Assignment 2

1. Label the first 5000 rows as class 1 and the remaining rows as class 2. Use SVM and Neural Network to classify the data and report 10-fold cross-validated accuracy. Describe the parameters of your classifiers.

**SVM:**

**Code:**

%Initiating the variable with no. of folds

kf=10;

% Taking the labels

Labels=zeros(10000,1);

Labels(1:5000,1)=1;

Indices = crossvalind('Kfold',Labels, kf); % Returns randomly generated indices for a K-fold cross-validation of N observations(Labels)

CP = classperf(Labels); % Evaluating the performance of classifier

trainingIndex=0; %Initiating the trainingIndex variable

testIndex=0; %Initiating the trainingIndex variable

for i = 1:kf

% Assigning training and test indices

z=(Indices==i);

testIndex = z; % Getting indices of test instances

trainingIndex = ~z; % Getting indices of training instances

% Training an SVM model over training instances

SVMStruct = svmtrain( data(trainingIndex,:), Labels(trainingIndex), 'Autoscale',true,'Showplot',false, 'Method','QP', 'BoxConstraint',2e-1, 'Kernel\_Function','rbf', 'RBF\_Sigma',0.2);

Group = svmclassify(SVMStruct, data(testIndex,:)); %classifies each row of the data in Sample, a matrix of data, using the information in a support vector machine classifier structure SVMStruct, created using the svmtrain function

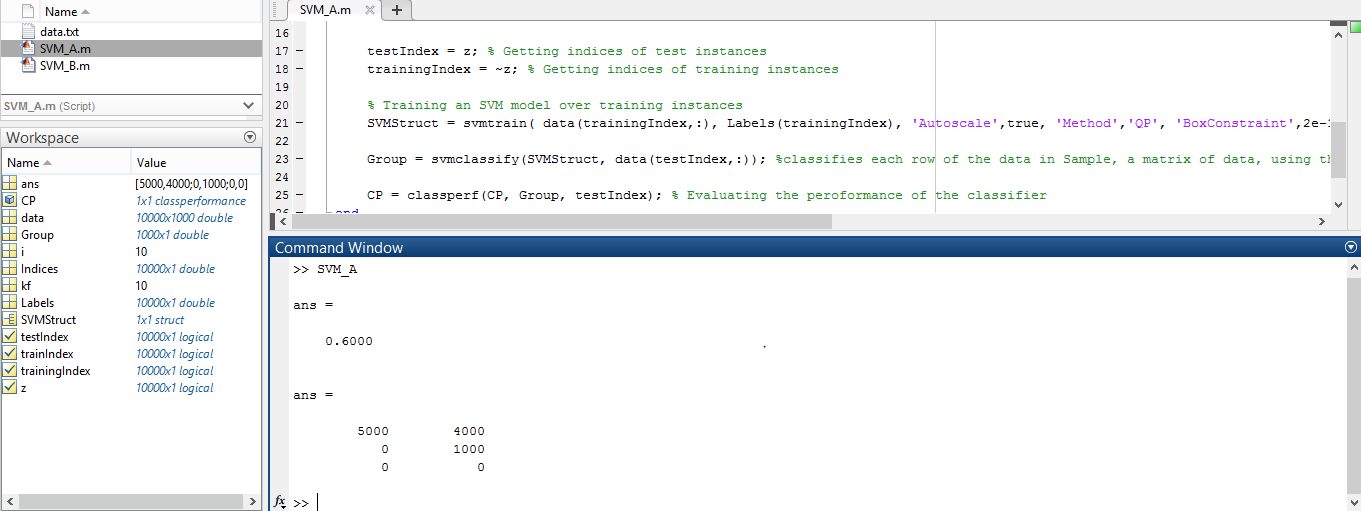
CP = classperf(CP, Group, testIndex); % Evaluating the peroformance of the classifier

end

CP.CorrectRate % Getting the Classified Samples

CP.CountingMatrix % Getting the classification confusion matrix

**MAtlab Screen and output:**

****

**Accuracy: 60%**

**Neural Network:**

**Code:**

nObser=10000;

