MCCCC

```
clc
clear all
close all
%generation of genes randomly
%generate 10 genes each of length 30 using binary encoding
pool = randi([0,1], 10, 30);
%fitness is decided based on summation of values for each gene
fitness = sum(pool,2);
%selection of best fittted genes
high_first = max(fitness);
for i = 1:10
if fitness(i) == high_first
a=i;
end
end
parent_one = pool(a,:);
disp('Parent 1:'), disp (parent_one)
high_second = max(fitness(fitness<max(fitness)));
for i = 1:10
if fitness(i) == high_second
a=i;
end
end
parent_two = pool(a,:);
disp('Parent 2:'), disp (parent_two)
%crossover is done at any random point
b = randi([1 , 30]);
for i = 1:30
if i <= b
child(i) = parent_one(i);
else
child(i) = parent_two(i);
end
end
disp('Crossover point:'), disp (b)
disp('Child after crossover:'), disp (child)
%mutation is done at any random point
c = randi([1 , 30]);
if child(c) == 0
child(c) = 1;
end
disp('Mutation point:'), disp (c)
disp('Child after mutation:'), disp (child)
```

PERCEPTRON

```
clear;
clc;
x=[1 1 -1 -1;1 -1 1 -1];
t=[1 -1 -1 -1];
w=[0 0];
```

```
b=0;
alpha=input('Enter Learning rate=');
theta=input('Enter Threshold Value=');
con=1;
epoch=0;
while con
con=0;
for i=1:4
yin=b+x(1,i)*w(1)+x(2,i)*w(2);
if yin>theta
y=1;
end
if yin<=theta && yin>=-theta
y=0;
end
if yin<-theta
y=-1;
end
if y-t(i)
con=1;
for j=1:2
w(j)=w(j)+alpha*t(i)*x(j,i);
end
b=b+alpha*t(i);
end
end
epoch=epoch+1;
end
disp('Perceptron for AND Function');
disp('Final Weight Matrix');
disp(w);
disp('Final Bias');
disp(b);
```

HEBB

```
%Hebb Net to recognize Two -Dimensional input patterns.
clear;
clc;
%Input Pattern
E=[1 1 1 1 1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 1 1];
F=[1 1 1 1 1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 -1 -1 -1];
X(1,1:20)=E;
X(2,1:20)=F;
w(1:20)=0;
t=[1 -1];
b=0;
for i=1:2
w=w+X(i,1:20)*t(i);
b=b+t(i);
end
```

```
disp('Weight Matrix');
disp(w);
disp('Bias');
disp(b);
```

MEMBER

```
clc

clear all
close all
x=(0:1:10)';
y1=trimf(x, [1 7 10]);
subplot(3,1,1)
plot(x,[y1]);
xlabel('x axis')
ylabel('membership of x')
title('Triangular membership function')

x=(0:1:10)';
y1=trapmf(x, [1 3 5 7]);
subplot(3,1,2)
plot(x,[y1]);
xlabel('x axis')
ylabel('membership of x')
title('Trapezoidal membership function')


x=(0:0.2:10)';
y1=gbellmf(x, [1 2 5]);
subplot(3,1,3)
plot(x,[y1]);
xlabel('x axis')
ylabel('membership of x')
title('Bell-Shaped membership function')
```

BACK

```
clear all;
close all;
clc;
input=xlsread('fv.xlsx');
target=xlsread('target.xlsx');
nntic=tic;
hiddenLayerSize = 10;
net = feedforwardnet(hiddenLayerSize,'traingd');
net.trainParam.lr = 0.05; %its not mandatory to give this value,
%automatic value will be taken
net.trainParam.epochs = 3000; %its not mandatory to give this
%value, automatic value will be taken
net.trainParam.goal = 1e-5; %its not mandatory to give this
%value, automatic value will be taken
net.divideParam.trainRatio = 70/100;
```

```
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
net=init(net);
[net,tr] = train(net,input,target); %training
output = sim(net,input); %simulation
figure,plotconfusion(target,output)
plotregression(target,output); %regresson plot
error = gsubtract(target,output);
performance = mse(error); %mean square error
figure, plotroc(target,output)
nntime=toc(nntic);
unknown=xlsread('unknown.xlsx');%let it is the unknown feature
%value
y = net(unknown);%results obtained for all classes
% initlay is a network initialization function that initializes
%each layer i according to
% its own initialization function net.layers{i}.initFcn.
% The weights and biases of each layer i are initialized
%according to net.layers{i}.initFcn.
```

GAAA

```
clc
clear all
close all
%generation of genes randomly
%generate 10 genes each of length 30 using binary encoding
pool = randi([0,1], 10, 30);
%fitness is decided based on summation of values for each gene
fitness = sum(pool,2);
%selection of best fittted genes
high_first = max(fitness);
for i = 1:10
if fitness(i) == high_first
a=i;
end
end
parent_one = pool(a,:);
disp('Parent 1:'), disp (parent_one)
high_second = max(fitness(fitness<max(fitness)));
for i = 1:10
if fitness(i) == high_second
a=i;
end
end
parent_two = pool(a,:);
disp('Parent 2:'), disp (parent_two)
%crossover is done at any random point
b = randi([1 , 30]);
for i = 1:30
if i <= b
```

```
child(i) = parent_one(i);
else
child(i) = parent_two(i);
end
end
disp('Crossover point:'), disp (b)
disp('Child after crossover:'), disp (child)
%mutation is done at any random point
c = randi([1 , 30]);
if child(c) == 0
child(c) = 1;
end
disp('Mutation point:'), disp (c)
disp('Child after mutation:'), disp (child)
```