UNIVERSIDADE
FEDERAL DO CEARÁ

ENGENHARIA DA COMPUTAÇÃO

INTELIGÊNCIA COMPUTACIONAL

STEFANE ADNA DOS SANTOS - 403249

16 DE OUTUBRO DE 2020
SOBRAL - CE

1. Algoritmos Genéticos

Um algoritmo genético é uma técnica derivada dos algoritmos evolutivos que é amplamente utilizada na ciência da computação. Ela se inspira na biologia evolutiva para encontrar soluções para alguns problemas, para isso, são utilizadas técnicas de hereditariedade, mutação, recombinação e seleção natural.

Em seu funcionamento, esse tipo de algoritmo tenta imitar a evolução dos seres vivos, criando inicialmente uma população de indivíduos aleatórios e depois combinações e mutações entre esses indivíduos, realizando esse processo por várias gerações. A Figura 01 exibe o esquema para o funcionamento de um algoritmo genético.

Figura 01: Funcionamento do algoritmo genético

A etapa de Inicialização da População é realizada gerando números binários de 20 bits de forma aleatória, onde cada número corresponde a um indivíduo da população, para esse trabalho a população conterá 100 indivíduos, como pode ser visualizado na Figura 02.

Figura 02: Inicializando a população

```
//Gera indivíduos com valores aleatorios de c
function[populacao]=Gera_Ind(tam_pop)
    individuos = round(rand(tam_pop,1)*2^20);
    populacao = dec2bin(individuos,20);//conv
endfunction
```

Na etapa de Avaliação dos Indivíduos são dados valores para cada indivíduo. Para isso, é necessário separar o número binário de 20 bits em duas partes, sendo os 10 bits da esquerda referente ao x e os 10 bits da direita referente ao y. Esses bits separados devem ser convertidos para decimal e então convertidos para o intervalo de $x \in [-5, 5]$ e $y \in [-5, 5]$. Após isso, eles são enviados para uma função de avaliação dada pela Equação 01, no qual, deve avaliar a qualidade do indivíduo dentro da população.

$$\mathbf{f(x, y) = (1 - x)^2 + 100(y - x^2)^2} \quad (01)$$

A Figura 03 exibe o código da etapa de avaliação dos indivíduos. As funções "ConverteX" e "ConverteY" recebem um indivíduo e retornam seus valores convertidos em decimal e no intervalo de x ∈ [-5 5] e y ∈ [-5 5]. A função "funcao" recebe os valores de x e y e retorna a avaliação do indivíduo correspondente de acordo com a Equação 01.

Figura 03: Avaliação dos indivíduos

```
//Recebe os valores binarios de 20 bits e converte para os valor
function[x]=ConverteX(populacao)
    xaux = part(populacao,1:10); //Separa os 10 bits referentes
    //Convertendo binario para decimal
    num_real_x = bin2dec(xaux);
    //Converte os valores para o intervalo -5 e 5
    lim_sup = 5;
    lim_inf = -5;
    x = lim_inf+(lim_sup - lim_inf)/((2^10)-1) * num_real_x;
endfunction

function[y]=ConverteY(populacao)
    yaux = part(populacao,11:20); //Separa os 10 bits referentes
    //Convertendo binario para decimal
    num_real_y = bin2dec(yaux);
    //Converte os valores para o intervalo -5 e 5
    lim_sup = 5;
    lim_inf = -5;
    y = lim_inf+(lim_sup - lim_inf)/((2^10)-1) * num_real_y;
endfunction

//Função de avaliação
function[G]=funcao(x,y)
    G = (1-x)^2+100*(y - x^2)^2;
endfunction
```

É necessário selecionar os melhores indivíduos da população para serem pais dos indivíduos da próxima geração. Para isso foi utilizado o método do torneio que consiste em colocar os indivíduos para disputar entre si, o que possuir a melhor avaliação ganha a rodada. Como o tamanho da população é 100, logo deve-se encontrar 100 pais para a próxima geração. Com isso, deve-se realizar 100 partidas, onde em cada partida o indivíduo com maior avaliação vence e se tornará pai da próxima geração, os pais poderão ser repetidos diversas vezes.