kf=10;

idx=zeros(nObser,kf);

c=cvpartition(nObser,'kfold',kf); % Returns cvpartition class object for kfold cross validation on n Observations(nObser)

for i=1:1:kf

idx(:,i)=test(c,i); %returns the logical vector idx of test indices for repetition i of an object c of the cvpartition class of type 'kfold'

end

%Finding the Training and Test indexes

trainingindex=zeros(9000,kf);

testindex=zeros(1000,kf);

n=1;

while n<=kf

k=1;

p=1;

j=1;

while j<=nObser

if(idx(j,n)==0)

trainingindex(k,n)=j;

k=k+1;

else

testindex(p,n)=j;

p=p+1;

end

j=j+1;

end

n=n+1;

end

% Creating a Pattern Recognition Network

hiddenLayerSize = kf;

net = patternnet(hiddenLayerSize);

% Choosing Input and Output Pre/Post-Processing Functions

net.input.processFcns = {'removeconstantrows','mapminmax'}; %Input Pre/Post-Processing Functions

net.output.processFcns = {'removeconstantrows','mapminmax'}; %Output Pre/Post-Processing Functions

acc=zeros(1,kf); % Defining the variable to store accuracies

% Taking the labels

Labels=zeros(nObser,1);

Labels(1:5000,1)=1;

Labels(:,2)=~Labels(:,1);

for i=1:1:kf

% Setup Division of Data for Training, Validation, Testing

% Choosing to divide the data randomly

net.divideFcn = 'divideind';

% Choosing Division Mode

net.divideMode = 'sample';

% Assigning the Training indexes

net.divideParam.trainInd=trainingindex(1:8100,i);

% Assigning the Validation indexes

net.divideParam.valInd=trainingindex(8101:9000,i);

% Assigning the Test indexes

net.divideParam.testInd=testindex(:,i);

% Choosing the Training Function

net.trainFcn = 'trainscg'; % Scaled conjugate gradient

% Choosing the Performance Function

net.performFcn = 'crossentropy'; % Cross-entropy

% Choosing the Plot Functions

net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...

'plotregression', 'plotfit'};

% Defining the network input X and network target T

X = data';

T = Labels';

% Training the Network

[net,tr] = train(net,X,T); % trains a network net with the input X and target T. Returns the new network 'net' and training record tr.

% Testing the Network

y = net(X); % Returns network output

errors = gsubtract(T,y); % Returns the errors

Tindices = vec2ind(T); % Converting target vector T to indices

yindices = vec2ind(y); % % Converting output vector Y to indices

percentErrors = sum(Tindices ~= yindices)/numel(Tindices);

performance = perform(net,T,y); % Calculates the network performance

% Recalculate Training, Validation and Test Performance

trainTargets = T.\* tr.trainMask{1};

valTargets = T .\* tr.valMask{1};

testTargets = T .\* tr.testMask{1};

trainPerformance = perform(net,trainTargets,y); % Training Performance

valPerformance = perform(net,valTargets,y); % Validation Performance

testPerformance = perform(net,testTargets,y); % Test Performance

[c,cm,ind,per] = confusion(T,y); % Returns the Confusion Value c, Confusion Matrix cm,indices ind and the percentges associated per

acc(1,i)=((1-c)\*100); % Calculating the accuracy

end

% To View the Neural Network Diagram

view(net)

