Application of k fold testing using Decision trees and SVM:

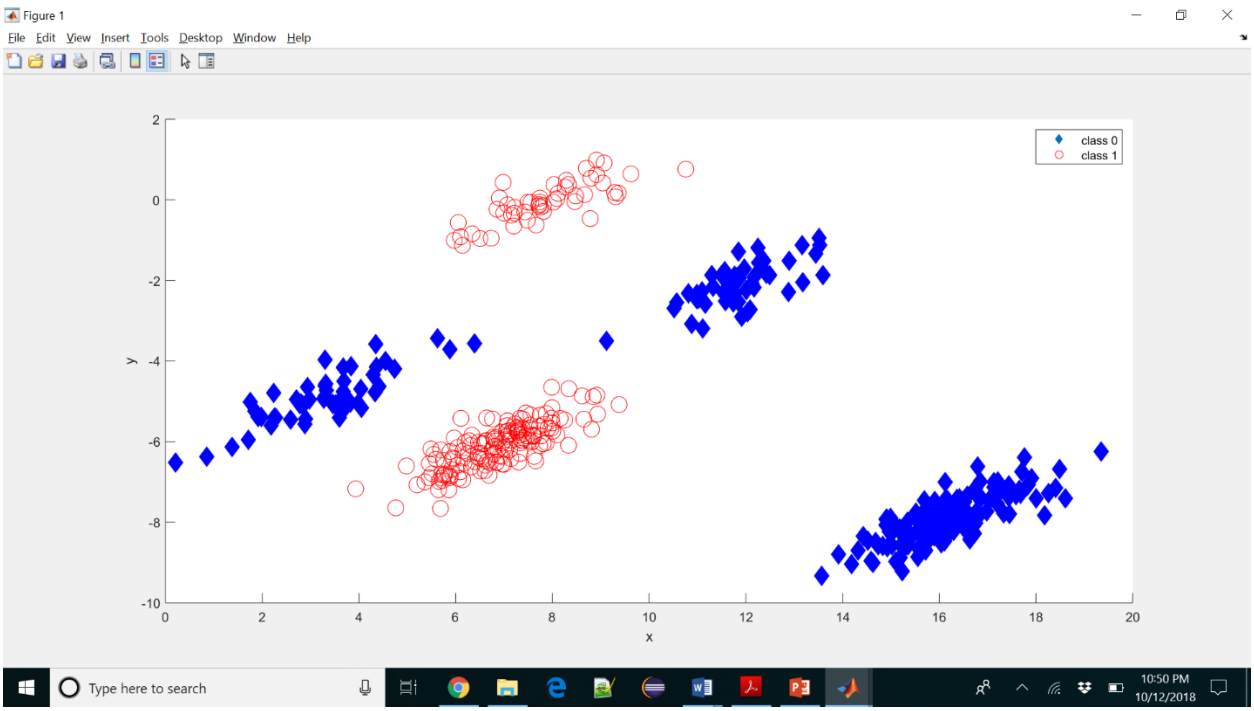
The dataset is uploaded in this project folder.

1) Visualizing the data through a 2D scatter plot:

MATLAB CODE:

```
% the given data is imported into a table named HW2data
HW2 = table2array(HW2data);
x = HW2(:,1);
y = HW2(:,2);
class = HW2(:,3);
scatter(x(class == 0), y(class == 0), 150, 'filled', 'd', 'b');
hold on;
scatter(x(class == 1), y(class == 1), 150, 'r', 'b');
legend('class 0', 'class 1');
xlabel('x');
ylabel('y');
```

OUTPUT:



2) 5-fold testing to find the best decision tree to fit to this dataset

To perform this task PARAMETER TUNING (Adjusting thee parameters in the fitctree function til the best tree is achieved) is done

My observations are as follows :

Parameter : MaxNumSplits

value	Average accuracy
2	0.9876
4	0.9907
6	0.9942
8	0.9929

- The MaxNumSplits means the decision tree has no more than the branches specified by MaxNumSplits or there can be fewer branches.
- The decision tree cannot have more branches (or node) than the maximum number of splits
- From the above table, it is clear that, when MaxNumSplits is set to 6, the average accuracy calculated over k fold decision trees accuracy is the highest.

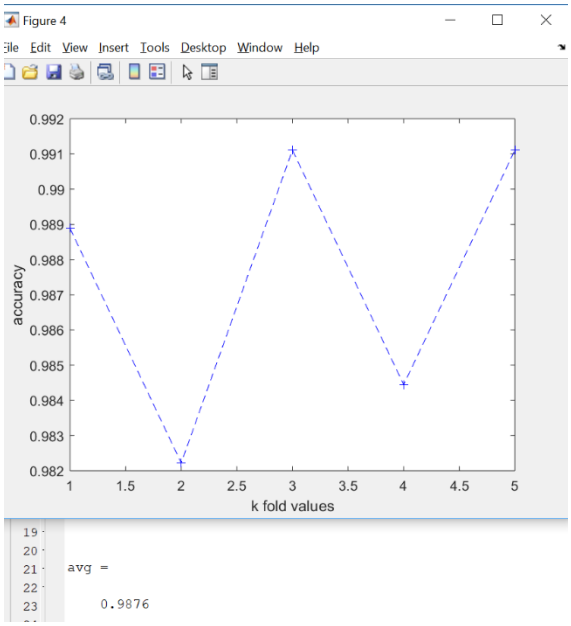
Thus, MaxNumSplits is set to 6.

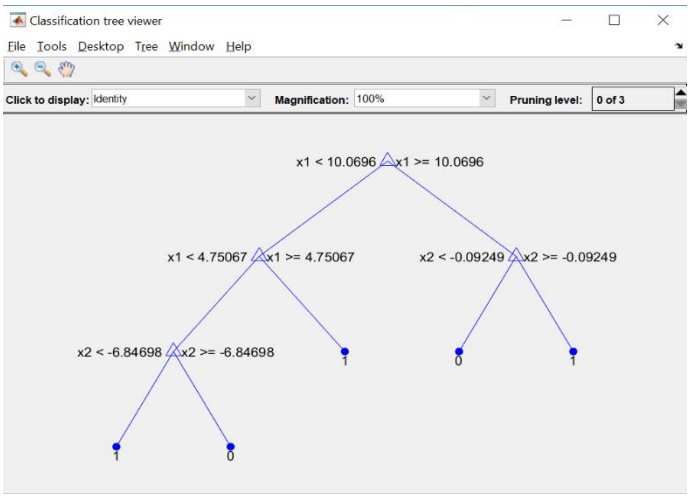
Now, more parameters tuning is performed in order to increase the accuracy to it's highest value.

The values specified in the above table are results implemented in MATLAB and the output is as follows:

When MaxNumSplits is 4

Here, no more than 4 branches are possible. The highest accuracy is given by tree with k fold 3, which is shown below



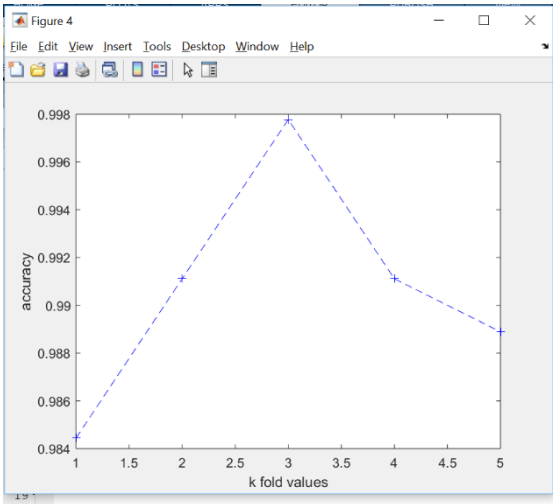



```
avg =

    0.9876
```

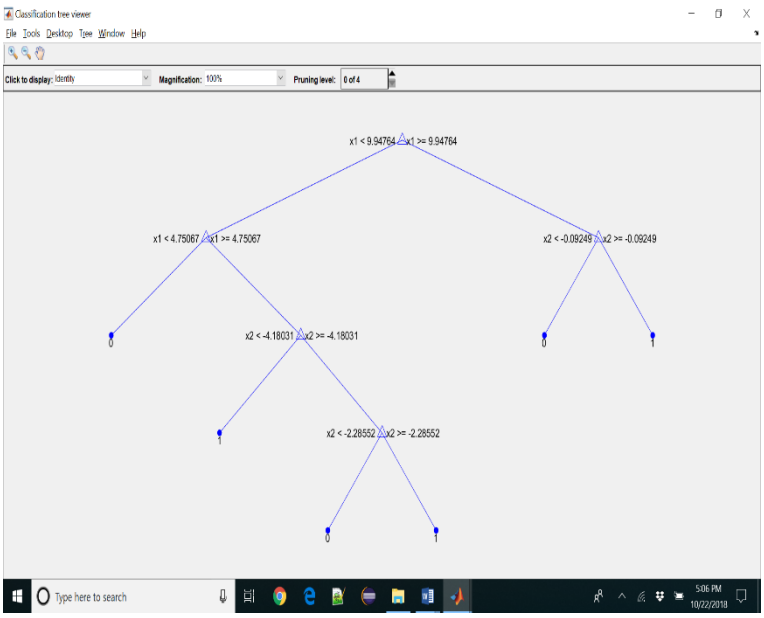
Here, avg in the output is the average accuracy given by the k fold decision trees. This is needed to compare the decision trees with some parameter to another set of k fold decision trees, so that we come to know how parameter needs to be tuned.

