In this problem, the first step is to write functions for calculation of entropy based on class distribution or based on the probabilities of the split obtained.

Next, we deploy our entropy calculators to determine the best attribute along which we split the data at a node. This is done by using the probabilities of the split data. We also ensure that the quality of the split is not the only parameter on which a decision attribute is selected. We also ensure that the respective bucket sizes are taken into consideration when a split occurs. The goal is to identify the attribute which splits the MOST amount of data in the BEST possible way.

After getting the desirable attribute, we train our decision tree. In the training, we recursively call the function which keeps on splitting the nodes based on the best attribute. Here, the stopping criteria is used to terminate the recursion. The criteria are :

1) When the split is perfect i.e there are only 1s or only 0s at the leaf node.
2) We have run out of attributes to split the data and we have yet not achieved a perfect split at the leaf nodes. In this case, we assign the decision based on majority vote.
3) In addition, to combat overfitting, I have used another criterion. If the number of data points at the leaf node is less than a specific threshold, we terminate the recursion. This ensures that we are not creating too many decision nodes if the buckets are small.
4) Another criterion which I have tried using is to limit the depth of the tree. This can be done by ensuring that we only use the best k out of the available n attributes such that $k < n$. In this case, n is 15 and k is 6.

After training, we deploy our decision tree on the given test samples. The results are documented below

Pseudo Codes:

1) entropy()

   Input: p-> The probabilities of the binary results

   P(0) is the probability that the data belongs to class 0
   P(1) is the probability that the data belongs to class 1

   H0 is the entropy of event 0 occurring
   H1 is the entropy of event 1 occurring

```
If P(0) = 0,
      H0 = 0
Else
      H0 = -p(0) * log(p(0))
If P(1) = 0,
      H1 = 0
Else
      H1 = -p(1) * log(p(1))

Return H0 + H1
```

2) entropy_of_class (class)

Input: class-> The class distribution of the given data

Class(0) is the number of 0s in the class distribution

Class(1) is the number of 1s in the class distribution

```
Let p0 = Class(0) / (Class(0) + Class(1))
    p1 = Class(1) / (Class(0) + Class(1))
```

```
P = [p0, p1]
return entropy(P)
```

//Use the function defined above

3) find_decision_attrib ()

let p_node be the class distribution at the node

h = entropy(p_node)

