Mapas auto-organizáveis de Kohonen aplicados em mineração e agrupamento de dados

Ítalo Herbert Siqueira Gabriel

Índice

Índice de figuras

Índice de tabelas

RESUMO

Esse trabalho foca o ensino/aprendizagem dos mapas auto-organizáveis de *Kohonen*, bem como, sua aplicação. A introdução diz respeito aos conceitos iniciais sobre mineração de dados como uma área da inteligência artificial. Em seguida, na fundamentação teórica, são mostradas opiniões de outras pessoas sobre mineração de dados e seus algoritmos. Logo após vem o terceiro capítulo que trata exclusivamente da mineração de dados, dando ênfase aos conceitos associados, bem como, o *KDD e* técnicas de mineração de dados como: memória associativa e regressão linear, assim como, introdução as redes neurais artificiais, finalizando o capítulo com a abordagem de três algoritmos bastante conhecidos, são eles: *K-Means*, *SOM* e *GNG*. Em seguida vem um capítulo inteiro dedicado aos mapas auto-organizáveis, em que são abordados os conceitos iniciais sobre o algoritmo como, vizinhança entre os neurônios, função de vizinhança, raio de vizinhança, taxa de aprendizado, medidas de avaliação e fases dos *SOM*, entre outros pontos importantes do algoritmo. Na sequência, vem o capítulo onde são mostrados os resultados obtidos com a aplicação dos mapas auto-organizáveis. Para tanto, foram utilizados o software e linguagem de programação suportada pelo *Octave*, além da implementação dos mapas auto-organizáveis como *framework* e software Java. Por fim, vem o capítulo com as considerações finais onde é considerado que o algoritmo *SOM* é ótima escolha para o agrupamento de dados, a pesar da sua desvantagem associada a complexidade de configuração. Inclusive, é considerada também a possibilidade de utilização dos *SOM* em conjunto com o *K-Means*.

Palavras Chave: mineração de dados, mapas auto-organizáveis, inteligência artificial, k-médias, gás neural evolutivo, redes neurais artificiais;

ABSTRACT

This work focuses the teaching/learning of the *Kohonen self organizing maps*, as well as, their application. The introduction conserns to initial consepts about datamining how a field of artificial inteligence. Then, in theoretical rationale, are shown reviews of other people about data dataminings and their algorithms. Soon after comes the third chapter that throws uniquely of datamining, enphasizing to associate concepts, well as, the *KDD and* datamining techniques hom: associative memory and linear regression, well as, artificial neural networks introduction, finishing the chapter with the approach of three algorithms very known, they are: *K-Means*, *SOM* e *GNG*. Then comes a whole chapter dedicated to self organizing maps, where are approached the initial concepts about the algorithm how, neighborhood among neurons, neighborhood function, neighborhood radial, learning rate, evaluation measures and *SOM fhases*, among others important points of the algorithm. Soon after, comes the chapter where are shown the archieved results with the application of self organizing maps. For so much, were used the software and programming language supported by *Octave*, beyond implementation of self organizing maps as *framework* and Java software. Finally, comes the chapter with the final considerations where is considered that the SOM algorithm is a optimal choice to data clustering, Although there are the downside associated the configuration complexity. Including, is also considered the possibility of using the *SOM* and *K-Means together*.

Keywords: data mining, self organizing maps, artificial inteligence, k-means, growing neural gas, redes neurais artificiais;

1 Introdução

O ser humano tem grande capacidade de raciocinar se comparado com os computadores. Isso porque, tarefas simples ao ser humano podem ser extremamente complexas de serem realizadas por uma máquina, e vice-versa. Por exemplo: o homem tem a capacidade de tomar decisões com base nos sentidos. De forma simples e intuitiva o ser humano discerne entre uma distância imprecisa entre ele e um determinado objeto. Se vai caminhando, ele pode perceber um obstáculo e parar de caminhar ou desviar dele. No entanto, para o computador ou robô, identificar um objeto a frente de sua câmera pode envolver técnicas de **visão computacional**, um ramo da **inteligência artificial (IA)** que tem relação, também, com o **processamento de imagens** para extração de informações ou conhecimento a partir de uma imagem digitalizada.

Conceitualmente, a **inteligência artificial** pode ser entendida como a capacidade de um computador tomar decisões "inteligentes" e, com base nelas, desempenhar (ou não) uma determinada função. Em contrapartida, há um ramo da **inteligência artificial** conhecido como **mineração de dados** que está associado, basicamente, ao processamento de grandes bases de dados para extração de características. Em analogia com a mineração de ouro, a mineração de dados pode extrair padrões e, a partir disto, gerar conhecimento. Enquanto que, a mineração de ouro pode extrair o mineral, separando-o das impurezas.

Há uma grande diversidade de aplicações para a **inteligência artificial**. Dentre elas: a robótica, a biometria, o reconhecimento de objetos em imagens, o reconhecimento de voz, o reconhecimento de faces, dentre muitas outras áreas. Já a **mineração de dados** tem aplicações, geralmente, no agrupamento de dados armazenados em sistemas de informação de alguma empresa. Seguindo as etapas de um processo conhecido como: **descoberta do conhecimento em bases de dados (KDD)**,

se pode conseguir visualizar grupos em grandes massas de dados, o que pode permitir, por exemplo, a classificação de pessoas, clientes ou usuários de uma empresa conforme suas características em comum e, com isso, gerar relatórios que possam auxiliar a tomada de decisões por parte de gestores.

É importante esclarecer que, pelo fato de as bases de dados de grandes empresas, geralmente, serem compostas por um volume munto grande de dados, há certa dificuldade associada a análise e classificação manual. Por isso, em se tratando, por exemplo, de um sistema de vendas, um gerente pode facilmente chegar a conclusão de que usuários desempregados ou sem renda fixa têm maiores chances de não pagar em dia por um produto ou serviço comprado. Mas à mineração de dados vai além de análises como esta, permitindo a análise, não apenas de um ou dois atributos, mas, de muitos. Por exemplo, será que há relação entre a possibilidade de inadimplência e o sexo do usuário, ou a idade, ou ainda, a altura? Aparentemente, não há relação nenhuma mas, na realidade, possivelmente, a mineração de dados pode mostrar que há.

Outra grande aplicação da mineração de dados é em sistemas de redes sociais. Com as devidas técnicas de inteligência artificial e mineração de dados se pode agrupar usuários de uma rede social com grande grau de semelhança e descobrir, por exemplo, quais produtos ou serviços um determinado usuário consome ou pode ser de seu interesse, dependendo do grupo ao qual ele pertence. Assim, alguns sistemas, não apenas redes sociais, expõem propagandas conforme o perfil do usuário.

Um motor de busca pode identificar interesses de um usuário com base no histórico de pesquisa associado ele e, com base nisto, expor-lhe propagandas pertinentes a sua área de atuação como, por exemplo, se o sistema identifica que o usuário é da área de informática, pode mostrar-lhe propagandas sobre cursos relacionados a informática ou computadores de última geração à venda.

Vale ressaltar que pode haver um processo complexo relacionado ao mecanismo de inferência de **sistemas inteligentes**. Tomando como exemplo um motor de busca que faz análise semântica dos termos informados para busca, ele precisa, não apenas relacionar os termos digitados a **palavras chave**, mas, inferir sobre o contexto ou o sentido relacionado aos termos em conjunto. A análise semântica de texto escrito e submetido a um sistema inteligente via Web abre espaço para uma nova área da computação: A **Web Semântica**.

Existem também os **sistemas especialistas**, considerados como um tipo de sistema inteligente. Tais sistemas, podem simular o trabalho de um especialista de uma determinada área. Por exemplo, um sistema computacional que simula o trabalho de um médico clínico geral, onde, o usuário pode fazer perguntas escritas em língua portuguesa e tê-las respondidas pelo sistema o que pode envolver **processamento de linguagem natural**. Outra classe de sistemas inteligentes são os **sistemas empíricos** que tomam decisões com base na incerteza. Na prática, quase todos os sistemas inteligentes são também empíricos, dado que, em bases de dados geralmente existem dados faltosos ou ruidosos e tais sistemas, geralmente, tomam decisões com base em estatísticas que, nem sempre, indicam 100% de precisão.

Os sistemas "inteligentes" podem, também, serem **baseados em conhecimento**. Isso significa que, tais sistemas, mantêm uma **base de conhecimento**, onde, o resultado, por exemplo, da aplicação de um algoritmo de mineração de dados é inserido. Inclusive, eles tomam decisões consideradas racionais a partir de consultas a base de conhecimento durante a tomada de decisões.

Muitos sistemas "inteligentes" tomam decisões com base no treinamento de um modelo de **rede de neurônios artificiais** (**RNA**) que, nada mais são que modelos matemáticos que podem ser aplicados a diversos problemas da computação. Entre elas estão o ***K-Means*** (ou K-médias), os mapas auto-organizáveis (***SOM*** – ***S****elf* ***O****rganizing* ***M****aps*), e o Gás Neural Evolutivo (***GNG*** – ***G****rowing* ***N****eural* ***G****as*). Os ***mapas auto-***

organizáveis são objeto de estudo deste trabalho.

1.1 Objetivos

Esse trabalho tem como **objetivo principal** a aplicação dos **mapas auto-organizáveis de *Kohonen*** em mineração de dados, dando ênfase a identificação automática de grupos em conjuntos de dados. Devido a falta de acessibilidade de bases de dados reais, optou-se pela geração de conjuntos de dados através de métodos matemáticos e algorítmicos. Abaixo algumas metas que espera-se que sejam atingidas para o alcance do objetivo principal:

- Esse trabalho tem como uma de suas **metas específicas** ser fonte de pesquisa a outros pesquisadores das áreas de mineração de dados;
- Outra meta específica desse trabalho é a geração de dados fictícios através de métodos matemáticos para que a teoria aqui apresentada possa ser posta em prática;
- Pretende-se, com a prática da teoria desse trabalho aplicar o processo de **descoberta do conhecimento em bases de dados (*KDD*)** para gerar conhecimento a partir do resultado do pré-processamento dos dados fictícios gerados;
- Após gerados os dados automaticamente, espera-se aplicar o algoritmo de mineração de dados **SOM** para: mineração dos dados, visualização e análise dos resultados;
- Será buscado atingir um bom, ou ótimo nível de ajuste da rede de neurônios aos dados de entrada com a aplicação do ***SOM***;

1.2 Justificativa

Este trabalho contribui por ser ótima fonte de informação sobre mineração de dados e, mais especificamente, sobre os **mapas auto-organizáveis de *Kohonen*** que são objeto de estudo e pesquisa nesse trabalho e serão detalhados em um capítulo dedicado exclusivamente ao algoritmo. Outra contribuição desse trabalho é a produção de um software e ***framework Java*** em que, como software, permite a configuração e acompanhamento em tempo real do treinamento de **mapas auto-organizáveis** e, como ***framework Java***, poderá ser integrado a algum aplicativo Java por outros programadores, além de poder ser executado para a mineração de dados de algum conjunto de dados de entrada.

2 Fundamentação teórica

Silva (2004) aborda em seu trabalho a mineração de dados como uma das etapas do processo de descoberta do conhecimento em bases de dados, o ***KDD – K****nowledge* ***D****iscovery in* ***D****atasets.* Ele cita todas as etapas do processo e explica que a mineração de dados é a etapa em ***KDD*** responsável pela seleção dos métodos a serem utilizados para localizar padrões de dados, seguida da efetiva busca por padrões de interesse, numa forma particular de representação, juntamente, com a busca pelo melhor ajuste dos parâmetros do algoritmo para a tarefa em questão.

Em Fayyad et al. (1996) o ***KDD*** é definido como um processo não trivial de identificar em dados, padrões que sejam: válidos, novos, potencialmente úteis e compreensíveis visando melhorar o entendimento de um problema ou um procedimento de tomada de decisão.

Segundo Figueiredo (2017), 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.

O foco do artigo de Oliveira et al. (2008) tem semelhança com os objetivos deste trabalho. Dado que membros do artigo escreveram sobre algoritmos de aprendizado competitivo em mineração de dados e deram ênfase aos algoritmos ***K-Means***, ***SOM*** e ***GNG***, abordando também o processo ***KDD***. O artigo também mostra gráficos de resultados do agrupamento automático de dados e, para tanto, foram aplicados os algoritmos de mineração de dados enfatizados no artigo.

Em SCIKIT-LEARN (2019), é escrito sobre uma definição do ***K-Means*** como um algoritmo de agrupamento de dados que tenta separar grupos de amostras com igual variância e, é descrito também sobre o efeito de inércia cuja definição é escrita no artigo

como: a medida da coerência entre os grupos. Em tal artigo sobre ***clustering*** também é afirmado que o ***K-Means*** visa a seleção de ***centroids*** (Referidos como as médias associadas aos grupos) que minimizem a inércia e maximizem a coerência. Também é abordado um algoritmo que é uma variação do ***K-Means***, o ***Mini Batch K-Means***, que tem o tempo de computação reduzido em comparação com o ***K-Means***. No entanto, o ***Mini Batch K-Means*** sofre uma perda na qualidade que pode ser minimizada para que as saídas do ***K-Means*** e sua variação sejam parecidas.

Foi Tuevo Kohonen que, em 1982, criou o algoritmo de rede neural artificial conhecido como Mapas auto-organizáveis O SOM.

Segundo Kohonen (2001): "*The SOM is a new, effective software tool for the visualization of high-dimensional data. It converts complex, nonlinear statistical relationsihps between high-dimensional data items into simple geometric relationships on a low-dimensional display. As it thereby compresses information while preserving the most important topological and metric relationships of the primary data items on the display, it may also be thought to produce some kind of abstractions*"

Tradução: "*O SOM é uma nova e efetiva ferramenta de software para visualização de dados de alta dimensionalidade. Ele converte relacionamentos estatísticos complexos e não lineares entre itens de dados de alta dimensionalidade em simples relacionamentos geométricos para uma visualização de baixa dimensionalidade. Desse modo, ele comprime informações enquanto preserva características muito importantes como a topologia e as relações métricas de itens de dados primários para visualização. Ele também pode ser utilizado para produzir alguns tipos de abstrações*."

Segundo Techopedia Inc (2019) um Self Organizing Map (SOM) é uma rede neural artificial que usa aprendizado não supervisionado para construir um mapa bidimensional do espaço de um problema. A diferença chave entre Self Organizing Maps e outras abordagens aplicadas à resolução de problemas é que Self Organizing Maps usa

aprendizado competitivo em vês de aprendizado por correção de erros assim como Backpropagation com Gradiente Descendente.

Em relação a distância e similaridade dos dados, Ahn e Syn (2005) escreveram: Existem numerosos caminhos para se encontrar as distâncias, no entanto, a mais comumente utilizada é a Distância Euclidiana e/ou Distância do cosseno.

Veja na figura abaixo o gráfico de resultados da aplicação do SOM por Ahn e Syn:

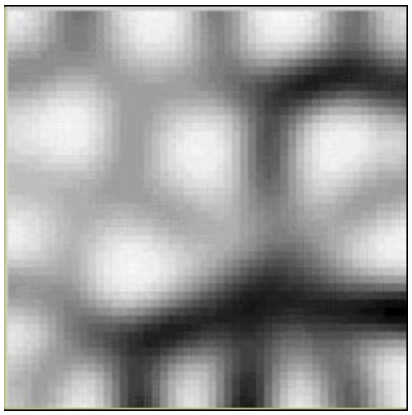


Figura 1: Resultados obtidos pela aplicação do SOM

Ahn e Syn (2005) comentam, ainda o resultado ilustrado na **Figura 1**: Se a distância média é alta, então os pesos envolvidos são muito diferentes e uma cor escura é associada a localização do peso. Se a média de distâncias é baixa, então uma cor clara é associada a localização do peso. O mapeamento resultante mostra que preto não é similar as partes brancas porque são linhas de preto que representam a não similaridade entre as partes brancas. Olhando o mapa, ele claramente mostra que as duas partes brancas separadas por preto entre elas não são muito similares. O gráfico pode estar dizendo que as partes brancas representam diferentes grupos e as linhas pretas representam a divisões de grupos.

Germano (1999) conclui sobre vantagens e desvantagens associadas ao ***SOM***.

Como vantagens ele destaca: Provavelmente, a melhor coisa do SOM é que ele é fácil de compreender. *Ele é muito simples. Outra grande coisa é que ele trabalha muito bem.* **Como desvantagens ele escreve***: Um grande problema do SOM é obter os dados certos. Outra coisa é que cada SOM é diferente e busca diferentes similaridades entre os vetores de amostras*.