In order to improve the accuracy the number of splits are increased to 5

When MaxNumSplits is 5



In the



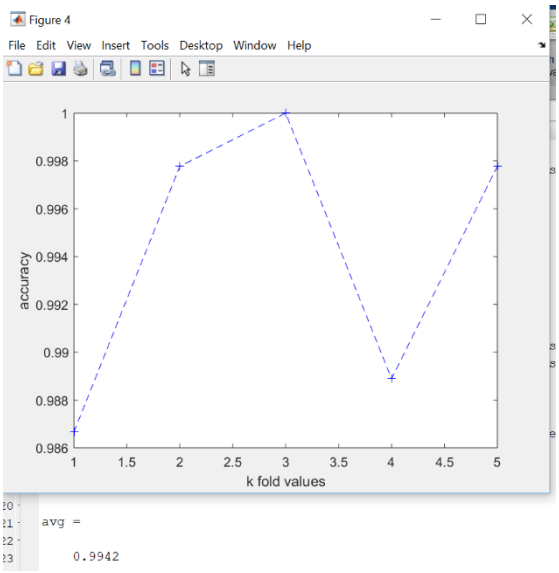

```
avg =

    0.9907
```

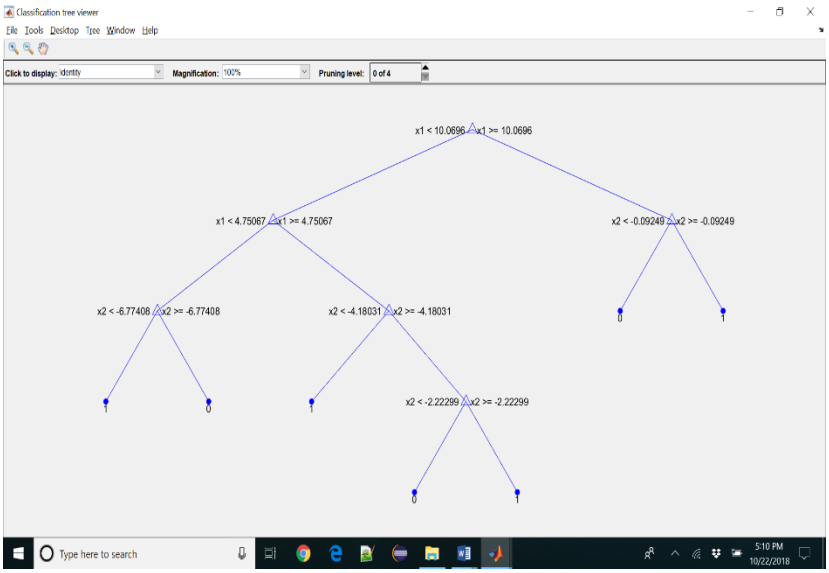
above decision tree no more than 5 branches are possible.

The tree which performs the best according to the accuracy is with the k fold value 3, shown in above figure.

To improve on accuracy further, next maximum number of splits are increased to 6 an we check avg accuracy

When MaxNumSplits is 6

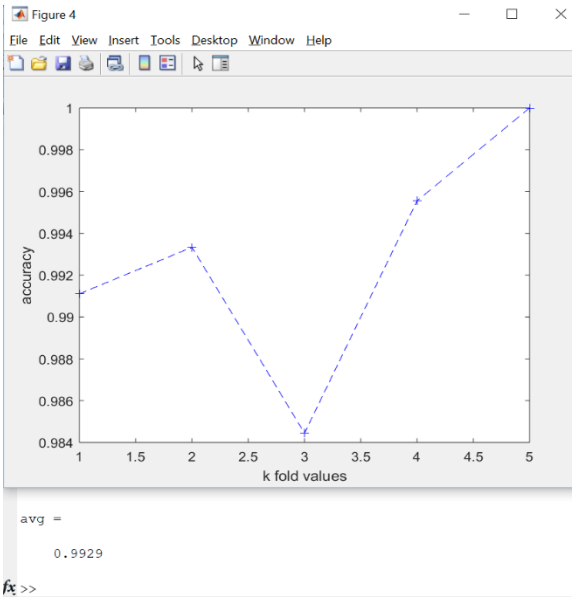


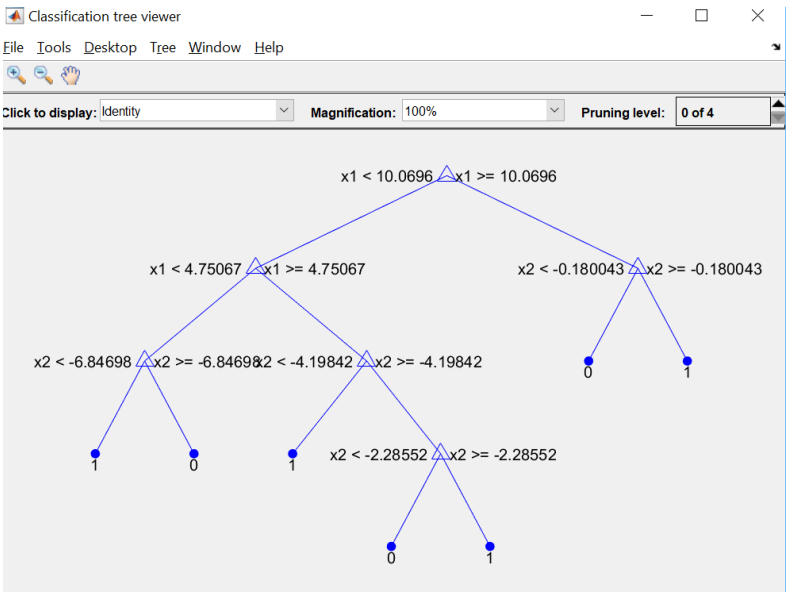


decision tree with k fold value 3 performs best and shown above

In order to improve the accuracy the number of splits are increased to 7 and avg accuracy is checked.

When MaxNumSplits is 7





There is a decrease in the value of average accuracy. If we split on further, it will be jut overfitting the data and decreasing the average accuracy also when the new features come in.

Conclusion: From the above observations the MaxNumSplits should be set as 6 to best fit our data.

Parameter: MinParentSize

Now I take up another parameter to tune in which is **MixParentSize**. The default value of MinParentSize in fictree function is 10. My aim is to change the value until a point that I get the best performance.

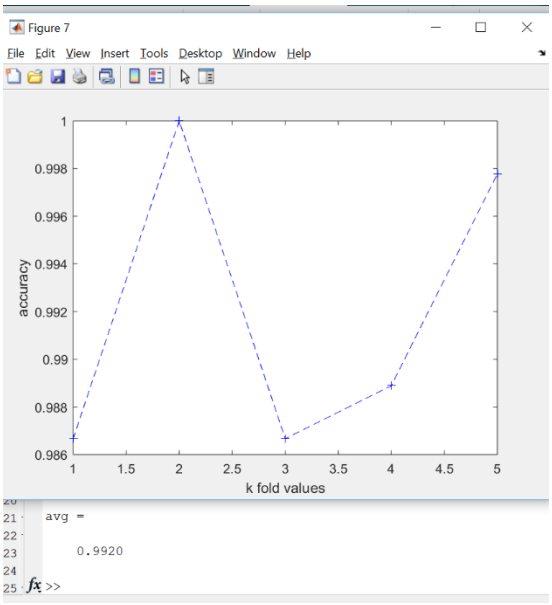
MinParentSize means the minimum number of data points to be present on that node for it to be parent, else it becomes leaf node.