for i = available attributes

M_1_1 = No of data points which belong to class 1 given the attribute i is 1

M_1_0 = No of data points which belong to class 0 given the attribute i is 1

M_0_1 = No of data points which belong to class 0 given the attribute i is 1

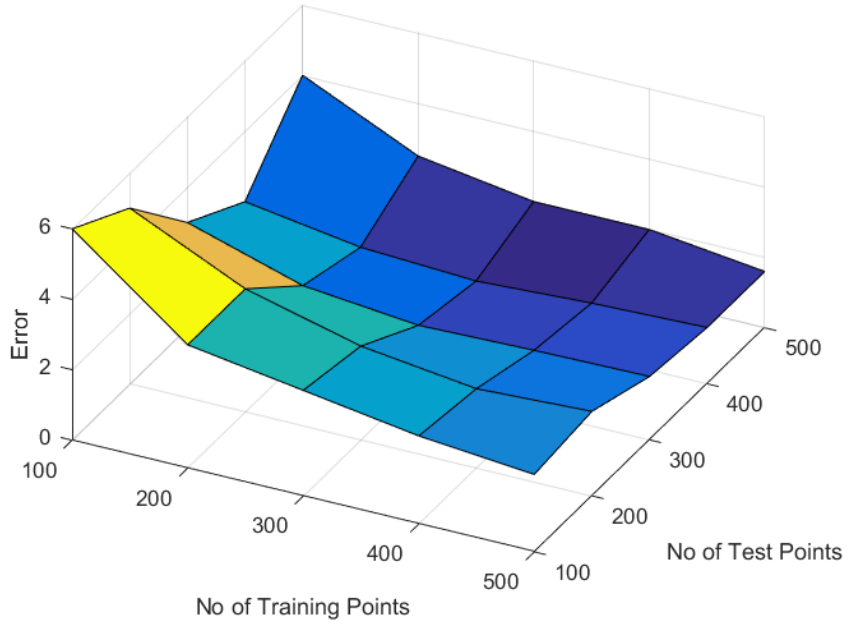
```
            M_0_0 = No of data points which belong to class 0 given the
            attribute i is 0

            Get the resulting probabilities P_left and P_right
            P_left_1 = M_1_1 / (M_1_1 + M_1_0)
            P_left_0 = 1 – p_left_1
            P_left = [P_left_1, P_left_0]
            Similarly, P_right = [P_right_1, P_right_0]

            Entropy of left H1 = entropy(p_left)
            Entropy of right H2 = entropy(p_right)

            Weight_left = M_1_1 + M_1_0
            Weight_right = M_0_1 + M_0_0

            H = Weight_left * P_left + Weight_right * P_right
            Info_gain(i) = h – H

End (Get next attribute)
Decision_attrib = max(Info_gain)
Available attributes = Available attributes – Decision_attrib

   4) Train()
   Input: attrib-> The attributes of the given data points
          Class-> The class values of all the given data data
   If (class = 1 for all points)
      Decision = 1;
   Return
   ElseIf (class = 0 for all points)
      Decision = 0;
   Return
```

```
ElseIf (available_attribs is empty)
   Decision = max(class = 1, class = 0);
Return
Else
   Find_decision_attrib(class, attrib)
   SubClass1 = (class == 1)
   SubClass0 = (class == 0)
   SubAttrib1 = (attrib == 1)
   SubAttrib0 = (attrib == 0)

Left_Node = new DecisionTreeNode()
Right_Node = new DecisionTreeNode()

This = parentOf(Left_Node)
This = parentOf(Right_Node)

Left_Node.train (SubClass1, SubAttrib1)
Right_Node.train (SubClass0, SubAttrib0)
```

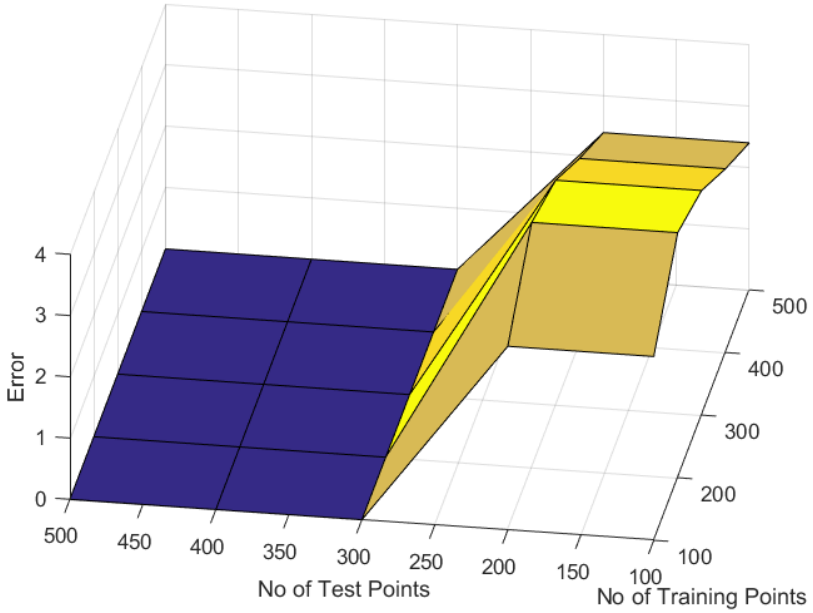
One of the bugs which I encountered in the implementation of the Decision tree was that in some cases, the information gain for all the attributes is the same. In this case, the program ran into an infinite recursion.

As discussed in lectures, this bug was resolved by selecting a random attribute among the available attributes and running the script as is.

