```
% Project 1 Clark Lakshminarayanan Sonawani
clear all
close all

% Generate Clusters

% % Training and Testing clusters for d = 2
% [Train1D2,Train2D2,Test1D2,Test2D2]=GenerateClusters(2);
% % Training and Testing clusters for d = -4
% [Train1Dn4,Train2Dn4,Test1Dn4,Test2Dn4]=GenerateClusters(-4);
% % Training and Testing clusters for d = -8
% [Train1Dn8,Train2Dn8,Test1Dn8,Test2Dn8]=GenerateClusters(-8);
r=10;
w=6;

% Seeds
seed1=1;    seed5=5;
seed2=2;    seed6=6;
seed3=3;    seed7=7;
seed4=4;    seed8=8;

% Cluster 1 Training (Rho-Magnitude Theta-angle[rad])
rng(seed1,'twister');
Cluster1RhoTrain = (r-w/2)+w*rand(1,1000);
rng(seed2,'twister');
Cluster1ThetaTrain = pi*rand(1,1000);

% Cluster 2 Training (Rho-Magnitude Theta-angle[rad])
rng(seed3,'twister');
Cluster2RhoTrain = (r-w/2)+w*rand(1,1000);
rng(seed4,'twister');
Cluster2ThetaTrain = -pi*rand(1,1000);

% Cluster 1 Testing (Rho-Magnitude Theta-angle[rad])
rng(seed5,'twister');
Cluster1RhoTest = (r-w/2)+w*rand(1,500);
rng(seed6,'twister');
Cluster1ThetaTest = pi*rand(1,500);

% Cluster 2 Testing (Rho-Magnitude Theta-angle[rad])
rng(seed7,'twister');
Cluster2RhoTest = (r-w/2)+w*rand(1,500);
rng(seed8,'twister');
Cluster2ThetaTest = -pi*rand(1,500);

% Convert to Carteasian Coordinate system
[Cluster1XTrain, Cluster1YTrain] =
 pol2cart(Cluster1ThetaTrain,Cluster1RhoTrain);
[Cluster2XTrain, Cluster2YTrain] =
 pol2cart(Cluster2ThetaTrain,Cluster2RhoTrain);
[Cluster1XTest, Cluster1YTest] =
 pol2cart(Cluster1ThetaTest,Cluster1RhoTest);
```

```
[Cluster2XTest, Cluster2YTest] =
 pol2cart(Cluster2ThetaTest,Cluster2RhoTest);

Cluster2XTrain=Cluster2XTrain+r;
Cluster2XTest=Cluster2XTest+r;
% Shift Cluster2 d=2
d=2;

Cluster2YTrain=Cluster2YTrain-d;
Cluster2YTest=Cluster2YTest-d;

Train1D2=[Cluster1XTrain;Cluster1YTrain];
Train2D2=[Cluster2XTrain;Cluster2YTrain];
Test1D2=[Cluster1XTest;Cluster1YTest];
Test2D2=[Cluster2XTest;Cluster2YTest];
% [C1Train,C2Train,C1Test,C2Test]
% Shift Cluster2 d=-4
d=-4;

Cluster2YTrain=Cluster2YTrain-d;
Cluster2YTest=Cluster2YTest-d;

Train1Dn4=[Cluster1XTrain;Cluster1YTrain];
Train2Dn4=[Cluster2XTrain;Cluster2YTrain];

Test1Dn4=[Cluster1XTest;Cluster1YTest];
Test2Dn4=[Cluster2XTest;Cluster2YTest];


% Shift Cluster2  d=-8
d=-8;

Cluster2YTrain=Cluster2YTrain-d;
Cluster2YTest=Cluster2YTest-d;

Train1Dn8=[Cluster1XTrain;Cluster1YTrain];
Train2Dn8=[Cluster2XTrain;Cluster2YTrain];

Test1Dn8=[Cluster1XTest;Cluster1YTest];
Test2Dn8=[Cluster2XTest;Cluster2YTest];


% % Organize Data in Clusters
TrainData={Train1D2,Train2D2;
           Train1Dn4,Train2Dn4;
           Train1Dn8,Train2Dn8};

TestData={Test1D2,Test2D2;
          Test1Dn4,Test2Dn4;
          Test1Dn8,Test2Dn8};



% Plot Clusters
```

```
fig1=figure(1);
fig1.Renderer='Painters';
set(fig1,'units','points','position',[200,200,700,600])
hold on;grid on;
scatter(Test1D2(1,:),Test1D2(2,:),20,'r','filled')
scatter(Test2D2(1,:),Test2D2(2,:),20,'b','filled')

fig2=figure(2);
fig2.Renderer='Painters';
set(fig2,'units','points','position',[200,200,700,600])
hold on;grid on;
scatter(Test1Dn4(1,:),Test1Dn4(2,:),20,'r','filled')
scatter(Test2Dn4(1,:),Test2Dn4(2,:),20,'b','filled')

fig3=figure(3);
fig3.Renderer='Painters';
set(fig3,'units','points','position',[200,200,700,600])
hold on;grid on;
scatter(Test1Dn8(1,:),Test1Dn8(2,:),20,'r','filled')
scatter(Test2Dn8(1,:),Test2Dn8(2,:),20,'b','filled')

fig4=figure(4);
fig4.Renderer='Painters';
set(fig4,'units','points','position',[860,200,700,600])

fig5=figure(5);
fig5.Renderer='Painters';
