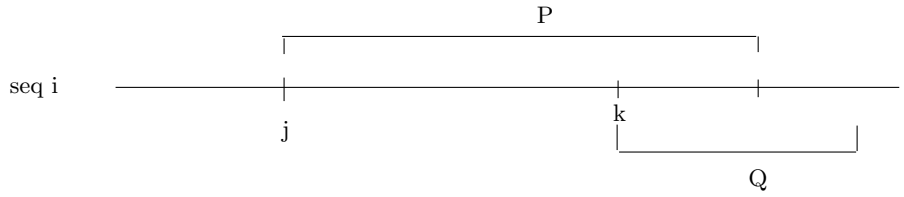
$$DF.A.T \oplus A.TSE = DF.A.TSE$$

$$AS.TF \oplus T.FDE = \epsilon$$

A seguinte proposição relaciona os offsets de P com os offsets de $P \oplus Q$:

Proposição 9.1 Sejam P e Q dois padrões que são juntáveis (a convolução de ambos é diferente do string vazio). Seja $R = P \oplus Q$. Então:

$$L_S(R) = \{(i,j) \in L_(P) \mid \exists (i,k) \in L_S(Q) \text{ tal que } k - j = \mid P \mid - \mid sufixo(P) \mid\}$$



Como fazer para ter certeza de que, somente utilizando a convolução, podemos

- gerar todos os padrões,
- identificar e descartar rapidamente os padrões que não são maximais.

A fim de se atingir estes dois objetivos, introduzimos duas ordens parciais no espaço de busca dos padrões, $\mathcal{C}$. Estas ordens serão utilizadas para guiar a maneira como as convoluções são realizadas. Usando estas ordens, poderemos garantir que : (a) todos os padrões são gerados e (b) um padrão maximal P é gerado antes que qualquer padrão não-maximal subentendido por P. Assim, um padrão não-maximal pode ser detectado com um esforço mínimo, simplesmente comparando-o com todos os padrões gerados até o momento para ver se ele é subentendido por algum destes que já foram gerados. Estas comparações podem ser feitas de maneira eficiente, armazenando os padrões maximais gerados até o momento, numa árvore hash.

Ordenando apropriadamente os elementos do espaço de busca

Sejam P e Q dois padrões. O seguinte procedimento vai dizer se P é menor do que Q em termos de prefixo (denotado $P <_{pref} Q$:

- Primeiro Passo : alinha-se os dois padrões de modo que os resíduos mais à esquerda de ambos estejam alinhados na primeira coluna.
- Segundo Passo : as colunas do alinhamento são examinadas uma a um, começando na mais a esquerda e prosseguindo para a direita. Caso se encontrar uma coluna onde um símbolo é um resíduo e outro é um curinga, o processo pára
- Se o resíduo está em P então $P <_{pref} Q$. Se for o resíduo estiver em Q dizemos que $Q <_{pref} P$.

Intuitivamente, P é menor do que Q em termos de prefixo se o primeiro curinga a partir da esquerda aparece em Q. Isto é, P é menos ambíguo à esquerda do que Q.

Consideremos o seguinte exemplo :

$P = ASD...F$
$Q = SE.ERF.DG$

Alinhamos P e Q pela esquerda :

A	S	**D**	.	.	.	F		
S	E	**.**	E	R	F	.	D	G

Veja que o primeiro curinga a partir da esquerda ocorre na posição 3 em Q. Logo, $P <_{pref} Q$.

De uma maneira análoga, definimos a ordem de acordo com os sufixos. O seguinte procedimento vai dizer se P é menor do que Q em termos de sufixo (denotado $P <_{suf} Q$:

- Primeiro Passo : alinha-se os dois padrões de modo que os resíduos mais à direita de ambos estejam alinhados na última coluna.
- Segundo Passo : as colunas do alinhamento são examinadas uma a uma, começando na mais a direita e prosseguindo para a esquerda. Caso se encontrar uma coluna onde um símbolo é um resíduo e outro é um curinga, o processo pára
- Se o resíduo está em P então $P <_{suf} Q$. Se for o resíduo estiver em Q dizemos que $Q <_{suf} P$.

Intuitivamente, P é menor do que Q em termos de sufixo se o primeiro curinga à partir da direita aparece em Q. Isto é, P é menos ambíguo à direita do que Q.

Consideremos o mesmo exemplo acima :

$P = ASD...F$
$Q = SE.ERF.DG$

Alinhamos P e Q pela esquerda :

		A	S	D	.	.	.	F
S	E	.	E	R	F	.	D	G

Veja que o primeiro curinga a partir da direita ocorre na penúltima coluna em P. Logo, $Q <_{suf} P$.

A idéia geral da fase de convolução é estender cada padrão P à esquerda com todos os padrões que podem ser juntados a P. Estes padrões serão considerados um a um, na ordem segundo o prefixo. Depois, estende-se cada padrão P à direita, com todos os padrões que podem ser juntados a P. Agora, estes padrões serão considerados um a um, na ordem segundo o sufixo.

9.5.3 Detalhes das Fases de Varredura e Convolução

Nesta seção, vamos ver os detalhes de execução das Fases de Varredura e Convolução do algoritmo TEIRESIAS bem como uma discussão sobre os resultados experimentais deste algoritmo. Todos os conceitos necessários para o entendimento do algoritmo foram introduzidos na aula passada. É importante que estes conceitos tenham sido bem assimilados antes de iniciar o estudo desta aula.

Lembramos que o input de TEIRESIAS é :

- Um banco de dados de n sequências S,
- Um número inteiro $2 \leq k \leq n$,
- Dois números inteiros L, W, com $L \leq W$.

O output será todos os $< L, W >$-padrões maximais com suporte $\geq k$ em S.

Lembramos que :

- **padrão** : começa e termina por um resíduo,

- $< L, W >$-padrão: todo subpadrão de comprimento $\geq W$ tem pelo menos L resíduos (este conceito restringe o universo dos padrões a padrões não muito "esparsos"),
- **padrão maximal** : todo padrão mais específico do que ele tem menos tem menos ocorrências do que ele,
- **suporte de um padrão** : número de sequências do banco de dados onde ele ocorre.
- **padrão elementar** : os menores $< L, W >$-padrões, isto é, os padrões de tamanho $\leq W$ com exatamente L resíduos.