Figueiredo (2007), em sua dissertação de mestrado, aborda o algoritmo Gás Neural Evolutivo (***GNG*** – ***G****rowing* ***N****eural* ***G****as*) e diz que o ***GNG*** é um processo de criação de grupos, também referidos como conjuntos de nós ou neurônios, que ocorre de forma evolutiva, no sentido de que o número de nós aumenta a medida que o algoritmo de treinamento é executado. Ele descreve ainda o ***GNG*** como sendo um algoritmo de rede neural artificial não supervisionado porque os grupos são formados automaticamente durante o treinamento da rede.

De acordo com Holmström (2002), o algoritmo ***GNG***, criado e publicado por Bernd Fritzke, é um algoritmo de agrupamento incremental e não supervisionado. Dada alguma entrada distribuída em $\mathbf{R}^n$, o ***GNG*** incrementalmente cria um grafo, ou rede de nós, onde, cada nó do grafo tem uma posição em $\mathbf{R}^n$. Ele descreve ainda a técnica conhecida como **quantização de vetor** ou, do inglês, **Vetor Quantization** (**VQ**) como o processo de quantizar vetores de entrada multidimensionais em um limitado conjungo de vetores de saída referidos na dissertação como ***code-vectors*** ou vetores de código. Ele enfatiza ainda que, ao conjunto de possíveis vetores de código, dar-se o nome de ***codebook*** ou livro de código. O livro de código é geralmente gerado pela classificação de um dado conjunto de treinamento (**training set**). O livro de código é usado para quantizar vetores de entrada.

3 Mineração de dados

Hoje em dia, as empresas acumulam grandes quantidades de dados em seus sistemas de informação. Com isso, a análise de tais dados torna-se cada vez mais difícil conforme a base de dados cresce. Claro, os sistemas de informação costumam oferecer relatórios e acesso rápido aos dados armazenados. Mas, nem sempre isso basta para a tomada de decisões nas empresas. Isso porque, tais organizações, buscam vantagem competitiva, almejam estar a frente da concorrência. E, para tanto, precisam obter informações preciosas sobre os dados armazenados.

A uma empresa de vendas, pode ser interessante gerar informações a partir de dados de clientes e vendas efetuadas. Mas, como obter tais informações? Um caminho para a obtenção de informações que os relatórios automáticos dos sistemas de informações de uma empresa não fornecem pode ser a mineração de dados. A partir dela, se pode gerar informação e conhecimento sobre bases de dados com grande volume.

Assim como a mineração de ouro, a mineração de dados busca extrair informações valiosas a partir de dados brutos. Por exemplo: Como agrupar usuários de uma rede social pelos seus perfis de usuário? Pelas suas características em comum. Como agrupar clientes de uma empresa de acordo com as chances probabilísticas de serem bons ou maus pagadores? Esse agrupamento, muitas vezes, vai além do que se pode obter através de relatórios gerados pelo sistema de informação de uma empresa.

Para o agrupamento de entidades armazenadas em bancos de dados, se pode aplicar um algoritmo de mineração de dados. Vale ressaltar que a mineração de dados é um ramo da inteligência artificial que tem aplicações na análise e interpretação de dados de entidades armazenadas em bancos de dados e, tais entidades podem ser usuários de uma rede social, clientes de uma loja de roupas, calçados, etc.

É comum em mineração de dados a utilização de algoritmos de redes neurais artificiais para extrair características e, com base nelas, agrupar ou classificar os dados. Inclusive, há uma relação entre a mineração de dados e a estatística, dado que, a estatística pode ser aplicada no pré-processamento e preparação dos dados a serem submetidos a rede de neurônios artificiais para extração de características. As vezes é necessário gerar dados estatísticos para, por exemplo, organizar clientes pela faixa etária ou pelo tempo que o cliente está cadastrado no sistema. Pode ser útil também gerar médias ou outras medidas estatísticas sobre essas classes de clientes para submeter tais dados ao algoritmo de agrupamento.

3.1 O processo KDD

Devido à complexidade envolvida com a mineração de dados, foi criado um processo conhecido como **KDD** – **K**nowledge **D**iscovery in **D**atabases (Descoberta de conhecimento em bases de dados). A **Figura 1** abaixo ilustra esse processo:

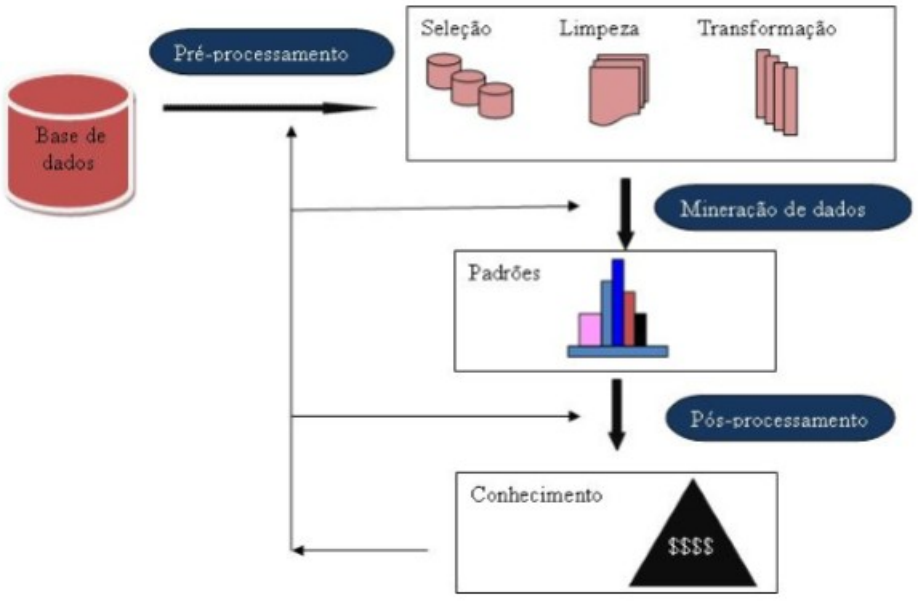


Figura 2: Processo de descoberta do conhecimento em bases de dados

Imagem retirada de Oliveira et al. (2007)

Conforme ilustrado na figura acima, O **KDD** é composto de várias etapas, são elas: pré-processamento, mineração de dados e pós-processamento. O pré-processamento é composto de três subetapas: seleção, limpeza e transformação. E o pós-processamento é composto pela visualização de resultados e geração de conhecimento obtido com a etapa de mineração de dados. Veja com mais detalhes, a seguir, uma breve descrição de cada etapa do processo **KDD**:

- **SELEÇÃO**: Nessa etapa devem ser selecionados os atributos de dados relevantes para resolução do problema que se deseje solucionar. **Exemplo**: para a classificação de clientes de uma loja, pode ser relevante considerar dados sobre o débito do cliente, renda, idade, etc.

- **LIMPEZA**: Após a seleção, é possível que dados de atributos selecionados estejam faltando ou ruidosos. Então, é necessário repará-los ou preenchê-los se possível. **Exemplo**: Por um erro de digitação, o operador de um sistema pode inserir no formulário de cadastro de clientes uma idade de 200 anos, quando, na verdade, queria digitar 20.

- **TRANSFORMAÇÃO:** Com a seleção e limpeza tendo sido realizadas, se pode passar para a etapa de transformação, onde, os dados são normalizados, isto é, dados não numéricos podem assumir valores numéricos em uma determinada faixa. **Exemplo**: O atributo nível de experiência profissional de um programador pode assumir valores como: nenhuma, júnior, pleno ou sênior e, numa faixa de 0 a 1, se pode assumir que nenhuma experiência tenha valor 0, nível junior pode ter valor 0,33, nível pleno pode ser associado ao valor 0,66 e, finalmente, nível sênior pode ter valor numérico 1. Perceba que foram utilizados cálculos de proporção para normalizar o atributo experiência.

- **MINERAÇÃO DE DADOS:** É nesta etapa que os dados normalizados são submetidos a um algoritmo de mineração de dados para extração de características. Grupos podem ser formados com base em similaridades entre os dados. **Exemplo:** pode-se utilizar algum algoritmo de rede neural artificial como os mapas auto-organizáveis (detalhado mais a frente) para identificar grupos.

- **VISUALIZAÇÃO DOS RESULTADOS:** A mineração de dados gera padrões que podem ser visualizados e interpretados. Pode-se gerar uma representação gráfica a partir dos padrões extraídos na etapa de mineração de dados. **Exemplo:** gráfico que ilustra os diferentes grupos detectados pelo algoritmo de mineração.

De acordo com a **Figura 4**, o processo **KDD** gera conhecimento a partir dos padrões e grupos detectados na etapa de mineração de dados. É importante considerar também que, se não forem obtidos os resultados desejados após a mineração de dados, pode-se voltar a etapas anteriores e, por isso, o ***KDD*** é considerado um **processo iterativo**. Mais detalhes sobre processo de descoberta do conhecimento podem ser encontrados em Oliveira et al. (2007).

Vale salientar que algumas bases de dados requerem a aplicação de um algoritmo de rede neural artificial. Mas, nem sempre o problema é tão complexo para que seja necessária a aplicação de algum algoritmo relativamente complexo. Existem os problemas que podem ser resolvidos com métodos estatísticos que, geralmente, são mais simples que a aplicação de um algoritmo de **RNA**. Como, por exemplo, regressão linear e regras de associação.

3.2 Regras de associação

Em bases de dados onde se pode determinar padrões de forma visual (Geralmente tais padrões são obtidos a partir da visualização dos resultados de um algoritmo de mineração de dados), é possível gerar associações entre os dados. Por

exemplo: imagine um site com certa diversidade de conteúdos. Esse sistema pode expor ao usuário uma página com um artigo como conteúdo e, expor também, sugestões de artigos relacionados. Ou ainda, o site pode ter um mecanismo de busca que associa os termos de pesquisa a palavras chave para expor conteúdo relevante ao usuário.

Na prática, após a visualização de informações extraídas de um banco de dados, se pode inserir registros que associam palavras chave as páginas de um site e, quando o usuário informar algum termo para busca, consultas são feitas ao banco de dados para mostrar links das páginas com conteúdo relevante para o usuário. Associações entre as páginas podem ser obtidas pela consulta por páginas indexadas com palavras chave em comum.

Outro ponto importante em se tratando de regras de associação é o processamento de conteúdo. Isto é, algum algoritmo precisa ser aplicado para extrair palavras chave do texto que compõe o conteúdo das páginas. Por exemplo, motores de busca como: Bing, Yahoo e Google, processam automaticamente o conteúdo das páginas, dado o gigantesco número de páginas de sites que esses buscadores têm indexada em seus bancos de dados.

3.3 Regressão linear

A regressão linear é um exemplo de método estatístico que pode ser utilizado em mineração de dados aplicada a problemas onde se pode determinar variáveis consideradas independentes e associar a elas uma variável que delas dependente.

A **regressão linear simples** caracterizada por casos onde os dados são bidimensionais, pode ter a variável **X** correspondendo a variável independente e Y, a variável dependente de **X**. Se os pontos estão distribuídos próximos de pontos que compõem uma reta, é possível determinar a equação da reta que melhor se ajusta aos dados. Veja a figura abaixo:

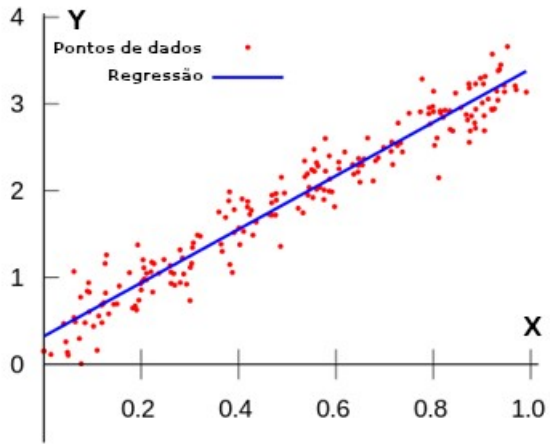


Figura 3: Gráfico que ilustra um exemplo de regressão linear simples

Imagem adaptada de WIKIPEDIA (2019)

No gráfico acima pode-se perceber os pontos vermelhos que representam dados, onde, pela distribuição dos dados, é possível identificar uma reta que se ajusta razoavelmente a eles. Isso significa que, se determinados o coeficiente angular da reta e a constante de interseção com o eixo **Y**, pode-se, com algum grau de precisão, predizer, por exemplo, o valor de **Y** para **X** igual a 1,2 ou para **X** igual a 3.

Existem métodos para o cálculo do coeficiente angular e constante de interseção com o eixo **Y**. Um deles é o método dos mínimos quadrados WIKIPEDIA (2018) que objetiva encontrar o melhor ajuste a um conjunto de dados tentando minimizar a soma do quadrado das diferenças entre os dados. Existe também a **regressão linear múltipla** que permite a estimativa de uma variável dependente de várias variáveis independentes e o método dos mínimos quadrados também se aplica a ela.

3.4 Regressão linear com o WEKA

Há um software livre conhecido como **WEKA – W**aikato **E**nvironment for **K**nowledge **A**nalisys, disponível sob licença **GPL** que pode ser utilizado para mineração

de dados. Mais detalhes em Waikato (2019). Ele é um software gratuito e produzido em ***Java*** que pode ser utilizado para motar árvores de decisão, para aplicação de regressão linear para previsão de valores com base em instâncias (ou registros), e classificação ou agrupamento com a aplicação de algoritmos de treinamento de redes neurais artificiais. Para obtenção de resultados com o **WEKA**, é necessária a criação de uma base de dados no formato ***ARFF***. Segue abaixo um exemplo que mostra dados organizados em classes no formato ***ARFF***:

```
@RELATION imóveis

@ATTRIBUTE areaterreno  REAL
@ATTRIBUTE numquartos NUMERIC
@ATTRIBUTE numbanheiros NUMERIC
@ATTRIBUTE valordolocal NUMERIC
@ATTRIBUTE valor REAL

@DATA
100,3,4,0.5,200000,Classe1
300,5,6,0.9,500000,Classe1
150,4,1,0.6,150000,Classe1
600,10,6,0.6,600000,Classe2
1000,0,0,0.8,500000,Classe2
80,3,1,0.4,100000,Classe2
```

No exemplo acima foi criada uma relação com nome: “imóveis” precedido da anotação ***@RELATION***. Logo abaixo, aparecem os atributos anotados com ***@ATTRIBUTE***, em que, após a anotação, aparecem os nomes dos atributos seguidos do tipo. No exemplo acima, os atributos são dos tipos ***REAL*** e ***NUMERIC***. Após a definição dos atributos, são definidos os dados da base de dados em si após a anotação ***@DATA***. Perceba que cada linha representa um registro (ou instância) dos dados e devem ter os mesmos números de atributos separados por vírgula que o número de atributos definidos. Em caso de cada registro da base de dados ter associado uma classe, após os valores dos atributos, deve ser definido o nome da classe a qual tal instância pertence. No exemplo acima, foram utilizadas duas classes de nomes: Classe1 e Classe2.

Como visto até aqui, é possível a predição de dados com base num histórico e,

como exemplo de mineração de dados através de **regressão linear múltipla**, suponha uma base de dados com valores de casas a venda. Cada casa tem atributos associados (ou variáveis independentes) como: área do terreno, número de quartos, número de banheiros, valor do local ou bairro numa escala de 0 a 1, etc. A variável independente nesse exemplo é o valor do imóvel. A base de dados pode ter registros dos dados das variáveis independentes e, associado a tais registros, a correspondente variável dependente (o valor do imóvel). Veja a tabela abaixo:

Tabela 1: Dados de imoveis a venda

Área do terreno	Número de quartos	Número de banheiros	Valor do local	Valor do imóvel
$100m^2$	3	4	0,5	200000
$300m^2$	5	6	0,9	500000
$150m^2$	4	1	0,6	150000
$600m^2$	10	6	0,6	600000
$1000m^2$	0	0	0,8	500000
$80m^2$	3	1	0,4	100000
$200m^2$	4	1	0,6	???

A tabela acima ilustra dados fictícios de imóveis a venda e o problema é, com base nos 6 primeiros registros determinar o valor do imóvel correspondente ao 7º imóvel da tabela. Perceba que a área do terreno está especificada em metros quadrados, as variáveis “número de quartos” e “número de banheiros” são valores numéricos, o valor do local está normalizado numa escala de 0 a 1.

O software **WEKA** foi utilizado para aplicar uma técnica de regressão que permite a predição do dado em falta com base nos registros armazenados na base de dados de imoveis representada pela **Tabela 1**.