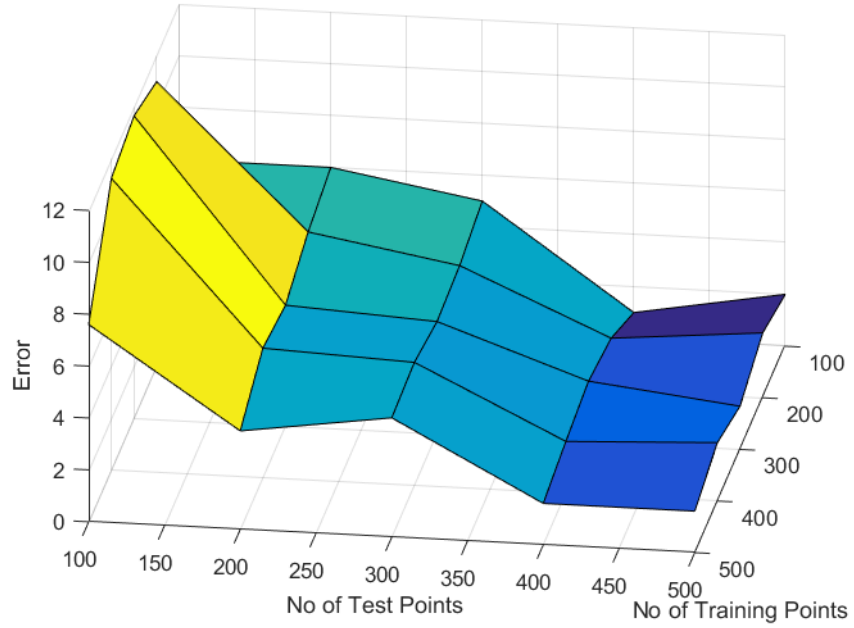
Results:



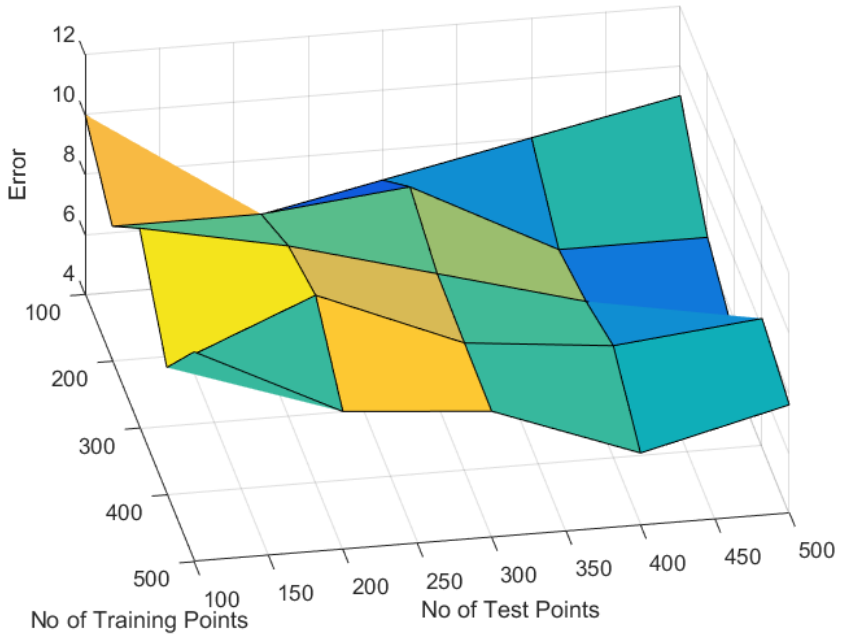
ORACLE 1		N Train				
		100	**200**	**300**	**400**	**500**
N Test	**100**	6	3.5	3	2.5	2.2
	200	5	3.5	2.666667	2.25	2.4
	300	3	2	1.666667	1.75	1.8
	400	2	1.5	1.333333	1.5	1.6
	500	4	2.5	2	2	1.6