Para isso, foi construída uma função "AvaliaPopu" que recebe um vetor de indivíduos e retorna a avaliação de cada indivíduo desse vetor. Outra função chamada "torneio" foi construída. Essa função contém um for que irá rodar 100 vezes, onde em cada rodada serão escolhidos e avaliados 6 indivíduos aleatórios da população e o que tiver melhor avaliação será o vencedor da rodada e se tornará pai da próxima geração. A função "torneio" retorna um vetor que contém os pais. A Figura 04 exibe a etapa de seleção dos indivíduos.

Figura 04: Seleção dos Pais

```
function[avaliacao_cromossomo]=AvaliaPopu(tam_pop,populacao)
    x = 0;
    y = 0;
    avaliacao_cromossomo = ones(tam_pop,1);
    for i=1:tam_pop
        x = ConverteX(populacao(i));
        y = ConverteY(populacao(i));
        avaliacao_cromossomo(i)=funcao(x,y);
    end
endfunction

//METODO DOS TORNEIOS
//Esse metodo retorna os pais dos individuos depois dele pas
function[pais]=torneio(tam_pop,populacao)
    concorrente = populacao(1:6);
    pais = populacao; //inicializando a variavel pais
    for i = 1:tam_pop
        for m = 1:6
            concorrente(m) = populacao(ceil(rand()*100));
        end
        avaliacao_cromossomo = AvaliaPopu(6,concorrente);
        [avaliacao posicao] = min(avaliacao_cromossomo);
        pais(i) = concorrente(posicao);
    end
endfunction
```

A quarta etapa consiste na geração e mutação dos indivíduos. Para gerar novos indivíduos deve ser feita a combinação entre os pais. Inicialmente, é escolhido um ponto de corte que consiste em um número aleatório entre 1 e 20. Após isso, os dois pais serão cortados em dois pedaços a partir da posição do ponto de corte. Então a primeira parte do pai1 será combinada com a segunda parte do pai2 e a primeira parte do pai2 será combinada com a segunda parte do pai1, gerando assim dois novos indivíduos a partir de dois pais.

A Figura 05 exibe o algoritmo da etapa de Cruzamento. A função “Cruzamento” recebe dois pais e realiza cruzamento, tendo como retorno dois filhos. A função “Gera_filhos” é responsável por fazer o cruzamento de todos os 100 pais e retornar um vetor onde cada posição contém um valor binário de 20 bits referente a um filho.

Figura 05:Cruzamento dos pais

```
function[filho1,filho2]=Cruzamento(pai1,pai2)
    pontocorte = int(rand()*19); //define um valor de 1 até 19 para
    //Separa o numero binario em duas partes de acordo com o ponto
    pai1_parte1 = part(pai1,1:pontocorte);
    pai1_parte2 = part(pai1,pontocorte+1:20);
    pai2_parte1 = part(pai2,1:pontocorte);
    pai2_parte2 = part(pai2,pontocorte+1:20);
    //gera o valor binario dos filhos de acordo com a junçao dos va
    filho1 = pai1_parte1 + pai2_parte2;
    filho2 = pai2_parte1 + pai1_parte2;
endfunction
//Função para gerar novos filhos a partir do cruzamento com os pais
function[filhos]=Gera_filhos(tam_pop,pais,populacao)
    filhos = populacao;//Inicia o vetor filhos
    for i=1:2:tam_pop
        [filhos(i) filhos(i+1)] = Cruzamento(pais(i),pais(i+1));
    end
endfunction
```

A mutação é feita de forma simples, segundo a aula ministrada pelo professor, um bom algoritmo conterá a probabilidade de 0.5% para realizar mutação em um bit, ele deve rodar em todos os 20 bits do indivíduo. Com isso, é gerado de forma aleatória um número de 0 a 100, se esse número contiver valor menor que 0.5 então o bit sofrerá mutação, ou seja, se seu valor for 1, então passará a ser 0 e se seu valor for 0, passará a ser 1.

A Figura 06 exibe o algoritmo da etapa de Mutação. A função “mutação” retorna um indivíduo com uma possível mutação em seus bits, e a função “Fazmutacao” retorna um vetor de tamanho 100, onde cada posição corresponde a um indivíduo que já sofreu mutação.