Após carregada no **WEKA** a base de dados ilustrada na **Tabela 1**, se pode perceber que o aplicativo gerou dados estatísticos e descritivos das variáveis da base de dados. O programa gerou valores mínimo e máximo, medias e desvios padrões para cada variável. Outros dados relevantes gerados pelo ***WEKA*** são: o número de instâncias

que corresponde ao número de registros da base de dados, o número de valores distintos e o número de valores únicos.

Visitando a aba ***Classify*** do **WEKA** se pode selecionar o método a ser empregado para que seja determinado o dado que se deseje estimar. No exemplo da **Tabela 1**, o método escolhido foi o ***LinearRegression*** ou **Regressão Linear**. Após a classificação, o **WEKA** gerou um modelo de regressão linear, onde o valor do imóvel teve associada a seguinte fórmula (ou modelo):

valor = (455,2248 * areaterreno) +
(47428,9627 * numbanheiros) +
(155009,5211 * valordolocal) +
-67984,8053

Perceba que o atributo "número de quartos" foi considerado irrelevante pelo modelo de regressão linear gerado pelo **WEKA**. Após gerado o modelo, para calcular o valor que está faltando na **Tabela 1**, basta multiplicar o valor das variáveis conhecidas pelos pesos encontrados pelo **WEKA** e, então, o cálculo fica como mostrado abaixo:

valor = (455,2248 * 200) + (47428,9627 * 1) + (155009,5211 * 0,6) – 67984,8053
= 91044,96 + 47428,9627 + 93005,71266 - 67984,8053
= 163494,83006

Logo, o valor estimado para o imóvel do último registro da **Tabela 1** é, aproximadamente, 163495 (ou R$ 136.492,00) reais. Detalhes sobre a geração de um modelo de regressão linear aplicado a predição do valor de uma casa a venda com base em um conjunto de atributos de outras casas pode ser encontrado em Abernethy (2010).

Vale esclarecer que, para o exemplo da **Tabela 1**, foram escolhidas uma pequena quantidade de variáveis. Em sistemas reais, geralmente, o número de variáveis é muito maior.

3.5 Redes neurais artificiais

Assim como o cérebro humano tem células neuronais que, juntas, formam uma rede de neurônios, existem os modelos de Redes Neurais Artificiais (RNAs). Tais modelos nada mais são que modelos matemáticos empregados para capacitar uma rede de neurônios a, geralmente, classificar ou agrupar dados de entrada. Uma área onde há aplicações de RNAs é a mineração de dados, onde, se deseja agrupar (com ou sem supervisão humana) grande volume de dados através da extração de características comuns a entidades que os dados representam.

Uma das grandes aplicações das RNAs é em problemas que exigem reconhecimento de padrões. Elas podem ser aplicadas em reconhecimento de voz, de faces, de dados biométricos e de caracteres em imagens, dentre várias outras aplicações que se pode imaginar. São também consideradas como o ramo da inteligência artificial conhecido como **inteligência artificial conexionista**. O termo "***conexionista***" é utilizado porque as RNAs têm base no ajuste de pesos que representam as conexões sinápticas entre os neurônios durante o treinamento da rede. Um dado curioso é que, enquanto as redes neurais do cérebro humano chegam a ter bilhões de neurônios, as redes neurais artificiais têm entre um (no caso do Perceptron simples) dezenas e, até, milhares de neurônios artificiais apenas.

O neurônio biológico tem basicamente os seguintes componentes: dendritos, núcleo (ou soma), axônio e sinapses. Onde, é nas sinapses (também chamadas de ligações sinápticas) que está o nosso aprendizado (A nossa cognição). A cada instante de nossas vidas, nos temos os nossos sentidos funcionando ativamente e, o nosso cérebro sofre modificações a cada sinal captado pelo nosso organismo, o que gera um aprendizado. Veja a imagem abaixo:

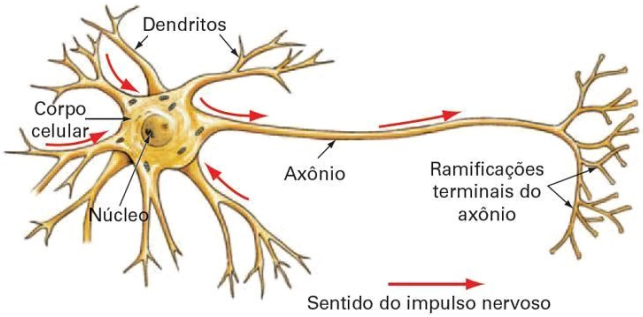


Figura 4: Neurônio biológico simplificado

Imagem retirada de DEEPLEARNINGBOOK (2019)

A imagem acima é uma ilustração de um neurônio (Unidade celular do nosso sistema nervoso). Superficialmente, o neurônio tem o axônio ligado aos dendritos que recebem sinais propagados por outros neurônios conectados. As sinapses ligam os axônios aos dendritos de neurônios vizinhos. Perceba que há uma ligação entre a ramificação terminal do axônio e os dendritos de outros neurônios. Cada neurônio recebe sinais de entrada pelos dendritos ligados por sinapses ao axônio de outros neurônios e, cada neurônio, processa sinais de entrada (impulsos nervosos) recebidos através de seus dendritos e propaga, através de seu axônio, o sinal resultante da alteração feita pelo núcleo. Para mais informação sobre o neurônio biológico simplificado e o neurônio artificial (ou matemático) acesse: DEEPLEARNINGBOOK (2019).

3.5.1 O treinamento da rede

As redes neurais artificiais, quando aplicadas a um problema, geralmente, são treinadas com a submissão de dados de entrada para o ajuste dos pesos. Os pesos podem ser valores de arestas e, caso os neurônios estejam representados em um espaço multidimensional, os pesos sinápticos podem ser as distâncias euclidianas entre os neurônios.

O treinamento das **RNAs** pode ser supervisionado ou não supervisionado. Esses modos de treinamento se distinguem pela necessidade, ou não, de intervenção humana

durante o treinamento. Por exemplo, o Perceptron simples e o multi-camadas são **RNAs** de aprendizado supervisionado, onde, um erro é calculado com base no valor do sinal de saída esperado para os dados de entrada submetidos e o sinal obtido. A intervenção humana está na necessidade que um algoritmo de **RNA** pode ter da especificação de saídas esperadas. Geralmente, as redes de aprendizado supervisionado têm seus pesos sinápticos ajustados após o calculo do erro e o treinamento consiste em ajustar os pesos de modo a aproximar o sinal de saída do sinal desejado, minimizando o erro.

Já o treinamento das **RNAs** de aprendizado não supervisionado não depende do calculo do erro com base nos dados de saída esperados. Nessas RNAs, geralmente, os neurônios se organizam automaticamente conforme os dados de entrada são submetidos a rede. São exemplos de redes neurais artificiais de aprendizado não supervisionado o **SOM** e o **KMeans**. Esses algoritmos geram grupos como resultado. No caso do **SOM**, durante o treinamento os neurônios tendem a aproximar-se dos dados de entrada e de neurônios com características similares, formando assim, os diferentes grupos. Já no caso do **k-means**, o número de grupos é informado previamente e o algoritmo agrupa os dados de entrada conforme o número de classes informadas ao algoritmo.

Alguns algoritmos de redes neurais artificiais são também classificados como de aprendizado competitivo, onde, os neurônios competem para representar dados de entrada submetidos a rede.

3.5.2 O Perceptron

O perceptron simples é o modelo de rede neural artificial mais básico e tem apenas uma unidade de processamento (O neurônio) que recebe sinais de entradas e realiza uma soma ponderada, onde, os dados (sinais) de entrada são somados e multiplicados por diferentes pesos. Tais pesos são ajustados a cada iteração de treinamento do perceptron dado o erro obtido na iteração corrente. Veja abaixo uma imagem que ilustra o modelo

perceptron:

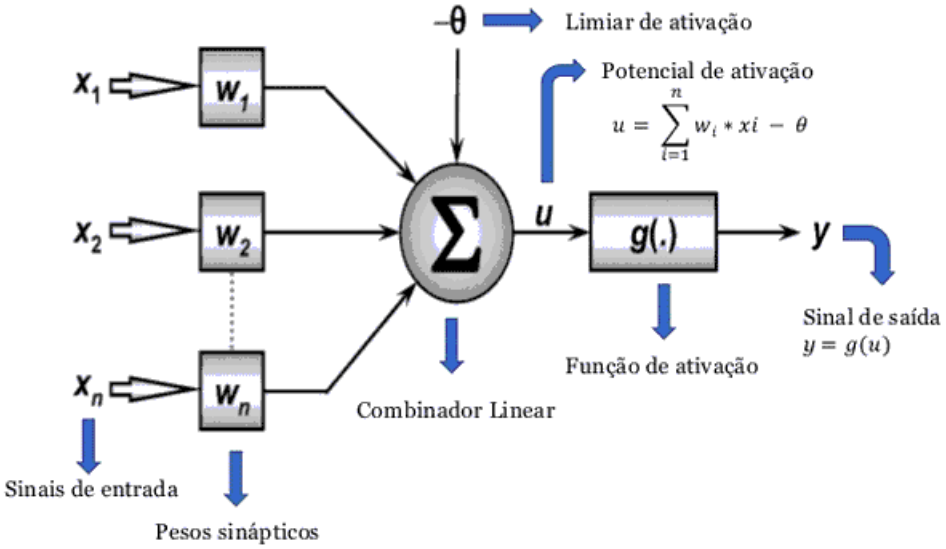


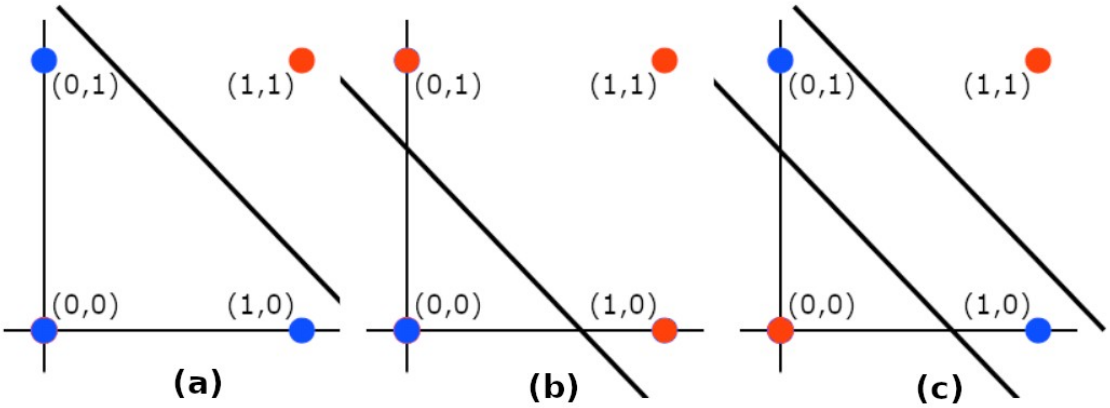
Figura 5: Modelo de representação do Perceptron

Imagem retirada de EMBARCADOS (2019)

Para o calculo do sinal de saída, há também um limiar de ativação e uma função de ativação. Ou seja, apos o perceptron realizar o calculo da soma ponderada dos dados de entrada multiplicados pelos pesos, o resultado é comparado a um limiar de ativação que é um fator numérico. Se o valor da soma obtida ultrapassar o valor do limiar, o valor da soma é submetido a uma função de ativação que normaliza a soma. Geralmente, a função de ativação toma como entrada a soma ponderada que, se passar no teste do limiar, passa a representar o potencial de ativação e gera como saída um valor binário, (0 ou 1). Assim como os pesos associados a cada sinal de entrada, o limiar de ativação também pode ser ajustado para se obter o erro mínimo, dada a sua influência no valor do sinal de saída. Em EMBARCADOS (2019), podem ser encontrados mais detalhes sobre o perceptron simples.

Além do perceptron simples, há o perceptron multicamadas que é mais complexo e resolve uma limitação do perceptron simples, dado que este só se aplica

com êxito em problemas com classes linearmente separáveis como, por exemplo, o calculo das operações ***booleanas*** **AND** e **OR** mas não se aplica a operação **XOR** porque esta não é linearmente separável conforme se pode ver na figura abaixo:



Perceba que os dados de entrada (0,0), (0,1), (1,0), (1,1) podem ter suas classes separadas linearmente para as operações ***booleanas*** **AND** e **OR**. Ou seja, no classificador **(a) – AND**, o resultado é 1 apenas para os dados de entrada (1,1) e uma linha pode separar as duas classes associadas aos valores de saída 1 e 0. Os mesmos dados submetidos ao classificador **(b) – OR** têm o valor zero como sinal de saída apenas se o valor de entrada for (0,0), o que permite separar as classes linearmente. Já para a implementação de um classificador **(c) – XOR**, o perceptron simples não se aplica, dado que são necessárias duas linhas (ou uma curva) para separar as duas classes. O algoritmo perceptron multicamadas pode ser utilizado para resolver o problema do **XOR**, mas, foge do escopo deste trabalho detalhá-lo.

O perceptron é um modelo de redes neurais artificiais de aprendizado supervisionado. Para o problema das operações ***booleanas***, por exemplo, para duas entradas diferentes, existem 4 entradas possíveis: (0,0), (0,1), (1,0), (1,1). E, para cada diferente conjunto de entradas, existe uma saída esperada. O erro é calculado com base na diferença entre o sinal esperado e o sinal obtido, caracterizando o modelo de **RNA** como de aprendizado supervisionado.

3.6 K-Médias

O ***K-Means*** ou **K-Médias** é um algoritmo de rede neural artificial simples e de aprendizado não supervisionado. Ele tem uma limitação em relação ao número de grupos se comparado com alguns outros algoritmos. O **K-Médias** depende que o número de grupos seja informado previamente, ao contrário de outros algoritmos que detectam os grupos automaticamente e sem necessidade prévia de definição do número de grupos.

O algoritmo é baseado no conceito de ***centroides*** (Ou neurônios) que são associados as classes. Os ***centroides*** têm como valor as médias dos dados pertencentes ao grupo ao qual eles representam.

Em SCIKIT-LEARN (2019) é escrito sobre o conceito de inércia como a medida da coerência entre os grupos. Isto é, busca-se, com a seleção dos centroides que representam os grupos, minimizar a inércia, maximizando a coerência entre os grupos.

Assim como outras redes neurais artificiais, o **K-Médias** depende de que os dados estejam representados numericamente e normalizados porque é necessário um cálculo da distância entre um determinado dado a ser agrupado e os centroides, o que exige, também, que os dados de entrada e os centroides tenham a mesma dimensão ou espaço de entrada. Geralmente o cálculo da distância entre dado de entrada e centroide é feito por meio da distância euclidiana, mostrada logo abaixo:

$$\|P-Q\| = \sqrt{(P1-Q1)^2 + (P2-Q2)^2 + \ldots + (Pn-Qn)^2} = \sqrt{\sum_{i=1}^{n} (Pi-Qi)^2}$$

Na fórmula acima, é calculada, por exemplo, a distância euclidiana entre um dado P e um centroide Q. Abaixo, uma representação do algoritmo como uma sequência de passos:

1. Definir o número de grupos **K**;

2. Inicializar os **K** centroides com valores aleatórios ou selecionados de modo a diminuir a inércia.

3. Gerar uma matriz de distâncias entre os **N** dados e os **K** centroides. Com isso, a matriz de distâncias terá a dimensão **NxK**;

4. Associar cada dado de entrada ao centroide mais próximo dele tomando por base a matriz de distâncias gerada no passo 3;

5. Calcular os novos centroides com base na média aritmética de todos os dados de entrada representados por eles;

6. Repetir a partir do passo 2 até a convergência ser atingida, isto é, até uma iteração em que nenhum dado de entrada mude de centroide;

Existe também uma variação do **K-Means**: o ***Mini Batch K-Means***, que tem o tempo de computação reduzido em comparação com o ***K-Means***. No entanto, o ***Mini Batch K-Means*** sofre perda na qualidade que pode ser minimizada. Mais detalhes sobre o Mini ***Batch K-Means*** podem ser encontrados em SCIKIT-LEARN (2019).

Imagine uma base de dados de 500000 registros, no ***K-Means*** original, todos esses registros são associados aos centroides. No ***Mini Batch K-Means***, os dados são organizados em lotes, por exemplo, 100000 lotes de 5 registros cada. O dado representante do lote pode ser escolhido aleatoriamente. Então, em vez de relacionar os centroides a cada dado unitário, os centroides são relacionados aos lotes, o que diminui muito o tempo de computação do algoritmo, nesse caso, em 5 vezes (500000/100000). Mais detalhes em Agrawal (2013).