A Fase da Varredura

A fase de varredura vai produzir o conjunto EP dos padrões elementares junto com a lista de seus respectivos offsets. Esta é a única vez em que o banco de dados de sequências é varrido. A fim de ilustrar cada etapa desta fase, vamos considerar o seguinte input:

$S = \{s_1 = \text{FASTS}, s_2 = \text{LFBST}, s_3 = \text{FAPK}\}$, $k = 3$, $L = 3$, $W = 4$.

1. **Etapa 1 :** Calcula os inícios dos possíveis padrões frequentes. Este conjunto é denotado por $\mathcal{I}$.

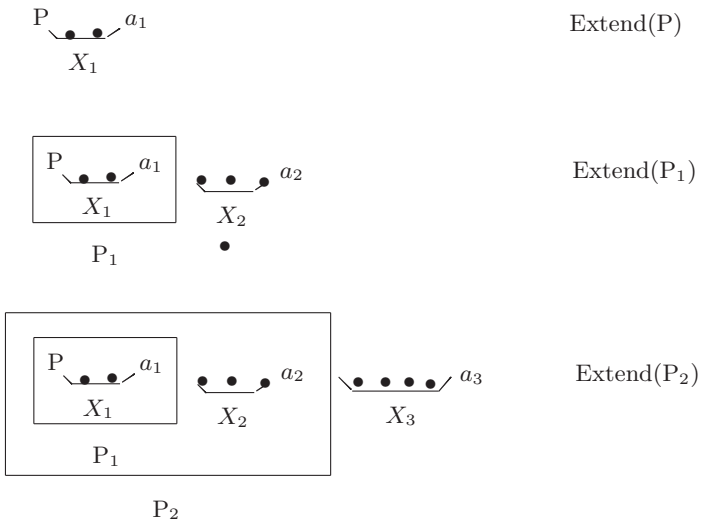
   Para cada símbolo σ do alfabeto Σ, calculamos o suporte de σ. Como $k = 3$, só vamos considerar aqueles que aparecem em pelo menos 3 sequências, juntamente com a lista de seus offsets. No nosso exemplo :

   $$\mathcal{I} = \{\ (F, \{(1,1),(2,2),(3,1)\})\}$$

2. **Etapa 2** : Processo recursivo que vai tentar estender cada símbolo de $\mathcal{I}$ a fim de produzir um padrão frequente (que aparece nas 3 sequências). É possível que existam diversas maneiras de estender cada símbolo de modo a construir um padrão que apareça nas 3 sequências.

   O procedimento recursivo responsável pela extensão de um padrão P é Extend(P). Repare que a primeira vez que este procedimento for evocado, será executado sobre um padrão constituído de apenas um símbolo σ do alfabeto (resíduo). Posteriormente, será executado sobre os outputs produzidos em fases precedentes, portanto, em padrões progressivamente mais longos.

   O procedimento Extend(P,ListaOff), onde P é um padrão frequente e ListaOff é a lista de seus offsets.



(a) Seja $A(P)$ = número de resíduos que aparecem em P. Se $A(P) = L$ então colocamos (P,ListaOff) no conjunto EP.

(b) Caso $A(P) < L$, vamos estender P de modo a completar os L resíduos. Veja que neste processo podemos introduzir mais resíduos de modo a completar o total de L, mas ao mesmo tempo acrescentar curingas. A única restrição é que não podemos ultrapassar o tamanho máximo permitid para o string que é W.

Para isso, a idéia é que cada vez que **Extend** seja executado sobre um padrão P ele concatene à direita de P um número X de curingas seguidos de um resíduo σ. O comprimento total do string obtido é no máximo W. Além disto, o número total de resíduos no string total é no máximo L. Assim:

$| P | + X + 1 \leq W$ e $A(P) + 1 = L$.

Logo, $X \leq (W$ - $| P |) - (L - (A(P))$

A figura abaixo ilustra a execução recursiva de Extend sobre o padrão P :

Para cada , o número de curingas inseridos entre o último símbolo do padrão P que está sendo expandido e o próximo resíduo é X_i. Este número pode variar entre 0 e $(W$ - $| P |) - (L - (A(P))$, como vimos acima. É preciso considerar todas estas possibilidades, a cada vez que se chama recursivamente o procedimento **Extend**.

(c) Para quais valores de X_i pode-se fazer a expansão de P ? Isto é, quantos curingas pode-se acrescentar depois de P para que possamos depois "fechar" o string com um resíduo a ? E qual seria este a com o qual vamos estender P ?

Ora, se y é a posição em que P começa na sequência, então é necessário que $y + \mid P \mid +$ número de curingas $<$ tamanho da sequência. Seja i o número de curingas depois de P. O símbolo a com o qual expandiremos P após os curingas deve ser aquele que aparece na posição $y + \mid P \mid + i$ da sequência.

(d) Agora, precisamos calcular o suporte do padrão extendido e descartar aquelas possibilidades de extensão que não correspondem a padrões frequentes. Vamos fazer isto sem varrer o banco de dados novamente. Construimos um array com tantas posições quanto for o número de símbolos de Σ (uma posição para cada símbolo; uma ordem qualquer foi estabelecida entre os símbolos de Σ). Para cada símbolo a com o qual estemos P segundo o procedimento acima e para cada (x, y) pertencente à lista de offsets de P (offsets permitidos, isto é, para os quais é possível se fazer a expansão com i curingas, mais um símbolo, sem depassar o tamanho da sequência s_x) inserimos na posição correspondente a a no array o par $(x, y+ \mid P \mid +i)$.
O suporte do padrão expandido $P \ . \ . \ . \ a$ é facilmente calculado contando-se o número de offsets na posição correspondente ao símbolo a no array. Caso este suporte for $\geq k$, continuamos expandindo $P' = P \ . \ . \ . \ a$, chamando recursivamente o procedimento **Expand**(P', $L_S(P')$).