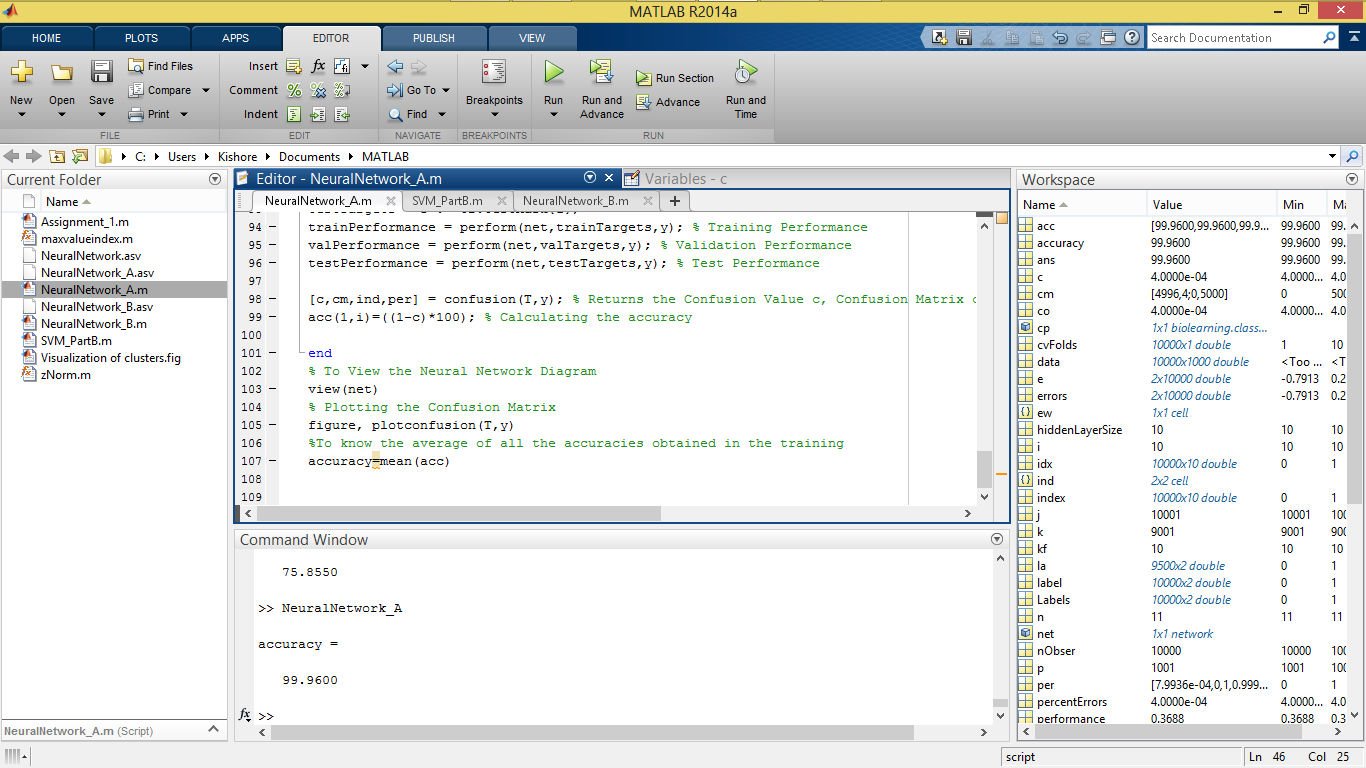
% Plotting the Confusion Matrix

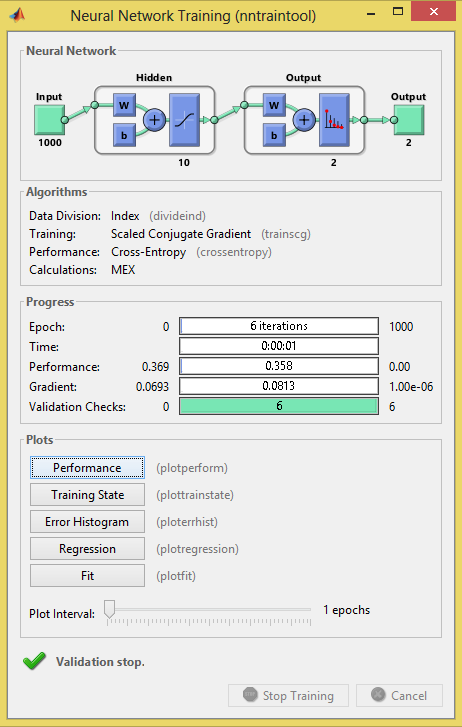
figure, plotconfusion(T,y)