3.7 Os mapas auto-organizáveis (SOM)

Os **mapas auto-organizáveis de Kohonen** ou **SOM** são um algoritmo de rede neural artificial de aprendizado não supervisionado e que, diferentemente do K-Médias, não depende de prévia informação sobre o número de grupos. Os ***SOM*** se organizam automaticamente conforme a rede de neurônios vai sendo treinada e, é também considerado um algoritmo de aprendizado competitivo, dado que os neurônios competem para representar um dado de entrada submetido a rede.

No ***SOM***, os neurônios, inicialmente, podem ser organizados em grade. A relação de vizinhança entre eles pode ser retangular ou hexagonal. Outra característica importante dele é a função de vizinhança (ou função de ativação) que, geralmente, é representada pela função gaussiana de distribuição normal utilizada na proporção de ajuste dos neurônios. Se a dimensão dos dados for multidimensional, a visualização dos grupos após o treinamento pode ser feita através da geração de uma matriz conhecida como **matriz-u** que é uma **Matriz** de Distâncias **U**nificada. Este trabalho dedicou um capítulo inteiro sobre o ***SOM***. Para mais detalhes, leia o capítulo 5.

3.8 Gás Neural Evolutivo (GNG)

O **Neural Gás** (ou Gás Neural) foi inspirado nos Mapas auto-organizáveis de Kohonen e foi criado em 1991 por Thomas Martinetz e Klaus Schulten. Posteriormente, foi criada uma variante do algoritmo que ficou conhecida como **Gás Neural evolutivo** (**GNG** – ***G****rowing* ***N****eural* ***G****as*).

O ***GNG*** é um algoritmo de aprendizado não supervisionado e é baseado em um grafo que conecta nós, formando relações de vizinhança entre eles. Diferentemente do ***SOM***, o ***GNG*** não tem uma rede de neurônios artificiais definida previamente, mas sim, de início, apenas 2 nós (ou neurônios), em que o número de nós cresce conforme a

evolução da rede (Ou gás), isto é, novos nós são inseridos, o que representa a evolução ou crescimento da rede representada pelo grafo.

Iniciando com 2 nós apenas, o algoritmo constrói o grafo em que os nós considerados vizinhos são ligados por arestas. O **GNG** é um algoritmo de aprendizado competitivo, onde, uma variante do **Aprendizado competitivo de *Hebbian*** (***CHL*** – ***C**ompetitive **H**ebbian **L**earning*) é aplicada da seguinte forma: Para cada sinal de entrada submetido a rede, uma aresta é inserida entre os dois nós mais próximos do sinal de entrada. A distância euclidiana é usada para medir o quanto cada nó está próximo de um determinado sinal de entrada.

O grafo gerado através do **CHL** é chamado de "***induced Delaunay triangulation***" Ou "triangulação induzida de Delaunay" e é um subgrafo da Delaunay Triangulação do conjunto de nós. Holmström (2002).

O ***GNG*** assume que cada nó do grafo tem os seguintes atributos: **vetor de referência** que representa a posição do nó no espaço de entrada, um **erro acumulado local** que é uma medida estatística usada para inserção de novos nós no grafo e um **conjunto de arestas** que o ligam com nós vizinhos. Cada aresta tem uma variável de tempo associada que é utilizada quando o algoritmo deve verificar por arestas antigas e, removê-las, se seu tempo de existência associado for maior que o tempo pré estabelecido para remoção de arestas.

3.8.1 O algoritmo

Assim como o ***SOM***, o ***GNG*** deve ser aplicado na etapa de mineração de dados do processo ***KDD***. Abaixo, uma representação dos passos que definem o algoritmo:

1. Criar dois nós iniciais em posições aleatórias no espaço vetorial dos dados de entrada e inseri-los no grafo, conectando-os por uma aresta;
2. Obter aleatoriamente uma amostra de entrada **s**:

- Encontrar o neurônio mais próximo da amostra, o ***BMU – Best Match Unit*** e o segundo mais próximo, $\mathbf{n_a}$ e $\mathbf{n_b}$. (**Obs:** a comparação para determinar os **BMUs** é feita entre a amostra e o vetor de peso – Ou vetor de código – do nó);
- Incrementar as idades de todas as arestas ligadas ao nó $\mathbf{n_a}$ **–** (***BMU)***;
- Acrescentar ao erro local acumulado do ***BMU*** a distância entre ele e a amostra escolhida **s**:

$$e_0 = e_0 + \left\| n_a - s \right\|$$

- Mover o **BMU** e seus vizinhos em direção ao dado de entrada **s**, isto é, atualizar os vetores de código dos nós $\mathbf{n_a}$ e $\mathbf{n_b}$, isto é, $\mathbf{c_a}$ e $\mathbf{c_b}$;
  - Atualizar o ***BMU*** da seguinte forma:

$$c_a = c_a + \varepsilon_a \cdot (s - c_a)$$

  - Atualizar os vetores de código $\mathbf{c_t}$ dos vizinhos do ***BMU*** da seguinte forma:

$$c_t = c_t + \varepsilon_t \cdot (s - c_t), \forall\, n_t \in Vizinhos(n_a)$$

Segundo Moral (2007) os valores para $\boldsymbol{\varepsilon_a}$ e $\boldsymbol{\varepsilon_t}$ são, respectivamente, 0,05 e $1x10^{-4}$

- Incrementar a idade das arestas que ligam o no $\mathbf{n_a}$ a seus vizinhos;
- Se não já existir, inserir uma aresta ligando $\mathbf{n_a}$ e $\mathbf{n_b}$. Existindo ou não a aresta, setar ou alterar a idade dela para 0;
- Remover todas as arestas com idade superior a uma constante **T**. De acordo com Moral (2007) um valor que pode ser atribuído a constante **T** é 88. Caso, com a remoção de arestas, exista algum nó isolado, sem nenhuma aresta ligando ele a outro nó, remover ele;

3. Caso o número de amostras já treinadas seja múltiplo de uma constante $\boldsymbol{\lambda}$ e a condição de parada do algoritmo não for atingida, inserir um novo nó aqui

referido como $\mathbf{n_{novo}}$ segundo os procedimentos abaixo:

- Encontrar o nó $\mathbf{n_u}$ que possua o maior erro local acumulado $\mathbf{n_{u\_erro}}$;
- Encontrar entre os nós vizinhos de $\mathbf{n_u}$, o nó vizinho $\mathbf{n_v}$ de maior erro local acumulado $\mathbf{n_{v\_erro}}$;
- Inserir o novo nó $\mathbf{n_{novo}}$ no ponto médio da aresta que liga os nós $\mathbf{n_u}$ e $\mathbf{n_v}$;

$$c_{novo} = \frac{c_u + c_v}{2}$$

- Remova a aresta que liga $\mathbf{n_u}$ a $\mathbf{n_v}$ e insira duas novas arestas: uma ligando $\mathbf{n_{novo}}$ e $\mathbf{n_u}$ e outra ligando $\mathbf{n_{novo}}$ e $\mathbf{n_v}$;
- Reduza os erros locais acumulados dos nós $\mathbf{n_u}$ e $\mathbf{n_v}$ ($\mathbf{n_{u\_erro}}$ e $\mathbf{n_{v\_erro}}$), multiplicando-os por uma constante $\boldsymbol{\alpha}$;

$$n_{u\_erro} = \alpha \, . \, n_{u\_erro}$$

$$n_{v\_erro} = \alpha \, . \, n_{v\_erro}$$

- Inicialize o erro acumulado de $\mathbf{n_{novo}}$ com o novo erro acumulado de $\mathbf{n_u}$ ($\mathbf{n_{u\_erro}}$);

$$n_{novo\_erro} = n_{u\_erro}$$

De acordo com Moral (2007) um valor que pode ser adequado para a constante $\boldsymbol{\lambda}$ é 600 e para a constante $\boldsymbol{\alpha}$ é ½.

4. Decrescer os valores dos erros de todos os nós do grafo de acordo com um parâmetro de entrada $\boldsymbol{\beta}$. Onde, cada erro deve receber a multiplicação dele mesmo por ($\mathbf{1-\beta}$). De acordo com Moral (2007), um valor adequado para $\boldsymbol{\beta}$ pode ser $1x10^{-5}$;
5. Caso a condição de parada não tenha sido atingida, retornar ao passo 2. Uma possível condição de parada pode ser a quantidade de nós do grafo atingir um determinado número **N**;

Os passos acima são uma representação do algoritmo ***GNG***. Vale salientar que os parâmetros ($\boldsymbol{\varepsilon_a}$, $\boldsymbol{\varepsilon_t}$, **T**, **λ**, **α**, **β** e **N**) podem assumir valores diferentes dos sugeridos por Moral (2007).

4 Os Mapas auto-organizáveis de Kohonen

O algoritmo de **RNA** chamado de **Mapas Auto-organizáveis de Kohonen** (Do inglês, **S**elf-**O**rganized **M**ap, ou, apenas, **SOM**) em homenagem a seu criador é bastante utilizado no agrupamento de dados conforme suas similaridades. Uma possível aplicação do SOM é no problema de classificar pessoas de, por exemplo, uma rede social. É possível ao algoritmo, agrupar pessoas com conta em uma determinada rede social conforme seus dados registrados no sistema como: Sexo, Idade, Formação acadêmica, Estado conjugal, etc.

O **SOM** necessita que os dados a serem submetidos a rede sejam numéricos e que sua dimensão que é o número de atributos representados pelos dados de entrada seja compatível com a dimensão dos pesos dos neurônios. Exemplo: Dados cujos registros representam a idade, a altura, o sexo e o peso, assumem quatro dimensões. Durante o ajuste dos pesos, é calculada a distância euclidiana entre a entrada submetida a rede e cada peso de cada neurônio, por isso a necessidade de representar os dados de entrada com números. Também é importante salientar que uma matriz **MxN** pode representar M registros (ou dados) de dimensão N (Com N podendo ser maior que 2).

4.1 Vizinhança entre neurônios

Os neurônios no SOM formam um mapa bidimensional que pode ser organizado inicialmente de acordo com um entre dois modos de relação de vizinhança: A vizinhança retangular e a hexagonal. Não confundir com a distância euclidiana entre os dados e os pesos de cada neurônio. Veja a imagem abaixo:

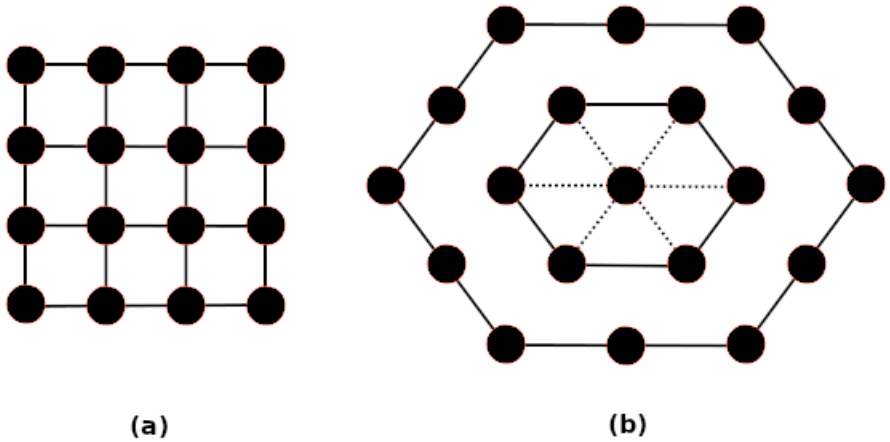


Figura 6: Vizinhanças retangular e hexagonal

Perceba que na figura acima os neurônios estão representados no espaço bidimensional, onde, no primeiro caso, em **(a)**, a vizinhança é retangular e cada neurônio tem apenas quatro vizinhos. No segundo caso, em **(b)**, a vizinhança é hexagonal e cada neurônio tem seis vizinhos. Contudo, há uma função de vizinhança que determina um fator de influência no ajuste dos pesos dos neurônios durante o treinamento da rede. Geralmente a função de vizinhança escolhida para o SOM é a gaussiana que está ilustrada na imagem abaixo:

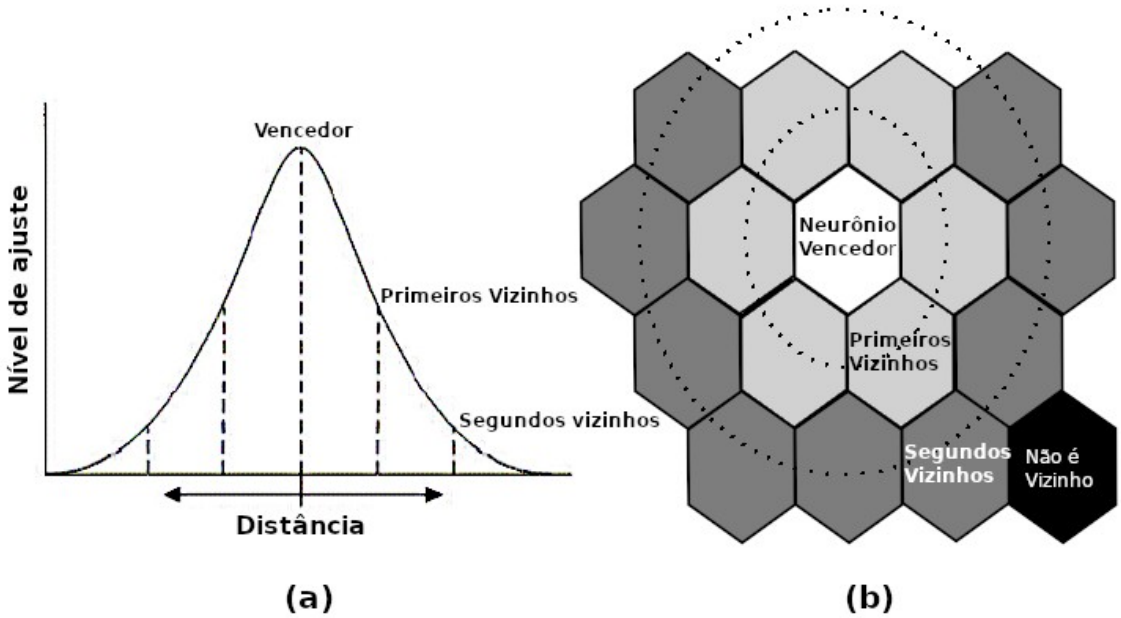


Figura 7: Função de vizinhança com efeito nos neurônios

Imagem adaptada de Moral (2008)

Em **(a)** está ilustrada a plotagem da função gaussiana em duas dimensões. E em **(b)** está ilustrado o raio de vizinhança entre os neurônios que sofrem alteração proporcional a distância a que se encontram do neurônio vencedor (Neurônio cujos valores dos atributos de seu peso mais se aproximam do sinal de entrada submetido)

4.2 O neurônio BMU

A cada iteração do ***SOM***, os dados de entrada são submetidos a rede e, para cada dado de entrada submetido, é calculado o neurônio cujo peso mais se aproxima do dado de entrada (O neurônio vendedor, ou **BMU** - **B**est **M**atch **U**nit). Tal cálculo pode ser feito, por exemplo, através da fórmula da distância euclidiana entre dois pontos, conforme será ilustrado a seguir:

A distância entre os pontos P(P1, P2,..., Pn) e Q(Q1, Q2,..., Qn) no espaço $\mathbf{R^n}$ pode ser calculada conforme a fórmula abaixo:

$$\|P-Q\| = \sqrt{(P1-Q1)^2 + (P2-Q2)^2 + \ldots + (Pn-Qn)^2} = \sqrt{\sum_{i=1}^{n} (Pi-Qi)^2}$$

Onde, quando aplicada ao SOM, **n** assume o valor correspondente a dimensão dos dados e pesos. A igualdade entre a dimensão dos dados e pesos é justamente para que se possa aplicar a fórmula acima para calcular a distância entre o dado **X** que pode ser associado na fórmula acima a variável **P** e o peso **W** que pode ser associado a variável **Q**. Um meio de representar o calculo do **BMU** ,conforme (MORAL, 2008), é:

$$c = argmin_i \{\|X - Wi\|\}, i \in \{1..N\}$$

Na fórmula acima a variável **c** recebe o índice do peso **Wi** do neurônio mais próximo do dado de entrada **X**. Assume-se, para a fórmula acima, que o número de neurônios é **N** e **i** varia de 1 a **N**. Então, apos determinado o índice do neurônio vencedor, podemos calcular o fator ajuste para os pesos através da função gaussiana de vizinhança.

