3 Agrupamento de dados e o algoritmo GNG

Processos de agrupamento de dados são utilizados para categorizar uma massa de dados em um número desejado de grupos, com a restrição de que os componentes de cada grupo compartilhem de características semelhantes.

Tanto o procedimento de extração de horizontes sísmicos quanto o de extração de falhas desenvolvidos neste trabalho se baseiam em uma fase de pré-processamento a ser aplicada ao volume sísmico sendo interpretado. Esse procedimento irá gerar um novo volume, obtido a partir dos dados existentes no volume original, e faz uso de um procedimento pertencente a uma classe de algoritmos denominados genericamente como processos de agrupamento de dados (*clusterization*). Nosso trabalho utiliza um algoritmo de agrupamento de dados particular, conhecido como Gás Neural Evolutivo ou GNG (*Growing Neural Gas)* [Fritzke 1995]. Neste capítulo apresentamos as definições do que vêm a ser os processos de agrupamento, assim como o algoritmo aqui adotado com um nível de detalhe que julgamos ser suficiente para servir de introdução a um leitor pouco familiarizado com métodos dessa natureza. O capítulo está organizado da seguinte maneira: primeiramente, apresentamos uma definição do que vem a ser o procedimento de agrupamento de dados, oferecendo um exemplo que ilustra o processo, e definindo-o mais formalmente. Feito isso, citamos alguns dos algoritmos mais difundidos na literatura, e em seguida apresentamos detalhadamente o Gás Neural Evolutivo (GNG), algoritmo que utilizamos como base da nossa proposta para identificação de horizontes e falhas em volumes sísmicos.

3.1 Processos de agrupamento de dados

Processos de agrupamento podem ser descritos informalmente como métodos cujo objetivo principal é, dada uma massa de dados inicial, separar essa massa de dados em um número desejado de grupos, com a restrição de que os componentes de cada grupo compartilhem de características semelhantes. Isso significa que o procedimento de separar os dados em conjuntos distintos faz uso de um critério de medida utilizado para definir o quanto dois componentes quaisquer da massa de dados são parecidos. Esse critério de medida geralmente é traduzido pelo uso de uma função de similaridade particular, que varia de acordo com a natureza dos dados sendo considerados.

3.1.1 Um exemplo simples

Como exemplo, podemos supor que nossa massa de dados inicial sejam todos os pixels de um determinado arquivo de imagem, do tipo RGB. Essa imagem possui um número de cores que conhecemos a priori. Foi-nos dada a tarefa de diminuir o número de cores da imagem original, com a condição de que o arquivo resultante seja o mais parecido possível com a imagem original. Nesse caso a massa de dados a ser agrupada seria formada por um conjunto de vetores tridimensionais, com cada pixel da imagem original sendo representando pelo seu vetor de cores correspondente (3 componentes de cor, R, G e B).

Uma estratégia para diminuir o número total de cores poderia ser encontrar na imagem original grupos de pixels cujas cores fossem bastante parecidas. Caso um grupo de pixels tivesse cores parecidas entre si, uma cor única poderia ser eleita para esse grupo (essa cor poderia ser a cor média entre todos os elementos do grupo, por exemplo), e todos os pixels pertencentes ao grupo na imagem original seriam substituídos por essa cor.

Como foi dito anteriormente, para que o procedimento funcione corretamente

é necessário estabelecer um critério que indique mais formalmente o quanto duas cores se parecem. No caso da imagem, uma função para medir o quanto duas cores são parecidas entre si poderia ser a distância euclidiana entre essas duas cores no espaço RGB ($\Re^3$). Assim a nossa interpretação para essa tarefa seria de que quanto menor for a distância euclidiana entre as duas cores mais elas se parecem.

Dessa forma caso queiramos diminuir o número de cores de uma imagem, por exemplo de 148.279 cores para 32 cores (um possível resultado pode ser visto na figura 3.1), basta que encontremos 32 grupos de cores próximas segundo a distância euclidiana a partir do conjunto de 148.279 cores inicial e substituir as cores iniciais pelas cores representantes do seu grupo.

De posse desse exemplo, podemos descrever mais formalmente os processos de agrupamento.

3.2 Definição do processo

Agrupamento de dados pode ser descrito como o processo de organizar uma coleção de vetores *k*-dimensionais em um número *n* de grupos cujos membros compartilham características similares segundo alguma função de medida. Depois de formado, cada um desses *n* grupos é representado por somente um vetor *k*-dimensional denominado vetor de código (*codevector*, outros nomes utilizados seriam *centre*, ou simplesmente *node*).

O objetivo do processo é, portanto, reduzir a diversidade de grandes quantidades de dados, para isso categorizando esses dados em *n* conjuntos de itens similares.

