28/11/2019: Trabalho 3 de inteligência computacional

Este repositório hospeda os arquivos necessários à execução do trabalho 3 realizado durante os estudos da disciplina de inteligência computacional

Universidade Federal do Ceará

Inteligência Computacional

Professor: Jarbas Joaci de Mesquita Sá Júnior

Construído em:

- Scilab 6.0.2 - Software open source para computação numérica
- Linux Mint 19.2 Tina - Sistema operacional usado

Autor:

- José Lopes de S. Filho - LinkedIn
- *Engenharia da Computação (UFC) - Matrícula # 389097*

Licença:

Questão 1

1. Implemente uma rede neural RBF para traçar uma superfície de decisão nas amostras do conjunto de dados twomoons.dat.

Obs.: na primeira coluna da base de dados constam as medidas da variável x1 e na segunda coluna as medidas da variável x2. A classe de cada vetor de medidas (x1, x2) é dada na terceira coluna.

Iniciando

Para a resolução desta questão e criação deste relatório foram usados os seguintes arquivos:

- Q1_RBF.sce - Código-fonte da aplicação proposta na questão
- twomoons.dat - Conjunto de dados da questão

- Q1_RBF_img1.png - Gráfico dos dados de treinamento twomoons clusterizado pelo k-means para encontrar as centroides
- Q1_RBF_img2.png - Retorno do console ao executar o código

Código comentado

```
// TERCEIRO TRABALHO DE INTELIGÊNCIA COMPUTACIONAL
// Questão 1
// Aluno: José Lopes de Souza Filho
// Matrícula: 389097
// Aplicação: Scilab, versão 6.0.2
// SO: Linux Mint 19.2 Tina
//-------------------------------------------------------------------------

clear;
clc;
clf;
base = fscanfMat('twomoons.dat');
```

PARTE 1: Cálculo das centróides usando K-means clustering

```
//Separa 90% dos dados para treino e 10% dos dados para teste
//Base de treino - 450 amostras da primeira metade da base e 450 da última
base_treino = base(1:450,:);
base_treino(451:900,:) = base(500:949,:);
//Base de teste - 50 amostras da primeira metade da base e 51 da última
base_teste = base(451:499,:);
base_teste(50:101,:) = base(950:1001,:);

//Define as entradas da rede (X) e a saída esperada (D)
X = base_treino(:,1:2);    //Entradas da rede
D = base_treino(:,3);      //Saídas da rede

//Implementa k-means
//cria duas centroides aleatorias - primeira coluna eixo X e segunda coluna eixo Y
//cada linha uma centroide
centroides = rand(2, 2) .* (max(X) - min(X)) + min(X);

//Calcula a distancia de todos os pontos para a centroide 1
distancias(:,1) = sqrt(((centroides(1,1)-X(:,1))^2) + ((centroides(1,2)-X(:,2))^2));
//Calcula a distancia de todos os pontos para a centroide 2
distancias(:,2) = sqrt(((centroides(2,1)-X(:,1))^2) + ((centroides(2,2)-X(:,2))^2));
//Testa qual a menor distância e classifica o ponto como cluster 1 ou 2.
X_classif = X; //copia a matriz de entrada para uma nova que terá a classificação
for i=1:900
    [min_valor, min_linha] = min(distancias(i,:));
    X_classif(i,3) = min_linha;
end
//Entra em loop de classificação até que as centroides não se movam mais
```

```
//calcula novas posições para as centroides
somax1=0;
somay1=0;
somax2=0;
somay2=0;

centroides_atuais = centroides;
centroides_anteriores = [0,0,0,0];
centroides_temporarias = [0,0,0,0];

while centroides_atuais <> centroides_anteriores,
//clf;
for i=1:900

    if X_classif(i,3) == 1 then
        somax1 = (somax1+X_classif(i,1));
        somay1 = (somay1+X_classif(i,2));
    elseif X_classif(i,3) == 2 then
        somax2 = (somax2+X_classif(i,1));
        somay2 = (somay2+X_classif(i,2));
    end
end
mediax1 = somax1/900;
mediay1 = somay1/900;
mediax2 = somax2/900;
mediay2 = somay2/900;

somax1 = 0;
somay1 = 0;
somax2 = 0;
somay2 = 0;

centroides_temporarias = [mediax1, mediay1; mediax2, mediay2];
centroides_anteriores = centroides_atuais;
centroides_atuais = centroides_temporarias;

end

//Cria duas matrizes classificadas e plota o gráfico classificado pelo k-means
k=1;
j=1;

for i=1:900

    if X_classif(i,3) == 1 then
        X_1(k,:) = X_classif(i,:);
        k = k+1
    elseif X_classif(i,3) == 2 then
        X_2(j,:) = X_classif(i,:);
        j = j+1
    end
end

cluster1x = X_1(:,1);
```

```
cluster1y = X_1(:,2);
scatter(cluster1x,cluster1y,26,"scilabred3","fill", ".");

cluster2x = X_2(:,1);
cluster2y = X_2(:,2);
scatter(cluster2x,cluster2y,26,"scilabgreen3","fill", ".");
xtitle("Gráfico twomoons.dat clusterizado pelo k-means. Cada cor representa um cluster
```