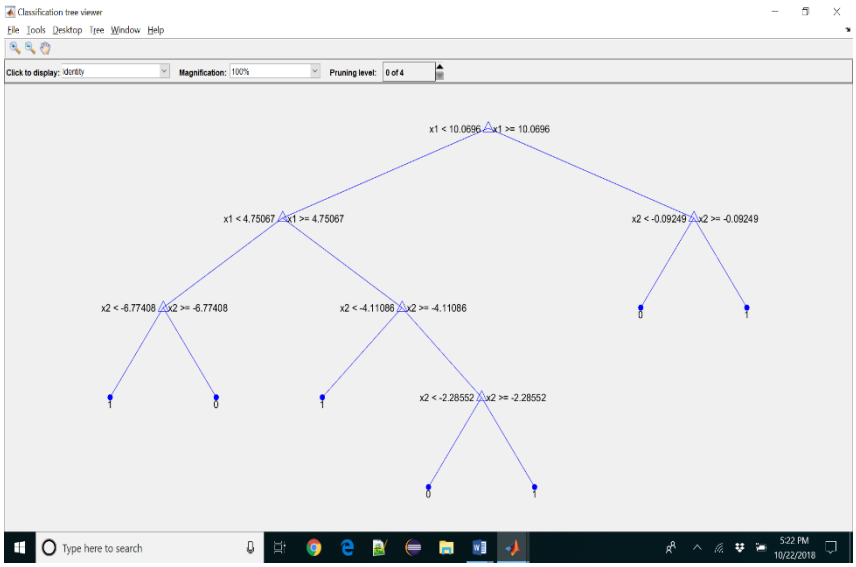
The value of MaxNumSplits I set as 6 here as it gives best accuracy until now.

MinParentSize

value	Average accuracy
30	0.9920
20	0.9929
15	0.9960
12	0.9942

MinParentSize 30

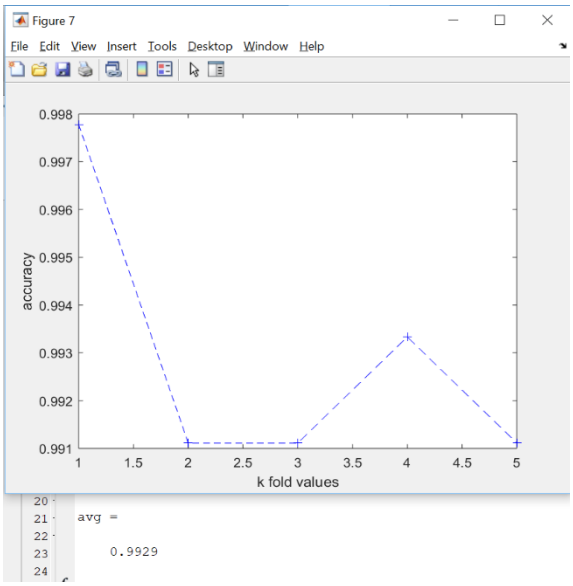


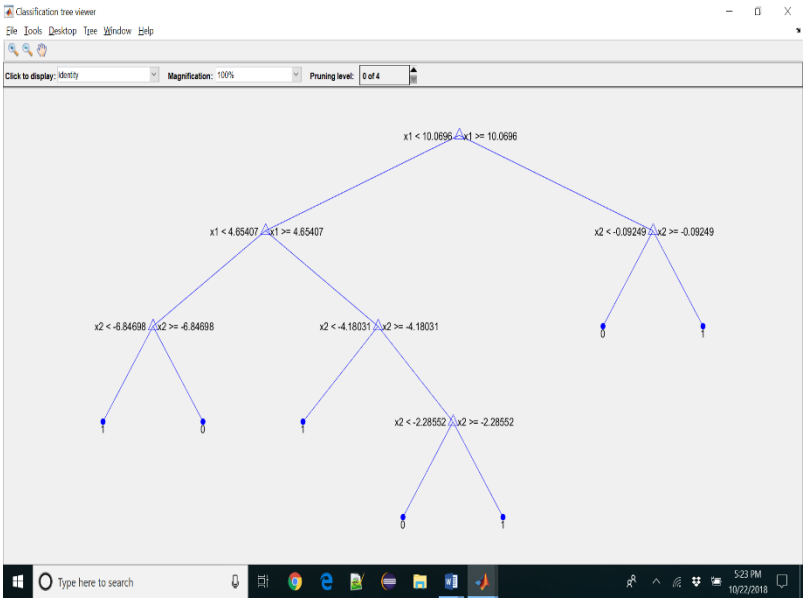


The overall depth of the tree decreases and this gives rise to underfitting as the features are not classified. More focus is on combining data sets and assigning it to one parent node.

The accuracy decreases, thus value is changed to 20 and aaccuracy is checked

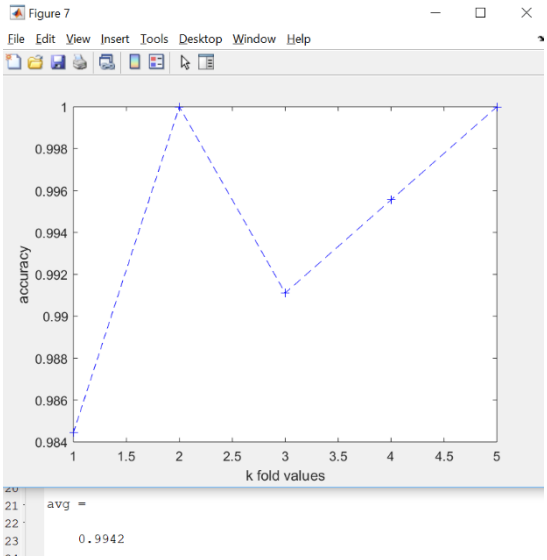
MinParentSize : 20

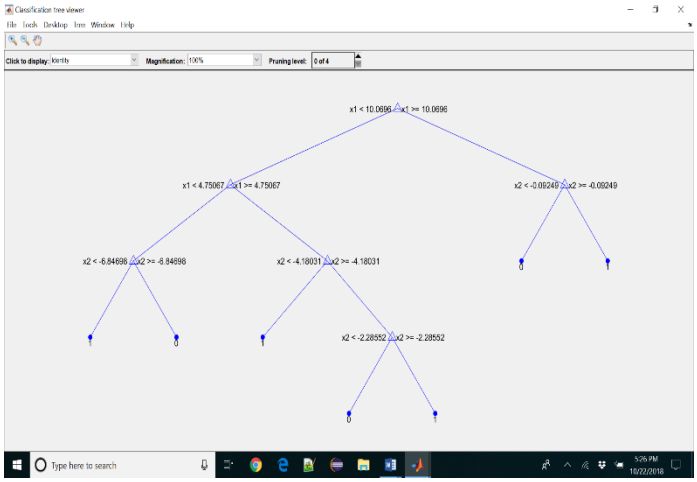




The accuracy is better than the previous case, but still not the best. Thus MinParentSize is further reduced

MinParentSize : 12

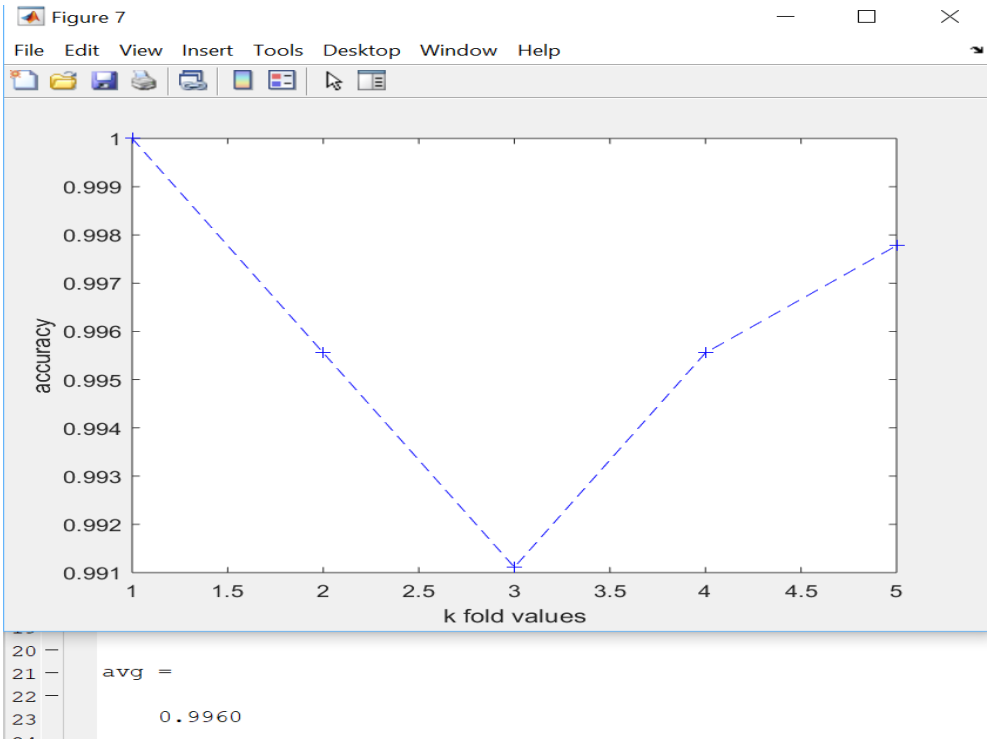




The avg accuracy now increases, there might be a case in between the MinParentSize 12 and 20 where the accuracy starts increasing.