Quantização vetorial (*Vector Quantization* – VQ) é o processo de quantizar vetores de entrada *k*-dimensionais em vetores de saída também *k*-dimensionais, denominados vetores de código. Ao conjunto de possíveis vetores de código dá-se o nome de livro de código (*codebook*). O livro de código é geralmente obtido através do agrupamento de um conjunto de vetores iniciais, denominado

conjunto de amostras (algumas vezes referido genericamente como *training set*).

Figura 3.1 – Ilustração do processo de quantização de cores de uma imagem através de agrupamento de dados. A imagem original possui 148.279 cores. Já a segunda imagem possui somente 32 cores (fonte: www.lenna.org).

3.3 Algoritmos difundidos

Ao longo das últimas décadas, vários algoritmos dessa classe têm sido desenvolvidos. Dentre eles, podemos citar alguns que mais se destacam, como K-means [MacQueen 1967], Mapas Auto-Organizáveis de Kohonen (*Kohonen's Self Organizing Map* – SOM) [Kohonen 1982] e o algoritmo Gás Neural (*Neural Gas algorithm*).

Aqui vamos apresentar um algoritmo específico, o Gás Neural Evolutivo (*Growing Neural Gas*), ou simplesmente GNG, desenvolvido por Bernd Fritzke [Fritzke 1995], algoritmo esse no qual os grupos, também denominados nós ou neurônios, são obtidos de forma incremental e evolutiva. Além do artigo original de Fritzke, outros artigos (como [Holmström 2002]) descrevem com detalhes o funcionamento do GNG.

3.4 O algoritmo Gás Neural Evolutivo (GNG)

O GNG é um algoritmo em que o processo de criação dos grupos (nós ou neurônios) ocorre de forma evolutiva, no sentido de que esse número de grupos aumenta à medida que o algoritmo de treinamento é executado, e não supervisionada, uma vez que os nós são formados de maneira autônoma durante o treinamento. O algoritmo recebe inicialmente um conjunto **C** de vetores de entrada (as amostras) de dimensionalidade conhecida ($\Re^k$).

Seguindo o exemplo do arquivo de imagem, o conjunto de amostras **C** seria o conjunto contendo as amostras de cada um dos pixels da imagem. Além disso, cada amostra (ou seja, cada cor) possui três dimensões, R, G e B, nesse caso k=3 ($\Re^3$).

Partindo de dois nós iniciais aleatoriamente posicionados, o algoritmo passa a acrescentar novos nós à medida que é executado, segundo regras descritas a seguir, e com isso cria um grafo no mesmo espaço vetorial das amostras de entrada, onde cada um dos nós é o representante de um grupo de amostras, possuindo uma posição nesse espaço dimensional das amostras. O algoritmo pode, então, ser utilizado em processos de quantização, bastando para isso substituir cada amostra pelo representante do grupo onde a amostra está classificada.

O GNG alcança bons resultados no sentido de que a topologia do grafo encontrado depois de inseridos todos os nós reflete fortemente a distribuição probabilística do conjunto das amostras de entrada. Para isso, o algoritmo faz uso de diversas heurísticas durante o processo de crescimento do grafo (que é geralmente chamado de processo de treinamento), visando obter a melhor distribuição possível das amostras entre os grupos.

Em resumo, dadas as amostras de entrada em $\Re^k$, cada nó $\mathbf{n}_p$ do grafo possui as seguintes características (cada característica será analisada em detalhes posteriormente):

a. um vetor de pesos de referência em $\Re^k$, que chamamos de $\mathbf{c}_p$;

b. uma variável que contém o erro acumulado do nó, que vamos chamar de n_{p_error};
c. uma lista de arestas contendo todas as ligações do nó $\mathbf{n}_p$ aos seus vizinhos. Cada uma das arestas da lista possui uma característica denominada sua idade.

O processo de criação (crescimento) do grafo se dá de forma incremental, e a cada iteração do algoritmo durante o processo uma amostra é escolhida segundo a distribuição de probabilidades do conjunto de amostras e utilizada no treinamento. Vamos chamar essa amostra escolhida no passo atual genericamente de amostra **s**.

- *Vetor de pesos*: dado um nó $\mathbf{n}_a$ qualquer do grafo, seu vetor de código (*codevector*) $\mathbf{c}_a$ pode ser interpretado como a posição desse nó no espaço vetorial das amostras. No caso de nosso exemplo, ele seria simplesmente as coordenadas RGB do nó do grafo, ou seja, a cor representante do grupo.

