ML

1. Warmup - Decision Trees :-

Soln 1.1

$$E(y) = -\sum_{y_i} P(y=y_i) \log_2 P(y=y_i)$$

where summation is over all possible values that y can take and P is probability.

~~$\Rightarrow E(y) =$~~

Now $P(y=-1) = \frac{6}{10}$, $P(y=1) = \frac{4}{10}$

So, we get

$$E(y) = -\left[P(y=-1)\log_2 P(y=-1) + P(y=1)\log_2 P(y=1)\right]$$

$$= -\left[\frac{6}{10}\log_2\frac{6}{10} + \frac{4}{10}\log\frac{4}{10}\right]$$

$$= -\left[-\frac{6}{10}(0.737) - \frac{4}{10}(1.322)\right]$$

$$= \left[0.4422 + 0.5288\right] = \boxed{0.971}.$$

Soln 1.2 $IG(x_i) = E(y) - E(x_i, y)$

$\Rightarrow$ Entropy of class variable - Entropy of feature x_i

bw Lets see for feature x_1

x_1 can take 0 and 1 values.

Lets see how x_1 affects y.

		y	
		-1	1
x_1	0	3	1
	1	3	3

Let's calculate $E(y, x_1)$

$$= P(x_1=0)\,E(3,1) + P(x_1=1)\,E(3,3)$$

$$= \frac{4}{10}\left[-\left(\frac{3}{4}\log_2\frac{3}{4} + \frac{1}{4}\log_2\frac{1}{4}\right)\right] + \frac{6}{10}\left[-2\left(\frac{1}{2}\log_2\frac{1}{2}\right)\right]$$

$$= \frac{4}{10}\left[\frac{3}{\cancel{4}}\times(0.415) + \frac{1}{\cancel{4}}\times 2\right] + \frac{6}{10}\left[1\right]$$

$$= \frac{\cancel{4}}{\cancel{10}}\frac{1}{10}\left[1.245 + 2\right] + 0.6 = 0.3245 + 0.6 = \underline{0.925}$$

$$\underline{\underline{IG(x_1)}} = 0.971 - 0.925 = \underline{\underline{0.046}}$$

Lets see for feature x_2

x_2 can take 0 and 1 values

		y	
		-1	1
x_2	0	1	4
	1	5	0

So $E(y, x_2) = P(x_2=0)\,E(1,4) + P(x_2=1)\,E(5,0)$

$$= \frac{5}{10}\left[-\left(\frac{1}{5}\log_2\frac{1}{5} + \frac{4}{5}\log_2\frac{4}{5}\right)\right] + \frac{5}{10}\underline{\underline{\left[0\right]}}$$

Scanned with CamScanner

$$= \frac{1}{2}\left(\frac{1}{5}\times 2.322 + \frac{4}{5}\times 0.322\right)$$

$$= \frac{1}{2}\left[0.4644 + 0.2576\right] = 0.361$$

$$\boxed{IG(x_2)} = 0.971 - 0.361 = \underline{0.610}$$

Lets see for x_3

		y	
		-1	1
x_3	0	2	1
x_3	1	4	3

$$E(y, x_3) = P(x_3=0)\,E(2,1) + P(x_3=1)\,E(4,3)$$

$$\Rightarrow \frac{3}{10}\left[-\left[\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}\right]\right] + \frac{7}{10}\left[-\left(\frac{4}{7}\log_2\frac{4}{7} + \frac{3}{7}\log_2\left(\frac{3}{7}\right)\right)\right]$$

$$= \frac{3}{10}\left[0.9183\right] + \frac{7}{10}\left[0.9852\right]$$

$$= 0.9651$$

So, $\boxed{IG(x_3)} = 0.971 - 0.9651 = 0.006$

For x_4

		y	
		-1	1
	0	1	2
x_4	1	5	2

$$E(y, x_4) = P(x_4=0)\,E(1,2) + P(x_4=1)\,E(5,2)$$

$$= \frac{3}{10}\left[-\left(\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}\right)\right] + \frac{7}{10}\left[-\left(\frac{5}{7}\log_2\frac{5}{7} + \frac{2}{7}\log_2\frac{2}{7}\right)\right]$$