MinParentSize 15 and MaxNumSplits 6

Figure: recall plotted against the k fold values



There is a noticeable change in the average accuracy which increases to 99.60%

Thus, this is the best case

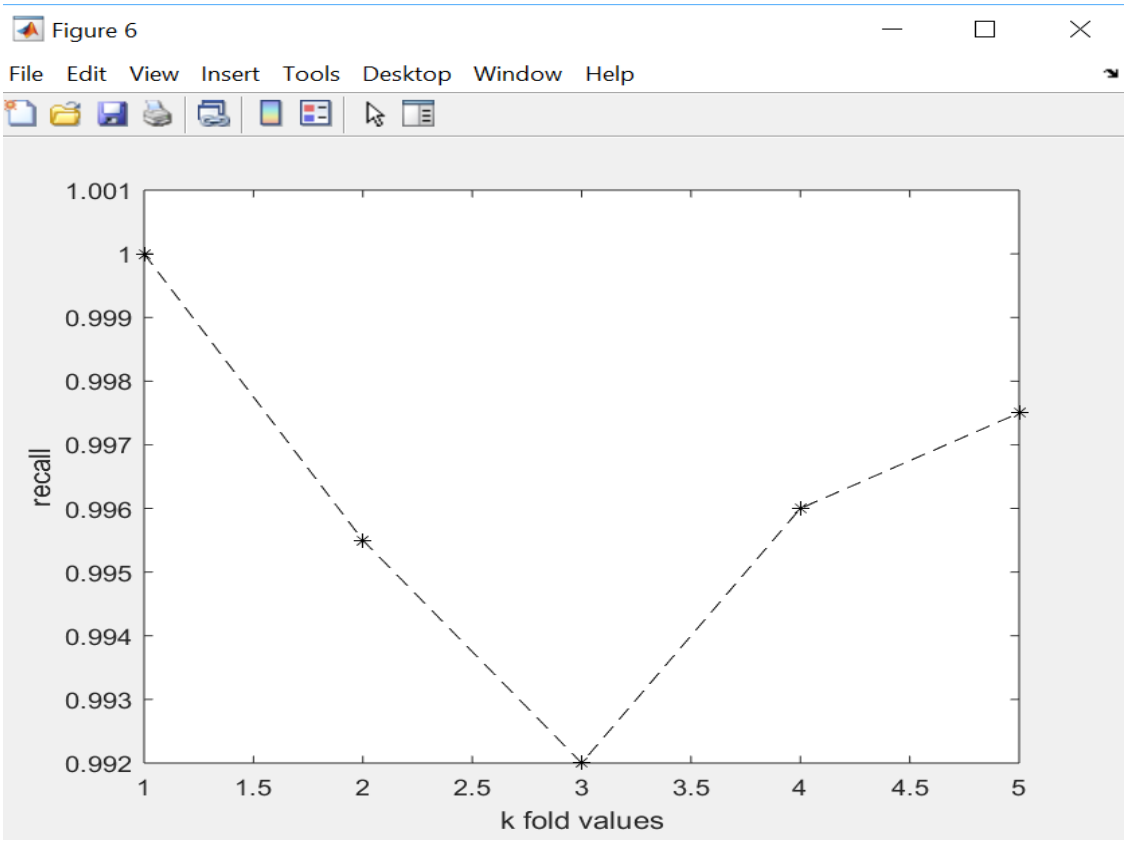


Figure: recall plotted against the k fold values

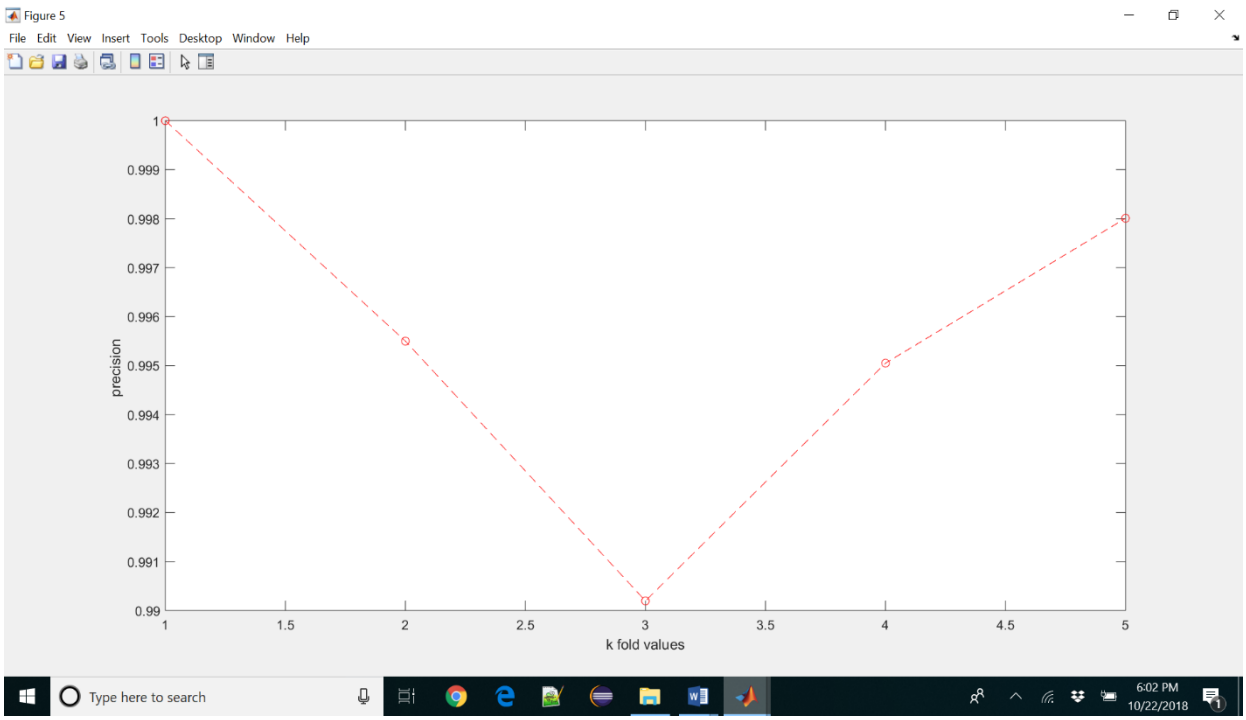
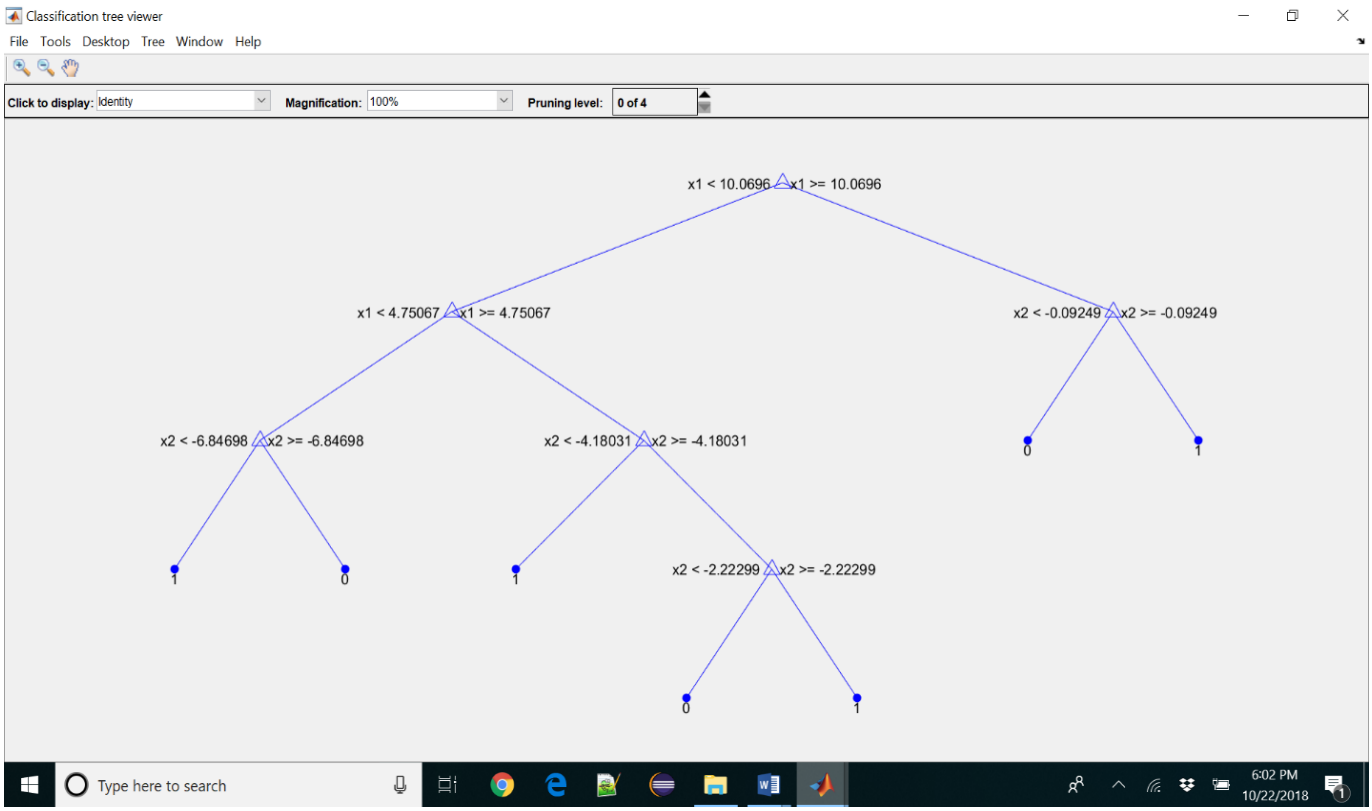