4.3 A função de vizinhança

Uma das funções de vizinhança mais aplicadas em mapa auto-organizáveis é a função gaussiana ou, como também é conhecida, distribuição normal. A função gaussiana aplicada aos ***SOM*** é uma variação da distribuição normal adaptada ao algoritmo. Veja abaixo uma formulação matemática da função de vizinhança gaussiana:

$$H_{ci}(t) = e^{-\left(\frac{\|r_c - r_i\|^2}{2 \cdot \sigma^2(t)}\right)}, i \in \{1..N\}$$

Na fórmula acima, a função de vizinhança $\mathbf{H_{ci}(t)}$ é aplicada a cada dado de entrada submetido a rede. Isso porque, quando o dado é submetido, o ***SOM*** calcula o neurônio **BMU** cujo índice na fórmula acima é **c**. O **(t)** varia de 1 até o número de dados de entrada multiplicado pelo número de neurônios. A variável $\mathbf{r_c}$ é a posição do neurônio **BMU** na grade de neurônios e a variável $\mathbf{r_i}$ é a posição do neurônio **i,** ao qual se está calculando o fator de ajuste de seu peso e, a constante **N**, corresponde a quantidade de neurônios. É muito importante entender a função do raio de vizinhança **σ**, pois ele determina o alcance de ajuste (Conforme ilustrado na **Figura 7**). Isto é, para neurônios cujas distâncias entre eles e o **BMU** na grade (**mapa bidimensional** – não confundir com a distância entre os dados e os pesos) são maiores que o valor do raio de vizinhança, nenhum ajuste é feito porque o valor retornado da função de vizinhança para o ajuste desses neurônios é zero. Já para os neurônios cuja distância do **BMU** é igual ou inferior ao raio de vizinhança, há um ajuste proporcional a tal distância. Quanto menor a distância, maior o ajuste.

4.4 Taxa de aprendizado

Um fator muito importante do **SOM**, em analogia com os neurônios biológicos, é a taxa de aprendizado que, no ser humano, diminui conforme o tempo passa e o indivíduo envelhece. No **SOM**, a taxa de aprendizado também vai decrescendo a cada iteração de treinamento. Então, é necessário determinar a nova taxa de aprendizado após cada iteração e, o calculo pode ser feito conforme a fórmula abaixo ou outra semelhante:

$$\alpha(t) = \alpha_i \cdot \left(\frac{\alpha_f}{\alpha_i}\right)^{\left(\frac{t}{T}\right)}$$

Onde, $\boldsymbol{\alpha(t)}$ é a taxa de aprendizado atual que recebe a taxa de aprendizado inicial $\boldsymbol{\alpha_i}$ multiplicada pela divisão do seu valor final $\boldsymbol{\alpha_f}$ pelo seu valor inicial $\boldsymbol{\alpha_i}$ elevados ao

valor da iteração atual **t** dividida pelo número máximo de iterações **T**. Fórmula adaptada de SOUZA (2018).

4.5 Raio de vizinhança

Assim como a taxa de aprendizado, o raio de vizinhança é também muito importante. Pois ela determina o alcance do ajuste na grade bidimensional que representa a vizinhança (relações topológicas) entre os neurônios. E, como a taxa de aprendizado, o raio de vizinhança também deve ter seu valor decrementado após o final de cada iteração de treinamento. A fórmula abaixo representa um possível meio de se determinar o ajuste do raio de vizinhança:

$$\sigma(t) = \sigma_i \cdot \left(\frac{\sigma_f}{\sigma_i}\right)^{\left(\frac{t}{T}\right)}$$

Na fórmula acima, o raio de vizinhança **σ(t)** recebe a multiplicação de seu valor inicial σ_i pela divisão de seu valor final σ_f pelo valor inicial σ_i elevados a divisão do número da iteração atual **t** pelo número máximo de iterações **T**. Fórmula adaptada de SOUZA (2018).

4.6 Ajuste dos pesos

O cálculo realizado através da função de vizinhança, assim como a taxa de aprendizado, determina apenas um fator que tem influência no ajuste dos pesos dos neurônios. Veja a fórmula completa de ajuste de cada peso ilustrada abaixo:

$$Wi(t+1) = Wi(t) + \alpha(t) \cdot H_{ci}(t) \cdot [X - Wi]$$

Na fórmula acima, o peso **Wi(t+1)** é o novo valor do neurônio que recebe a soma

de seu valor atual **Wi** somado a taxa de aprendizado atual **α** multiplicada pelo fator obtido pelo calculo utilizando a função de vizinhança $\mathbf{H_{ci}(t)}$ e pela diferença entre o dado de entrada atual e o peso do neurônio corrente. O **i** varia de 1 ao número de neurônios da rede.

4.7 O algoritmo SOM

O algoritmo de mineração de dados deve ser aplicado na devida etapa do processo ***KDD***. Entretanto, é importante considerar que, os resultados do algoritmo dependem das etapas anteriores a mineração de dados. Isto é, a escolha das variáveis, a limpeza de dados ruidosos e a normalização são a base para os bons resultados do algoritmo, bem como, é também de muita importância a escolha da configuração adequada dos SOM conforme o problema que se queira aplicar o algoritmo. Abaixo uma sequência de passos simplificada do SOM:

1. Gerar a grade de neurônios conforme a vizinhança retangular ou hexagonal;
2. Iniciar os neurônios com pesos aleatórios;
3. Para cada iteração, escolher um dado de entrada aleatório, calcular o ***BMU*** e ajustar os pesos dele e de seus vizinhos;
4. Ajustar a taxa de aprendizado e o raio de vizinhança
5. Voltar ao passo 3 até que a convergência ou nível de treinamento desejado seja alcançado ou o número máximo de iterações seja atingido.

4.8 Visualização dos resultados com a matriz-u

A última etapa do processo de descoberta do conhecimento é a visualização dos resultados e vem logo após a mineração dos dados, onde, é aplicado o algoritmo de

classificação ou agrupamento dos dados. Existem diversas formas de visualizar os resultados do processo de mineração de dados. Algumas, permitem até a visualização dos resultados representados em espaço multidimensional. Geralmente, os dados de entrada a serem submetidos a rede SOM representam vários atributos de entidades as quais se quer, por exemplo, encontrar características em comum. Dada esta representação, que pode ser multidimensional, surge o problema de visualização de resultados em espaços de dimensão maior que três.

Uma das formas para visualização dos resultados do SOM é através da geração de uma matriz bidimensional de números chamada **matriz-u** (**matriz de distâncias unificada**), ou, do inglês, **u-matrix**. A **matriz-u** deve ser calculada para representar as distâncias entre os neurônios. Por exemplo, veja as imagens abaixo:

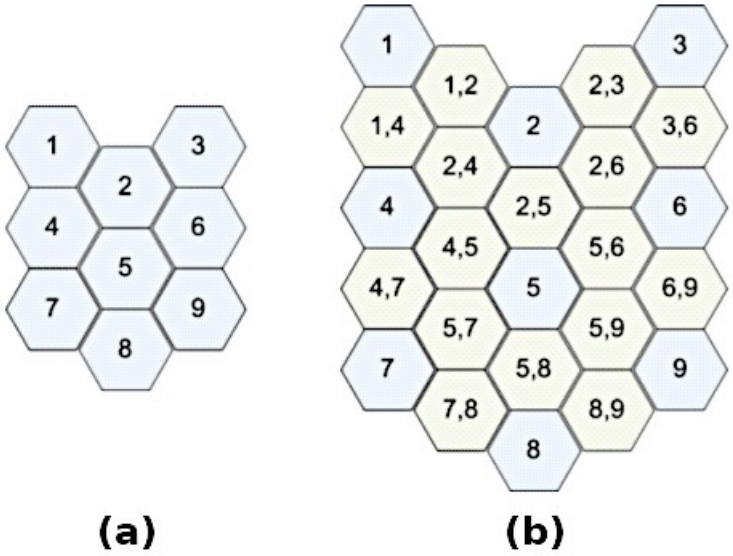


*Figura 8: Matriz de neurônios **(a)** e Matriz-U hexagonal correspondente **(b)**.*

Imagem adaptada de STACKOVERFLOW(2019)

Perceba que, em **(a)**, está a representação dos neurônios em uma grade 3x3. A vizinhança neste exemplo, como se pode perceber, é hexagonal. Então a **matriz-u** pode

ser gerada a partir dos pesos dos neurônios conforme é ilustrado em **(b)**. Para o cálculo da dimensão da **matriz-u**, se pode utilizar a seguinte fórmula:

M = 5 + ((grade_larg – 3) * 2);
N = 5 + ((grade_alt – 3) * 2);

Onde **M** corresponde a largura da ***matriz-u*** e **N**, a altura e, **grade_larg** representa o valor da largura da grade de neurônios e, **grade_alt** a altura. Por exemplo, para uma grade **3x3**, a **matriz-u** que representa suas distâncias é **5x5**. Se a grade for **5x5**, a **matriz-u** assume a dimensão **9x9**. Quanto as distâncias que a **matriz-u** representa, elas são distribuídas utilizando a seguinte lógica: Tomando como exemplo o neurônio 5 seus vizinhos são os neurônios: 2, 4, 6, 7, 8, 9. Logo, entre os neurônios 5 e 4 há outro valor na **matriz-u** que corresponde a distância entre os neurônios 5 e 4 (conforme seus pesos), esse valor é numerado como 4,5 na representação da **matriz-u** acima. A distância que é representada pelo número 5 é igual média aritmética das distâncias vizinhas, isto é, após calculadas as distâncias: 2,5; 4,5; 5,6; 5,7; 5,8; 5,9, a distância numerada com 5 assume a média aritmética das desses valores. O mesmo pode ser repetido para o cálculo de cada elemento da **matriz-u**. E se a grade de neurônios for retangular, para uma grade **3x3**, a **matriz-u** se parece com a representação ilustrada na imagem abaixo:

1	1,2	2	2,3	3
1,4	?	2,5	?	3,6
4	4,5	5	5,6	6
4,7	?	5,8	?	6,9
7	7,8	8	8,9	9

*Figura 9: **Matriz-U** de vizinhança retangular*

Perceba que os neurônios são representados com elementos de cor azul, as distâncias entre os neurônios vizinhos são representados pelos elementos de cor

amarela e, os elementos de cor vermelha correspondem a média aritmética das distâncias vizinhas. No caso, as distâncias de cor amarela, vizinhas do elemento de cor vermelha. É considerável que a **matriz-u hexagonal** representa melhor a topologia da rede.

É importante considerar que, dado que a **matriz-u** representa distâncias, quanto mais próximos estiverem os elementos de um determinado grupo de neurônios, menor a distância. O que significa, também, que quanto mais distantes estiverem os neurônios um do outro, maiores as chances de pertencerem a classes distintas. Ou seja, é possível, com base na **matriz-u** gerada, calcular um fator proporção entre a distância representante de cada elemento da **matriz-u** e a diferença entre a maior e a menor distância. Com essa normalização das distâncias representadas na **matriz-u**, é possível calcular a intensidade numa escala de zero a um de cada distância. Por exemplo, veja abaixo a visualização das distâncias representadas por uma **matriz-u**:

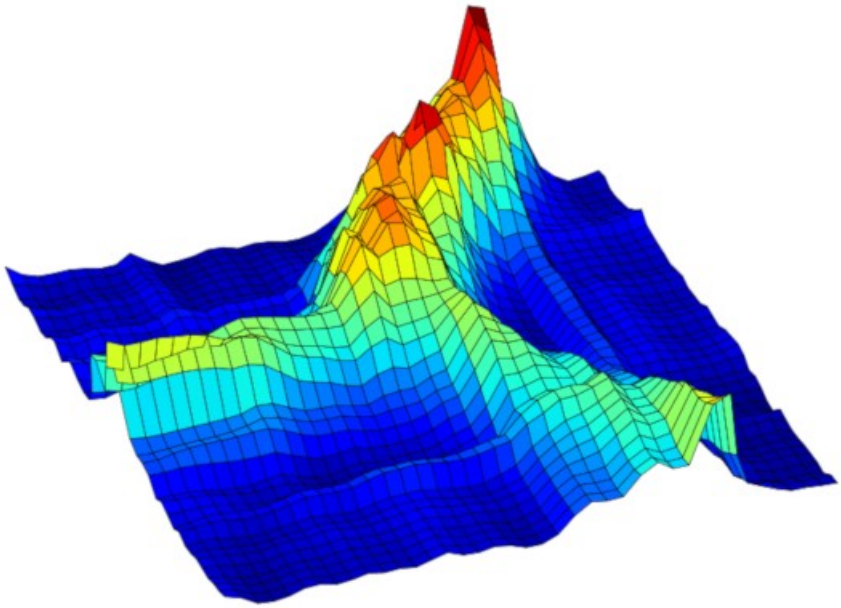


Figura 10: Visualização de distâncias topológicas conforme matriz-u.

Imagem de QUAZOO (2014)

De acordo com a imagem acima, pode-se perceber que os valores de cada elemento da matriz variam de azul para vermelho, ou seja, quanto mais próximo da cor vermelha, maior a distância. Quanto mais próximo cor azul, menor a distância.

4.9 O conhecimento

A partir da **matriz-u** de distâncias, pode-se determinar os neurônios que pertencem aos diferentes grupos. Dado que, os grupos são delimitados por valores altos na escala de 0 a 1. Pode-se assumir o valor de uma constante que, quando ultrapassado, indique que aquele neurônio não pertence a nenhuma classe, mas, sim, ao grupo de neurônios que, conforme a **matriz-u**, delimitam os grupos. Então, um determinado dado de entrada pode ser comparado com o neurônio mais próximo a ele para, assim, ser associado ao grupo ao qual o neurônio pertence.

Após associados os neurônios a seus grupos, se pode gerar dados estatísticos com base neles. Por exemplo, considerando uma base de dados de clientes, se pode fazer a média aritmética do atributo idade de todos os clientes de um determinado grupo e descobrir que, os clientes pertencentes aquele grupo são de uma determinada faixa etária, por exemplo: entre 20 e 30 anos.

4.10 Cálculo de erros

Em algoritmos de aprendizado supervisionado o erro costuma ser calculado após cada saída da rede de neurônios artificiais (UFOP, 2017). Já em se tratando do **SOM**, se pode comparar a diferença entre os dados de entrada e os pesos dos **BMUs** durante cada submissão de dados de entrada a rede. Existem basicamente dois tipos de erros que podem ser calculados durante o treinamento da rede, são eles: **erro de quantização** e **erro topográfico**.

O calculo do erro relacionado ao treinamento de **RNAs**, é de suma importância dado que, com ele, é possível compreender a diminuição das distâncias entre os neurônios ***BMUs*** e os dados de entrada (Erro de quantização) ou, ainda, compreender se a topologia da rede está sendo preservada durante o treinamento através do erro topográfico. Por exemplo, pode-se perceber que, após um determinado número de iterações, a distância entre os **BMUs** e os dados de entrada diminuiu consideravelmente, o que significa que os neurônios da rede se tornaram mais parecidos com os dados de entrada, o que tem influência na diminuição do erro.

4.10.1 Erro de quantização

O erro de quantização mede a evolução de aproximação dos **BMUs** e dos dados de entrada. Conforme as iterações vão sendo executadas, os **BMUs** tentem a se ajustarem de modo a ficarem mais próximos dos dados de entrada. Com isso, a distância entre eles e os dados de entrada diminuem. Veja a imagem abaixo:

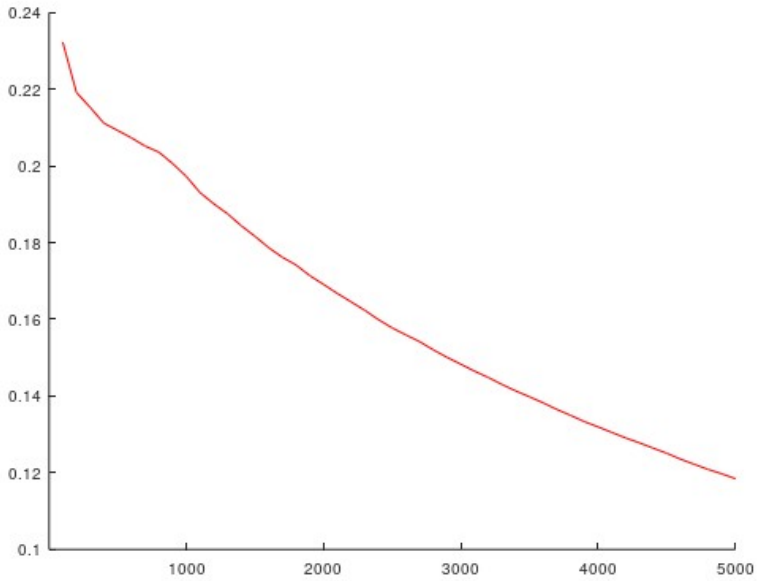


Figura 11: Diminuição do erro de quantização

Na figura acima se pode interpretar que o erro começou próximo de 0,23 e foi diminuindo conforme o avanço do número de iterações. Pode-se perceber também que a figura acima representa o gráfico da função referente a tendência do erro que, neste exemplo, é decrescente. Veja a fórmula abaixo:

$$QErro = \frac{1}{I} \cdot \sum_{i=1}^{I} \left\| X_i - W_{ci} \right\|$$

Na fórmula acima, o **I** representa a iteração atual a qual se está calculando o erro de quantização. Por exemplo, se 1000 iterações forem divididas em 5 partes, o erro de quantização é calculado com o **I** recebendo os respectivos valores: 200, 400, 600, 800, 1000. Isso se a intenção for gerar um gráfico de, por exemplo, quais os valores do erro de quantização nas iterações escolhidas. O $\mathbf{X_i}$ representa o dado de entrada submetido na iteração **i** e o $\mathbf{W_{ci}}$ é o neurônio **BMU**, o que mais se aproximou de $\mathbf{X_i}$ na referida iteração. O valor somado é a distância euclidiana entre $\mathbf{X_i}$ e $\mathbf{W_{ci}}$. Em resumo, a fórmula acima calcula a média aritmética das distâncias entre os dados de entradas e os **BMUs** até determinada iteração.