- *Variável de erro acumulado*: durante o processo de criação do grafo a cada iteração do algoritmo, é medido o nível de similaridade (segundo a função de similaridade utilizada) entre a amostra **s** sendo treinada e o nó $\mathbf{n}_a$ que a representa. O erro existente (ou seja, a distância existente) entre a amostra **s** e o vetor $\mathbf{c}_a$ do nó que representa essa amostra é acumulado na variável de erro do nó (n_{a_error}). É através desses erros acumulados que o algoritmo vai decidir em qual posição serão incluídos os novos nós, à medida que o grafo está sendo criado. A idéia aqui é que, depois de um certo número de iterações, o nó do grafo que possui o maior erro acumulado está sendo o representante de muitas amostras, e portanto um novo nó a ser adicionado deveria ser incluído próximo desse, dividindo o número de amostras representadas e com isso melhorando a distribuição das amostras entre todos os nós do grafo.

- *Idade das arestas do grafo***:** a idade é utilizada para que o algoritmo possa remover as arestas mais antigas, e essa é uma das características que permitem ao algoritmo criar grafos cuja forma vai sendo modificada ao longo do treinamento, obedecendo à distribuição espacial das amostras de entrada.

Cada um dos passos do algoritmo é descrito abaixo. O algoritmo trabalha com 6 parâmetros: ε_b, ε_t, a_{max}, λ, α, β. Cada um deles será apresentado à medida que for utilizado na descrição do algoritmo, dada abaixo:

Passo 0: o primeiro passo consiste em criar um grafo inicial, composto por somente dois nós situados inicialmente em posições aleatórias no espaço das amostras e com valor de erro acumulado zero. Esses dois neurônios devem ser ligados por uma aresta. A idade da aresta que liga os dois neurônios deve receber valor zero.

Passo 1: obter aleatoriamente (segundo a distribuição de probabilidades do conjunto de amostras de entrada) uma amostra de entrada qualquer **s**. A partir do grafo, encontrar o primeiro e o segundo nós mais próximos da amostra **s** segundo a função de similaridade sendo utilizada. Esses dois nós são denominados $\mathbf{n}_a$ e $\mathbf{n}_b$, respectivamente.

Passo 2: o neurônio mais próximo da amostra **s**, $\mathbf{n}_a$, é considerado o vencedor (denominado BMU – *Best Match Unit*) e o seu valor de erro acumulado deve ser atualizado:

$$n_{a\_error} = n_{a\_error} + dist(\mathbf{n}_a, \mathbf{s})$$

Passo 3: mover a posição de $\mathbf{n}_a$ no espaço (ou seja, seu vetor de código $\mathbf{c}_a$) e seus vizinhos topológicos (chamamos de vizinhos topológicos de $\mathbf{n}_a$ a todos os outros nós do grafo ligados a ele através de uma aresta; chamaremos esses nós genericamente de $\mathbf{n}_t$) em direção à amostra **s**. Essa movimentação é feita através de médias ponderadas entre as posições dos neurônios no espaço e na amostra, segundo os parâmetros ε_b e ε_t:

$$\mathbf{c}_a = \mathbf{c}_a + \varepsilon_b \,.\, (\mathbf{s} - \mathbf{c}_a)$$

$$\mathbf{c}_t = \mathbf{c}_t + \varepsilon_t \,.\, (\mathbf{s} - \mathbf{c}_t),\ \forall\ \mathbf{n}_t \in \text{Vizinhos}(\mathbf{n_a})$$

Valores típicos para esses parâmetros são $\varepsilon_b = 0.05$ e $\varepsilon_t = 1\times10^{-4}$.

Passo 4: a idade de todas as arestas que ligam $\mathbf{n}_a$ aos seus vizinhos topológicos recebe o incremento de uma unidade;

Passo 5: inserir uma aresta ligando os nós $\mathbf{n}_a$ e $\mathbf{n}_b$. Caso essa aresta já exista, sua idade deve ser reiniciada (receber valor zero);

Passo 6: remover do grafo todas as arestas cujo valor de idade for maior que o valor estabelecido pelo parâmetro de entrada a_{max}. Após essas arestas terem sido removidas, caso existam nós desconexos (que não possuam pelo menos um vizinho topológico) esses nós devem ser excluídos do grafo. Um valor exemplo para a_{max} é 88.

