IA - Trabalho Prático 5 - Neural Networks - Identificação da Vinícula

- Aluno: Quenio Cesar Machado dos Santos
- Matrícula: 14100868
- Semestre: 2015-2

Introdução

O reconhecimento de padrões através de redes neurais é demonstrado neste trabalho. Para tanto, usaremos dados de análise química de vinhos coletados de três vinhedos. O objetivo é descobrir, dado sua análise química, de qual vinhedo pertence uma amostra de vinho.

O `MatLab`, com seu o `toolbox` de `Neural Networks`, foi a ferramenta utilizada para o tratamento dos dados, treinamento da rede neural e verificação do seu desempenho.

Os dados foram fornecidos em duas matrizes carregáveis no `MatLab`:

- `x`: que contém os dados da análise química dos vinhos, onde cada linha representa o componente químico analisado e cada coluna representa uma amostra de vinho.
- 't': contendo a informação de qual vinhedo pertence a amostra de vinho, onde cada linha representa um vinhedo e cada coluna representa a amostra de vinho correspondendo à coluna de mesmo número na matrix `x`.

Normalização dos Dados de Entrada

Antes de definir e treinar a rede neural, é preciso normalizar os dados das análises químicas no intervalo [-1, 1]. A normalização permite o melhor desempenho da função de treinamento da rede quando se usa a função de transferência `tangente sigmoidal`, que tem maior variação no intervalo [-1, 1].

Para a normalização, utilizamos a seguinte função do `Neural Net Toolbox`:

```
xn = mapminmax(x);
```

`mapminmax` recebe como entra a matriz de dados original e retorna uma matriz de mesma dimensão com os dados normalizados no intervalo [-1, 1].

Esta normalização foi feita nos scripts `quick_net.m` e `verified_net.m` que serão usados nas próximas seções.

Saturação dos Dados de Saída

Na matriz `t`, que representada a saída esperada, tem-se o valor `1` para o vinhedo que é a origem de uma amostra de vinho e o valor `0` para os demais vinhedos. No treinamento da rede neural, a função de treinamento vai ajustar a rede para que esta se aproxime dos valores encontrados em `t`. Como é apenas uma aproximação, precisamos de uma função que converta os valores de saída em zeros e uns. Isto nos

permitirá comparar as saídas da rede com os valores em `t` a fim the verificar o desempenho da rede. Este processo é chamado de "saturação".

A função de saturação é mostrada abaixo:

```
function s = saturate(output)
    row_size = size(output, 1);
    col_size = size(output, 2);
    s = zeros(row_size, col_size);
    for i = 1:col_size
        v = output(:, i);
        if v(1) > v(2) && v(1) > v(3)
            s(1, i) = 1;
        elseif v(2) > v(1) && v(2) > v(3)
            s(2, i) = 1;
        else
            s(3, i) = 1;
        end
    end
end
```

Observe no código acima que criamos uma nova matriz de mesma dimensão que a matriz de saída (`t`). Inicialmente, esta matriz tem zeros em todas as suas célculas. Depois o valor `1` é colocado na célula da linha que contém o maior valor em `t`, ou seja, o valor mais perto de `1`.

Esta função é utiliza pelos scripts `quick_net.m` e `verified_net.m` - usados nas próximas seções - para "saturar" a saída da rede e permitir a comparação com `t`.

Protótipação Rápida

A fim de nos familiarizarmos com as funções da `Neural Net Toolbox`, e com o processo de treinamento e verificação, implementamos um script chamado `quick_net.m` que cria, treina e valida uma rede usando o mesmo conjunto de dados.