Na etapa de concepção da nova geração, todos os indivíduos da população são substituídos pelos filhos gerados e mutados, gerando assim uma nova geração da população. Este processo de encontrar os pais, realização de combinação e mutação, gerando novos indivíduos deve se repetir por inúmeras gerações. Para melhor exibição do funcionamento do algoritmo deste trabalho, foram utilizadas 100 gerações. A Figura 07 exibe a função do treinamento de cada geração e a plotagem do gráfico, ela retorna a população após o treinamento.

Figura 06: Etapa de Mutação

```
function[indmutado]=mutacao(individuo)
    prob = rand()*100; //prob vai receber um valor aleatorio d
    //Este for vai analizar cada bit separadamente, existira u
    //Se o valor da variavel 'prob' for menor do que 1.0 então
    //Na mutação se o bit tiver valor 0, passará a ter valor 1
    for j = 1:20
        bit = part(individuo,j);
        if prob < 0.5
            if bit == "1" then
                bit = "0";
            else
                bit = "1";
            end
        end

        if j == 1 then
            indmutado = bit;
        else
            indmutado = indmutado + bit;
        end
    end
endfunction
//Função para fazer a mutação dos cromossomos
function[filhos_mutados]=Fazmutacao(tam_pop,filhos,populacao)
    filhos_mutados = populacao;//Inicia o vetor filhos
    for i=1:tam_pop
        filhos_mutados(i) = mutacao(filhos(i));
    end
endfunction
```

Figura 07:Função de treinamento

```
//Função para fazer o treinamento do algoritmo
function[populacao,x,y]=treinamento(tam_pop,populacao)
for i = 1:quat_geracoes
    x = ones(tam_pop,1);
    y = ones(tam_pop,1);

    //Retorna os valores reais de x e y
    for j = 1:tam_pop
        x(j) = ConverteX(populacao(j));
        y(j) = ConverteY(populacao(j));
    end
    //retorna o valor da avaliação dos individuos
    avaliacao_cromossomo = AvaliaPopu(quat_geracoes,populacao);

    pais = torneio(tam_pop,populacao); //recebe os pais que fora
    filhos = Gera_filhos(tam_pop,pais,populacao); //gera os fil
    filhos_mutados = Fazmutacao(tam_pop,filhos,populacao); //fa
    populacao = filhos_mutados; //substitui a população pelos f

    //PLOTANDO OS INDIVIDUOS
    p=gca();
    r=p.rotation_angles;
    h=scatter3d(x,y,avaliacao_cromossomo,"fill");
    a=gca();
    f=gcf();
    f.figure_name='Geração:'+string(i);
    a.rotation_angles = r;
    sleep(100);
    if(~(i==100)) then
        delete(h);
    end
```

Por fim, foi realizada a plotagem dos gráficos, e os valores de x, y, mínimo da função de Rosenbrock foram exibidos no terminal. A Figura 08 exibe a última parte do código.

Figura 08: Parte final do código

```
//PLOTAGEM DO GRAFICO EM 3D
quat_pontos = 30;
xl=linspace(-5,5,quat_pontos);
yl=linspace(-5,5,quat_pontos);
zl=zeros(quat_pontos,quat_pontos)
for k=1:quat_pontos
    for m=1:quat_pontos
        zl(k,m)=funcao(xl(k),yl(m))
    end
end
plot3d(xl,yl,zl)
graf3d=gcf();
graf3d.color_map = hotcolormap(4);

//INICIA AS VARIAVEIS PARA INICIO DO ALGORITMO
tam_pop = 100;
quat_geracoes = 100;
populacao = Gera_Ind(tam_pop); //gera 100 individuos de uma
avaliacao_cromossomo = AvaliaPopu(quat_geracoes,populacao);
[populacao x y] = treinamento(tam_pop,populacao); //retorna

//Resultados
[avaliacao posicao] = min(avaliacao_cromossomo);
disp("O valor minimo encontrado:");
disp(avaliacao);
disp("A posição do individuo na população é:");
disp(posicao);
disp("O individuo é:");
disp(populacao(posicao));

disp("O valor de X é:" + string(x(posicao)));
disp("O valor de Y é:" + string(y(posicao)));
```

A Figura 09 exibe o gráfico após 3 gerações e a Figura 10 exibe o gráfico após 100 gerações.