Passo 7: caso o número de amostras já treinadas no passo atual do algoritmo seja múltiplo do parâmetro de entrada λ e a condição de parada do treinamento não foi atingida, um novo nó (que vamos chamar de nó $\mathbf{n}_r$) será inserido de acordo com o seguinte procedimento:

- encontrar no grafo o nó $\mathbf{n}_u$ que possua o maior erro acumulado n_{u_error};
- encontrar, dentre todos os vizinhos topológicos de $\mathbf{n}_u$, o nó $\mathbf{n}_v$ que possui o maior erro acumulado n_{v_error};
- criar o novo nó $\mathbf{n}_r$ no ponto médio da aresta que liga $\mathbf{n}_u$ e $\mathbf{n}_v$, ou seja:

$$\mathbf{c}_r = (\mathbf{c}_u + \mathbf{c}_v) / 2$$

- criar duas novas arestas, uma ligando $\mathbf{n}_r$ a $\mathbf{n}_u$ e outra ligando $\mathbf{n}_r$ a $\mathbf{n}_v$.
- decrescer n_{u_error} e n_{v_error} de acordo com o parâmetro de entrada α. Feito isso, o valor inicial do erro acumulado do novo neurônio $\mathbf{n}_r$ deve ser o mesmo valor de erro do neurônio $\mathbf{n}_u$:

$$n_{u\_error} = \alpha \,.\, n_{u\_error};$$

$$n_{v\_error} = \alpha \cdot n_{v\_error};$$

$$n_{r\_error} = n_{u\_error};$$

Valores para o parâmetro λ podem ser da ordem de 600 e do parâmetro α, da ordem de ½.

Passo 8: decrescer os valores de erro de todos os nós do grafo de acordo com o parâmetro de entrada β, multiplicando os valores de erro acumulado de todos os neurônios do grafo por (1-β). Valores típicos de β são da ordem de 5×10^{-3}

Passo 9: caso a condição de parada do algoritmo não tenha sido atingida (essa condição pode ser, por exemplo, que o grafo possua um número pré-definido de nós), retornar ao passo 1.

3.4.1 Comentários adicionais sobre o funcionamento do algoritmo

Neste tópico comentaremos os principais parâmetros do algoritmo, procurando esclarecer sua função durante o processo de treinamento.

3.4.1.1 O erro local acumulado

No passo 2 a variável de erro local acumulado do nó vencedor é atualizada. Essa variável mantém o valor do somatório das distâncias entre o nó e as amostras para as quais ele é o nó vencedor (situadas à menor distância). Manter esse somatório é a forma encontrada para identificar nós que estejam cobrindo uma porção relativamente grande da distribuição das amostras. Dessa forma o erro local de um nó serve como uma medida estatística no sentido em que nós que estejam cobrindo uma porção relativamente grande da distribuição das amostras de entrada terão, estatisticamente, seu erro total acumulado maior do que os outros nós. Como o objetivo do algoritmo é minimizar o erro total do grafo em relação às amostras, saber quais são os nós que têm maior erro

acumulado é um bom indicativo de em que áreas os novos nós devem ser incluídos durante o processo de crescimento do grafo.

3.4.1.2 A movimentação dos nós

No passo 3 vemos que, uma vez encontrado o nó vencedor para cada amostra, existe uma movimentação desse nó, bem como de seus vizinhos topológicos, na direção da amostra segundo frações definidas por dois dos parâmetros fornecidos ao algoritmo, ε_b para o nó vencedor e ε_i para seus vizinhos topológicos (ambos parâmetros de valores variando entre 0 e 1). Isso é feito com o objetivo que, ao longo do treinamento, o nó tenha a todo momento sua posição no espaço (seu vetor de código) situada o mais próximo possível do centro do espaço compreendido pelo conjunto de amostras para as quais esse nó é o representante.

Os deslocamentos são lineares em relação à distância entre o vetor de pesos do nó e a amostra, sendo que o valor do parâmetro que define o deslocamento do vencedor (ε_b) é geralmente muito maior que o valor que define os deslocamentos dos seus vizinhos (ε_i).

Esses dois parâmetros variam caso a caso. Valores relativamente altos tendem a favorecer a criação de grafos cuja posição dos nós é muito instável, variando fortemente no decorrer do treinamento. Já valores excessivamente baixos irão tornam a adaptação do grafo lenta e ineficaz.

3.4.1.3 A inserção de novos nós no grafo

No algoritmo GNG os nós são inseridos sempre após a execução de um determinado número pré-definido de treinamentos (esse também é um dos parâmetros do algoritmo, λ). A cada λ treinamentos um novo nó é inserido no grafo, sendo sua posição de inserção definida pelo nó que possui o maior erro acumulado no momento. É dessa forma que se consegue manter uma boa distribuição dos nós do grafo em relação à distribuição espacial das amostras de treino.

Uma observação importante a ser feita diz respeito ao impacto que o valor escolhido para λ tem no desempenho do algoritmo. Caso esse parâmetro tenha seu valor excessivamente baixo, a distribuição inicial dos nós no grafo será mal executada, pois ao se incluir um novo nó os nós existentes não terão seu vetor de código situado em posições realmente representativas da distribuição amostral. No entanto, escolhas de valores altos para λ farão com que o tempo de execução do algoritmo seja demasiadamente aumentado, visto que serão necessários λ treinamentos antes da inclusão de cada novo nó.