No nosso exemplo, teremos as seguintes possibilidades de expansão do padrão unitário F na *primeira* chamada de Extend(P,$L_S(P)$):

$FA, F.S, F..T, F...S, FB, F.P, F..K$

Se k fosse 2, somente FA, $F.S$ e $F..T$ teriam suporte aceitável. A cada um destes padrões aplica-se o procedimento **Extend** (no segundo nível da recursão).

Exercício 1 : Como calcular $L_S(P')$ em função do conjunto $L_S(P)$? Melhorar o pseudo-código do programa apresentado no final desta seção, acrescentando linhas de código que permitem calcular $L_S(P')$ em função de $L_S(P)$ e dos elementos do array (veja quarta linha de baixo para cima no pseudo-código abaixo).

O pseudo-código da Fase de Varredura é o seguinte:

Fase da Varredura

$EP = [\ ]$;
for all $\sigma \in \Sigma$

Calcule $L_S(\sigma)$;
if $\sup(\sigma) \geq k$
Extend(σ, $L_S(\sigma)$)
end-if
end-for

Procedure **Extend**(P, $L_S(P)$)
array B[| Σ |]

$A(P)$ = número de resíduos em P
if $A(P) = L$
$EP = append(EP, (P, L_S(P)))$
return
end-if
For $i = 0$ **to** $(W- \mid P \mid) - (L - A(P))$
for all $\sigma \in \Sigma$
$B[\sigma] = \emptyset$;
end-for
$P' = P$ concatenado com i curingas;
for $(x, y) \in L_S(P)$
if $(y+ \mid P \mid +i) < \mid s_x \mid$;
$\sigma = s_x[y+ \mid P \mid +i]$
(= % *elemento da posição* $y+ \mid P \mid +i$ *da sequência* s_x*).*
Insira $(x, y+ \mid P \mid +i)$ em $B[\sigma]$
end-if
end-for
for all $\sigma \in \Sigma$
if $\sup(P'\sigma)) \geq k$
(% *calcula-se este suporte a partir do conteúdo de* $B[\sigma]$);
Extend(P',$L_S(P')$)
end-if
end-for
end-for

Exercício 2 : Execute a fase de varredura de TEIRESIAS sobre o input $S = \{s_1 =$ FASTS, $s_2 =$ LFBST, $s_3 =$ FAPK$\}$, $k = 3$, $L = 3$, $W = 4$.

A Fase da Convolução

Nesta fase, vamos calcular o conjunto dos padrões frequentes maximais. Este conjunto, que denominamos Maximal, é inicializado como o conjunto vazio.

Ordena-se todos os padrões elementares calculados na fase de varredura, utilizando a ordem segundo o prefixo. Cada padrão elementar será considerado como uma semente com a qual construiremos progressivamente novos padrões, convoluindo padrões elementares à esquerda e à direita da semente e dos padrões já convoluidos.

Primeiro round de convoluções

1. Consideramos o menor padrão elementar P segundo a ordem do prefixo. Caso ele não seja subentendido por um padrão que está no conjunto Maximal, inserimo-lo no stack St (no primeiro round de convoluções, o conjunto Maximal é vazio, mas no segundo round ele não é e portanto este teste faz sentido) ;

2. Seja T o padrão do topo do stack. No início, este elemento é P. Sabemos com certeza que ele é maximal até o momento. Seja w o prefixo de T. Consideramos todos os padrões elementares que terminam em w.

   - Ordenamos estes padrões pela ordem segundo o sufixo.
   - Convoluímos cada um destes padrões à direita de T.
   - A cada vez que fazemos a convolução obtendo um novo padrão R, fazemos os testes (1) | $L_S(R) = L_S(T)$ e (2) $\sup(R) > k$ e R não é subentendido por nenhum padrão dentro do conjunto Maximal.

     Temos 4 possibilidades a considerar:

     (a) (1) e (2) são positivos: neste caso, retiramos T do topo do stack e desistimos de convoluir T à direita. Inserimos R no stack e começamos tudo novamente com R como semente, isto é, começamos a convoluir R à direita com todos os padrões elementares onde a convolução for possível.

     (b) (1) é positivo e (2) é negativo : neste caso, retiramos T do topo do stack mas continuamos a convoluí-lo com os restantes dos padrões elementares.

     (c) (1) é negativo e (2) é positivo: neste caso, T fica no stack (T é potencialmente maximal), R entra no topo do stack e reiniciamos todo o processo de convolução à direita com R como semente, interrompendo a convolução de T com os restantes dos padrões elementares à esquerda.

     (d) (1) e (2) são negativos: neste caso, T fica no topo do stack e continuamos a convoluí-lo com os restantes dos padrões elementares.

- No momento em que o padrão que estiver no topo do stack não puder ser mais estendido à esquerda, o mesmo processo é aplicado tentando-se estender o padrão do topo do stack pela direita.
- Quando todas as extensões à direita e à esquerda se esgotam, o topo atual do stack é retirado do stack e inserido no conjunto Maximal e retornado como resposta.
- Todo o processo se reinicia novamente com o novo topo do stack.

Segundo round de convoluções

Quando o stack se esvazia, o próximo padrão P_2 na ordem dos prefixos é entrado no stack e o segundo round de convoluções se inicia agora com a nova semente sendo P_2.

Os outros rounds são análogos: um para cada padrão elementar, na ordem dos prefixos. O processo termina quando todos os padrões elementares foram utilizados como sementes para as convoluções à esquerda e à direita.

Na lista de exercícios s ao propostos exercícios que visam a provar a corretude do algoritmo TEIRESIAS, isto é, que realmente o processo descrito acima produz todos os padrões maximais frequentes e somente eles.

Abaixo, descrevemos o pseudo-código da Fase de Convolução. Vamos utilizar a seguinte subrotina :

Is-maximal(P) : retorna 0 se o padrão P é subentendido por um padrão $Q \in$ Maximal (onde Maximal = conjunto dos padrões maximais), e retorna 1 em caso contrário.