set(fig5,'units','points','position',[860,200,700,600])

fig6=figure(6);
fig6.Renderer='Painters';
set(fig6,'units','points','position',[860,200,700,600])

fig7=figure(7);
fig7.Renderer='Painters';
set(fig7,'units','points','position',[860,200,700,600])
hold on;grid on;
scatter(Test1Dn8(1,:),Test1Dn8(2,:),20,'r','filled')
scatter(Test2Dn8(1,:),Test2Dn8(2,:),20,'b','filled')

fig8=figure(8);
fig8.Renderer='Painters';
set(fig8,'units','points','position',[860,200,700,600])

% Setup Experiment 1
Test1.Algorithms={'traingd','traingdm','trainlm'};
Test1.C={'-r','--r',':r';'-g','--g',':g';'-b','--b',':b'};
Test1.d=[2,-4,-8];
Test1.Lrate=[ 1.6,0.9,0.2];
Test1.Nneurons=5;
Test1.xvec=(-15:1:25);
Test1.yvec=(15:-1:-15);
Test1.grid=zeros(length(Test1.yvec),length(Test1.xvec));
[Test1.X,Test1.Y]=meshgrid(Test1.xvec,Test1.yvec);
```

```
% Train Experiment1
for n=1:length(Test1.Algorithms)
    for o=1:length(Test1.Lrate)
        for m=1:length(Test1.d)
            rng(2);
            % prep training input
            y1 = ones(1,1000);
            y2 = zeros(1,1000);
            train_set = [TrainData{m,1},TrainData{m,2}];
            order = randperm(2000);
            train_set = train_set(:,order);
            target = [y1,y2];
            target = target(order);

            % setup net
            net1{m,n,o} =
 feedforwardnet(Test1.Nneurons,Test1.Algorithms{n});
            net1{m,n,o} = configure(net1{m,n,o},train_set,target);
            net1{m,n,o}.trainParam.lr = Test1.Lrate(o);
            net1{m,n,o}.trainParam.epochs=2000;
            net1{m,n,o}.divideParam.trainRatio = 0.75;
            net1{m,n,o}.divideParam.valRatio = 0.25;
            net1{m,n,o}.divideParam.testRatio = 0.0;
            net1{m,n,o}.trainParam.max_fail=2000;
            if m==2
                net1{m,n,o}.trainParam.mc = 0.9;
            end

            % training algorithm
            [net1{m,n,o},TR] = train(net1{m,n,o},train_set,target);
            train_op = net1{m,n,o}(train_set);


            % testing algorithm
            order = randperm(1000);
            target=[ones(1,500) zeros(1,500)];
            target=target(:,order);
            testing_set = [TestData{m,1},TestData{m,2}];
            testing_set = testing_set(:,order);
            test_op = net1{m,n,o}(testing_set);

            % Boundary Function
            for r=1:length(Test1.xvec)
                for t=1:length(Test1.yvec)
                   classifierGrid(t,r)=net1{m,n,o}
([Test1.xvec(r);Test1.yvec(t)]);
                end
            end

            % Plot Boundaries
            figure(m);
            hold on;grid on;
            contour(Test1.X,Test1.Y,classifierGrid-.5,[0
 0],Test1.C{n,o},'lineWidth',1)
```

```
            xlim([-15 25]);
            ylim([-15 15]);

            %Plot Performance
            q=m+3;
            figure(q);
            hold on;grid on;
            plot(TR.epoch,TR.vperf,Test1.C{n,o},'lineWidth',1)
            xlim([0 2000]);
            ylim([0 .2]);

            for i=1:1000
                 if(test_op(i)>=0.5)
                    test_op(i)=1;
                 else
                     test_op(i)=0;
                 end
            end


            plotconfusion(target,test_op);


            pause;
        end
    end
end


% Setup Experiment 2
Test2.Algorithms={'traingd','traingdm','trainlm'};
Test2.C={'-r','--r',':r';'-g','--g',':g';'-b','--b',':b'};
Test2.d=[2,-4,-8];
Test2.Lrate=0.9;
Test2.Nneurons=[2 5 11];
Test2.xvec=(-15:1:25);
Test2.yvec=(15:-1:-15);
Test2.grid=zeros(length(Test2.yvec),length(Test2.xvec));