4.10.2 Erro topológico

O erro topológico mede a proporção, num intervalo de 0 a 1, de o quanto o mapa permanece organizável em uma determinada iteração. Ou seja, permite verificar se a topologia da rede está sendo preservada durante o treinamento. Veja a imagem abaixo:

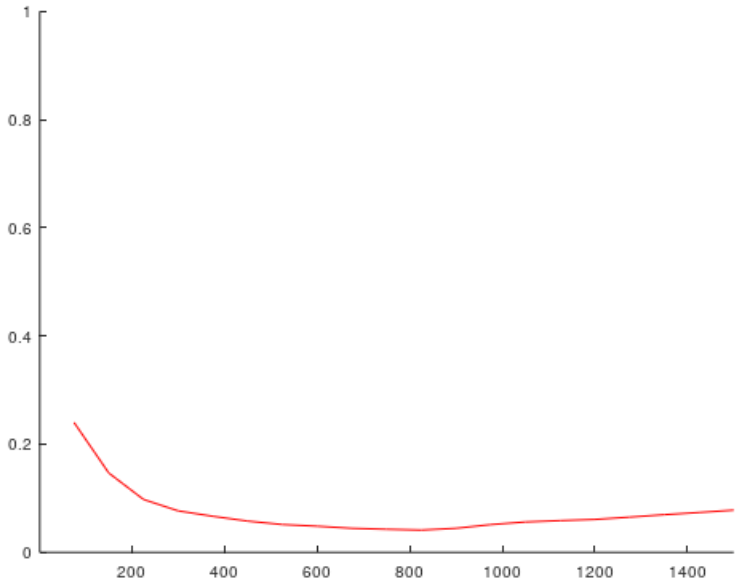


Figura 12: Ilustração do erro topológico

Perceba na figura acima que o erro topológico permanece quase constante em torno de, aproximadamente, 0,1. O que é normal, pois, o erro topológico varia de 0 a 1 e, se o erro permanece quase constante e próximo de 0, significa que a topologia foi preservada. Na figura acima, a medida de erro topológico na iteração 400 é quase o mesmo das iterações 600, 800, 1000 e 1500. Veja abaixo a fórmula que pode ser utilizada para o cálculo do erro topológico:

$$TErro = 1 - \frac{1}{I} \cdot \sum_{i=1}^{I} f(X_i)$$

Na fórmula acima, assim como na do erro de quantização, o **I** representa o número da iteração a qual se deseja calcular o erro topológico. Por exemplo, pode-se querer gerar um gráfico sobre os erros topológico nas iterações 500, 1000, 1500 e 2000. Já o $\mathbf{f(X_i)}$, representa uma função que retorna o valor 1 se o primeiro ***BMU*** e o segundo ***BMU*** de $\mathbf{X_i}$ são vizinhos e 0 caso contrário. O somatório é uma soma de zeros e uns que é dividido pelo número da iteração.e a média aritmética encontrada é então subtraída

de 1.

4.11 Fases do SOM

Para a obtenção de melhores resultados, é comum a divisão do treinamento feito pelo SOM em fases. Geralmente, são duas fases apenas. A primeira, a fase de ordenação e, a segunda, a fase de ajuste fino ou refinamento ou, ainda, fase de convergência.

Durante a primeira fase, se pode treinar os mapas auto-organizáveis de modo que, se os pesos dos neurônios estiverem representados em 2 ou 3 dimensões, eles sejam ordenados e organizados (dai o nome auto-organizáveis) para, juntos, se parecerem com uma superfície malhada, mesmo que os pesos dos neurônios recebam valores aleatórios.

Durante a segunda fase, busca-se a convergência, ou uma aproximação dela. Geralmente a convergência é baseada em um critério. Por exemplo, pode-se ter como critério de convergência a distância mínima, dado um valor de distância mínima, entre os neurônios e algum dado de entrada, o que significa que, se todos os neurônios tiverem distância do dado de entrada mais próximo igual ou menor que o valor de distância mínima, então, o treinamento atingiu o nível de convergência desejado e o erro foi minimizado o suficiente. É importante frisar que, se o nível de convergência não for atingido, um número de iterações máximo deve finalizar a execução do algoritmo.

Normalmente, na fase de convergência, a taxa de aprendizado recebe de início um valor pequeno e o raio de vizinhança também, com o raio de vizinhança permanecendo constante ou pode decrescer lentamente e a taxa de aprendizado deve apenas decrescer lentamente em direção a taxa de aprendizado final (ou mínima).

As configurações do SOM podem ser muito diferentes para diferentes aplicações. Por isso, é difícil estabelecer um valor padrão para a taxa de aprendizado e raio de

vizinhança na fase de convergência. Mas, muitas vezes, no início da fase de convergência, a taxa de aprendizado pode assumir, por exemplo, o valor 0,02 e o raio de vizinhança, assumir o valor 1.

4.12 Os SOM em conjunto com o K-Means

O ***K-Médias*** pode ser aplicado ao agrupamento dos neurônios resultantes do treinamento com aplicação dos ***SOM***. Isto é, após executado o algoritmo ***SOM*** para agrupamento dos dados, se pode visualizar a quantidade de grupos a partir da **matriz de distâncias unificada** (a ***matriz-u***) e, então, saber qual o número de centroides deve ser passado para o algoritmo ***K-Médias***.

Após detectado o número de grupos (representados por centroides pelo ***K-Means***), se pode agrupar os neurônios e, então, associar os dados de entrada ao neurônio mais próximo (através da distância euclidiana por exemplo) e, assim, saber a qual grupo os dados pertencem e, então, gerar outros dados estatísticos para cada grupo detectado pelos ***SOM***.

5 Prática da teoria

Esse capítulo trata de expor e explicar resultados obtidos de experimentações com os algoritmos ***SOM***, ***K-Means*** e ***GNG***. A intenção é a distinção de grupos gerados pelos algoritmos de modo que se possa associar dados a serem agrupados a seus devidos grupos, de acordo com a semelhança entre tais dados e os pesos dos neurônios (ou centroides) das classes.

Para pôr em prática os conceitos aqui apresentados, optou-se por utilizar a linguagem de programação Java e um software gratuito: O ***Octave***, que tem uma linguagem de programação própria semelhante ao ***Matlab.***

5.1 A linguagem Java

A linguagem de programação Java é uma das mais robustas que existem. Ela pode ser aplicada a diversos tipos de projetos. É multiplataforma, o que significa que, softwares criados a partir dela, podem executar em sistemas operacionais diferentes. Ela pode, ainda, ser utilizada para produção de sistemas de grande porte e é considerada uma linguagem com ótimo nível de confiabilidade. Isso porque, em comparação com as linguagens de programação C/C++, a linguagem Java dificulta a possibilidade de erros de programação.

A orientação a objetos (OO) é um paradigma de linguagens de programação que oferece diversas vantágens em comparação com o paradigma de linguagens estruturadas. Dentre os benefícios de se escolher uma linguagem OO para trabalhar, estão a organização do software em classes, o que permite organizar melhor o código e aplicar padrões de projeto que facilitam o trabalho em equipe e diminuem as chances de fracaço de um projeto. Java é uma linguagem de programação orientada a objetos assim como C++: linguagem de que Java é derivada. Já as linguagens C e Pascal são

estruturadas onde o programa é organizado em instruções sequenciais e funções ou procedimentos que são subprogramas e podem ser referenciados em um programa principal ou outro subprograma.

A linguagem Java pode ser aplicada em manipulação de imagens, interface gráfica com o usuário através do pacote de classes "javax.swing", o que permite o desenvolvimento de aplicativos desktop. A tecnologia Java também pode ser aplicada em programação de dispositivos móveis, sistemas de grande porte como os que automatizam transações bancárias e outras operações pertinentes ao sistema financeiro. Ela tem suporte a: multithread, programação em rede, desenvolvimento de sistemas WEB com ***JSP/Servlets*** e conexão com sistema de bancos de dados através da tecnologia **JDBC** - ***J**ava **D**ata**B**ase **C**onnection*. Entre outros recursos disponíveis pela linguagem. A empresa que desenvolve e mantém a tecnologia Java atualmente é a ORACLE (2019).

Os recursos gráficos disponíveis pela linguagem Java para o desenho de formas geométricas básicas, além da característica de orientação a objetos, a torna uma ótima escolha para aplicação de algoritmos de mineração de dados e visualização dos resultados obtidos. Através da linguagem de programação Java, também se pode manipular uma uma base de dados e minerar a partir dos dados armazenados nela.

Para esse trabalho, a linguagem Java foi utilizada para alguns exemplos em que o SOM e K-Means foram aplicados. Foi utilizada também a linguagem própria do Octave que será detalhado logo a seguir.

5.2 O Octave

O ***Octave*** é um software com sintaxe orientada a matemática com ferramentas de plotagem e visualização de dados embutida. É desenvolvido e mantido pela GNU (2019). Trata-se de um software livre, disponibilizado em versões para os sistemas

operacionais Windows, GNU/Linux, BSD e MacOS e tem grande compatibilidade com scripts do ***MatLab***, outro software com sintaxe orientada a matemática. Veja abaixo um pequeno script que pode ser escrito na linguagem do ***Octave*** que tem certo grau de compatibilidade com a linguagem própria do ***MatLab***:

```
1;

x=-pi:0.1:pi;
y=sin(x);

plot(x,y)
```

O código acima carrega uma variavel **x** como um vetor cujos elementos variam de **-pi** a **pi**, com esse intervalo dividido com espaços de 0.1 entre cada elemento do vetor. Logo após, a variável **y** é carregada com o vetor correspondente ao valor da função **seno** aplicada a cada respectivo valor do vetor **x**. E, então, o comando ***plot*** é chamado para a plotagem da função **seno** em função de **x** conforme os valores de **x** e **y**. Para executar o código acima, você pode escrevê-lo em um arquivo texto, salvá-lo com extensão "**.m**" e executá-lo. Bastando para isso, digitar o nome do arquivo (sem a extensão **.m**) na linha comandos do software ***Octave*** seguido do pressionamento da tecla enter. (**Obs**: o script deve ser iniciado com **1;**). Por exemplo: se o arquivo foi salvo com nome **prog.m**, basta navegar até a pasta onde o arquivo está (através do comando **cd**) e, então, digitar o nome **prog** e pressionar enter. Veja abaixo o resultado do código acima:

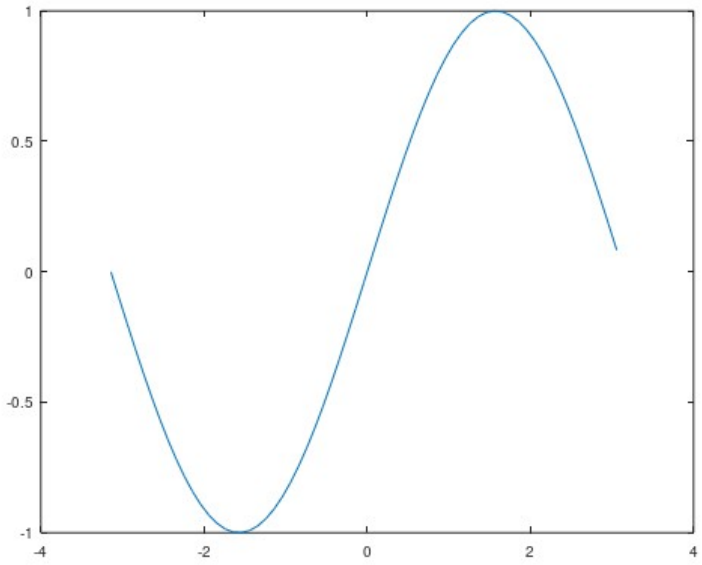


Figura 13: Gráfico da função seno no Octave

Perceba que também se pode plotar a função acima apenas digitando os devidos comandos na linha de comandos do octave. Os scripts geralmente são utilizados como linguagem de programação para aplicações mais complexas conforme será feito e mostrado mais adiante na aplicação do **SOM**.

5.3 Experimentos com o SOM

Como visto no **capítulo 5** dedicado ao ***SOM***, se pode treinar os mapas auto-organizáveis de modo que, se os pesos dos neurônios estiverem representados em 2 ou 3 dimensões, eles sejam ordenados e organizados para, juntos, se parecerem com uma superfície malhada, mesmo que os pesos dos neurônios recebam valores aleatórios de início. A ordenação também acontece em se tratando de pesos com dimensão maior que 3, mas, é de difícil visualização.

5.3.1 Ordenamento e agrupamento de cores

Um exemplo clássico de aplicação dos mapas auto-organizáveis é no

agrupamento de cores. Por exemplo, pode-se gerar um mapa de cores aleatórias e associar os valores ***RGB*** de cada cor aleatória a seu devido neurônio. O ***SOM***, quando aplicado a esse problema, tende a ordenar (organizar) as cores de modo a formar classes de cores parecidas.

Para geração do mapa de cores organizado, foi escolhida a linguagem Java porque ela dispõe de ótimos recursos gráficos, além da robustez associada a orientação a objetos suportado pela mesma. Foi desenvolvida em Java uma pequena caixa de ferramentas (***framework***) para aplicação do ***SOM*** a algum problema e, nesse caso, o problema em questão é a organização de cores dispostas de modo aleatório em um mapa de cores bidimensional. Vale esclarecer que a dimensão dos pesos e dados de entrada é 3, já que cada cor tem 3 atributos. O conjunto de amostras de cores utilizado para treinar a rede está logo abaixo:

(255,0,0) – Azul
(0,128,0) – Verde
(0,0,255) – Azul
(0,100,0) – Verde escuro
(0,0,139) – Azul escuro
(255,255,0) – Amarelo
(255,165,0) – Laranja
(128,0,128) – Lilas

Cada uma das 8 amostras acima representa os valores RGB de uma cor. A configuração da rede está mostrada logo abaixo:

Grade = 15x20
Distância bidimensional entre cada neurônio = 5
Taxa de Aprendizado Inicial = 0.5
Taxa de Aprendizado Final = 0.01
Raio de Vizinhança Inicial = 5
Raio de Vizinhança Final = 1
Tipo de Vizinhança = Hexagonal
Pesos inicializados aleatoriamente com valores RGB

Veja abaixo ilustrações dos resultados alcançados com a aplicação dos ***SOM*** no problema de organização de cores:

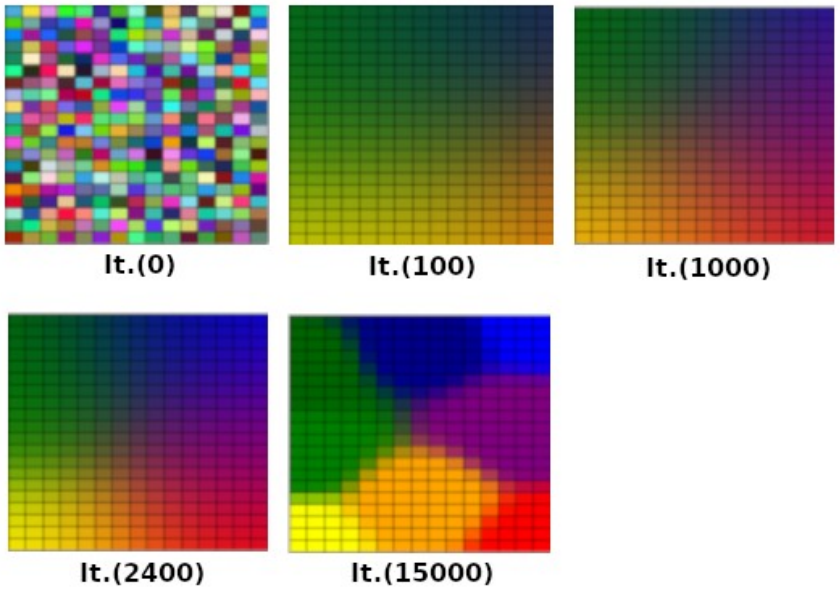


Figura 14: Classificação de cores com os SOM

Na figura acima, se pode perceber que, na iteração 0 [**It.(0)**], os pesos têm valores de cores aleatórias. Perceba que o mapa é organizado após 15000 iterações e cada cor tem um conjunto de neurônios associado. São 8 grupos no total, nesse exemplo, um grupo por amostra.