Figure: precision plotted against the k fold values



The Decision tree as viewed in Command Prompt :

```
Decision tree for classification
 1  if x1<10.0696 then node 2 elseif x1>=10.0696 then node 3 else 0
 2  if x1<4.75067 then node 4 elseif x1>=4.75067 then node 5 else 1
 3  if x2<-0.09249 then node 6 elseif x2>=-0.09249 then node 7 else 0
 4  if x2<-6.84698 then node 8 elseif x2>=-6.84698 then node 9 else 0
 5  if x2<-4.18031 then node 10 elseif x2>=-4.18031 then node 11 else 1
 6  class = 0
 7  class = 1
 8  class = 1
 9  class = 0
10  class = 1
11  if x2<-2.22299 then node 12 elseif x2>=-2.22299 then node 13 else 1
12  class = 0
13  class = 1
```

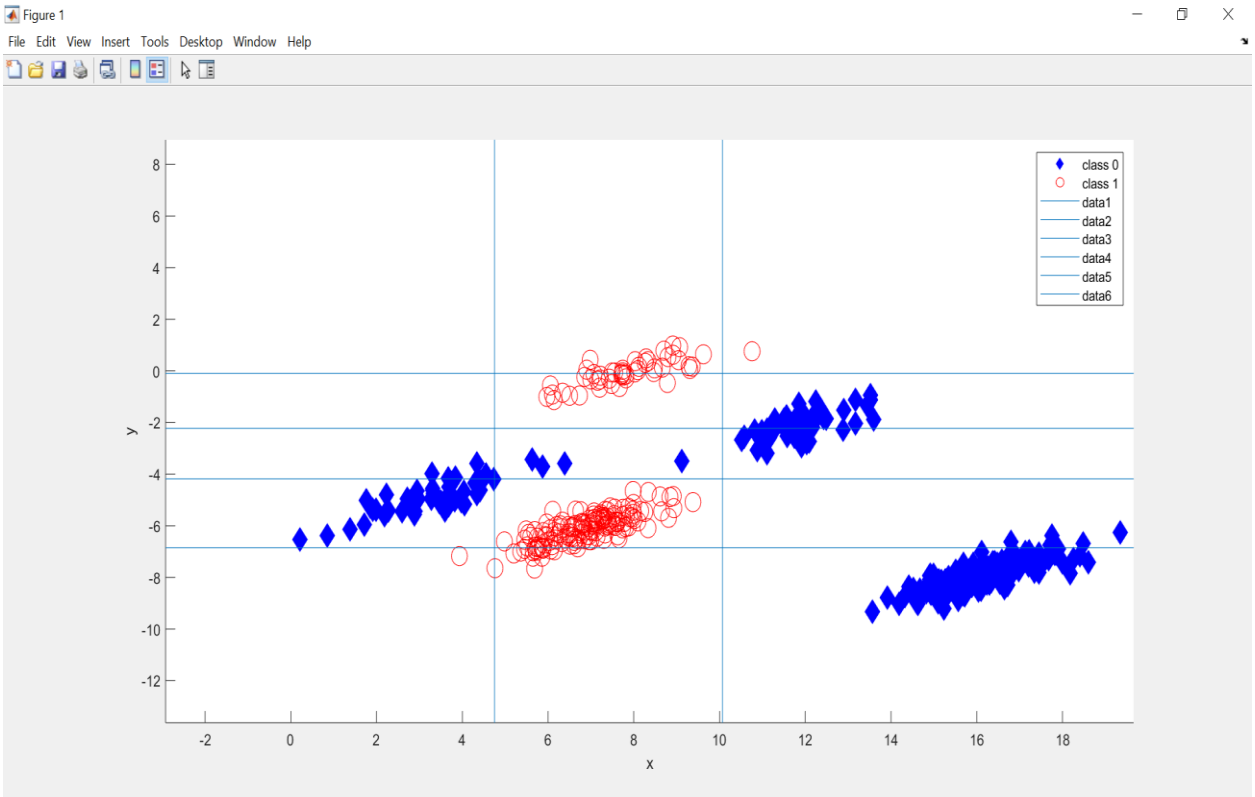
Code in MATLAB for the best decision tree output with parameter tuning:

```
HW2 = table2array(HW2data);
% assigning the Predictor Data to X
X = HW2(:,1:1:end-1);
% assigning the Class Labels to Y
Y = HW2(:,end);

%trees = fitctree(X,Y,);
tree1 = fitctree(X,Y,'MaxNumSplits',6, 'MinParentSize', 15);
tree = crossval(tree1, 'Kfold', 5);
%there are 2 ways to view a decision tree
view(tree.Trained{1}) %this is the text representation which shows up in the Command Window
view(tree.Trained{2})
view(tree.Trained{3})
view(tree.Trained{4})
view(tree.Trained{5})

view(tree.Trained{1},'Mode','graph'); %this is the graphical representation which shows up in the
classification tree viewer tab

view(tree.Trained{2},'Mode','graph');
view(tree.Trained{3},'Mode','graph');
view(tree.Trained{4},'Mode','graph');
```

```
view(tree.Trained{5},'Mode','graph');

%predict function is used to predict the class of test data
label1 = predict(tree.Trained{1}, X);
% confusion matrix is created which is then further used to calculate
% precision, recall and accuracy
c1 = confusionmat(Y,label1);
[p1, r1, pn1, rn1 ,q1 ,pf1, rf1] = measure(c1);

label2 = predict(tree.Trained{2}, X);
c2 = confusionmat(Y,label2);
[p2, r2, pn2, rn2, q2, pf2, rf2 ] = measure(c2);

label3 = predict(tree.Trained{3}, X);
c3 = confusionmat(Y,label3);
[p3, r3, pn3, rn3 ,q3 ,pf3, rf3] = measure(c3);

label4 = predict(tree.Trained{4}, X);
c4 = confusionmat(Y,label4);
[p4, r4, pn4, rn4 ,q4 ,pf4, rf4] = measure(c4);

label5 = predict(tree.Trained{5}, X);
c5 = confusionmat(Y,label5);
[p5, r5, pn5, rn5 ,q5 ,pf5, rf5] = measure(c5);


%precesion, recall and accuracy values of decision trees with training data 1,2,3,4,5 respectively
p = [pf1,pf2,pf3,pf4,pf5];
r = [rf1,rf2,rf3,rf4,rf5];
q = [q1,q2,q3,q4,q5];
t = [1,2,3,4,5];
% a combined plot of precision, recall and accuracy against min records per
% leaf node
plot(t,p, '--or');
hold on;
plot(t,r, '--*k');
plot(t,q, '--+b');
xlabel('k fold values');
f2 = figure();
f2 = plot(t,p, '--or');% precision against min records per
% leaf node
xlabel('k fold values');
ylabel('precision ');
f3 = figure();
f3 = plot(t,r, '--*k');% recall against training data
xlabel('k fold values');
ylabel('recall');
```

```
avg = (q1 + q2 + q3 + q4 + q5)/5

f8 = figure();
f8 = plot(t,q,'--+b'); % accuracy against min records per
% leaf node

xlabel('k fold values');
ylabel('accuracy');

%%to calculate precision, recall and accuracy. I have created a
%%function that takes the confusion matrix as input and gives the
%%precision, recall and accuracy as output. x corresponds to the
%%confuion matrix sent as an argument to the function.

function [p, r, pn, rn, a, pf, rf] = measure(x)
tp = x(1,1); %true positive
fn = x(1,2); %false negative
fp = x(2,1); %false positive
tn = x(2,2); %true negative
p = tp/(tp + fp); %positive precision value
r = tp/(tp + fn); %positive recall value
pn = tn/(tn + fn); %negative precision value
rn = tn/(tn + fp); %negative recall value
a = (tp + tn)/ (tp + tn + fp + fn);%accuracy
pf = (p + pn)/2; %mean precision
rf = (r + rn)/2; %mean recall
end
```