Figura 09: Gráfico de 3 gerações

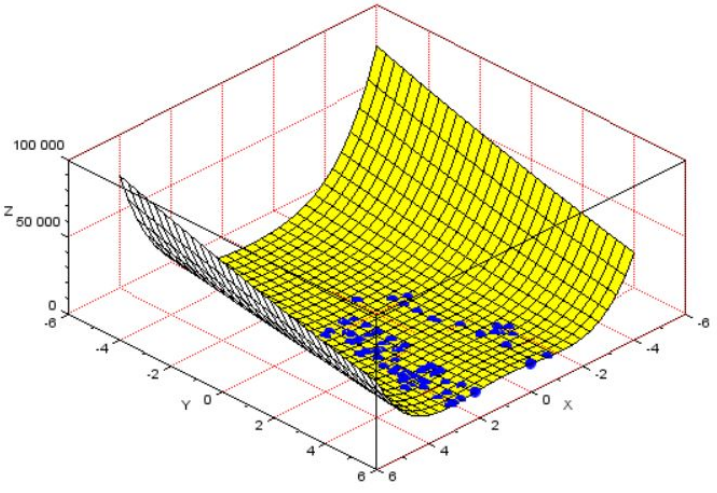
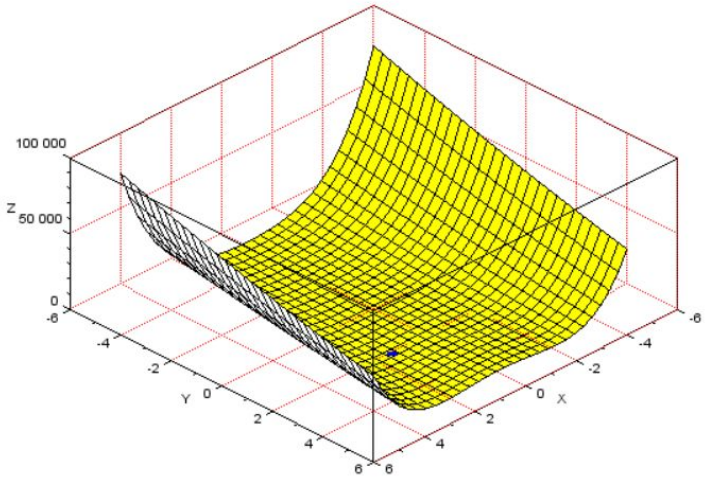


Figura 10: Gráfico de 100 gerações



A Figura 11 exibe a saída no terminal do Scilab.

Figura 11: Saída do terminal do Scilab

```
"O valor minimo encontrado: "

 0.5769484

"A posição do individuo na população é:"

 73.

"O individuo é:"

"01100110110110011101"

"O valor de X é:0.9824047"

"O valor de Y é:0.9628543"
```

2. Rede ELM

A ELM, Máquina de Aprendizado Extremo são redes neurais avançadas que podem ser utilizada para muitos fins, entre eles o de classificação e regressão. Um algoritmo ELM contém três fases principais, que serão abordadas neste trabalho.

A Fase 1 corresponde a inicialização aleatória dos pesos dos neurônios ocultos. A Fase 2 corresponde ao cálculo da função de ativação, dada pela Equação 02 e 03.

$$u(t) = wt * xt \quad (02)$$

$$Z(t) = 1 / (1+\exp(-u(t))) \quad (03)$$

A Fase 3 corresponde a encontrar os pesos dos neurônios de saída, utilizando o método dos mínimos quadrados. Como pode ser mostrado na Equação 04.

$$M = DZ'*(ZZ)'^{-1} \quad (04)$$

Para este trabalho foi utilizado o dataset “two_classes.dat” que contém 1000 amostras, com duas classes e duas características. Inicialmente, o dataset foi importado, e utilizando a função “datasetform” as duas primeiras colunas referente às características, foram associadas a matriz X, e a última coluna referente aos rótulos foi associada a matriz D. A Figura 12 exibe a importação e o tratamento inicial do dataset.

Figura 12: Importação do dataset

```
dataset = fscanfMat('C:\Users\STEFA\Desktop\ic\trab2\two_classes.dat');
function[X,D]=datasetform(len,dataset)
    X = dataset(:,1:2)';
    X = [(-1)*ones(1,len);X]; //adiconando o bias
    D = dataset(:,3)';
endfunction
```

Após isso, utilizando a função “W” foram geradas valores de pesos aleatórios. Ademais, a função “fase2” foi utilizada para realizar o cálculo das funções de ativação e tem como retorno uma matriz Z. A Figura 13 exibe o código das fases 1 e 2 do algoritmo.