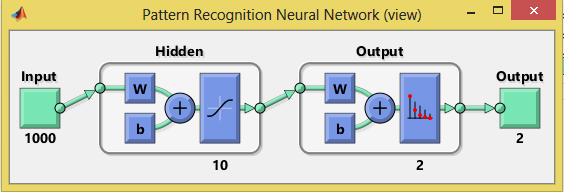
%To know the average of all the accuracies obtained in the training

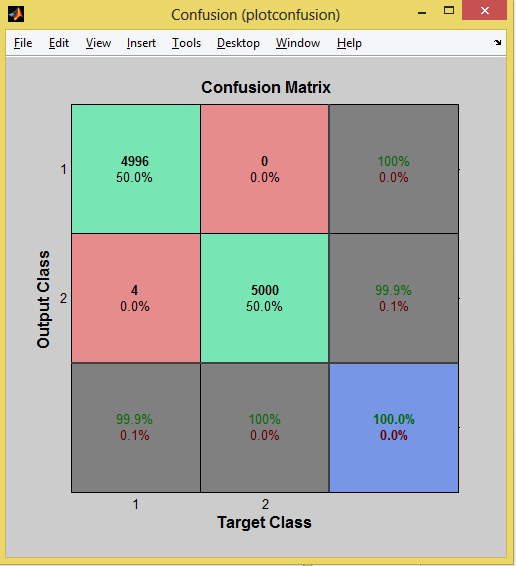
accuracy=mean(acc)

**MAtlab Screen and output:**









**Accuracy: 99.96%**

1. Label the rows [1:500,1001:1500,2001:2500,3001:3500,4001:4500,5001:5500,6001:6500,7001:7500,8001:8500,9001:9500]  
   as class 1 and the remaining rows as class 2. Use SVM and Neural Network to classify the data and report 10-fold cross-validated accuracy. Describe the parameters of your classifiers.

**SVM:**

**Code:**

%Initiating the variable with no. of folds

kf=10;

% Taking the labels

Labels=zeros(10000,1);

Labels([1:500,1001:1500,2001:2500,3001:3500,4001:4500,5001:5500,6001:6500,7001:7500,8001:8500,9001:9500],1)=1;

Indices = crossvalind('Kfold',Labels, kf); % Returns randomly generated indices for a K-fold cross-validation of N observations(Labels)

CP = classperf(Labels); % Evaluating the performance of classifier

trainingIndex=0; %Initiating the trainingIndex variable

testIndex=0; %Initiating the trainingIndex variable

for i = 1:kf

% Assigning training and test indices

z=(Indices==i);

testIndex = z; % Getting indices of test instances

trainingIndex = ~z; % Getting indices of training instances

%testIndex = (Indices == i); %# get indices of test instances

%trainingIndex = ~testIndex;

% Training an SVM model over training instances

SVMStruct = svmtrain( data(trainingIndex,:), Labels(trainingIndex), 'Autoscale',true,'Showplot',false, 'Method','QP', 'BoxConstraint',2e-1, 'Kernel\_Function','rbf', 'RBF\_Sigma',0.2);

Group = svmclassify(SVMStruct, data(testIndex,:)); %classifies each row of the data in Sample, a matrix of data, using the information in a support vector machine classifier structure SVMStruct, created using the svmtrain function