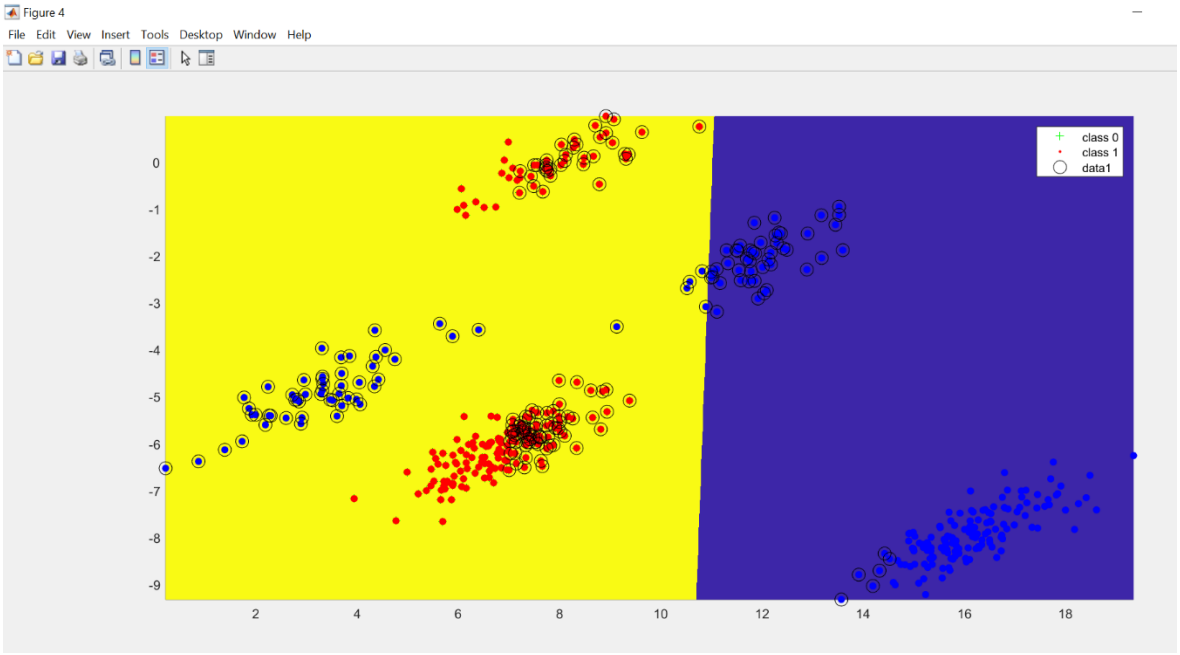
3) Visualizing the boundaries learnt by the best decision tree




```
% the given data is imported into a table named
HW2data
HW2 = table2array(HW2data);
x = HW2(:,1);
y = HW2(:,2);
class = HW2(:,3);
scatter(x(class == 0), y(class == 0), 150,
'filled', 'd', 'b');
hold on;
scatter(x(class == 1), y(class == 1), 150, 'r',
'b');
legend('class 0', 'class 1');
xlabel('x');
ylabel('y');
line([10.0696,10.0696],[-20,20 ]);
```

```
line([4.75067,4.75067],[-20,20 ]);
line([-20,20 ],[-0.09249,-0.09249]);
line([-20,20], [-6.84698,-6.84698]);
line([-20,20], [-4.18031, -4.18031]);
line([-20,20], [-2.22299, -2.22299]);
```

a) Using SVM to classify data



The support vectors are encircled in black. The blue dots represent class 0. Red dots represent class 1

There are 2 regions in which the svm separates the points, represented by yellow and blue in the above figure.

There are 55 misclassified points where the class 0 points are classified as class 1.

The accuracy of the bet classifier after the cross validation using k fold is 87.77%

The parameter tuning is done using **c(class weight) and the slack variable.**

It turns out that soft margin approach is followed here in order to classify certain amount of error is acceptable.

But the svm given by the default parameter values is the best one and s shown in above diagram

Code is as follows:

```
HW2 = table2array(HW2data);
% assigning the Predictor X to X
X = HW2(:,1:1:end-1);
% assigning the Class Labels to Y
Y = HW2(:,end);

x = HW2(:,1);
y = HW2(:,2);
class = HW2(:,3);
scatter(x(class == 0), y(class == 0),30,'filled',
'b');
hold on;
scatter(x(class == 1), y(class == 1),30,'r');
legend('class 0', 'class 1');
hold off

SVMModel = fitcsvm(X,Y, 'KernelFunction',
'linear');
linear_out=predict(SVMModel,X);
```

```
[~,acc_linear,fm_Linear]=confusionMatrix(Y,linear_o
ut);

x1range=min(X(:,1)):0.005:max(X(:,1));
x2range=min(X(:,2)):0.005:max(X(:,2));
[x1 x2]=meshgrid(x1range,x2range);
X1n=reshape(x1,1,size(x1,2)*size(x1,1));
X2n=reshape(x2,1,size(x2,2)*size(x2,1));
Xn=[X1n' X2n'];
yn=predict(SVMModel,Xn);
Yn=reshape(yn,size(x1));
figure
hold on
imagesc([min(X(:,1)) max(X(:,1))], [min(X(:,2))
max(X(:,2))], Yn);
scatter(X(Y==1,1),X(Y==1,2),'+g')
scatter(X(Y==0,1),X(Y==0,2),'.r')
axis([min(X(:,1)) max(X(:,1)) min(X(:,2))
max(X(:,2))])
hold on
x = HW2(:,1);
y = HW2(:,2);
class = HW2(:,3);
scatter(x(class == 0), y(class == 0),30,'filled',
'b');
hold on;
scatter(x(class == 1), y(class ==
1),30,'filled','r');
hold on
sv = SVMModel.SupportVectors;

plot(sv(:,1),sv(:,2),'ko','MarkerSize',10)


function [cm acc fm ] = confusionMatrix(
y_actual,y_predicted )
tp=0;tn=0;fp=0;fn=0;
for i=1:length(y_actual)
```

```
if y_actual(i)>0
    if y_actual(i)==y_predicted(i)
        tp=tp+1;
    else
        fn=fn+1;
    end
end
if y_actual(i)<0
    if y_actual(i)==y_predicted(i)
        tn=tn+1;
    else
        fp=fp+1;
    end
end
end
disp(tn)
disp(tp)
disp(fn)
disp(fp)

cm=[tn fp; fn tp];
acc=(tp+tn)/(tp+fn+fp+tn);
sens=tp/(tp+fn);prec=tp/(tp+fp);
fm=(2*prec*sens)/(prec+sens);
end
```

b) Using rbf Kernel in SVM

Since the data is **non linearly separable**, this is the **best** method to separate out the data.