Também utilizaremos a seguinte notação :
$DirP(w)$ = todos os padrões elementares começando com w
$DirS(w)$ = todos os padrões elementares terminando com w

Fase da Convolução

Ordena EP segundo a ordem dos prefixos,
Maximal := $\emptyset$;
while $EP \neq \emptyset$
  P = menor elemento de EP na ordem dos prefixos;
  if Is-maximal(P) **then**
   push(P);
  end-if

```
    while stack ≠ ∅
start:
      T = topo do stack;
      w = prefixo(T);
      U = {Q ∈ DirS(w) | Q, T ainda não foram convoluídos};
      while U ≠ ∅
       Q = menor elemento de U de acordo com a ordem dos sufixos
       R = Q ⊕ T

       if | L_S(R) |=| L_S(T) | then
        retire o topo do stack
       end-if

       if sup(R) ≥ k e Is-maximal(R) then
        push(R)
        goto start
       end-if
      end-while

      w = sufixo(T);
      U = {Q ∈ DirP(w) | Q, T ainda não foram convoluídos};
      while U ≠ ∅
       Q = menor elemento de U de acordo com a ordem dos prefixos
       R = T ⊕ Q

       if | L_S(R) |=| L_S(T) | then
        retire o topo do stack
       end-if

       if sup(R) ≥ k e Is-maximal(R) then
        push(R)
        goto start
       end-if
      end-while
      T = pop stack (retira o topo);
      Maximal := Maximal ∪ {T};
      Retorne T
    end-while
end-while
```

9.5.4 Discussão sobre os resultados Experimentais de TEIRESIAS

Durante os experimentos realizados com o algoritmo, observou-se aue o fator mais importante que afeta a performance do mesmo é a quantidade de similaridades entre as sequências de input.

Nos experimentos foi utilizada uma sequência P de 400 amino-ácidos; esta sequência foi gerada por um gerador aleatório de proteínas que pode ser encontrado em :

http://bo.expasy.org/tools/randseq.html

Fixou-se uma porcentagem X e utilizou-se a proteina P para obter 20 novas proteínas derivadas, cada uma delas tendo uma similaridade de $X\%$ com P ($X\%$ de coincidências entre as duas proteínas). Utilizou-se 6 conjuntos de proteínas assim obtidos : para X sendo 40%, 50%, 60%, 70%, 80% e 90%. Para cada um dos conjuntos de proteínas, executou-se TEIRESIAS utilizando-se um diferente k (nível mínimo de suporte). Para cada escolha do suporte, vários valores de W foram utilizados (entre eles 10 e 15). O parâmetro L foi fixado em 3. A cada execução do algoritmo, foi testado (a) o tempo de execução e (2) o número total de padrões maximais retornados pelo algoritmo.

Como esperado, o tempo de execução do algoritmo foi de alguns segundos quando o suporte mínimo k era próximo do número total de sequências. Além disto, a variação entre a similaridade dos grupos de sequências não teve muito efeito para suportes próximos ao total de sequências. A razão para isto é que, não importa o quanto similares são as sequências, não há muitos padrões que ocorrem em todas (ou quase todas) as sequências de input.

Entretanto, quando o suporte diminui, o grau de similaridade entre as sequências é um fator importante no desempenho do algoritmo. Mais precisamente, quanto maior a similaridade, maior é o tempo de execução. Isto é uma consequência direta do fato de que à medida que cresce a similaridade entre as sequências de input, maior é o número de padrões possivelmente frequentes. Só para se ter uma idéia, o pior tempo de execução foi pouco mais de 15 minutos, para $W = 15$, suporte mínimo de 12 sequências em 20 e grau de similaridade 90%. Neste caso foram produzidos perto de 230 mil padrões frequentes maximais.

Os resultados experimentais mostraram que o tempo gasto na execução do algoritmo é proporcional ao número de padrões retornados. Conclui-se que o tamanho do output parece ser o fator preponderante que afeta a performance do algoritmo: o tempo de execução permance razoavelmente baixo mesmo quando o input é grande, se estes contém um número moderado de padrões frequentes. Isto prova que, na verdade, a quantidade de trabalho executada pelo algoritmo é o absolutamente necessário.

Capítulo 10

Data Mining Incremental

Neste capítulo, vamos tratar do problema da mineração incremental de dados, isto é, como calcular os padrões frequentes com relação a um banco de dados obtido inserindo-se novas tuplas em um banco de dados D, tendo já calculado os padrões frequentes com relação ao banco de dados D. Isto é, queremos estudar a possibilidade de se aproveitar o conhecimento minerado *antes* de um update, no processo de mineração de padrões no banco de dados modificado. A figura 1, ilustra o processo:

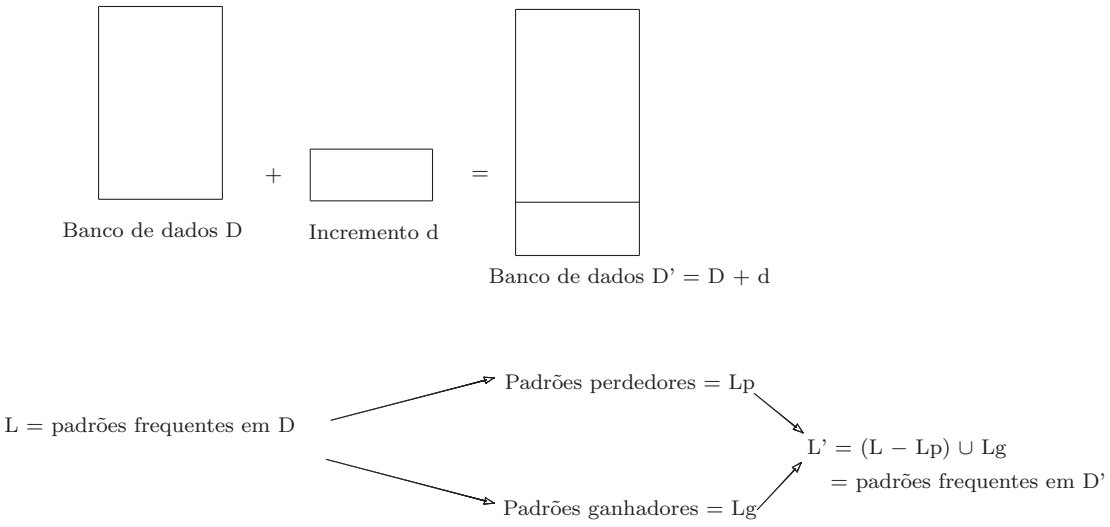


Figura 10.1: Update dos itemsets frequentes

Vamos estudar o problema de mineração incremental no caso onde os padrões são *itemsets* (ou mais particularmente, regras de associação). O algoritmo FUP (Fast Update) que vamos apresentar aqui, foi introduzido em [35]. Outro problema de grande

importância é o da mineração incremental de padrões sequenciais, que não tem recebido muita atenção dos pesquisadores. Um trabalho importante e recente neste tema e que recomendamos ao leitor é [36].