[Test2.X,Test2.Y]=meshgrid(Test2.xvec,Test2.yvec);

% Train Experiment2
for n=1:length(Test2.Algorithms)
    for o=1:length(Test2.Nneurons)
        for m=3:length(Test2.d)
            rng(2);
            % prep training input
            y1 = ones(1,1000);
            y2 = zeros(1,1000);
            train_set = [TrainData{m,1},TrainData{m,2}];
            order = randperm(2000);
            train_set = train_set(:,order);
            target = [y1,y2];
            target = target(order);
```

```
            % setup net
            net2{m,n,o} =
 feedforwardnet(Test2.Nneurons(o),Test2.Algorithms{n});
            net2{m,n,o} = configure(net2{m,n,o},train_set,target);
            net2{m,n,o}.trainParam.lr = Test2.Lrate;
            net2{m,n,o}.trainParam.epochs=2000;
            net2{m,n,o}.divideParam.trainRatio = 0.75;
            net2{m,n,o}.divideParam.valRatio = 0.25;
            net2{m,n,o}.divideParam.testRatio = 0.0;
            net2{m,n,o}.trainParam.max_fail=2000;
            if m==2
                net2{m,n,o}.trainParam.mc = 0.9;
            end

            % training algorithm
            [net2{m,n,o},TR] = train(net2{m,n,o},train_set,target);
            train_op = net2{m,n,o}(train_set);


            % testing algorithm
            order = randperm(1000);
            target=[ones(1,500) zeros(1,500)];
            target=target(:,order);
            testing_set = [TestData{m,1},TestData{m,2}];
            testing_set = testing_set(:,order);
            test_op = net2{m,n,o}(testing_set);

            % Boundary Function
            for r=1:length(Test2.xvec)
                for t=1:length(Test2.yvec)
                   classifierGrid(t,r)=net2{m,n,o}
([Test2.xvec(r);Test2.yvec(t)]);
                end
            end

            % Plot Boundaries
            figure(7);
            hold on;grid on;
            contour(Test2.X,Test2.Y,classifierGrid-.5,[0
 0],Test2.C{n,o},'lineWidth',1)
            xlim([-15 25]);
            ylim([-15 15]);

            %Plot Performance
            figure(8)
            hold on;grid on;
            plot(TR.epoch,TR.vperf,Test2.C{n,o},'lineWidth',1)
            xlim([0 2000]);
            ylim([0 .2]);
            for i=1:1000
                 if(test_op(i)>=0.5)
                    test_op(i)=1;
                 else
                     test_op(i)=0;
```

```
                end
            end


            plotconfusion(target,test_op);
            pause;
        end
    end
end



% Save Graphs
% Experiment 1
figure(1)
title('Decision Boundaries for Clusters D=2','FontSize',15)
ldg1=legend( 'Cluster 1 (d=2)','Cluster 2 (d=2)',....
        'Back Propogation - Learning Rate 1.6','Back Propogation -
 Learning Rate 0.9','Back Propogation - Learning Rate 0.2',....
        'Back Propogation with Momentum - Learning Rate 1.6','Back
 Propogation with Momentum - Learning Rate 0.9','Back Propogation with
 Momentum - Learning Rate 0.2',....
        'Levenberg Marquardt - Learning Rate 1.6','Levenberg Marquardt
 - Learning Rate 0.9','Levenberg Marquardt - Learning Rate 0.2')
ldg1.FontSize=7;
print('-painters','-depsc','Exp1_DB2')

figure(2)
title('Decision Boundaries for Clusters D=-4','FontSize',15)
ldg2=legend( 'Cluster 1 (d=-4)','Cluster 2 (d=-4)',....
        'Back Propogation - Learning Rate 1.6','Back Propogation -
 Learning Rate 0.9','Back Propogation - Learning Rate 0.2',....
        'Back Propogation with Momentum - Learning Rate 1.6','Back
 Propogation with Momentum - Learning Rate 0.9','Back Propogation with
 Momentum - Learning Rate 0.2',....