PARTE 2: Implementação da rede RBF

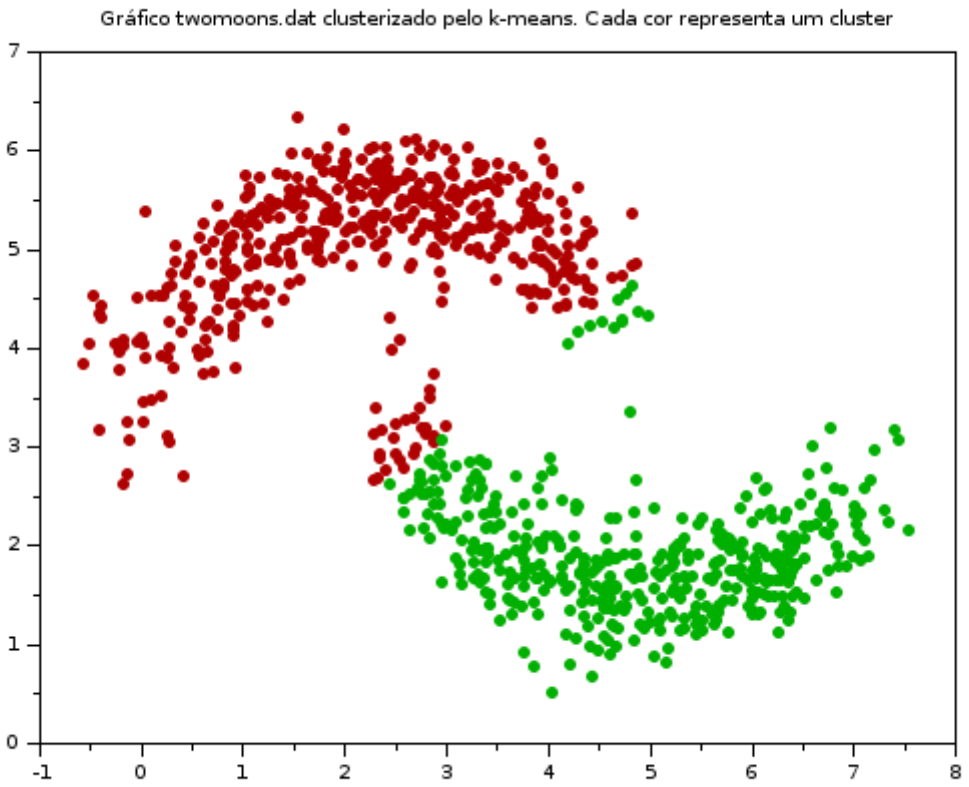
```
//Par de RBF de saída do neurônio
for i=1:900
    G(i,1) = 1;
    if X_classif(i,3) == 1 then
        G(i,2) = exp(-(sqrt(((X_classif(i,1)-centroides(1,1))^2) + ((X_classif(i,2)-ce
    elseif X_classif(i,3) == 2 then
        G(i,3) = exp(-(sqrt(((X_classif(i,1)-centroides(2,1))^2) + ((X_classif(i,2)-ce
    end
end

// Matriz de pesos W

W = [(((G' * G) \ G') * D(:, 1))'];

// Calcula a saída da rede (d)

d = G * W';
```