Por que o interesse em mineração de dados incremental, já que o objetivo da mineração de dados é basicamente extrair padrões frequentes ocorrendo em um banco de dados suficientemente grande e depois utilizar estes padrões para a tomada de decisões sobre outros dados ? A razão é que os padrões extraídos de um banco de dados somente refletem o estado atual do banco de dados. Para que estes padrões sejam considerados *estáveis e confiáveis* é necessário que sejam extraídos de um grande volume de dados que evolua durante um certo período de tempo. Somente após um certo tempo, os padrões que apresentarem estabilidade durante a evolução do banco de dados serão considerados confiáveis.

Pode-se imaginar duas maneiras de realizar esta tarefa :

1. a cada update, aplica-se o algoritmo de mineração do padrão específico que se quer minerar sobre o novo banco de dados e compara-se os diferentes padrões extraídos do banco de dados antigo e os extraídos do novo banco de dados.

2. aproveita-se os padrões extraídos do banco de dados antigo, dos quais serão eliminados os padrões que deixarão de ser frequentes no novo banco de dados e minera-se somente os novos padrões que não eram frequentes antes e que passarão a ser frequentes no novo banco de dados. Isto é, se L é o conjunto dos padrões frequentes segundo o banco de dados antigo e L' é o conjunto dos padrões frequentes com relação ao novo banco de dados (após o update) então :

$$L' = (L - Lp) \cup Lg$$

onde :

- $L_p \subseteq L$ = padrões que eram frequentes antes do update e que deixam de ser frequentes após o update. Estes padrões são chamados de *padrões perdedores*.
- $Lg \subseteq \overline{L}$[1] = padrões que não eram frequentes antes do update e que passam a ser frequentes após o update. Estes padrões são chamados de *padrões ganhadores*.

Vamos apresentar o algoritmo FUP que opera de acordo com a segunda idéia descrita acima. Na seção final destas notas de aula, discutimos os resultados experimentais que foram realizados sobre dados sintéticos e que comparam as performances de FUP com sucessivas aplicações de Apriori e de DHP sobre o banco de dados modificado sem levar em conta os padrões extraídos anteriormente.

[1] $\overline{L}$ denota o complemento de L com relação ao conjunto de todos os padrões.

10.1 Idéia geral do algoritmo FUP

A idéia geral do algoritmo FUP é a seguinte: suponha que D é o banco de dados atual e que um conjunto d de novas transações é inserido a D obtendo o novo banco de dados D'. Suponha também que já foram minerados itemsets frequentes com relação ao banco de dados D. Para cada k, denotamos por L_k o conjunto de itemsets frequentes extraídos do banco de dados D. Para cada itemset frequente $I \in L_k$, temos armazenado seu contador(I, D), isto é, o número de transações de D que suporta I. Vamos denotar por L'_k o conjunto de itemsets frequentes extraídos de D' utilizando o algoritmo de mineração incremental FUP. Este conjunto é obtido na iteração k de FUP, da seguinte maneira:

1. **Fase da identificação e eliminação dos perdedores**: esta fase é realizada em duas etapas:

   (a) utilizamos a seguinte propriedade para podar elementos de L_k : se $X \in L_k$ e X contém um $(k-1)$-itemset que é um perdedor na iteração $k-1$, então X não pode ser frequente e portanto pode ser podado. Esta fase de poda só é realizada nas iterações $k > 1$.

   (b) após a poda de L_k, os contadores dos itemsets restantes de L_k são atualizados varrendo-se somente o incremento d. Obtém-se assim contador(X,D'). Os perdedores X são identificados, fazendo-se o seguinte teste:

   $$\text{contador}(X, D') < minsup * (\mid d \mid + \mid D \mid) \ ?$$

   É claro que se este teste for negativo, então X é um perdedor e deve ser retirado de L_k. Denotamos por L_k^{old} o conjunto restante dos elementos de L_k (isto é, os não-perdedores).

2. **Fase da Geração de Candidatos** : esta fase, na verdade, é executada antes que a segunda poda de perdedores se inicie. Calcula-se o conjunto de candidatos C_k, juntando-se (como em Apriori) L'_{k-1} com L'_{k-1}. Quando $k = 1$ (na primeira iteração), C_1 é simplesmente os itemsets unitários constituídos de itens que não estão presentes em L_1. Retira-se desta junção os elementos que estão em L_k, pois estes já sabemos tratar: ou estão em L_k^{old} e portanto são frequentes (escaparam da poda na fase anterior), ou não estão em L_k^{old} e portanto, não são frequentes.

3. **Fase da Poda dos novos candidatos** : ao mesmo tempo em que se varre o incremento d a fim de se podar os perdedores em L_k, faz-se a contagem do suporte dos novos candidatos C_k: para cada nova transação t de d verifica-se quais dos novos candidatos são suportados por t, incrementando-se os contadores daqueles

candidatos que são suportados por t. No final da varrida de d, verifica-se, para cada candidato X se:

$$\text{contador}(X, d) < minsup* \mid d \mid$$

Se X é tal que contador$(X, d) < minsup* \mid d \mid$ então X não tem chance nenhuma de ser frequente com relação ao banco de dados $D' = D \cup \{d\}$. Para concluir isto, utilizamos a seguinte proposição :

Proposição 10.1 Se contador$(X, d) < minsup*d$ então contador$(X, D') < minsup*$ $(\mid D \mid + \mid d \mid)$.