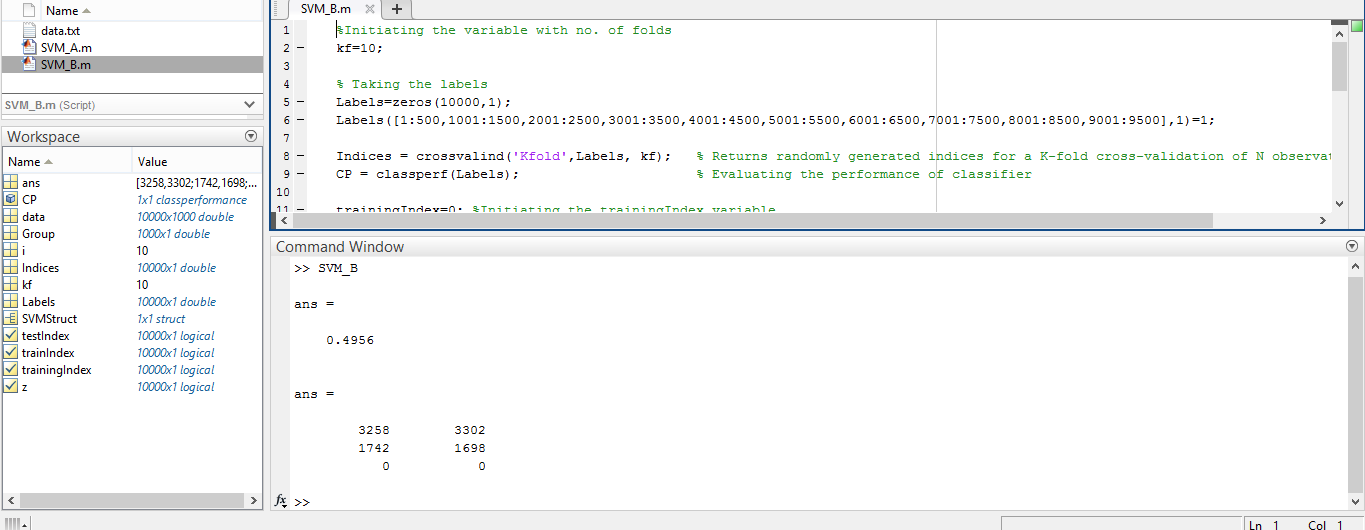
CP = classperf(CP, Group, testIndex); % Evaluating the peroformance of the classifier

end

CP.CorrectRate % Getting the Classified Samples

CP.CountingMatrix % Getting the classification confusion matrix

**MAtlab Screen and output:**

****

**Accuracy: 49.56%**

**Neural Network:**

**Code:**

nObser=10000;

kf=10;

idx=zeros(nObser,kf);

c=cvpartition(nObser,'kfold',kf); % Returns cvpartition class object for kfold cross validation on n Observations(nObser)

for i=1:1:kf

idx(:,i)=test(c,i); %returns the logical vector idx of test indices for repetition i of an object c of the cvpartition class of type 'kfold'

end

%Finding the Training and Test indexes

trainingindex=zeros(9000,kf);

testindex=zeros(1000,kf);

n=1;

while n<=kf

k=1;

p=1;

j=1;

while j<=nObser

if(idx(j,n)==0)

trainingindex(k,n)=j;

k=k+1;

else

testindex(p,n)=j;

p=p+1;

end

j=j+1;

end

n=n+1;

end

% Creating a Pattern Recognition Network

hiddenLayerSize = kf;

net = patternnet(hiddenLayerSize);

% Choosing Input and Output Pre/Post-Processing Functions

net.input.processFcns = {'removeconstantrows','mapminmax'}; %Input Pre/Post-Processing Functions

net.output.processFcns = {'removeconstantrows','mapminmax'}; %Output Pre/Post-Processing Functions

acc=zeros(1,10); % Defining the variable to store accuracies

% Taking the labels

Labels=zeros(nObser,1);

Labels( [1:500,1001:1500,2001:2500,3001:3500,4001:4500,5001:5500,6001:6500,7001:7500,8001:8500,9001:9500],1)=1;

Labels(:,2)=~(Labels(:,1)); %

for i=1:1:kf

% Setup Division of Data for Training, Validation, Testing

% Choosing to divide the data randomly

net.divideFcn = 'divideind';

% Choosing Division Mode

net.divideMode = 'sample';

% Assigning the Training indexes

%net.divideParam.trainInd=trainingindex(i,1:8000);

net.divideParam.trainInd=trainingindex(1:8100,i);

% Assigning the Validation indexes

%net.divideParam.valInd=trainingindex(i,8001:9000);

net.divideParam.valInd=trainingindex(8101:9000,i);

% Assigning the Test indexes

%net.divideParam.testInd=testindex(i,:);

net.divideParam.testInd=testindex(:,i);

% Choosing the Training Function

net.trainFcn = 'trainscg'; % Scaled conjugate gradient

% Choosing the Performance Function

net.performFcn = 'crossentropy'; % Cross-entropy

% Choosing the Plot Functions

net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...