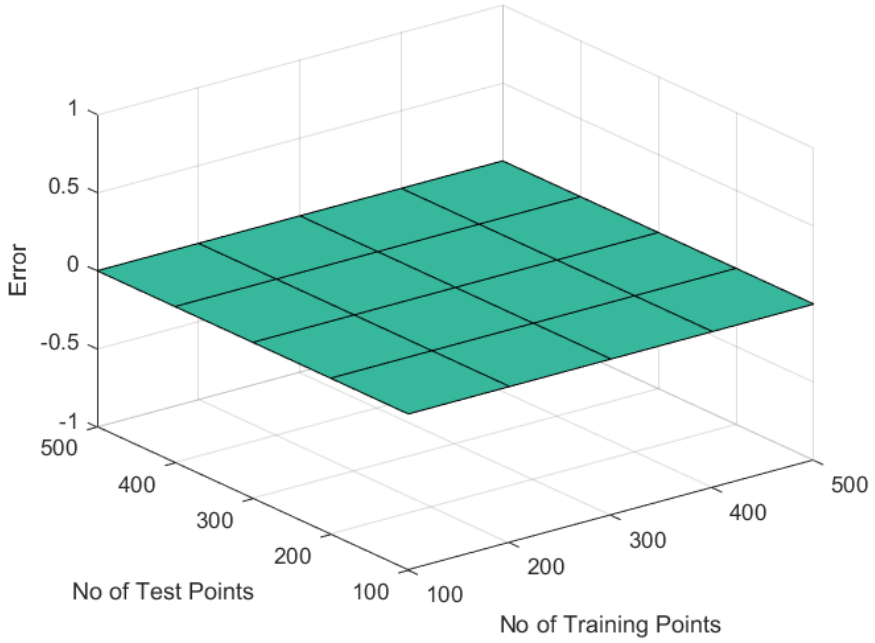
ORACLE 2		N Train				
		100	**200**	**300**	**400**	**500**
N Test	**100**	3	4	3.666667	3	2.4
	200	3	4	3.666667	3	2.4
	300	0	0	0	0	0
	400	0	0	0	0	0
	500	0	0	0	0	0



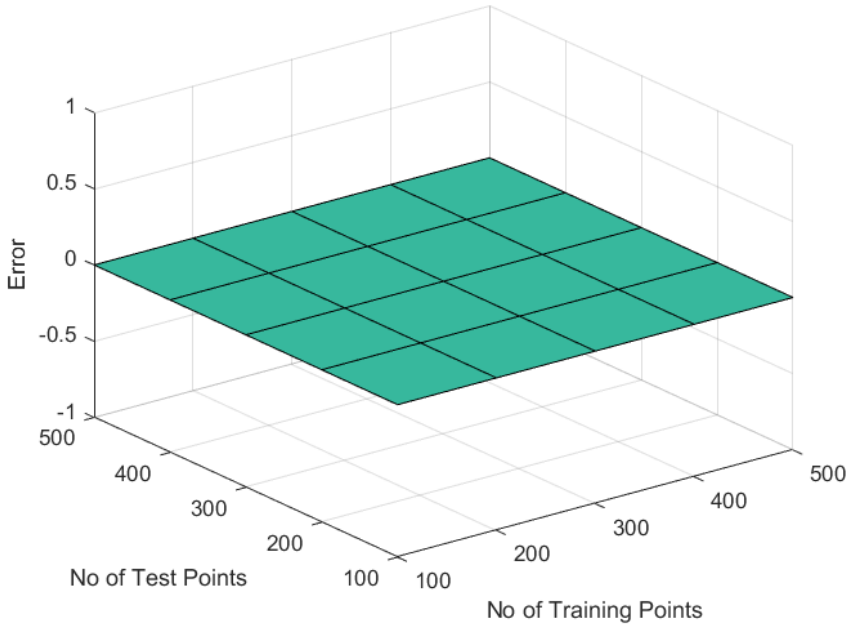
ORACLE 3		N Train				
		100	**200**	**300**	**400**	**500**
N Test	**100**	7	10	9.666667	8.5	7.6
	200	7	7	6.333333	3.25	3.6
	300	6	3.5	5	4.25	4.6
	400	0	1.5	1.333333	1.5	1.2
	500	2	2.5	2	1.5	1.2