Prova: contador(X, D') = contador(X, D) + contador(X, d). Como sabemos que X não é frequente com relação a D (lembre-se que excluimos de C_k os elementos de L_k), então contador$(X, D) < minsup* \mid D \mid$. Se contador$(X, d) < minsup* \mid d \mid$ então teríamos:

$$\text{contador}(X, D') = \text{contador}(X, D) + \text{contador}(X, d) < minsup(\mid D \mid + \mid d \mid)$$

4. **Fase da identificação dos ganhadores ou cálculo do suporte dos novos candidatos** : Após a poda dos novos candidatos, os que restam são testados, varrendo-se o banco de dados D. Na fase anterior, de poda dos candidatos, os contadores dos mesmos foram estocados. Quando se varre o banco de dados D, estes contadores são incrementados ou não a cada transação de D testada: caso esta suporte o candidato, o contador do mesmo é incrementado. No final da varrida de D, todos os candidatos com contador superior a minsup $*(\mid D \mid + \mid d \mid)$ são identificados como os novos itemsets frequentes *ganhadores*. Este conjunto é denotado por L_k^{new}. O conjunto L'_k dos itemsets frequentes na iteração k é dado por:

$$L'_k = L_k^{old} \cup L_k^{new}$$

10.2 Exemplo de uso

Primeira Iteração

Vamos supor que temos um banco de dados D com tamanho 1000 no qual foram minerados os conjuntos de 1-itemsets frequentes :

L_1	suporte
$\{I_1\}$	32
$\{I_2\}$	31

Estamos supondo um nível mínimo de suporte minsup = 3%. Assim, para que um itemset seja frequente é preciso que seja suportado por ao menos 30 transações. Na tabela acima, são registrados, na coluna *suporte* o número de transações de D que suportam o itemset.

O conjunto de itens inicial é $\{I_1, I_2, I_3, I_4\}$. Suponhamos agora que um update é realizado sobre D, com um incremento d de tamanho 100 (100 novas transações são inseridas em D).

1. **Fase da identificação e eliminação dos perdedores**

   Na iteração 1, a primeira etapa da poda de L_1 não ocorre. Passa-se diretamente para a segunda etapa: calcula-se o contador de cada um dos elementos de L_1 com relação ao incremento d. Suponhamos que contador(I_1,d) = 4, contador(I_1,d) = 1. Temos que 3% 1100 = 33. Logo,

   contador($\{I_1\}, D'$) = $32 + 4 = 36 > 33$
   contador($\{I_2\}, D'$) = $31 + 1 = 32 < 33$

   Logo, $\{I_2\}$ é perdedor em L_1 e é podado. Portanto, $L_1^{old} = \{I_1\}$.

2. **Fase da Geração de Candidatos** : $C_1 = \{\{I_3\}, \{I_4\ \}\ \}$, já que estes são os itens que não aparecem em L_1.

3. **Fase da Poda dos Candidatos**: Suponhamos que os contadores dos itemsets de C_1 com relação a d são:

   contador($\{I_3\}$, d) = 6
   contador($\{I_4\}$, d) = 2

   Ora, 3% de 100 = 3. Logo, somente $\{I_3\}$ tem chance de ser frequente no banco de dados $D' = D \cup \{d\}$. O candidato $\{I_4\}$ é podado de C_2.

4. **Fase da identificação dos ganhadores**: Agora, varre-se o banco de dados D para calcular o suporte de $\{I_3\}$. Suponhamos que contador($\{I_3\}$, D) = 28. Então, contador($\{I_3\}$, D') = 28 + 6 = 34 > 33. Logo, $\{I_3\}$ é um ganhador. Temos então :

$$L_1' = \{\{I_1\}, \{I_3\}\}$$

Na verdade, na primeira iteração não é necessário varrer-se o banco de dados D para calcular-se o contador dos candidatos, pois podemos supor que durante o cálculo de L_1 foram estocados os contadores de todos os itens com relação ao banco de dados D. Caso algum novo item apareça em d (que não aparecia em D) , seu contador é zero com relação a D.

Segunda Iteração

Suponhamos a mesma situação anterior, onde dispomos do mesmo conjunto de itens $\{I_1, I_2, I_3, I_4\}$, um banco de dados D e um incremento d com tamanhos idênticos ao exemplo anterior, e um nível de suporte mínimo igual a 3%. Suponhamos que L_2 é dado por:

L_2	suporte
$\{I_1, I_2\}$	50
$\{I_2, I_3\}$	31

Suponhamos também que a primeira iteração do algoritmo FUP produziu o conjunto:

$$L'_1 = \{\{I_1\}, \{I_2\}, \{I_4\}\}$$

Vamos agora encontrar L'_2 na segunda iteração de FUP:

1. **Fase da identificação e eliminação dos perdedores**:

   (a) Etapa da poda de L_2: o primeiro elemento de L_2 não é podado, pois todos os seus 1-subitemsets estão em L'_1. Mas o segundo elemento é podado pois contém o 1-itemset $\{I_3\}$ que não está em L'_1.

   (b) Etapa da varredura de d : varremos d para incrementar o contador dos elementos restantes (isto é, de $\{I_1, I_2\}$). Suponhamos que contador($\{I_1, I_2\}$,d) = 3. Logo, contador($\{I_1, I_2\}$,D') $= 50 + 3 = 53 > 33$. Logo, $L_2^{old} = \{\ \{I_1, I_2\}\ \}$.