Scanned with CamScanner

$$= \frac{3}{10}\left[0.9183\right] + \frac{7}{10}\left[0.8631\right]$$

$$= 0.2755 + 0.6042 = 0.8797$$

$$\boxed{IG(x_4)} = 0.971 - 0.880$$

$$= 0.091$$

For x_5

		y	
		-1	1
x_5	0	4	3
	1	2	1

$$E(y, x_5) = P(x_5=0)\,E(4,3) + P(x_5=1)\,E(1,2)$$

$$= \frac{7}{10}\left[-\left(\frac{4}{7}\log_2\frac{4}{7} + \frac{3}{7}\log_2\frac{3}{7}\right)\right] +$$

$$\frac{3}{10}\left[-\left(\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}\right)\right]$$

$$= \frac{7}{10}\left[0.9852\right] + \frac{3}{10}\left[0.9183\right]$$

$$= 0.9651$$

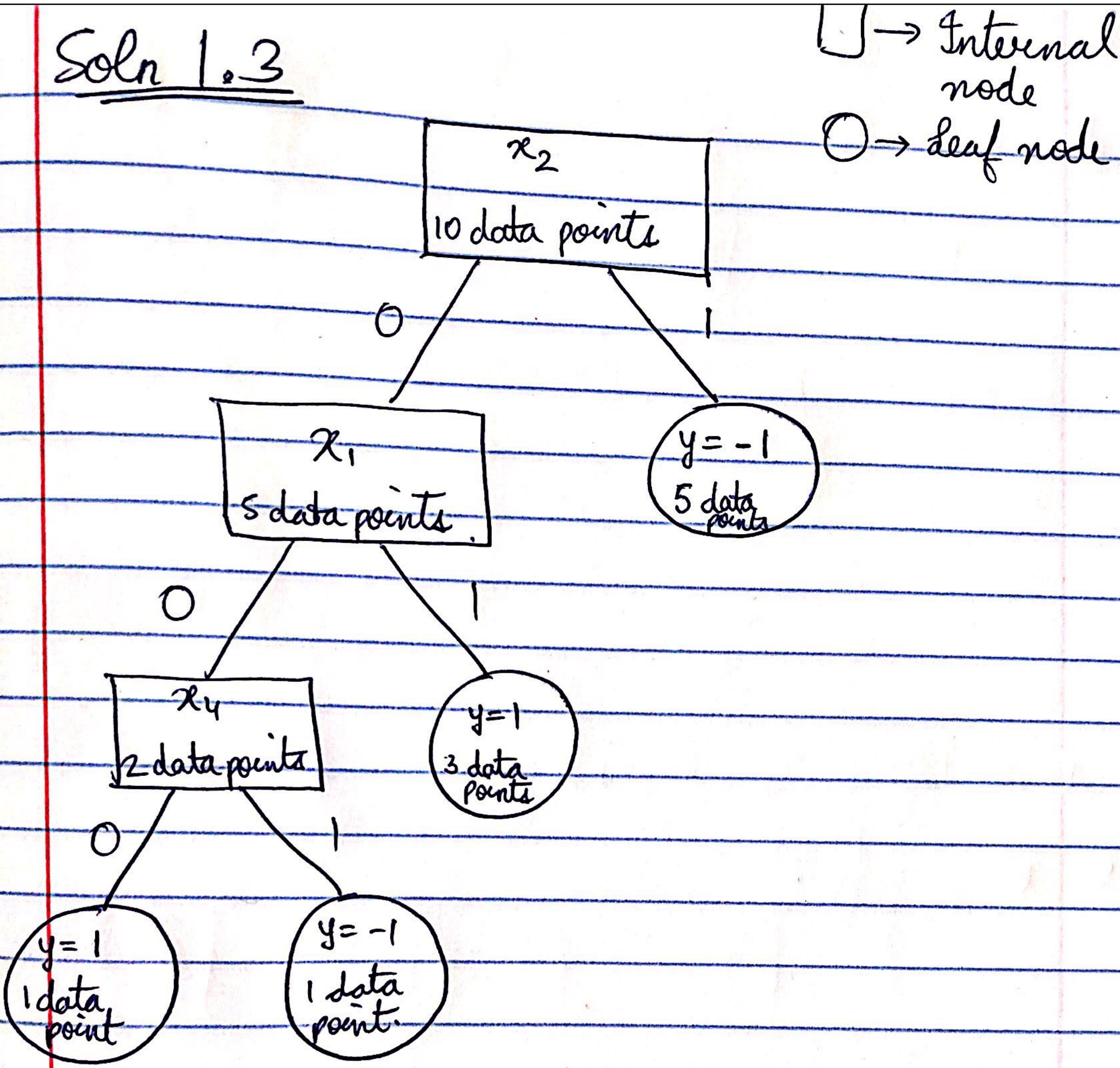
$$\boxed{IG(x_5)} = 0.971 - 0.965$$

$$= 0.006$$

As $IG(x_2)$ is the most, first feature used for splitting is x_2

Scanned with CamScanner

Soln 1.3



□ → Internal node
○ → Leaf node
x_2
10 data points
0
1
x_1
5 data points
$y = -1$
5 data points
0
1
x_4
2 data points
$y = 1$
3 data points
0
1
$y = 1$
1 data point
$y = -1$
1 data point.

Prob 2.

Soln 2.1

In [2]:
```
import numpy as np
import mltools as ml
import matplotlib.pyplot as plt

X = np.genfromtxt('data/X_train.txt', delimiter=None)
Y = np.genfromtxt('data/Y_train.txt', delimiter=None)
X,Y = ml.shuffleData(X,Y)
```

In [3]:
```
print('Minimum of all the features')
print(X.min(axis = 0))
print('\nMaximum of all the features')
print(X.max(axis = 0))
print('\nMean of all the features')
print(X.mean(axis = 0))
print('\nVariance of all the features')
print(X.var(axis = 0))
```

```
Minimum of all the features
[ 1.9300e+02  1.9000e+02  2.1497e+02  2.0542e+02  1.0000e+01  0.0000e+00
  0.0000e+00  0.0000e+00  6.8146e-01  0.0000e+00  0.0000e+00  0.0000e+00
  1.0074e+00 -9.9990e+02]

Maximum of all the features
[2.5300e+02 2.5050e+02 2.5250e+02 2.5250e+02 1.7130e+04 1.2338e+04
 9.2380e+03 3.5796e+01 1.9899e+01 1.1368e+01 2.1466e+01 1.4745e+01
 2.7871e+02 7.8250e+02]

Mean of all the features