5.3.2 Exemplo de visualização de Matriz-U

Agora será abordado sobre um experimento com os mapas auto-organizáveis aplicados em uma base de dados gerada matematicamente em que os dados são gerados e organizados em 4 grupos distantes uns dos outros. Após o treinamento, é gerada uma **matriz-u** que permite a visualização dos 4 grupos.

A configuração da rede para esse exemplo é mostrada logo abaixo:

Grade = 13x13
Distância bidimensional entre cada neurônio (**grade_neurs_dist**) = 5
Taxa de Aprendizado Inicial = 0.5
Taxa de Aprendizado Final = 0.02
Raio de Vizinhança Inicial = 13 * **grade_neurs_dist** * 0.5
Raio de Vizinhança Final = 1;
Tipo de Vizinhança = Retangular
Número de iterações = 1500
Pesos inicializados com valores aleatórios

Para o treinamento dos ***SOM***, foi utilizado o ***octave***. Ele não dispõe de uma ***SOM toolbox*** como o ***MatLab*** e, por isso, foi necessária a implementação dos algoritmos: ***SOM*** e visualização em matriz de distâncias unificada (**matriz-u**). Veja abaixo os resultados obtidos com o treinamento dos mapas auto-organizáveis e a visualização dos resultados com base na ***matriz-u*** gerada a partir dos neurônios treinados:

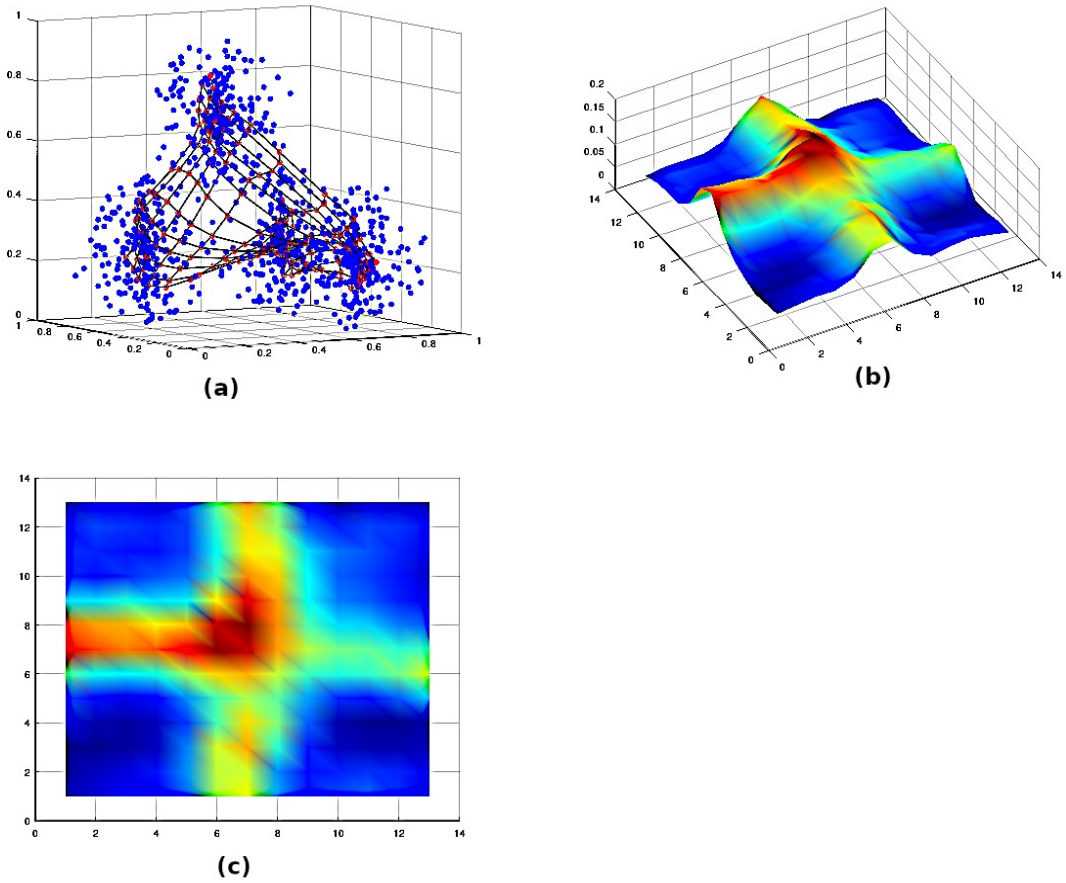


Figura 15: Plotagem da rede, dados e matriz-u

Perceba na figura acima que em **(a)**, são mostrados: os neurônios de cor vermelha, as arestas de cor preta e os dados de entrada (ou amostras) de cor azul. Já em **(b)**, é mostrada a visualização baseada na **matriz-u** gerada. Os picos representam delimitações dos grupos que são associados a cor azul. São quatro grupos no total como se pode observar em **(a)** comparando com **(b)**. Também cabe explicação sobre: a similaridade entre os neurônios cujas distâncias topográficas são representadas na **matriz-u** com cor azul (eles compõem um grupo) e as diferenças das distâncias topográficas dos neurônios delimitadores de grupos cujas representações na **matriz-u** tendem a cor vermelha. Em **(c)** é mostrada em duas dimensões a mesma **matriz-u** de

(b).

Os gráficos dos erros de quantização e topográfico gerados com a execução do algoritmo **SOM** com base na configuração do exemplo da **Figura 15** é mostrado abaixo:

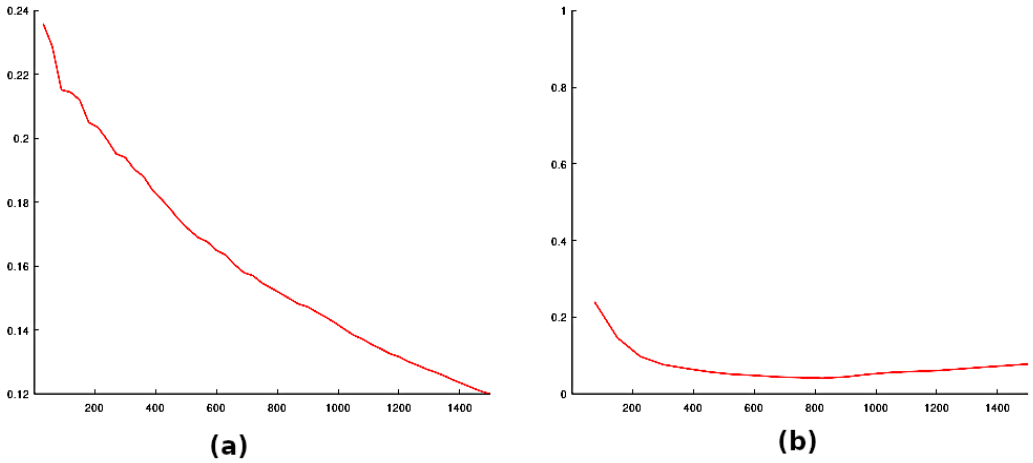


Figura 16: Plotagem dos erros de quantização e topológico

Na figura acima, em **(a)** é mostrado o gráfico do erro de quantização onde é feito o calculo com base na fórmula mostrada em 4.10.1. E, a direita, em **(b)**, é mostrado o gráfico do erro topológico (calculado pela fórmula exibida em 4.10.2) que permanece quase constante em torno do valor 0,1, o que é um bom resultado, dado que, conforme a rede vai sendo treinada, há uma pequena diminuição no fator que mede o quanto a topologia da rede está sendo preservada. **Obs:** os erros exibidos na figura acima são calculados a cada 30 iterações de 1500 no total.

5.3.3 Aplicativo de execução do SOM

Com esse trabalho foi desenvolvido um aplicativo que permite a execução do algoritmo ***SOM*** com pesos em duas, três ou múltiplas dimensões. A partir de agora, esse software e ***framework*** será referenciado pelos nomes: ***aplicativo SOM*** e ***framework SOM***, respectivamente. Com esse aplicativo é possível visualizar, em tempo real, o

treinamento da rede, bem como, o valor de seus parâmetros em cada iteração da execução do algoritmo.

Com o aplicativo ***SOM***, é possível também a configuração inicial do mapa, além da geração automática de dados de entrada em que os dados são gerados automaticamente pelo algoritmo de geração de dados conforme a especificação de posições de que os dados gerados devem ser relativos. O algoritmo de geração automática gera os dados: (a) internos a uma esfera, se os dados devem ter três dimensões ou, (b) internos a um círculo, se os dados devem ser gerados com apenas duas dimensões. O raio do círculo ou esfera é configurável.

Também é possível o carregamento de dados armazenados em arquivo de texto. Por exemplo, se pode gerar dados de entrada em softwares como: ***Octave*** e ***MatLab*** e carregá-los no aplicativo. Inclusive, é possível também, carregar os dados de uma aplicação que possa gerar os dados de entrada a partir de dados armazenados em um banco de dados.

5.3.4 Execução do aplicativo SOM com dados 2D

Abaixo um exemplo em que os dados de entrada e pesos do mapa de neurônios estão no espaço bidimensional.

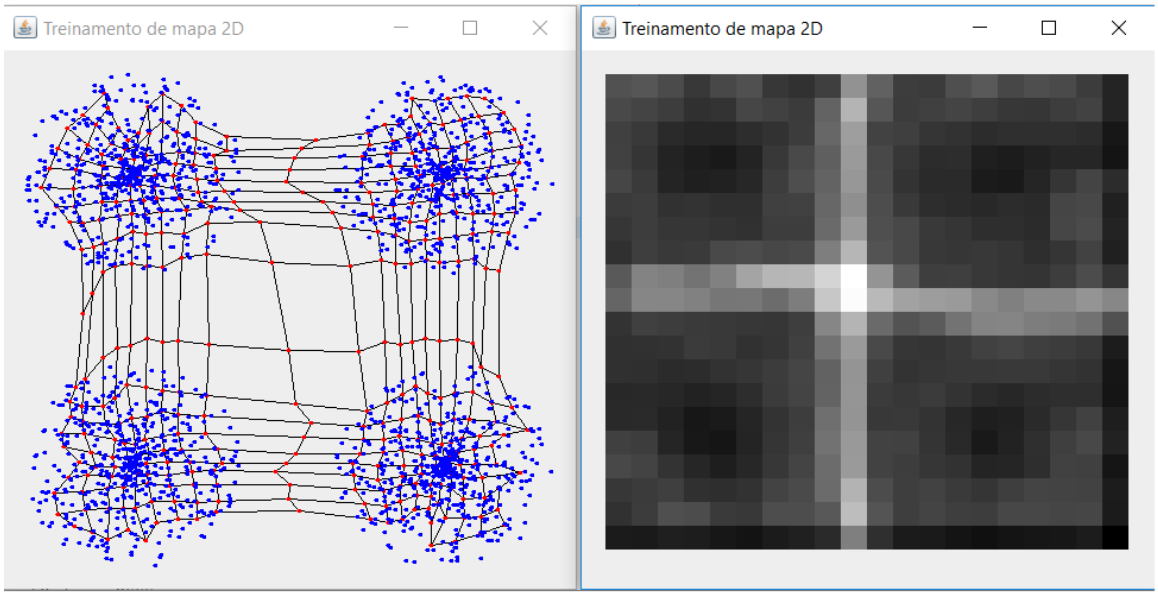


Figura 17: Aplicativo SOM 2D

A imagem acima é uma captura de imagem do aplicativo de treinamento e organização de um mapa de neurônios de pesos com espaço de entrada bidimensional. Analisando **(a)** na imagem acima, se pode perceber os pontos azuis. Eles representam amostras de treinamento (dados de entrada). Os pontos vermelhos estão associados aos pesos dos neurônios e as arestas pretas, as relações de vizinhança. Perceba o resultado do treinamento onde os neurônios do meio estão distantes dos próprios vizinhos e, os neurônios próximos dos dados de entradas, estão também próximos de seus vizinhos. Em **(b)** está a **matriz-u** referente ao resultado do treinamento exibido em **(a)**.

5.3.5 Execução do aplicativo SOM com dados 3D

Agora um exemplo em que os dados de entrada estão representados em 3D e são gerados automaticamente pelo algoritmo de geração de dados do **aplicativo *SOM***. Se os dados carregados pelo aplicativo são em 3D, o programa permite a execução em

tempo real do algoritmo, permitindo também a rotação com uso do mouse para visualização em vários ângulos da rede sendo treinada. Veja a imagem abaixo:

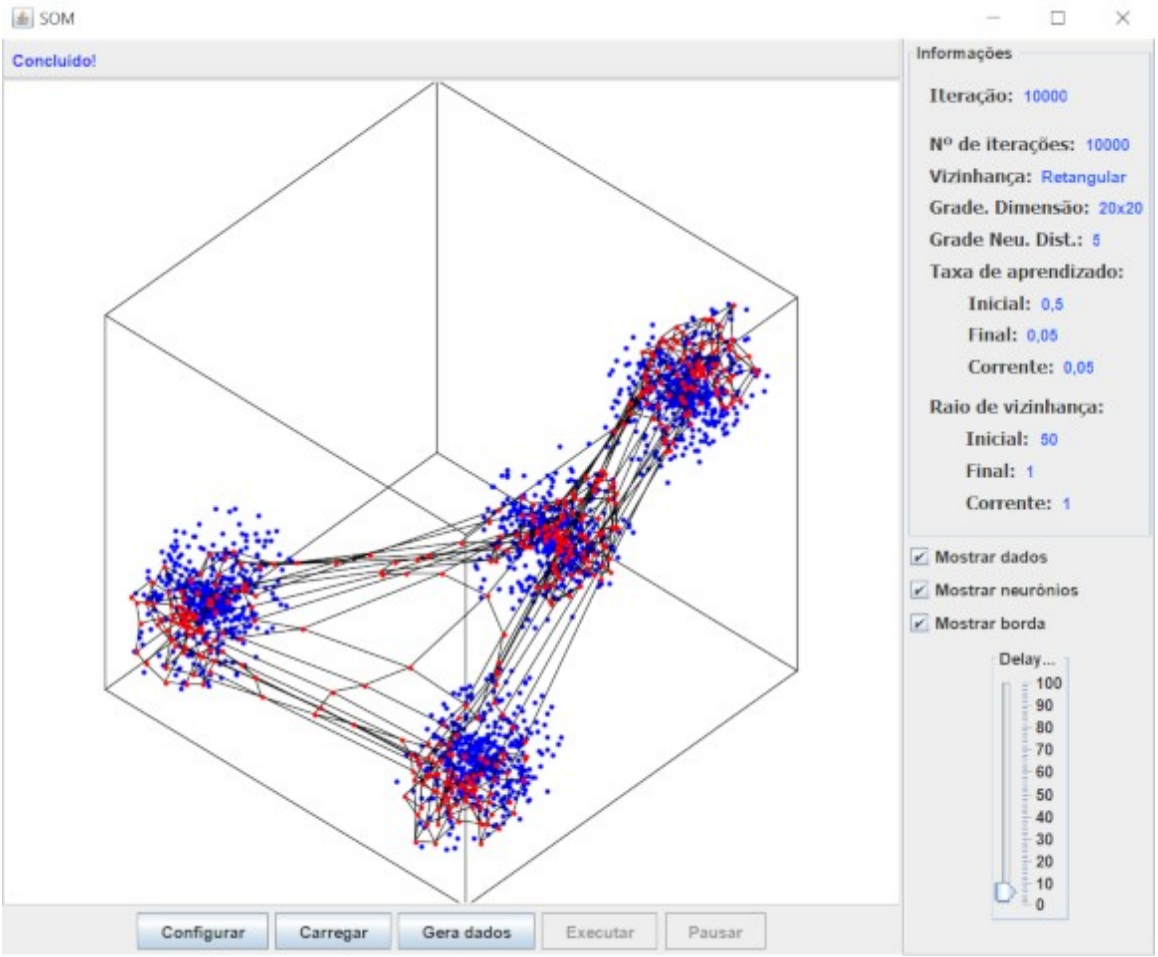


Figura 18: Aplicativo SOM 3D

A imagem acima ilustra a execução do algoritmo SOM com dados de entrada 3D gerados automaticamente. Perceba que é possível através do aplicativo, várias configurações como mostrar ou esconder bordas, dados e neurônios e configurar o ***delay*** (tempo de parada após cada iteração do algoritmo). É possível também a visualização dos parâmetros da rede, bem como, a pausa e continuação da execução do algoritmo. Após o final da execução do algoritmo é mostrada uma visualização da

matriz-u correspondente ao resultado do agrupamento dos dados. Veja abaixo a visualização da ***matriz-u*** resultante do treinamento da rede:

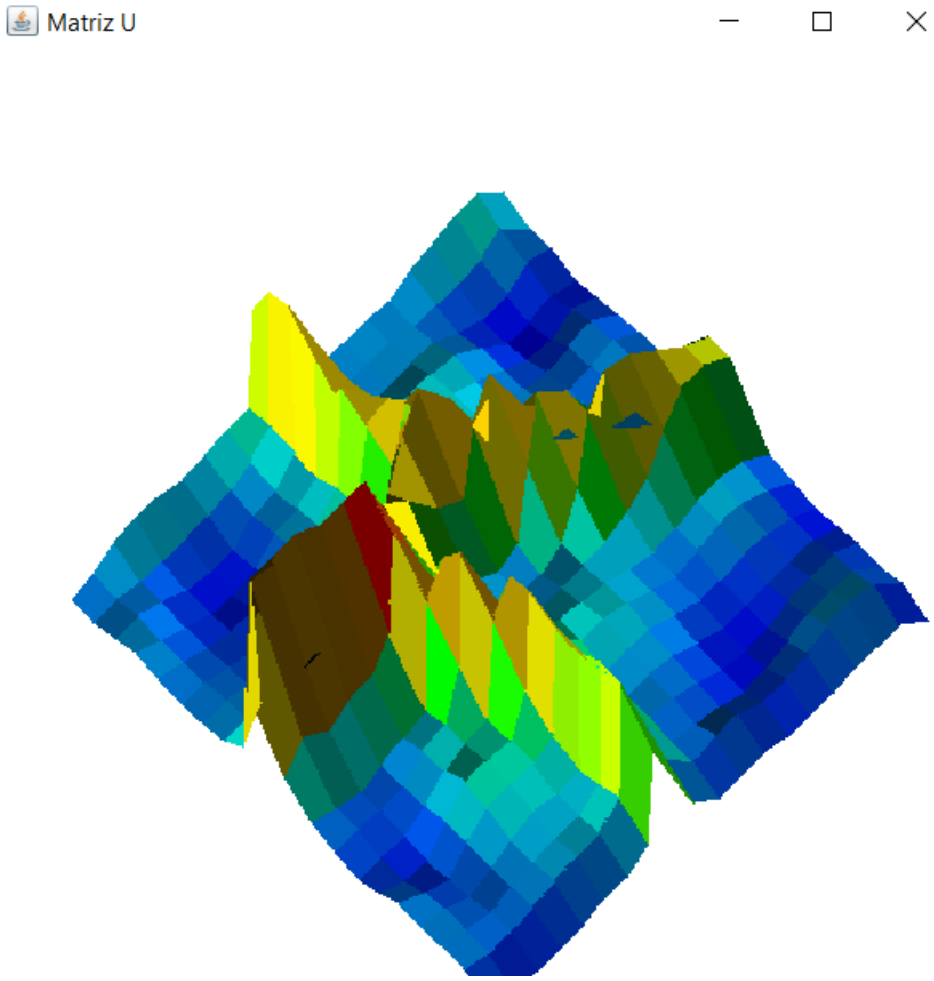


Figura 19: Matriz-U correspondente ao treinamento

A figura acima ilustra uma **matriz-u** onde os picos representam delimitações de grupos e as partes de azul representam neurônios próximos que formam grupos. Facilmente pode ser notada a existência de quatro grupos. No entanto, vale salientar que, em se tratando de bases de dados reais, geralmente a **matriz-u** é mais complexa de ser interpretada que a visualização da **matriz-u** ilustrada na **Figura 19.**

.

5.3.6 Aplicativo SOM 3D (superfície esférica)

Para exemplificar a geração de dados por um software externo e carregamento pelo **aplicativo *SOM***, foi utilizado o software ***Octave*** para gerar dados sorteados na superfície de uma esfera com um determinado raio. O arquivo gerado forma um arquivo de texto com espaços entre os atributos e quebras de linha separando os registros. Os dados gerados são representados em ponto flutuante.

Após carregados os dados da superfície esférica no **aplicativo *SOM***, o algoritmo foi executado para auto-organizar o mapa. Veja o resultado abaixo:

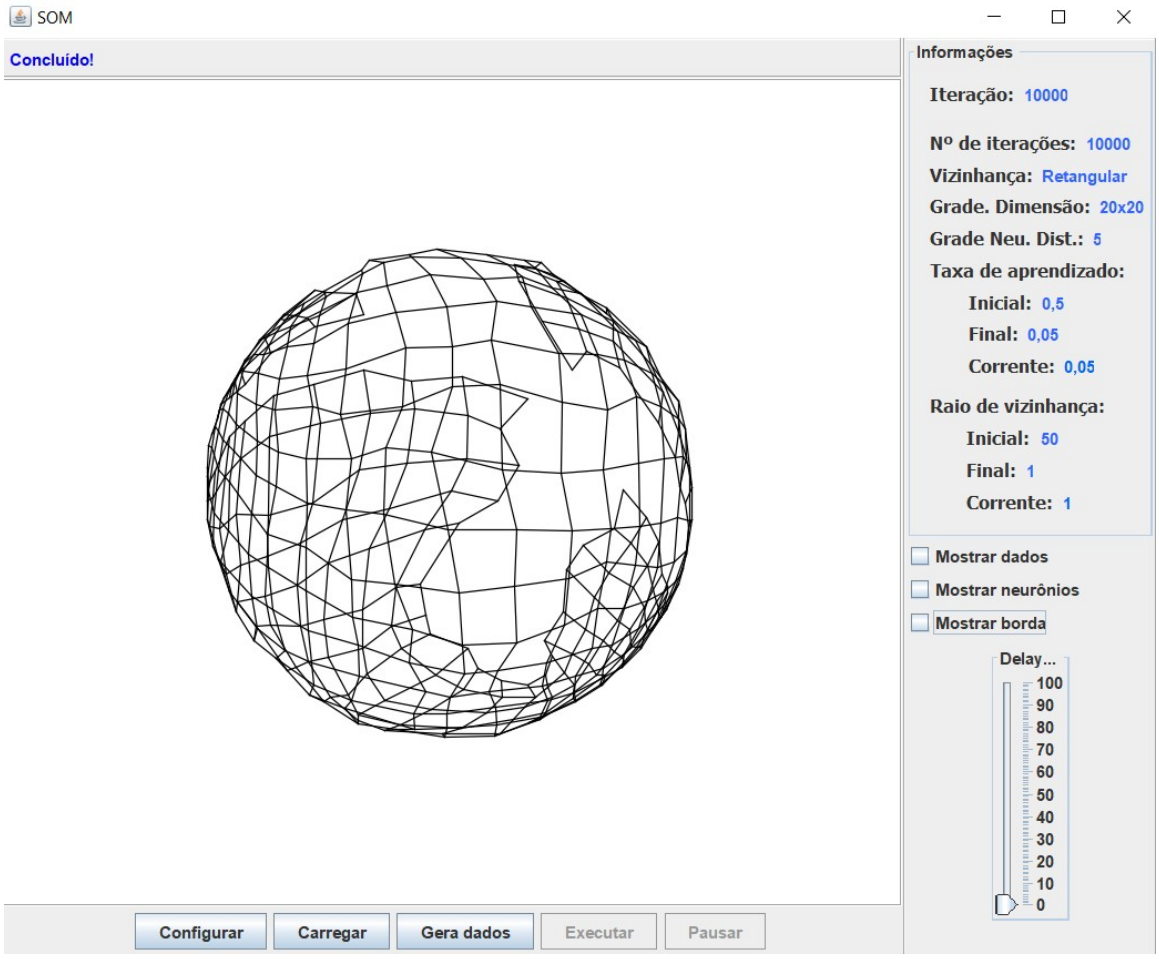


Figura 20: Resultado da organização do mapa

Perceba que após a execução do algoritmo, a rede tomou a forma de uma esfera, já que os dados de entrada representam dados aleatórios da superfície de uma esfera. Perceba também que os dados de entrada e neurônios foram omitidos para visualização apenas da rede resultante do treinamento.

6 Considerações finais

Com esse trabalho e os experimentos realizados, foi constatado que os mapas auto-organizáveis de Kohonen são ótimas opções de aplicação para o agrupamento de dados onde não se conhece o número de agrupamentos presentes. Trata-se de um algoritmo relativamente simples se comparado com algoritmos de redes neurais artificiais como, por exemplo, o perceptron multicamadas.

Foi percebido também que os SOM necessitam de diferentes configurações para diferentes aplicações, o que torna o algoritmo, embora simples de implementar, complexo de configurar, em que, muitas vezes, para se chegar a configuração ideal, é necessário recorrer a tentativa e erro. Essa é uma desvantagem de utilização do SOM como algoritmo de agrupamento.

Entretanto, as vantagens do algoritmo SOM superam a desvantagem da complexidade na configuração. Por isso, é notável que o SOM é um dos algoritmos de mineração de dados mais utilizados por profissionais da área.

Outra característica percebida dos mapas auto-organizáveis é a facilidade de avaliação da tendência do treinamento. Isto é, com os SOM, é possível determinar durante ou depois da execução do algoritmo se a topologia da rede está sendo preservada e o erro de quantização está diminuindo conforme a execução avança. Com isso, se pode considerar que, há maior confiabilidade relacionada aos resultados obtidos com aplicação do algoritmo em que erros topológicos e de quantização resultam em valores adequados para um bom treinamento da rede.

De um modo geral, foi constatado com esse trabalho que o ***SOM*** é uma ótima escolha de algoritmo de mineração de dados para o agrupamento de grandes massas de dados provenientes de um banco de dados de alguma empresa. Foi percebido também que ele pode, inclusive, ser utilizado em conjunto com o ***K-Médias***.

REFERÊNCIAS

Agrawal (2013) – **Machine Learning – Mini Batch K-Means**. 26 de Julho. Acessado em: 01/07/2019. Disponível em: https://algorithmicthoughts.wordpress.com/2013/07/26/machine-learning-mini-batch-k-means/

Abernethy (2010) – Michael Abernethy – **Mineração de dados com WEKA, introdução a regressão**. Acessado em: 29/06/2019. Disponível em: https://www.ibm.com/developerworks/br/opensource/library/os-weka1/index.html

Ahn e Syn (2005) – Jae W. Ahn, Sue Y. Syn – **Self Organizing Map Tutorial System**. 27 de Abril. Acessado em 29/06/2019. Disponível em: http://www.pitt.edu/~is2470pb/Spring05/FinalProjects/Group1a/tutorial/som.html

DEEPLEARNINGBOOK (2019) - **Capítulo 4 - O neurônio, biológico e matemático.** Disponível em: https://deeplearningbook.com.br/o-neuronio-biologico-e-matematico/. Acessado em 18/06/2019. 2019.

EMBARCADOS (2019) – **Rede perceptron de uma única camada.** Acessado em: 27/06/2019. Disponível em: https://www.embarcados.com.br/rede-perceptron-de-uma-unica-camada/.

Fayyad et al. (1996) - Fayyad, U. M.; Piatesky-Shapiro, G.; Smyth, P – "F**rom Data Mining to Knowledge Discovery: An Overview**". In: **Advances in Knowledge Discovery and Data Mining**, AAAI Press.

Figueiredo (2007) – Aurélio M. Figueiredo – **Mapeamento Automático de Horizontes e Falhas em Dados Sísmicos em 3D baseado no Algoritmo de Gás Neural Evolutivo.** Dissertação de Mestrado. Programa de Pós-Graduação em Informática da PUC-Rio. Rio de Janeiro, 29 de Junho.

Germano (1999) – Tom Germano – **Self Organizing Maps**. 23 de Março. Disponível em: davis.wpi.edu/~matt/courses/soms/. Acessado em: 29/06/2019.

GNU (2019) – **GNU Octave**. Acessado em: 06/07/2019. Disponível em: https://www.gnu.org/software/octave/

Holmström (2002) – Jim Holmström – **Growing Neural Gas: Experiments with GNG, GNG with Utility e Supervised GNG.** Uppsala University. Departament de Information Tecnology. Suécia, 03 de agosto.

Kohonen (2001) – Teuvo Kohonen – **Self Organizing Maps.** New York: Springer-Verlag, 3ª Extended Ed.

MATWORKS (2019) – **MatWorks**. Acessado em 06/07/2019. Disponível em: https://www.mathworks.com/products/matlab.htm.

MÉNDES, GUARDIA (2008) - Iasmín S. Méndes, Luiz E. T. Guardia - **PROBLEMA DO CAMINHO MAIS CURTO - ALGORITMO DE DIJKSTRA**. Universidade Federal Fluminence. Acessado em: 09/06/2019. Disponível em: https://www.marinha.mil.br/spolm/sites/www.marinha.mil.br.spolm/files/006_1.pdf.

MORAL (2008) – Emílio D. Moral – **Princípios da neurocomputação.** EPUSP. Acessado em: 11/06/2019. Disponível em: http://www.lsi.usp.br/~icone/psi5886/2008/PEE5886_2008_Aula08_09.pdf.

Oliveira, R. S et al, (2007) – **Uso de algoritmos de aprendizado competitivo em mineração de dados**. Acessado em 28/06/2019.

ORACLE (2019) – **ORACLE**. Acessado em 06/07/2019. Disponível em: http://www.oracle.com.

QUAZOO (2014) – **U-Matrix**. Disponível em: http://www.quazoo.com/q/u-matrix. Acessado em 26/06/2019.

SCIKIT-LEARN (2019) – **Clustering**. Acessado em: 01/07/2019. Disponível em: https://scikit-learn.org/stable/modules/clustering.html#k-means.

Silva (2004) – Marcelino P. S. Silva – **Mineração de dados – Conceitos, aplicações e Experimentos com WEKA**. Universidade do estado do Rio Grande do Norte (UFRN), Instituto Nacional de Pesquisas Espaciais (INPE).

SOUZA (2018) – Miguel A. A. Souza – **Metodologias para desenvolvimento de mapas auto-organizáveis de Kohonen executados em FPGA**. Universidade de São Paulo, São Paulo, 2018.

STACKOVERFLOW (2019) – **How do make U-matriz**. Acessado em: 09/07/2019. Disponível em: https://stackoverflow.com/questions/13631673/how-do-i-make-a-u-matrix.

TECHOPEDIA, Inc. (2019) – **Self Organizing Map (SOM)**. Acessado em: 29/06/2019. Disponível em: https://www.techopedia.com/definition/32068/self-organizing-map-som

UFOP (2017) – **Introdução aos self organizing maps.** Acessado em 28/06/2019. Disponível em: http://www.decom.ufop.br/imobilis/self-organizing-maps/.

Waikato (2019) – **Weka 3: Machine Learning Software in Java**. Universidade de Waikato. Nova Zelândia. Disponível em: https://www.cs.waikato.ac.nz/ml/weka/index.html. Acessado em: 28/06/2019.

WIKIPEDIA (2019) – **Regressão linear**. Acessado em: 29/06/2019. Disponível em: https://pt.wikipedia.org/wiki/Regressão_linear.

WIKIPEDIA (2018) – **Método dos mínimos quadrados**. Acessado em: 29/06/2019. Disponível em: https://pt.wikipedia.org/wiki/Método_dos_mínimos_quadrados

WIKIPEDIA (2016) – **Distância euclidiana.** Acessado em 28/06/2019. Disponível em: https://pt.wikipedia.org/wiki/Distância_euclidiana.