2. **Fase do Cálculo dos novos candidatos**: Faz-se a junção de L'_1 com L'_1 obtendo-se :

$$C_2 = \{\{I_1, I_2\}, \{I_1, I_4\}, \{I_2, I_4\}\}$$

Retira-se de C_2 os elementos de L_2, pois estes já foram tratados anteriormente: sabemos que são frequentes, caso estejam em L_2^{old}, e sabemos que não são frequentes, caso tenham sido podados de L_2. Assim, os candidatos C_2 passam a ser:

$$C_2 = \{\{I_1, I_4\}, \{I_2, I_4\}\}$$

3. **Fase da Poda dos Candidatos**: Suponhamos que os contadores dos itemsets de C_2 com relação a d são:

   contador($\{I_1, I_4\}$,d) = 5
   contador($\{I_2, I_4\}$, d) = 2

   Como contador($\{I_2, I_4\}$,d) = $2 < 3 = 3\% \times 100$, então $\{I_2, I_4\}$ é podado. O outro itemset $\{I_1, I_4\}$ não é podado.

4. **Fase da identificação dos ganhadores**: Agora, varre-se o banco de dados D para incrementar o contador do candidato que sobrou $\{I_1, I_4\}$. Suponha que contador($\{I_1, I_4\}$, D) = 30. Então :

   contador($\{I_1, I_4\}$, D) = $30 + 5 = 35 > 33$

   Assim, $\{I_1, I_4\}$ é um ganhador. O resultado final da iteração 2 é :

$$L'_2 = \{\{I_1, I_2\}, \{I_1, I_4\}\}$$

Exercício: Na primeira lista de exercícios, sobre regras de associação, foi proposto um exercício que envolvia o cálculo dos itemsets frequentes após um update. A solução proposta foi baseada na idéia de considerar o banco de dados D' contendo duas partições D e d. Aplica-se Apriori sobre D, depois sobre d. Os dois resultados são testados em $D \cup d$ para ver quais dos itemsets localmente frequentes permanecem frequentes globalmente. Faça uma análise comparativa desta solucão com a solução proposta pelo algoritmo FUP. Qual, na sua opinião, é a proposta com melhor performance ? Justifique sua opinião.

10.3 Resultados Experimentais: Comparação com Apriori e DHP

O algoritmo FUP foi testado em dados sintéticos construídos utilizando um gerador de itemsets. Este gerador foi adaptado para gerar incrementos adequados. Recomendamos ao leitor a leitura cuidadosa da seção 4 de [35] onde são discutidos com detalhes os resultados experimentais do algoritmo FUP.

FUP versus Apriori e DHP variando-se o suporte

Comparou-se a performance de FUP com a de Apriori e de DHP. Verificou-se que para suportes pequenos, FUP é 3 a 6 vezes mais rápido do que DHP e 3 a 7 vezes mais rápido do que Apriori. Isto é, se executarmos Apriori sobre D e D' e Apriori sobre D e em seguida FUP sobre D, obtemos uma melhor performance com a execução incremental Apriori+FUP. Para suportes grandes, o custo computacional da primeira opção (não-incremental) diminui (pois o número de itemsets frequentes diminui), mas mesmo assim FUP é mais rápido (2 a 3 vezes mais rápido).

Performance de FUP com relação ao tamanho do incremento

Em geral, quanto maior for o incremento, mais tempo vai demorar para se realizar o update dos itemsets. E com isso, o ganho de performance de FUP sobre Apriori e DHP diminui. Realizou-se testes sobre um mesmo banco de dados, com incrementos variando de 1K, 5K e 10K, para diferentes graus de suporte. Por exemplo, quando o suporte é de 2%, o ganho de performance decresceu de 5,8 vezes (relativo ao menor incremento) para 3,7 (relativo ao maior incremento).

Testou-se também se para algum incremento suficientemente grande, a performance de FUP poderia ser inferior à performance de Apriori e de DHP. Para isto, variou-se o tamanho do incremento de 1K a 350K, sendo que o tamanho do banco de dados original era de 100K (portanto, testou-se mesmo um incremento 3,5 vezes maior do que o banco de dados original). A performance de FUP decresceu conforme o tamanho do incremento aumentou, mas mesmo para o maior incremento, nunca foi inferior a de Apriori e DHP. Recomendamos ao leitor estudar os gráficos de performance apresentados na seção 4 de [35].

Overhead de FUP

Um outro teste realizado foi o de analisar o *overhead* de FUP. Seja T_1 o tempo gasto para se calcular o conjunto dos itemsets frequentes L no banco de dados D utilizando um algoritmo qualquer de mineração $\mathcal{A}$. Seja T_{FUP} o tempo gasto por FUP para calcular os itemsets frequentes L' no banco de dados incrementado $D \cup d$, utilizando L. Seja T_2 o tempo gasto pelo algoritmo $\mathcal{A}$ para extrair L' diretamente de $D \cup d$. Em geral :

$$T_1 + T_{FUP} > T_2$$

Observamos que nos testes anteriores comparamos os tempos T_{FUP} e T_2 !

A diferença entre T_{FUP} e $T_2 - T_1$ é o *overhead* de FUP. Esta diferença mede de fato o tempo gasto no *update* dos itemsets. Se esta diferença é pequena, então o *update* foi realizado de forma eficiente. Testes foram realizados para avaliar esta diferença. Constatou-se que quanto maior for o incremento, menor é o *overhead* de FUP. Nos experimentos descobriu-se que quando o incremento é muito menor do que o banco de dados original, a porcentagem de *overhead* varia de 10 a 15%. Quando o incremento é maior do que o tamanho original, o overhead cai muito rapidamente de 10 a 5%.

Este resultado {e bastante encorajador, pois mostra que FUP não só pode se beneficiar de incrementos pequenos (como vimos no parágrafo sobre a performance de FUP com relação ao tamanho do incremento) como também funciona bem no caso de grandes incrementos, quando consideramos o fator *overhead*.

FUP versus Apriori e DHP variando-se o tamanho do banco de dados

Realizou-se experimentos com um banco de dados contendo 1 milhão de transações. Verificou-se que a taxa de performance entre DHP e FUP para este banco de dados variou entre 6 a 16 vezes. Este resultado mostra que o ganho de performance de FUP com relação a DHP aumenta a medida que o banco de dados torna-se maior.

Bibliografia

[1] Agrawal, R., Srikant, R. : Fast Algorithms for Mining Association Rules. Proc. 20th Int. Conf. Very Large Data Bases, VLDB, 1994.