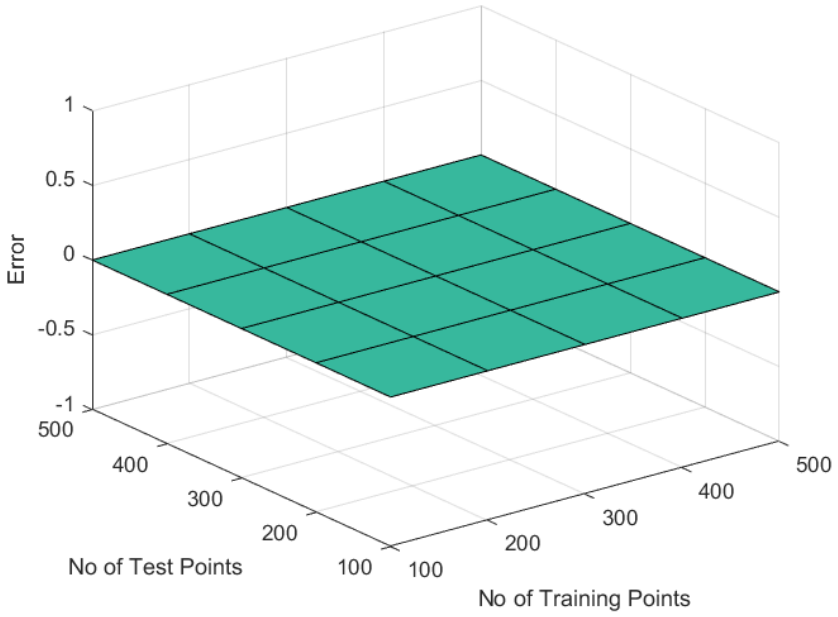
ORACLE 5		N Train				
		100	**200**	**300**	**400**	**500**
N Test	**100**	10	7.5	7	9.5	8.2
	200	6	8.5	7.333333	9.75	9.8
	300	7	7.5	8.666667	9.75	8
	400	8	8.5	6	8.25	6.2
	500	9	8	5.666667	7.5	8.4



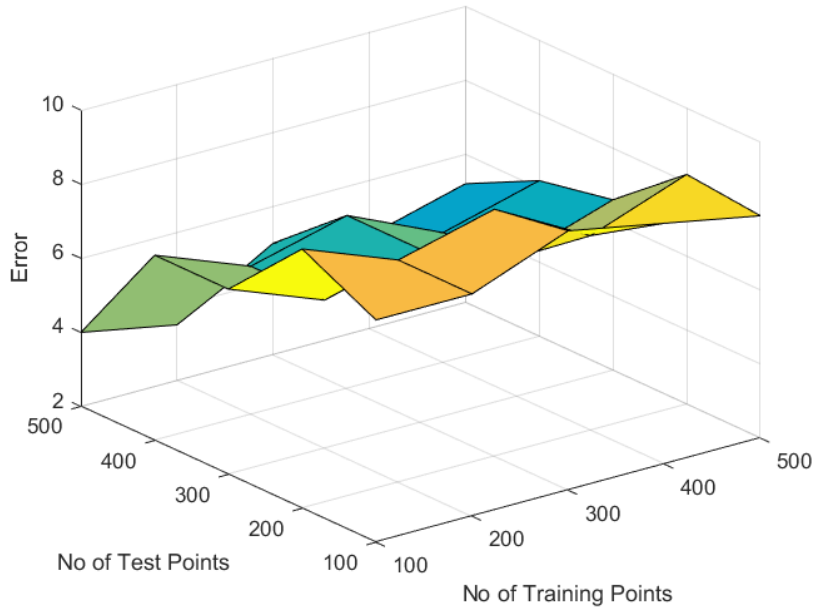
ORACLE 7		N Train				
		100	**200**	**300**	**400**	**500**
N Test	**100**	0	0	0	0	0
	200	0	0	0	0	0
	300	0	0	0	0	0
	400	0	0	0	0	0
	500	0	0	0	0	0



ORACLE 8		N Train				
		100	**200**	**300**	**400**	**500**
N Test	**100**	0	0	0	0	0
	200	0	0	0	0	0
	300	0	0	0	0	0
	400	0	0	0	0	0
	500	0	0	0	0	0