Veja o código abaixo:

```
function quick_net(net_size, x, t)
    xn = mapminmax(x);

    net = newff(xn, t, net_size, {'tansig','tansig'}, 'trainlm');
    net.trainParam.epochs = 1000;
    net.trainParam.goal = 0;

    net = train(net,xn,t);
    y = saturate(sim(net, xn));
    plotconfusion(t, y);
end
```

Na função `quick_net` listada acima, observe o seguinte:

- Além do conjunto de dados de entrada `x` e do conjunto alvo `t`, a função também recebe como entrada o número de neurônios da camada intermediária.
- Primeiramente, a função normaliza os dados usando `mapminmax`.
- Logo após, a rede é criada usando a função de transferência `tangente sigmoidal` tanto para os neurônios da camada intermediária, quanto para os neurônios da camada de saída. Fizemos assim a

rede ser compatível com os dados normalizados do conjunto de entrada.
- O treinamento terá a seguinte configuração:
  - 'trainlm': Levenberg-Marquardt é usado no treinamento para que se alcance mínimo do gradiente no menor número de épocas possível.
  - `epochs = 1000`: Mil épocas é mais que suficiente para Levenberg-Marquardt.
  - `goal = 0`: Com erro zero desejamos atingir o mínimo da função, ao invés de apenas uma aproximação.
  - a proporcão de treinamento, validação e teste: como não foram especifados os conjuntos de treinamento, validação e de teste, estes serão aleatóriamente construídos numa proporção de 60%, 20% e 20%, respectivamente.
- Uma vez treinada a rede, a função `quick_net`, executa a rede sobre os mesmos dados de treinamento e compara o resultado com os dados originais num gráfico. Esta não é a melhor de verificar a rede, como veremos nas próximas seções, mas é suficiente para nossa prototipação rápida.

Para fazer experimentos com o script acima, criou-se redes de 3 a 7 neurônios intermediários sobre todas as treze análises químicas fornecidas no conjunto de dados original. Após várias execuções, o resultado da melhor performance de cada rede foi colocado na pasta `testes` em arquivos com o prefixo `quick_13i_`.

Observamos nos experimentos que foi possível gerar redes que alcançam 100% de acertos em todas as configurações - de 3 a 7 neurônios - como demonstrado nos arquivos da pasta `testes`. Porém, nem todas a redes geradas atingiram este nível de acerto. A taxa de acerto variava entre 95% a 100%.

Como já dissemos antes, não é recomendado usar o mesmo conjunto de treinamento para os testes. A próxima seção cuidará desta questão.

Conjuntos de Treinamento e de Teste

Usando o mesmo conjunto para treinamento e para a verificação da rede neural, como foi feito na seção anterior de prototipação, não vai mostrar como a rede se comporta com novas amostras.

Portanto, antes de definir a rede neural que irá reconhecer a origem das amostras de vinho, é preciso separar os dados em dois conjuntos distintos:

- *conjunto de treinamento*: que contém os dados que fazem o treinamento da rede neural;
- *conjunto de teste*: contendo os dados que verificam o desempenho da rede.

Também é preciso definir o tamanho da fatia do conjunto original de dados que será usado para treinamento e o tamanho da fatia que será usada para testes. Para nossos experimentos, usamos as seguintes proporções:

- *dois terços para treinamento*: a maior parte das amostras de vinho foi usada para treinamento, pois uma rede bem treinada deve ter um melhor desempenho na determinação da origem do vinho.
- *um terço para testes*: a menor parte dos dados foi usada para testes, pois é possível verificar o desempenho da rede com um número menor de amostras, desde que sejam representativas da população de amostras de vinho.

Como os dados de amostra foram ordenados por vinhedo, usou-se o código seguinte para distribuir igualmente os vinhedos entre os conjuntos de treinamento e de teste:

```
x_train = [xn(:,1:3:end), xn(:,3:3:end)];
```

```
x_test = xn(:,2:3:end);

t_train = [t(:,1:3:end), t(:,3:3:end)];
t_test = t(:,2:3:end);
```

Observe que no código acima intercalamos entre as colunas das matrizes para conseguir uma variedade entre os conjuntos na proporção desejada. O conjunto de treinamento (`x_train`) recebeu duas vezes mais columas (ou amostras) do que o conjunto de teste (`x_test`). Da mesma forma ocorreu com os conjuntos de saída usados para treinamento e teste.