[2] R. Srikant, Q. Vu, R. Agrawal: Mining Association Rules with Item Constraints Proc. of the 3rd Int'l Conference on Knowledge Discovery in Databases and Data Mining, Newport Beach, California, August 1997.

[3] Agrawal, R., Srikant, R. : Mining Sequential Patterns. In Proc. of 1995 Int. Conf. on Data Engineering, Taipei, Taiwan, March 1995.

[4] Agrawal, R., Srikant, R. : Mining Sequential Patterns : Generalizations and Performance Improvements. Proc. 5th EDBT, 3-17, 1996.

[5] M. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: sequential pattern mining with regular expression constraints. Proc. VLDB, 223-234, 1999.

[6] M. Garofalakis, R. Rastogi, and K. Shim. Mining Sequential Patterns with Regular Expression Constraints. IEEE Transactions on Knowledge and Data Engineering Vol. 14, No. 3, May/June 2002, pp. 530-552/

[7] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery, 1(3), 259-289, 1997.

[8] A.Savasere et al. : An Efficient Algorithm for Mining Association Rules in Large Databases. VLDB 1995.

[9] Ian H. Witten, Eibe Frank : Data Mining : Practical Machine Learning Tools and Techniques with Java Implementations,Capítulo 6, Academic Press, 2000.

[10] D.E.Rumelhart, G.E.Hinton, R.J.Williams: Learning internal representations by error propagation. In D.E.Rumelhart and J.L.McClelland, editors, Parallel Distributed Processing, Cambridge, MA,MIT Press, 1986.

[11] S.M.Weiss, C.A. Kulikowski : Computer Systems that learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning and Expert Systems. San Mateo, CA:Morgan Kaufmann, 1991.

[12] H. Lu, R. Setiono, H. Liu : Neurorule: A connectionist approach to data mining. In Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95), 478-489, Zurich, Switzerland, Sept. 1995.

[13] Asim Roy : Artificial Neural Networks - A Science in Trouble SIGKDD Explorations, Vol. 1, Issue 2, January 2000, pp. 33-38.

[14] R. Setiono: A Penalty-Function Approach for Pruning Feedforward Neural Networks. Neural Computation, Vol. 9, n. 1, pages 185-204, 1997.

[15] Guha, S., Rastogi, R., Shim, K. : Cure: An Efficient Clustering Algorithm for Large Databases. In Proc. ACM/SIGMOD Conference on Management of Data, 1998.

[16] Tian Zhang, Raghu Ramakrishnan, Miron Livny: Birch: An efficient data clustering method for very large databases. In Proc. ACM/SIGMOD Conference on Management of Data, 1996.

[17] M.Ester, H.-P. Kriegel, J. Sander, X.Xu: A density-based algorithm for discovering clusters in large spatial databases with noise . In Proc. 1996 International Conference on Knowledge Discovery and Data Mining (KDD'96), pages 226-231, Portland, USA, Aug. 1996.

[18] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest: Introduction to Algorithms. The MIT Press, Massachussets, 1990.

[19] Ng, R.T., Han, J. : Efficient and Effective Clustering Methods for Spatial Data Mining. Proceedings of the 20th International Conference on Very Large Data Bases, 1994.

[20] Edwin M. Knorr, Raymond T. Ng: Algorithms for Mining Distance-Based Outliers in Large Datasets. Proceedings of the 24th International Conference on Very Large Databases, VLDB 1998, New York, USA.

[21] D. Hawkins : Identification of Outliers. Chapman and Hall, London, 1980.

[22] Kosala, R., Blockeel, H.: Web Mining Research : A Survey. SIGKDD Explorations, Vol. 2, Issue 1, July 2000.

[23] Cooley, R., Mobasher, B., Srivastava, J.: Web Mining: Information and Pattern Discovery on the World Wide Web. Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97), November 1997.

[24] Soumen Chakrabarti et al.: Mining the Link Structure of the World Wide Web. IEEE Computer, 32(8), August 1999.

[25] Soumen Chakrabarti: Data mining for hypertext: A tutorial survey. ACM SIGKDD Explorations, 1(2):1-11, 2000.

[26] Stephen J. DeRose: What do those weird XML types want, anyway ? Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

[27] Victor Vianu : A Web Odyssey: from Codd to XML. In ACM PODS (Symposium on Principles of Database Systems), Santa Barbara, CA, May 2001.

[28] Garofalakis, M., Rastogi, R., Seshadri, S., Shim, K.: Data Mining and the Web: Past, Present and Future. ACM Workshop on Web Information and Data Management (WIDM), pp. 43-47, 1999.

[29] M.S. Chen, J. S. Park, P.S. Yu : Efficient Data Mining for Path Traversal Patterns. IEEE Transactions on Knowledge Discovery and Data Engineering 10(2), 209-221, Mars 1998.

[30] M.S. Chen, J. S. Park, P.S. Yu : An Effective Hash-Based Algorithm for Mining Association Rules. In ACM SIGMOD International Conference on Management of Data, May 1995.

[31] Brejova, B., DiMarco, C., Vinar, T., Hidalgo, S.R., Holguin, G., Patten, C. : Finding Patterns in Biological Sequences. Project Report, Department of Biology, University of Waterloo, 2000.

[32] Floratos, A. Pattern Discovery in Biology: Theory and Applications. Ph.D. Thesis, Department of Computer Science, New York University, Jan. 1999.

[33] Brazma, A., Jonassen, I., Eidhammer, I., Gilbert, D.: Approaches to the automatic discovery of patterns in biosequences. Journal of Computational Biology. 5 (2): 279-305, 1998.

[34] L.C.K. Hui : Color set size problem with applications to string matching. In Combinatorial Pattern Matching, pages 230-243, 1992.

[35] Cheung, D.W., Han, J., Ng, V.T., Wong, C.Y.: Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Tecnique. In International Conference on Data Engineering, ICDE 1996.

[36] Zheng Q., Xu K., Ma S., Lv W. : The algorithms of Updating Sequential Patterns. Proc. of 5th International Workshop on High Performance Data Mining, in conjunction with 2nd SIAM Conference on Data Mining, 2002.