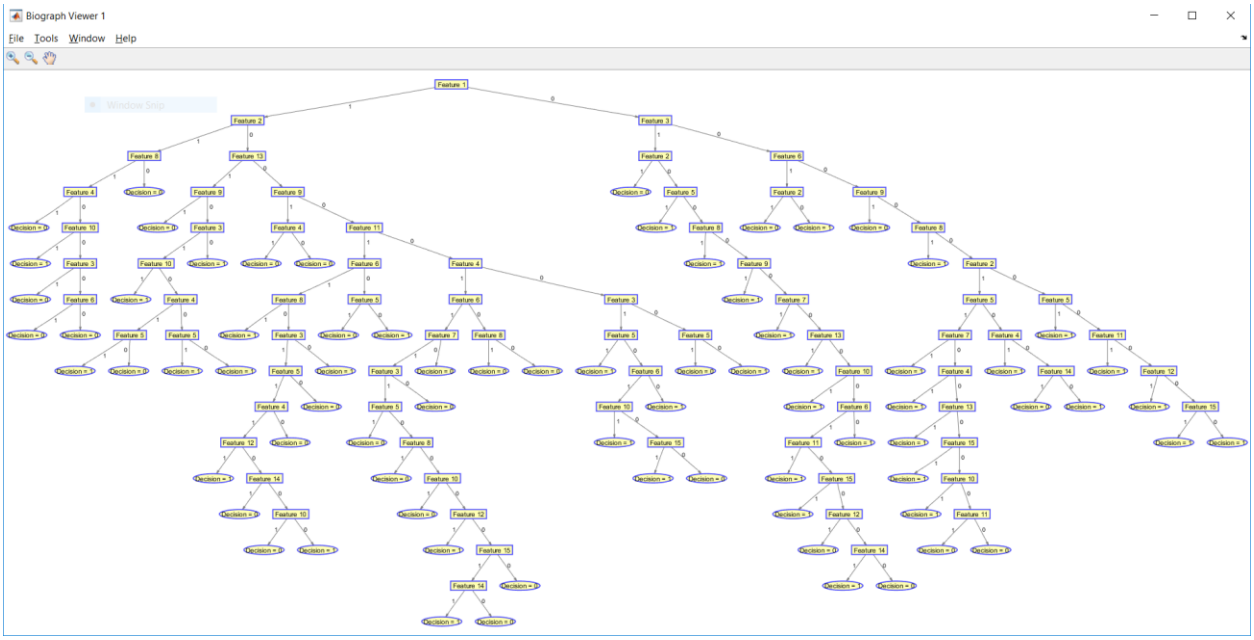


ORACLE 12		N Train				
		100	**200**	**300**	**400**	**500**
N Test	**100**	0	0	0	0	0
	200	0	0	0	0	0
	300	0	0	0	0	0
	400	0	0	0	0	0
	500	0	0	0	0	0



ORACLE 15		N Train				
		100	**200**	**300**	**400**	**500**
N Test	**100**	8	8	9	8.5	8
	200	9	8	8.666667	7.25	8.2
	300	7	6	6.666667	5.75	6.6
	400	7	6	6.666667	5.5	6.2
	500	4	3.5	5	4.5	5.2