Parameter tuning is shown as follows:

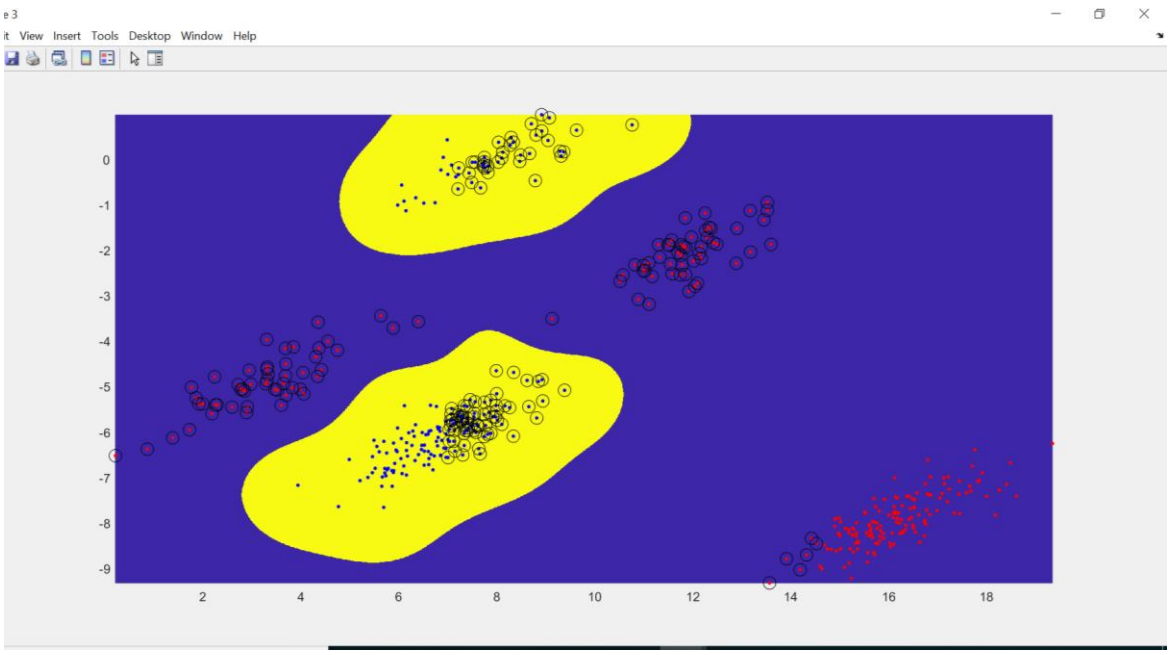
After using the k fold validation method (5 folds), the accuracy of all the kfold validated svm's came out to be 1

For all following figures:

The blue points represent class 1

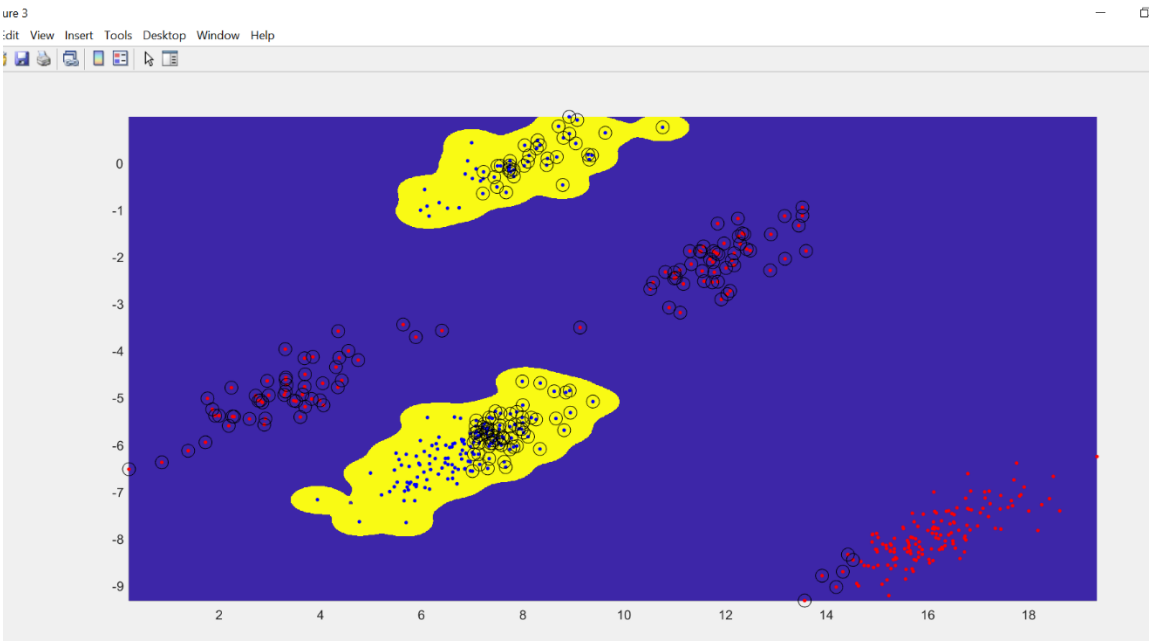
The red points represent class 0

The support vectors are encircled by black color



I have shown what changes might be using parameter tuning below.

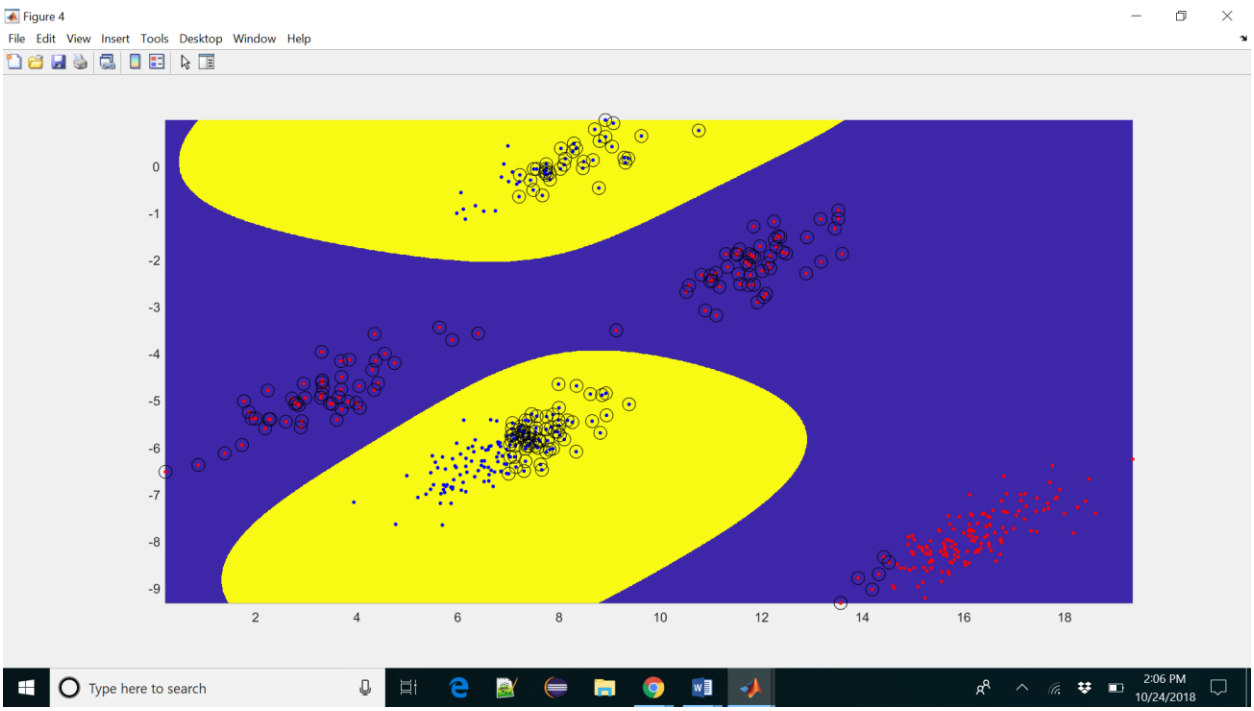
1)Thus, when the **Standardize parameter** is set to false. The following svm is output. Although, accuracy remains the same. But it will not be a good classifier if new data points apart from 450 points tested.



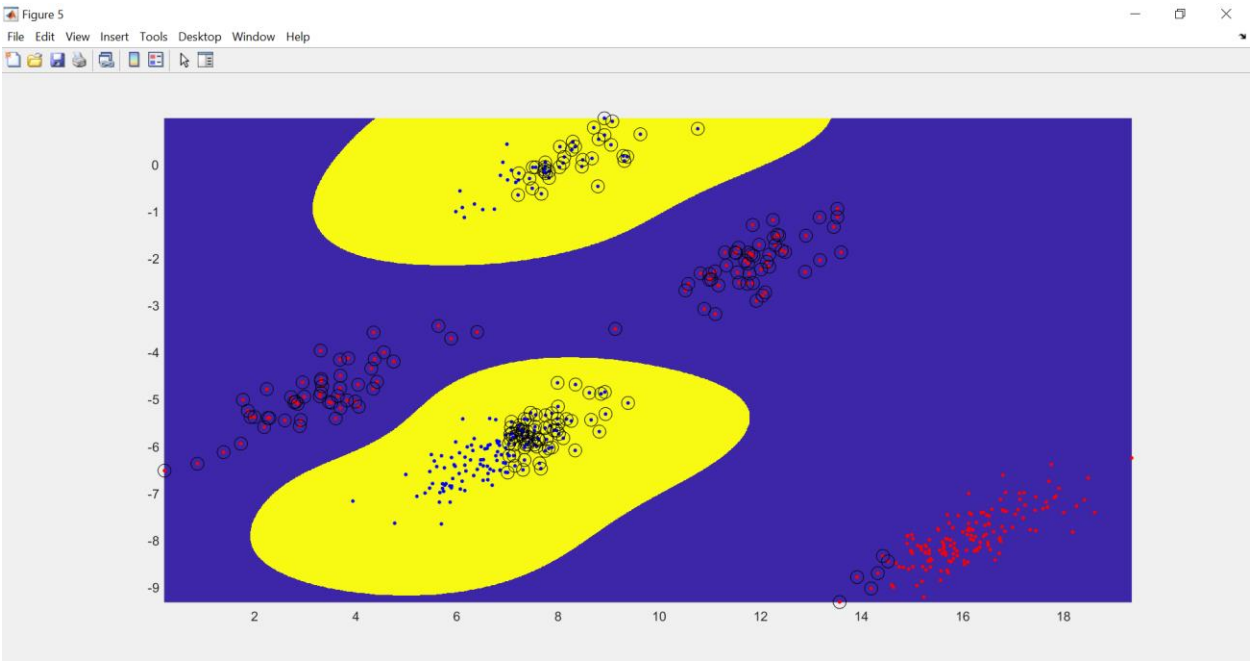
So, let us move on to changing another parameter