PARTE 3: Mostra as saídas da rede no console

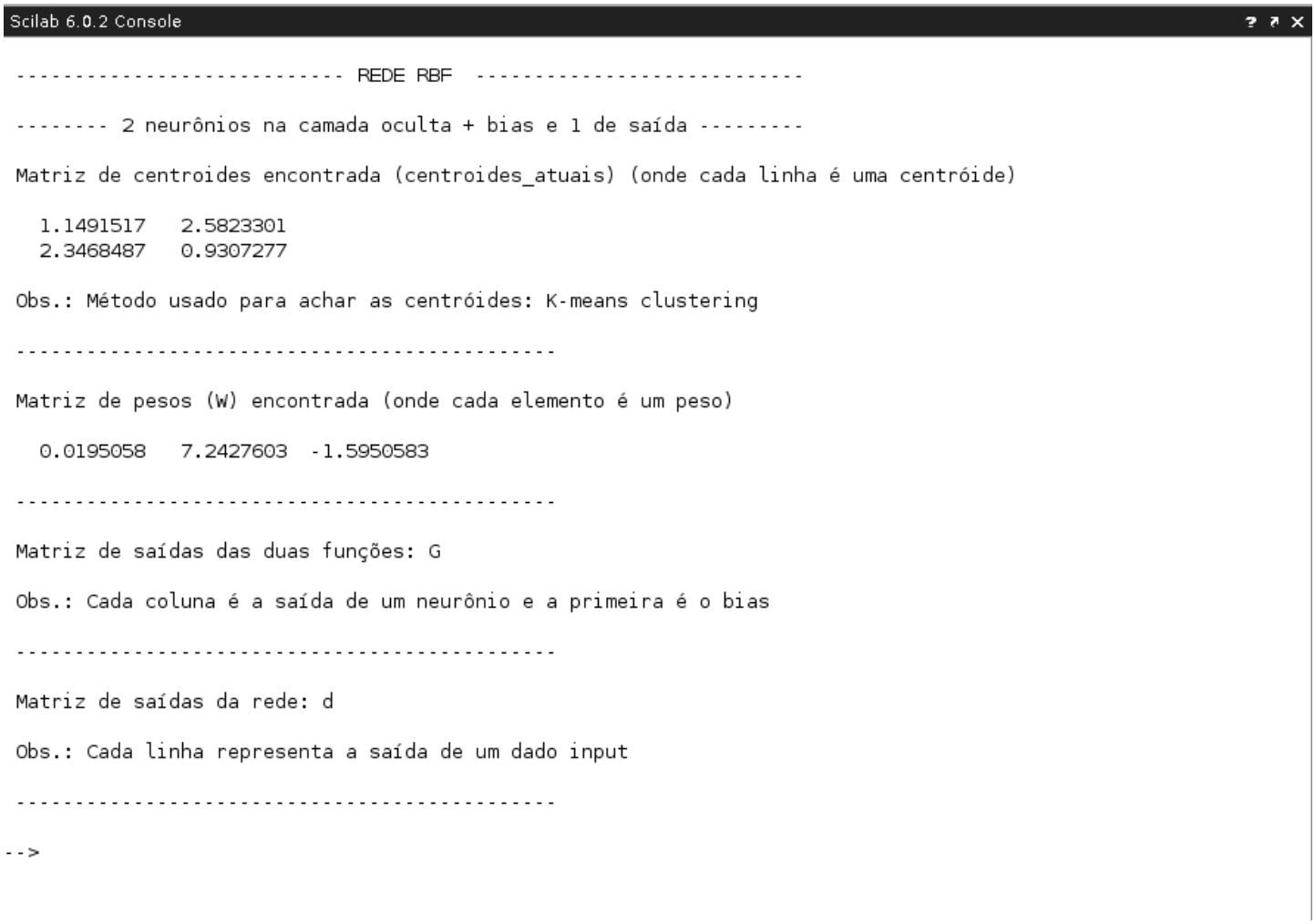
```
disp("---------------------------- REDE RBF  -----------------------------");
disp("-------- 2 neurônios na camada oculta + bias e 1 de saída ---------")
disp("Matriz de centroides encontrada (centroides_atuais) (onde cada linha é uma centr
disp(centroides_atuais);
disp("Obs.: Método usado para achar as centróides: K-means clustering");
disp("-----------------------------------------------")
disp("Matriz de pesos (W) encontrada (onde cada elemento é um peso)");
disp(W);
disp("-----------------------------------------------")
disp("Matriz de saídas das duas funções: G");
disp("Obs.: Cada coluna é a saída de um neurônio e a primeira é o bias");
disp("-----------------------------------------------")
disp("Matriz de saídas da rede: d");
disp("Obs.: Cada linha representa a saída de um dado input");
disp("-----------------------------------------------")
```

Discussão dos resultados obtidos

Ao executar o arquivo Q1_RBF.sce no Scilab, será retornado no console algumas informações da rede assim como uma janela gráfica com os dados clusterizados pelo algoritmo k-means durante a criação das centróides. Cada cluster identificado está de uma cor diferente.