Arquitetura da Rede Neural

No script `verified_net.m`, nós criamos os conjuntos de treinamneto e de teste - como mostrado na seção anterior - e logo após criamos a rede neural a ser verificada.

Veja o script abaixo:

```
function verified_net(net_size, x, t)
    xn = mapminmax(x);

    x_train = [xn(:,1:3:end),xn(:,3:3:end)];
    x_test = xn(:,2:3:end);

    t_train = [t(:,1:3:end), t(:,3:3:end)];
    t_test = t(:,2:3:end);

    net = newff(x_train, t_train, net_size, {'tansig','tansig'}, 'traingda');

    net.trainParam.epochs = 1500;
    net.trainParam.goal = 0;

    net.divideParam.trainRatio = 1.0;
    net.divideParam.valRatio = 0;
    net.divideParam.testRatio = 0;

    net = train(net, x_train, t_train);
    y = saturate(sim(net, x_test));
    plotconfusion(t_test, y);
end
```

Ao criar a rede neural, estabelecemos a sua arquitetura desta forma:

- *rede feed-forward*: a rede escolhida é do tipo "feed-forward" com "back propagation". Este tipo de rede utiliza o produto do vetor de entrada com o vetor dos pesos para calcular a saída de cada *perceptron*, juntamente com a função de transferência.
- *tansig*: a função de transferência usada nas camadas intermediárias e de saída é "tangente sigmoidal", que é compatível com os dados de entrada normalizados.
- *traingda*: aqui - diferentemente da rede protótipo das seções anteriores - foi escolhido a função de treinamento "gradiente descendente" apenas para verificar a necessidade de mais épocas para achar o mínimo da função.
- Nesta rede não haverá validação ou testes durante o treinamento. Apenas a verificação do erro.

Uma vez treinada a rede usando somente o conjunto de treinamento, a rede é executada para o conjunto de teste. A comparação é feita então entre a saída já saturada e a matriz de saída original `t`.

Na próxima seção, rodamos vários experimentos para verificar o desempenho desta rede.

Verificação da Rede Neural

Assim como foi feito com o protótipo, verificamos o desempenho da rede neural definida na seção anterior com o número de neurônios da camada intermediária variando de 3 a 7. Os testes foram feitos sobre todas as treze análises químicas fornecidas no conjunto de dados original, porém com a divisão do conjunto de treinamento e de teste, como implementado em `verified_net.m`. Após várias execuções, o resultado do melhor desempenho de cada rede foi colocado na pasta `testes` em arquivos com o prefixo `verified_13i_`.

Para as configurações de 4, 5 e 6 neurônios, foi possível gerar redes que alcançaram 100% de acertos quando testadas sobre os dados de treinamento. Já, nos testes com 3 e 7 neurônios, conseguimos gerar redes com no máximo 98% de acerto. Também observamos que as redes com 5 neurônios resultando em 100% de acerto foram geradas com mais frquência do que na configuração de 4 e 6 neurônios. O que nos faz pensar que 5 neurônios parece ser o número ideal para este conjundo de dados e esta arquitetura de rede.

Tentatiza de Otimização dos Parâmetros de Entrada

Outro experimento que realizamos foi a tentativa de diminuição do número de análises químicas necessárias para a determinação da origem do vinho. A intenção aqui seria diminuir o custo de uso da rede neural, visto que cada análise química adicional aumenta o custo. A questão então foi selecionar quais análises químicas seriam suficientes para a seleção confiável da origem das amostras de vinho.

Para avaliar as análises químicas, implementamos o script `data_view_matrix.m`, que imprime o gráfico do arquivo `data_view_matrix.pdf`. Este desenha na verdade vários gráficos, cada um combinando duas análises químicas e mostrando os pontos de cada vinhedo em cores diferentes. O resultado é uma matriz de gráficos mostrando todas as combinações possíveis entre as análises químicas. Além disso, na diagonal principal da matriz, foi impresso o histograma de cada análise quimica, com a distribuição de vinhedos.