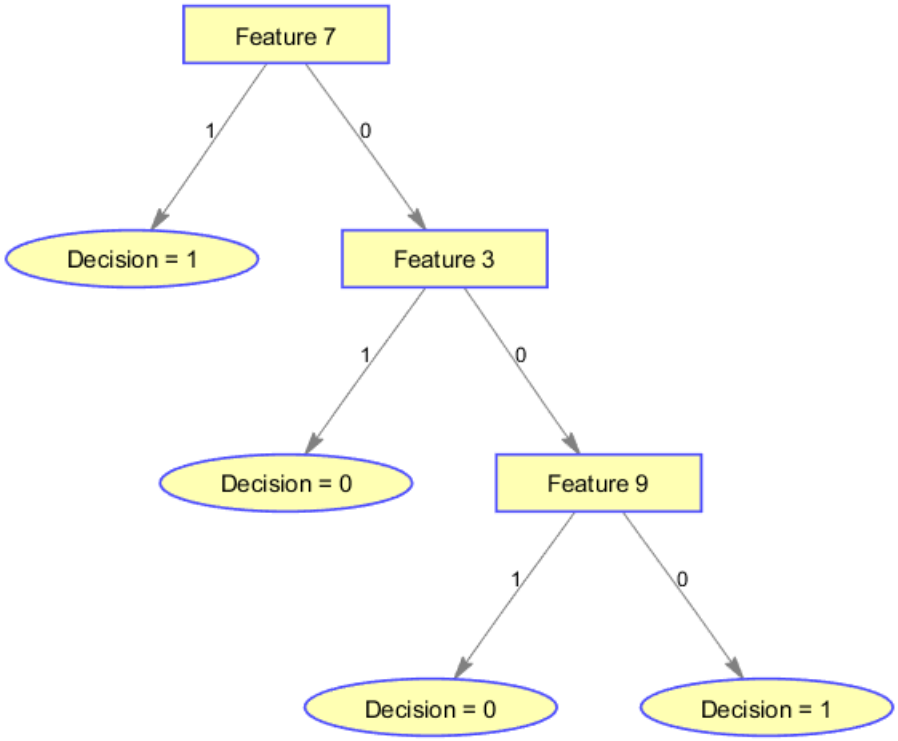
Overfitting of data for Oracle 5



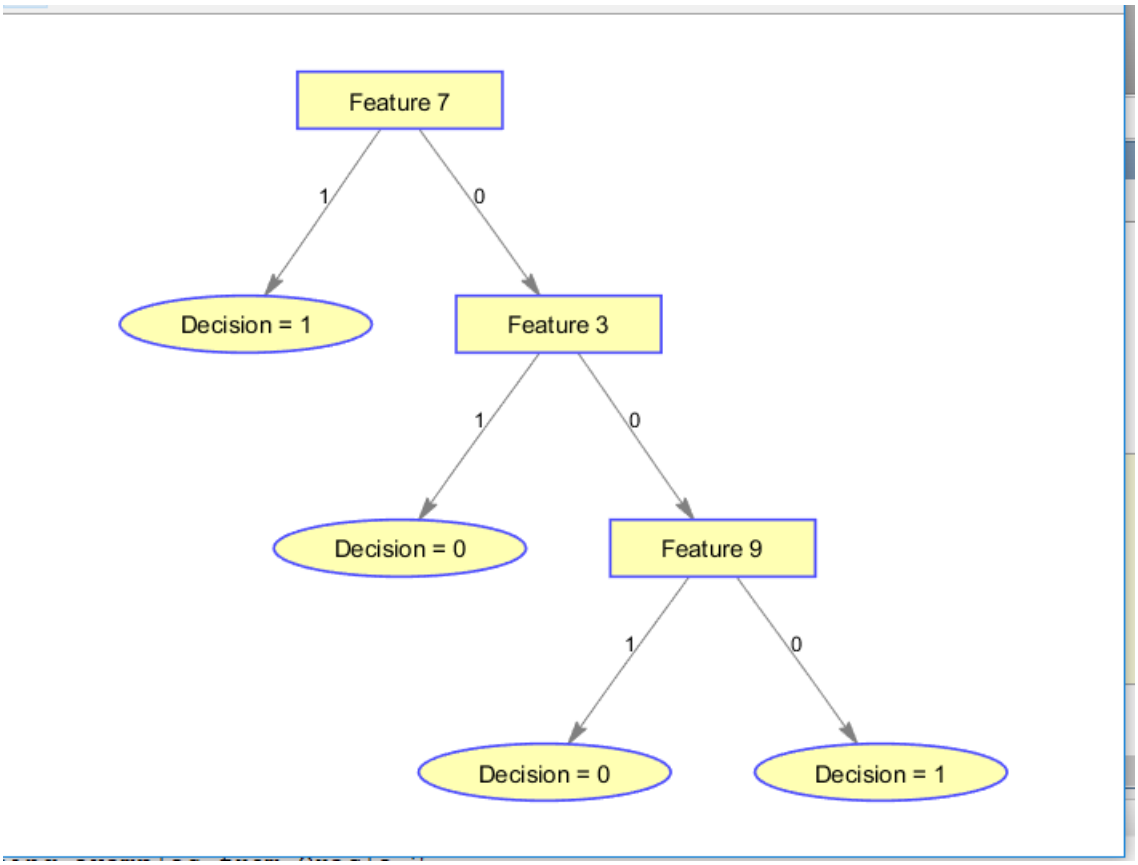
Measures taken to counter this are:

1) Assign a majority vote to nodes after a depth of 8
2) Assign a majority vote to the nodes after the bucket size is below a threshold

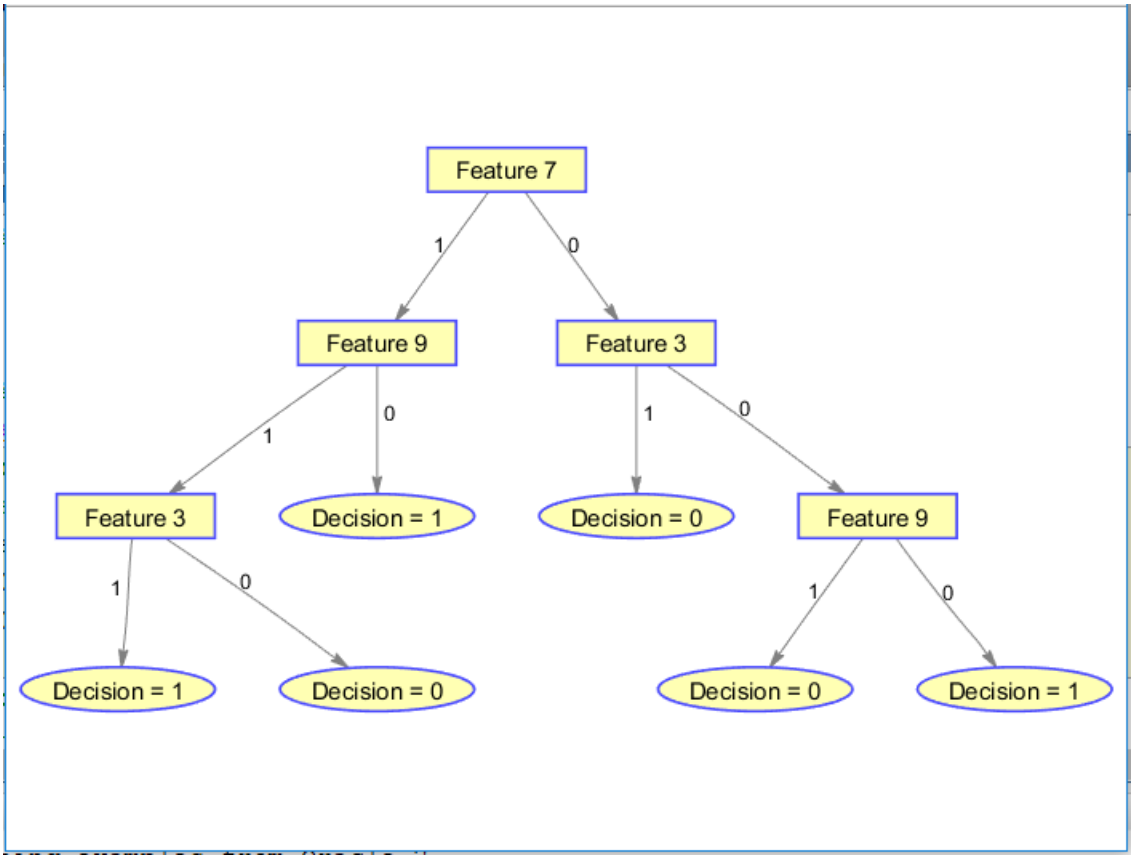
Oracle 2 Train 50 Test 100



Oracle 2 Train 100 Test 100



Oracle 2 Train 500 Test 100


Feature 7
1
0
Feature 9
Feature 3
1
0
1
0
Feature 3
Decision = 1
Decision = 0
Feature 9
1
0
1
0
Decision = 1
Decision = 0
Decision = 0
Decision = 1