'plotregression', 'plotfit'};

% Defining the network input X and network target T

X = data';

T = Labels';

% Training the Network

[net,tr] = train(net,X,T); % trains a network net with the input X and target T. Returns the new network 'net' and training record tr.

% Testing the Network

y = net(X); % Returns network output

errors = gsubtract(T,y); % Returns the errors

Tindices = vec2ind(T); % Converting target vector T to indices

yindices = vec2ind(y); % % Converting output vector Y to indices

percentErrors = sum(Tindices ~= yindices)/numel(Tindices);

performance = perform(net,T,y); % Calculates the network performance

% Recalculate Training, Validation and Test Performance

trainTargets = T.\* tr.trainMask{1};

valTargets = T .\* tr.valMask{1};

testTargets = T .\* tr.testMask{1};

trainPerformance = perform(net,trainTargets,y); % Training Performance

valPerformance = perform(net,valTargets,y); % Validation Performance

testPerformance = perform(net,testTargets,y); % Test Performance

[c,cm,ind,per] = confusion(T,y); % Returns the Confusion Value c, Confusion Matrix cm,indices ind and the percentges associated per

acc(1,i)=((1-c)\*100); % Calculating the accuracy

end

% To View the Neural Network Diagram

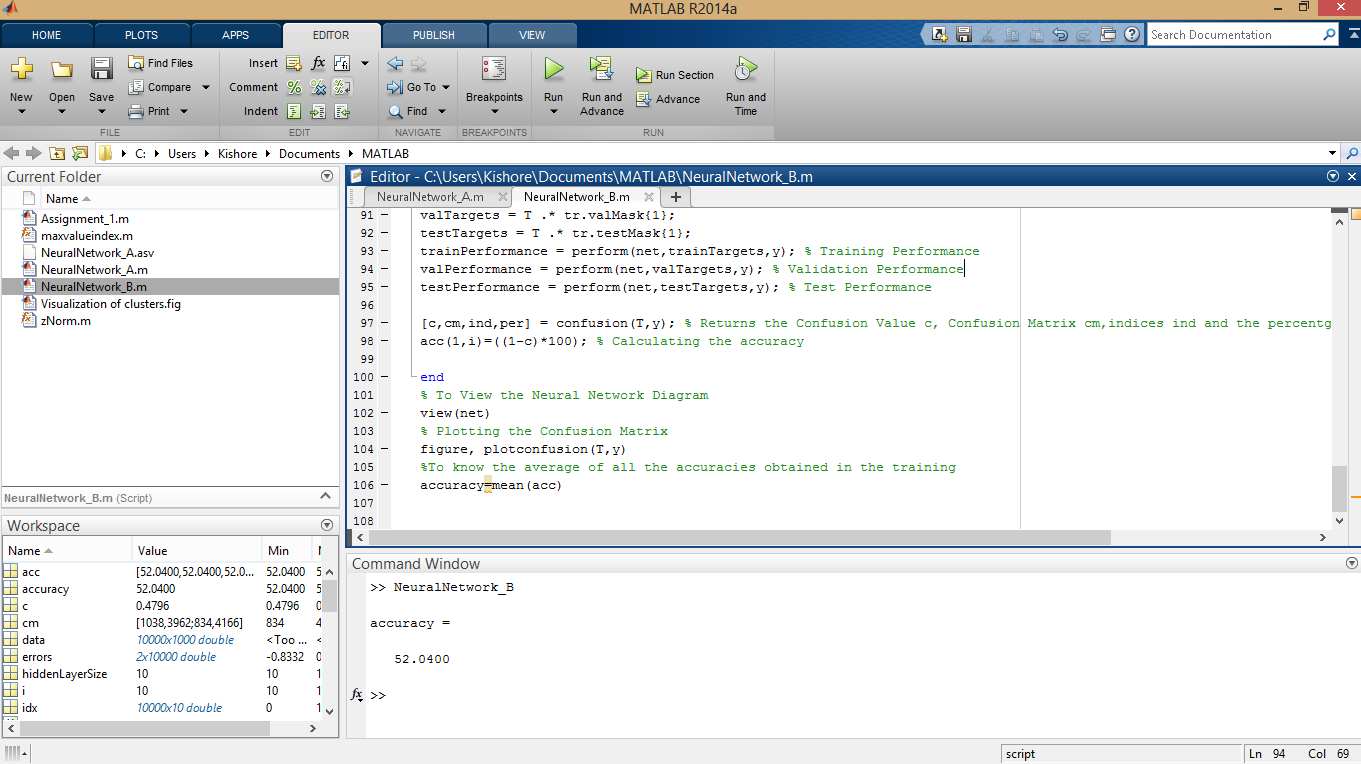
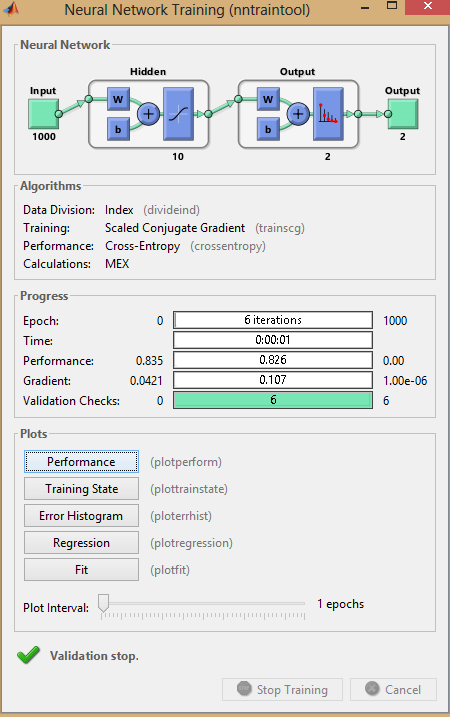
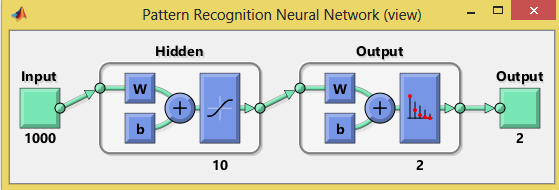
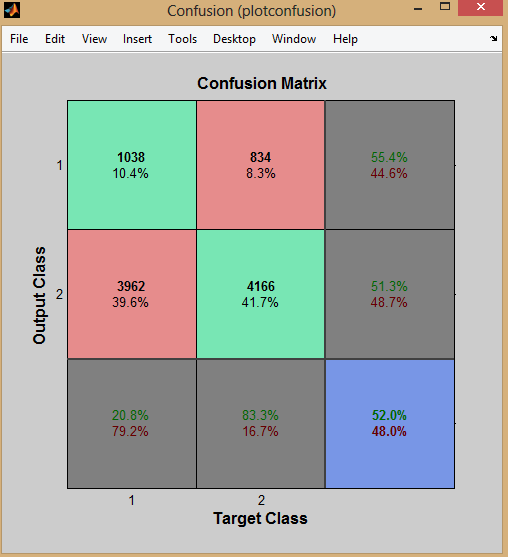
view(net)

% Plotting the Confusion Matrix

figure, plotconfusion(T,y)

%To know the average of all the accuracies obtained in the training

accuracy=mean(acc)

**MAtlab Screen and output:**   

**Accuracy: 52.04%**