Janela gráfica exibindo os dados de treino da base twomoons.dat



Retorno do console ao executar o código

No console, será retornado a matriz de centróides encontrada no problema, a matriz de pesos W e foram criadas mais duas matrizes. A matriz *G* retorna a saída das funções de base radial, sendo cada coluna referente a saída de um neurônio (para ver a matriz basta digitar (G) no console e a matriz *d* que contém as saídas da rede. Ela possui apenas uma coluna (1 neurônio de saída). Para visualizá-la basta digitar (d) no console. Observação.: Os valores e comportamento do gráfico podem ser diferentes a cada execução em função do caráter aleatório inicial de algumas etapas.

Questão 2

2. Crie um algoritmo genético para achar o máximo da função f(x,y) = |xsen(yπ/4) + ysen(xπ/4)|. Cada indivíduo da população é um vetor binário de 20 bits, em que os 10 primeiros representam x e os restantes representam y. As variáveis x e y pertencem ao intervalo entre 0 e 20. O crossover a ser usado é de 1 ponto.

Iniciando

Para a resolução desta questão e criação deste relatório foram usados os seguintes arquivos:

- Q2_GA.sce - Código-fonte do algoritmo genético
- Q2_GA_img1.png - Retorno do console ao executar o código

Código comentado

PARTE 1: Criação da população de cromossomos

Para esta etapa foram criados uma população de 100 cromossomos de 20 bits. 10 bits para as 10 primeira posiçãoes e 10 para as últimas.

```
// TERCEIRO TRABALHO DE INTELIGÊNCIA COMPUTACIONAL
// Questão 2
// Aluno: José Lopes de Souza Filho
// Matrícula: 389097
// Aplicação: Scilab, versão 6.0.2
// SO: Linux Mint 19.2 Tina
//-------------------------------------------------------------------------

clear;
clc;
p = grand(100,20, "uin", 0, 1); //gera uma matriz binaria 100x20
```

PARTE 2: Atribui notas aos cromossomos de acordo com a função de ativação

```
ant = 0;
passo = 20/1023; // valor correspondente a um bit
for q = 1:1000 //quantidade de gerações
    nota = [];

    for i = 1:100
        x=0;
        y=0;
        //conversão da representação binária para real
        for j = 1:10
            if p(i,j)==1
                x = x +(2^(10-j)*passo);
            end
            if p(i,j+10)==1
                y = y +(2^(10-j)*passo);
            end
        end
        a = abs(x*sin(y*%pi/4)+y*sin(x*%pi/4));
        if a > ant
            xm = x;
            ym = y;
        end
        if ant < a
            ant = a;
        end
        nota = [nota; a];

    end
```

PARTE 3: Seleção dos pais e crossover

```
    p = [p nota];

    p = gsort(p,'r');

    pais = p(1:50,1:20);
    p = [];
    for i = 1:2:100
       pai1 = grand(1,1,"uin",1,50);
       pai2 = grand(1,1,"uin",1,50);
       xcross = grand(1,1,"uin",1,20);
       p(i,:) = [pais(pai1,1:xcross) pais(pai2,xcross+1:20)];
       p(i+1,:) = [pais(pai2,1:xcross) pais(pai1,xcross+1:20)];
    end
end
```

PARTE 4: Exibição dos resultados no console

```
disp(ant);
disp("valor de x e y é: ");
disp(xm);
disp(" e ");
disp(ym);
```

Discussão dos resultados obtidos

Ao executar o arquivo Q2_GA.sce no Scilab, podemos verificar que o código tem inicio definindo a população no caso uma matriz 100 por 20, preenchida apenas com ‘zeros’ e ‘uns’ aleatoriamente, representando uma população de 100 indivíduos. Próximo passo foi converter os dados binários para números reais para aplicar na função e atribuir as notas de cada indivíduo. Em seguida a população é ordenada de acordo com sua nota em ordem decrescente, os 50 melhores indivíduos são selecionados para ser os pais dos próximos indivíduos. Em seguida são gerados dois índices aleatórios representando dois pais, estes indivíduos são combinados gerando dois filhos, esse processo é repetido até definir a nova população. O algoritmo repete estes passos 1000 vezes e no fim mostra: a maior nota e as coordenadas ‘x’ e ‘y’ correspondentes.

- Retorno do console ao executar o algoritmo