2)Kernel scale: The software divides all elements of the predictor matrix `X` by the value of `KernelScale`. Then, the software applies the appropriate kernel norm to compute the Gram matrix. (from Matlab Documentation)

If the parameter KernelScale is not applied. Then the following figure is the output. This again results in covering the datapoints in class 1 and classifying them to 0 , in case when new data is added to test.



3) Set Standardize to true and KernelScale to auto



The above result is the best result

Also, from output of accuracy we can see that accuracy is 1, it gives the best performance amongst all.

The Code is as follows(code is in MATLAB)

```
HW2 = table2array(HW2data);
% assigning the Predictor X to X
X = HW2(:,1:1:end-1);
% assigning the Class Labels to Y
Y = HW2(:,end);
x = HW2(:,1);
y = HW2(:,2);
class = HW2(:,3);
scatter(x(class == 0), y(class == 0),30,'filled', 'b');
hold on;
scatter(x(class == 1), y(class == 1),30,'r');
legend('class 0', 'class 1');
hold off
rbf_in=fitcsvm(X,Y,'KernelFunction','rbf', 'KernelScale','auto','Standardize',true);


rbf_out=predict(rbf_in,X);
[~,acc_RBF,fm_RBF]=confusionMatrix(Y,rbf_out)
x1range=min(X(:,1)):0.005:max(X(:,1));
x2range=min(X(:,2)):0.005:max(X(:,2));
[x1 x2]=meshgrid(x1range,x2range);
X1n=reshape(x1,1,size(x1,2)*size(x1,1));
X2n=reshape(x2,1,size(x2,2)*size(x2,1));
Xn=[X1n' X2n'];
yn=predict(rbf_in,Xn);
Yn=reshape(yn,size(x1));
figure
hold on
imagesc([min(X(:,1)) max(X(:,1))], [min(X(:,2)) max(X(:,2))], Yn);
scatter(X(Y==1,1),X(Y==1,2),70,'.b')
scatter(X(Y==0,1),X(Y==0,2),70,'.r')
axis([min(X(:,1)) max(X(:,1)) min(X(:,2)) max(X(:,2))])
hold on
sv = SVMModel.SupportVectors;
plot(sv(:,1),sv(:,2),'ko','MarkerSize',10)
```

```
function [cm acc fm ] = confusionMatrix( y_actual,y_predicted )
tp=0;tn=0;fp=0;fn=0;
for i=1:length(y_actual)
if y_actual(i)>0
   if y_actual(i)==y_predicted(i)
      tp=tp+1;
   else
      fn=fn+1;
   end
end
if y_actual(i)<0
   if y_actual(i)==y_predicted(i)
      tn=tn+1;
   else
      fp=fp+1;
   end
end
end
cm=[tn fp; fn tp];
acc=(tp+tn)/(tp+fn+fp+tn);
sens=tp/(tp+fn);prec=tp/(tp+fp);
fm=(2*prec*sens)/(prec+sens);
end
```

CONCLUSION

The given data to be classified in **non linearly separable.**

Thus linear svm and decision trees are not so good classifiers for the data as the non-linear svm using RBF Kernel.

The conclusion is that non-linear svm using RBF Kernel is the best classifier.

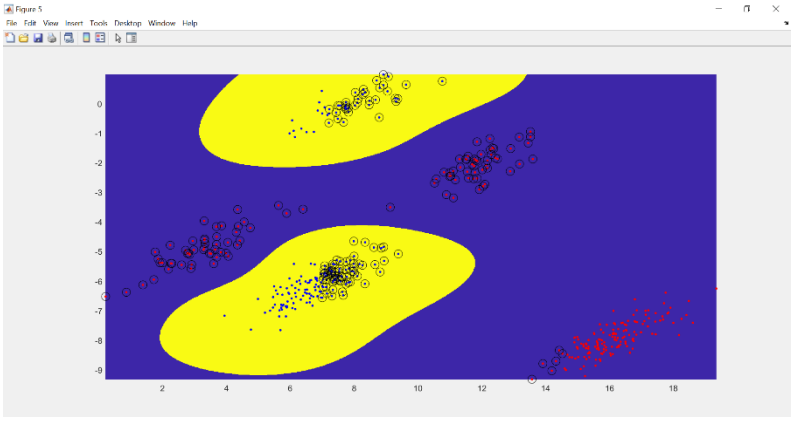


Figure 1) non-linear SVM

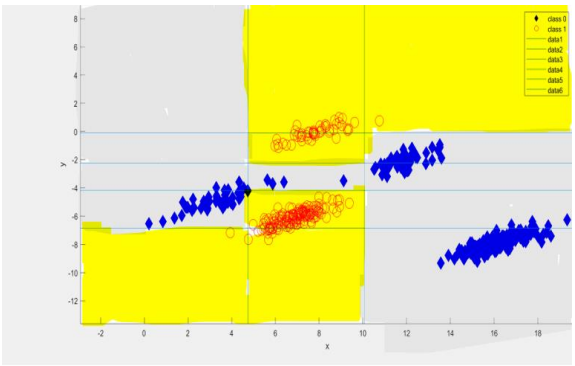


Figure 2) decision tree using

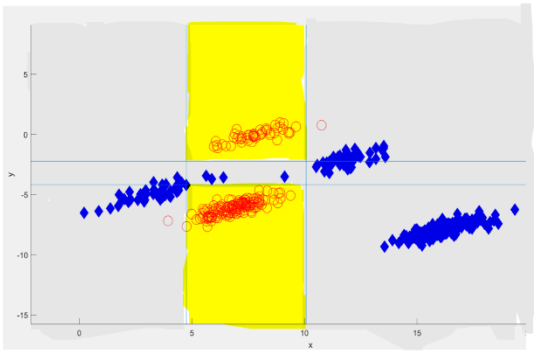


Figure 3)Ideal decision tree

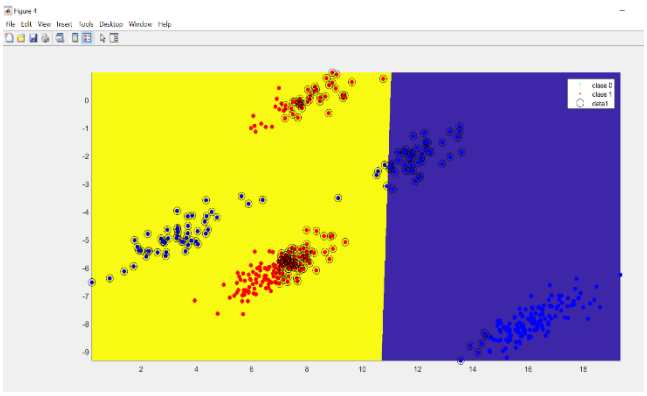


Figure 3) linear SVM

From the above figures it is clearly visible that non-linear svm has the highest accuracy and it completely correctly classifies the data to their respective classes

The accuracy is as follows

non-linear SVM using Kernel RBF	Decision tree(k fold)	Ideal Decision Tree	Linear SVM
1 (accuracy)	0.9960 (Accuracy)	0.9955 (accuracy)	0.8777 (accuracy)
Avoids overfitting and underfitting	Results in overfitting	Avoids overfitting	Underfits data
Classify all points correctly	Classifies 99.6%points correctly	Misclassifies 2 points	Misclassifies lot of points
Best classifier	Good classifier	Good Classifier	Bad classifier
lowest training error	Low training error	Higher training error	Highest training error

Lowest test error	**Low test error**	**Low test error**	**Highest testing error**