Veja abaixo o script que gera os gráficos:

```
function data_view_matrix(data, targets, var_names)
    column_size = size(data, 2);
    group = zeros(1, column_size);
    for i = 1:column_size
        group(1, i) = group_no(targets, i);
    end
    gplotmatrix(transpose(data), [], transpose(group), ['c','b','m'], [], [], false);
    var_size = size(var_names, 1);
    text(((1:1:var_size)/var_size)-0.10,...
         repmat(-.05,1,var_size),...
         var_names, 'FontSize',8);
    text(repmat(-.04,1,var_size),...
         ((var_size:-1:1)/var_size)-0.10,...
         var_names, 'FontSize',8, 'Rotation',90);
end

function result = group_no(targets, column)
    if targets(1,column) == 1
        result = 1;
    elseif targets(2,column) == 1
```

```
        result = 2;
    else
        result = 3;
    end
end
```

Visto que o conjunto de dados tem várias dimensões, estes gráficos permitem uma visualização mais global da qualidade dos dados do que seria possível num único gráfico 3D com apenas três análises químicas. Após analisar os gráficos, selecionamos oito análises químicas que ofereciam a melhor diferenciação entre as origens dos vinhos, ou seja, aqueles gráficos que mostravam as menores interseções entre pelo menos dois vinhedos.

As análises químicas selecionadas foram as seguintes:

1. Alcohol
2. Malic Acid
3. Total Phenols
4. Flavanoids
5. Color Intensity
6. Hue
7. OD280/OD315
8. Proline

Os gráficos com a combinação das análises químicas selecionadas estão em `data_view_matrix_selected.pdf`.

Após executar `verified_net.m` sobre os dados selecionados, para nossa supresa, verificamos um desempenho inferior daquele que vimos sobre o conjunto de dados com as treze análises. Esperávamos que o desempenho seria tão bom, ou ainda melhor, porque removemos aquelas análises químicas que não diferenciavam os vinhedos. Porém, nunca conseguimos acerto de 100% com as oito análises químicas selecionadas. Os resultados estão nos arquivos com o prefixo `verified_8i_`.

Já, quando executamos `quick_net.m`, foi possível alcançar acertos de 100%. Neste caso, porém os dados de teste eram os mesmos dados de treinamento, o que faz do resultado pouco expressivo. Os resultados estão nos arquivos com prefixo `quick_8i_`.

Conclusão

Este trabalho nos permitiu aprender a usar o `MatLab` e sua `Neural Net Toolbox` para criar e testar redes neurais que podem ser aplicadas a áreas diversas. Também nos mostrou que existe todo um processo a ser seguido para que as redes criadas possam ser verificadas e posteriormente utilizadas.

Além disso, através da tentativa de otimização, percebemos que o resultado dos experimentos nem sempre são os esperados. Que é preciso iterar sobre os resultados e fazer mais experimentos até que se possa chegar no resultado desejado. Por exemplo, no caso da otimização, um próximo passo seria tentar identificar - talvez através de uma visualização diferente dos dados - porque as análises quimicas adicionais - que pareciam não diferenciar as amostras - puderam na prática melhorar o desempenho. Baseado nestas conclusões, talvez seria possível selecionar um novo sub-conjunto das análises químicas que teria um desempenho melhor.

Outra questão que seria interessante explorar após este trabalho, seria como transferir a rede neural

desenvolvida no `MatLab` para um software desenvolvido nas plataformas convencionais, como o Java, a fim de incorporá-lo numa aplicação que pudesse benefeciar os usuários finais.

Certamente, esta técnica, e tantas outras, mostram o potencial da área de Inteligência Artificial como diferencial no desenvolvimento de aplicações.