        'Levenberg Marquardt - Learning Rate 1.6','Levenberg Marquardt
 - Learning Rate 0.9','Levenberg Marquardt - Learning Rate 0.2')
ldg2.FontSize=7;
print('-painters','-depsc','Exp1_DBn4')

figure(3)
title('Decision Boundaries for Clusters D=-8','FontSize',15)
ldg3=legend( 'Cluster 1 (d=-8)','Cluster 2 (d=-8)',....
        'Back Propogation - Learning Rate 1.6','Back Propogation -
 Learning Rate 0.9','Back Propogation - Learning Rate 0.2',....
        'Back Propogation with Momentum - Learning Rate 1.6','Back
 Propogation with Momentum - Learning Rate 0.9','Back Propogation with
 Momentum - Learning Rate 0.2',....
        'Levenberg Marquardt - Learning Rate 1.6','Levenberg Marquardt
 - Learning Rate 0.9','Levenberg Marquardt - Learning Rate 0.2')
ldg3.FontSize=7;
print('-painters','-depsc','Exp1_DBn8')

figure(4)
```

```
title('Learning Curves for Clusters D=2','FontSize',15)
ldg4=legend(....
        'Back Propogation - Learning Rate 1.6','Back Propogation -
 Learning Rate 0.9','Back Propogation - Learning Rate 0.2',....
        'Back Propogation with Momentum - Learning Rate 1.6','Back
 Propogation with Momentum - Learning Rate 0.9','Back Propogation with
 Momentum - Learning Rate 0.2',....
        'Levenberg Marquardt - Learning Rate 1.6','Levenberg Marquardt
 - Learning Rate 0.9','Levenberg Marquardt - Learning Rate 0.2')
ldg4.FontSize=7;
print('-painters','-depsc','Exp1_LC2')

figure(5)
title('Learning Curves for Clusters D=-4','FontSize',15)
ldg5=legend(....
        'Back Propogation - Learning Rate 1.6','Back Propogation -
 Learning Rate 0.9','Back Propogation - Learning Rate 0.2',....
        'Back Propogation with Momentum - Learning Rate 1.6','Back
 Propogation with Momentum - Learning Rate 0.9','Back Propogation with
 Momentum - Learning Rate 0.2',....
        'Levenberg Marquardt - Learning Rate 1.6','Levenberg Marquardt
 - Learning Rate 0.9','Levenberg Marquardt - Learning Rate 0.2')
ldg5.FontSize=7;
print('-painters','-depsc','Exp1_LCn4')

figure(6)
title('Learning Curves for Clusters D=-8','FontSize',15)
ldg6=legend(....
        'Back Propogation - Learning Rate 1.6','Back Propogation -
 Learning Rate 0.9','Back Propogation - Learning Rate 0.2',....
        'Back Propogation with Momentum - Learning Rate 1.6','Back
 Propogation with Momentum - Learning Rate 0.9','Back Propogation with
 Momentum - Learning Rate 0.2',....
        'Levenberg Marquardt - Learning Rate 1.6','Levenberg Marquardt
 - Learning Rate 0.9','Levenberg Marquardt - Learning Rate 0.2')
ldg6.FontSize=7;
print('-painters','-depsc','Exp1_LCn8')

% Experiment 2
figure(7)
title('Decision Boundaries for Clusters D=-8','FontSize',15)
ldg7=legend( 'Cluster 1 (d=-8)','Cluster 2 (d=-8)',....
        'Back Propogation - Neurons 2','Back Propogation - Neurons
 5','Back Propogation - Neurons 11',....
        'Back Propogation with Momentum - Neurons 2','Back Propogation
 with Momentum - Neurons 5','Back Propogation with Momentum - Neurons
 11',....
        'Levenberg Marquardt - Neurons 2','Levenberg Marquardt -
 Neurons 5','Levenberg Marquardt - Neurons 11')
ldg7.FontSize=7;
print('-painters','-depsc','Exp2_DBn8')

figure(8)
title('Learning Curves for Clusters D=-8','FontSize',15)
```

```
ldg8=legend(....
        'Back Propogation - Neurons 2','Back Propogation - Neurons
 5','Back Propogation - Neurons 11',....
        'Back Propogation with Momentum - Neurons 2','Back Propogation
 with Momentum - Neurons 5','Back Propogation with Momentum - Neurons
 11',....
        'Levenberg Marquardt - Neurons 2','Levenberg Marquardt -
 Neurons 5','Levenberg Marquardt - Neurons 11')
ldg8.FontSize=7;
print('-painters','-depsc','Exp2_LCn8')
```

Published with MATLAB® R2017a