Figura 13: Fase 1 e Fase 2

```
//FASE1
//definindo os pesos aleatorios
function[wt]=W(num_neuronios,num_classes)
    wt = rand(num_neuronios,num_classes+1,'normal');
endfunction

//FASE 2
function[Z]=fase2(len,wt,X)
    ut = wt*X; //função de ativação
    Z = (1./(1+exp(-ut)));
    Z = [(-1)*ones(1,len);Z];
endfunction
```

Para esse trabalho, são utilizados 30 neurônios, pois foram realizados alguns testes e essa quantidade de neurônios foi a que obteve os melhores resultados. A Figura 14 exibe o código para o cálculo dos pesos dos neurônios de saída.

Figura 13: Pesos dos Neurônios de Saída

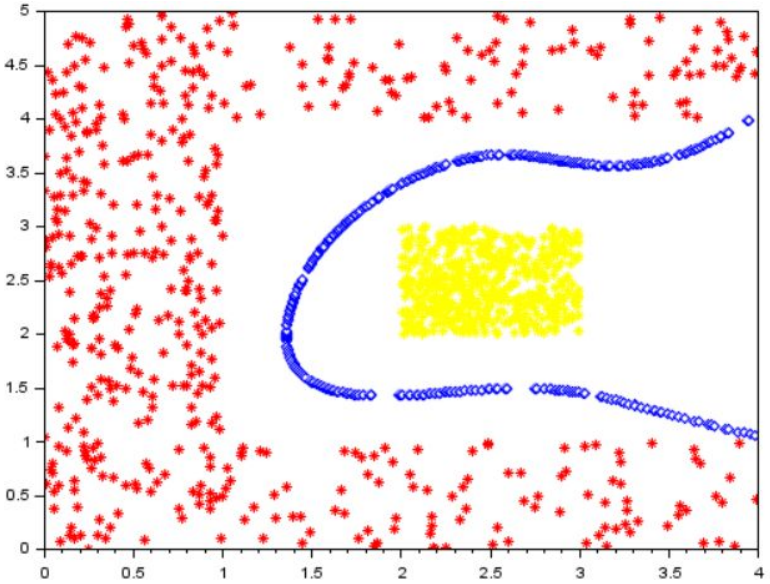
```
num_neuronios = 30; //define o numero d
num_classes = 2;
len = 1000; //quantidade de amostras d
[X D] = datasetform(len,dataset);

//FASE 1
//Recebe um vetor de pesos aleatorios
wt = W(num_neuronios,num_classes);

//FASE2
Z=fase2(len,wt,X);

//FASE 3
M = D*Z'*(Z*Z')^(-1); //metodo dos min
```

Para fazer a plotagem dos pontos, o dataset foi dividido em duas partes, sendo as primeiras 500 linhas associadas a classe 1 e as 500 ultimas linhas associadas a classe 2. Na plotagem os pontos amarelos são referentes a classe 1 e os pontos vermelhos são referentes a classe 2. Além disso, também foi realizada a plotagem da superfície de decisão, que tem como objetivo separar as duas classes. A Figura 14 exibe o código da plotagem. O gráfico de saída é exibido na Figura 15.

Figura 14: Plotagem dos Resultados

```
class1 = dataset(1:500,1:2);
class2 = dataset(501:1000,1:2);

plot(class1(:,1),class1(:,2),'y*')
plot(class2(:,1),class2(:,2),'r*')


//Criando 1000 pontos do plano
x = linspace(0,4, 1000);
y = linspace(0,4, 1000);

//Fazendo a plotagem da curva de decisã
for i = 1:1000
    for j = 1:1000
        xm = [-1 x(i) y(j)]';
        zm = fase2(1,wt,xm);
        s = M*zm;

        if s <0.001 & s > -0.001 then
            plot(x(i),y(j),"d");
        end
    end
end
```

Figura 15: Gráfico de Saída



É importante notar que ele não plotou os pontos onde “s” é muito próximo de zero. Isso ocorre porque esses pontos podem trazer uma determinada incoerência para o algoritmo pois não possuem classe bem definida.