[2.41797220e+02 2.28228260e+02 2.41796298e+02 2.33649299e+02
 2.86797959e+03 8.84073295e+02 1.73553355e+02 3.04719572e+00
 6.35196722e+00 1.92523232e+00 4.29379349e+00 2.80947178e+00
 1.03679146e+01 7.87334450e+00]

Variance of all the features
[8.26945619e+01 9.09573945e+01 3.57255796e+01 9.52608539e+01
 1.06194180e+07 3.25702985e+06 7.40656134e+05 7.42244277e+00
 6.33229913e+00 4.28448703e+00 4.04684087e+00 1.98218303e+00
 1.66679252e+02 1.41079679e+03]
```

Soln 2.2

In [4]:
```
Xtr, Xva, Ytr, Yva = ml.splitData(X, Y)
Xt, Yt = Xtr[:5000], Ytr[:5000] # subsample for efficiency (you can go higher
XtS, params = ml.rescale(Xt) # Normalize the features
XvS, _ = ml.rescale(Xva, params) # Normalize the features
```

In [5]:

```
print('Minimum of rescaled training data features')
print(XtS.min(axis = 0))
print('\nMaximum of rescaled training data features')
print(XtS.max(axis = 0))
print('\nMean of rescaled training data features')
print(XtS.mean(axis = 0))
print('\nVariance of rescaled training data features')
print(XtS.var(axis = 0))
```

```
Minimum of rescaled training data features
[ -4.85067996  -3.97777055  -4.40332092  -2.87419042  -0.87193012
  -0.50034384  -0.20016868  -1.13539968  -2.18113307  -0.93052417
  -2.09970056  -1.97355843  -0.70334666 -28.90336056]

Maximum of rescaled training data features
[ 1.21667282  2.17019299  1.78956627  1.92437529  4.44744949  6.5374828
 10.92700516  8.07152428  4.19042172  4.48701818  5.79066362  8.2895552
 14.17289443 17.82596059]

Mean of rescaled training data features
[ 2.87925239e-16 -1.61485270e-15  4.67923478e-15 -1.37753156e-13
 -6.60360655e-17 -7.69162511e-17  4.38427072e-16 -1.15285559e-15
 -1.49323331e-15 -3.96454675e-15 -8.75897133e-15 -2.99421599e-15
  1.09028342e-15  1.37900802e-16]

Variance of rescaled training data features
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [9]:

```
print('Minimum of rescaled validation data features')
print(XvS.min(axis = 0))
print('\nMaximum of rescaled validation data features')
print(XvS.max(axis = 0))
print('\nMean of rescaled validation data features')
print(XvS.mean(axis = 0))
print('\nVariance of rescaled validation data features')
print(XvS.var(axis = 0))
```

```
Minimum of rescaled validation data features
[ -5.06737113  -3.97777055  -4.45825782  -2.87419042  -0.87224085
  -0.50034384  -0.20016868  -1.13539968  -2.25827922  -0.93052417
  -2.09970056  -1.97355843  -0.70818046 -28.90336056]

Maximum of rescaled validation data features
[ 1.21667282  2.32649715  1.78956627  1.92437529  4.44744949  6.5374828
 10.92700516  9.62960761  5.07643395  4.54043803  8.29520293  8.2895552
 19.75460729 21.06113362]

Mean of rescaled validation data features
[ 0.00334438  0.00436023  0.00735564  0.00086075  0.01994065  0.00760846
  0.01088606  0.00327045 -0.0069943  -0.00319134 -0.01798589 -0.01181682
 -0.01577494  0.00798332]

Variance of rescaled validation data features
[0.97265754 0.98225423 0.98486302 0.98289612 1.04348891 1.08293282
 1.08870809 1.03543049 0.98860298 0.98326309 0.93969279 0.95682393
 0.95878219 1.07965658]
(5000, 14)
(40000, 14)
```

Prob 3.

Soln 3.1

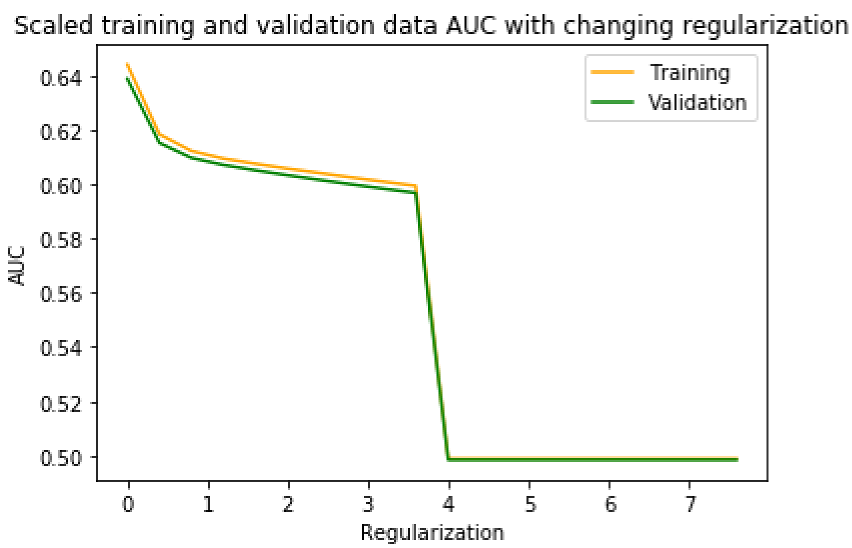
In [10]:

```
regularization = [None] * 20
tS_AUC = [None] * 20
vS_AUC = [None] * 20

for i in range(20):      #Incrementing regularization in chunks of 0.4
    currReg = (i*2)/5
    regularization[i] = currReg
    learner = ml.linearC.linearClassify()
    learner.train(XtS,Yt,reg=currReg,initStep=0.5,stopTol=1e-6,stopIter=100)
    tS_AUC[i] = learner.auc(XtS, Yt)
    vS_AUC[i] = learner.auc(XvS, Yva)
```

```
C:\Users\Karan\Desktop\Study\Spring 2019\CS273P- Machine Learning\ML Models
\HW4\mltools\linearC.py:122: RuntimeWarning: invalid value encountered in t
rue_divide
  sigx  = np.exp(respi) / (1.0+np.exp(respi))
C:\Users\Karan\Desktop\Study\Spring 2019\CS273P- Machine Learning\ML Models
\HW4\mltools\linearC.py:121: RuntimeWarning: invalid value encountered in g
reater
  yhati = 1.0 if respi > 0 else 0.0   # convert to 0/1 prediction
```

In [12]:

```
plt.plot(regularization,tS_AUC,'orange')
plt.plot(regularization,vS_AUC,'green')

plt.title('Scaled training and validation data AUC with changing regularizati
plt.xlabel('Regularization')
plt.ylabel('AUC')
plt.legend(['Training','Validation'])
plt.show()
```



Soln 3.2

In [13]:
```
print('Before transformation,the number of features were '+str(Xt.shape[1]))
XtP = ml.transforms.fpoly(Xt,2,bias = False)
print('After transformation,the number of features were '+str(XtP.shape[1]))
```

```
Before transformation, the number of features were 14
After transformation, the number of features were 119
```

Initially, there are 14 features. But when we transform the features to degree 2, each feature gets multiplied with every other feature and with itself. So, possible ways of selecting 2 features is 14C2 = 91. Apart from this, there are 14 initial features and 14 of the squares of each features. So, the total number of final features is 91 + 14 + 14 = 119.

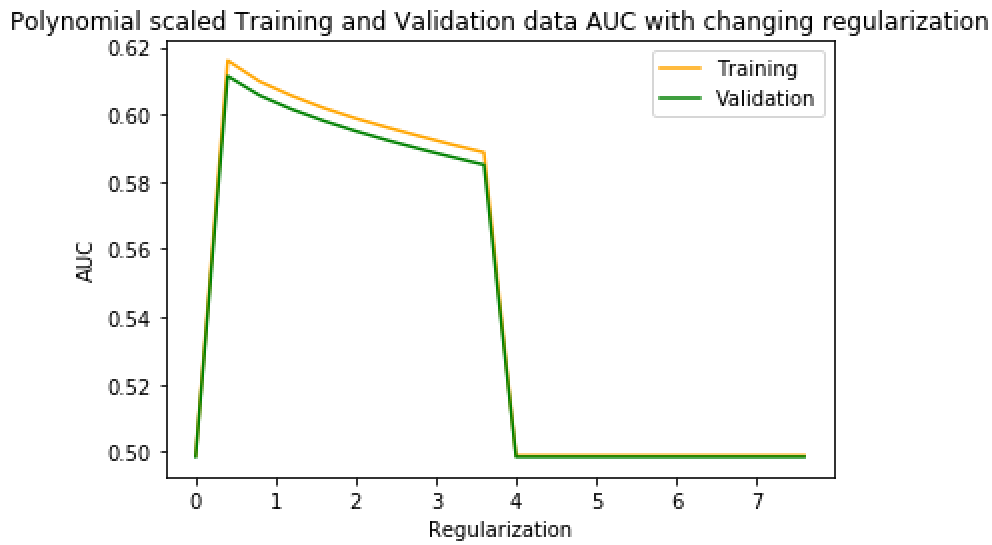
Soln 3.3

In [14]:
```
XtP = ml.transforms.fpoly(Xt,2,bias=False)
XvP = ml.transforms.fpoly(Xva,2,bias=False)

XtPS, params = ml.rescale(XtP)
XvPS, _ = ml.rescale(XvP,params)
```

In [15]:
```
regularization = [None] * 20
tPS_AUC = [None] * 20
vPS_AUC = [None] * 20

for i in range(20):      #Incrementing reg by 0.4
    currReg = (i*2)/5
    regularization[i] = currReg
    learner = ml.linearC.linearClassify()
    learner.train(XtPS,Yt,reg=currReg,initStep=0.5,stopTol=1e-6,stopIter=100)
    tPS_AUC[i] = learner.auc(XtPS, Yt)
    vPS_AUC[i] = learner.auc(XvPS, Yva)
```

```
C:\Users\Karan\Desktop\Study\Spring 2019\CS273P- Machine Learning\ML Models
\HW4\mltools\base.py:96: RuntimeWarning: divide by zero encountered in log
  return - np.mean( np.log( P[ np.arange(M), Y ] ) ) # evaluate
C:\Users\Karan\Desktop\Study\Spring 2019\CS273P- Machine Learning\ML Models
\HW4\mltools\linearC.py:134: RuntimeWarning: invalid value encountered in d
ouble_scalars
  done = (it > stopIter) or ( (it>1) and (abs(Jsur[-1]-Jsur[-2])<stopTol) )
```

In [16]:
```
plt.plot(regularization,tPS_AUC,'orange',label = 'Training')
plt.plot(regularization,vPS_AUC,'green',label = 'Validation')

plt.title('Polynomial scaled Training and Validation data AUC with changing r
plt.xlabel('Regularization')
plt.ylabel('AUC')
plt.legend(loc = 'best')
plt.show()
```



Prob 4.

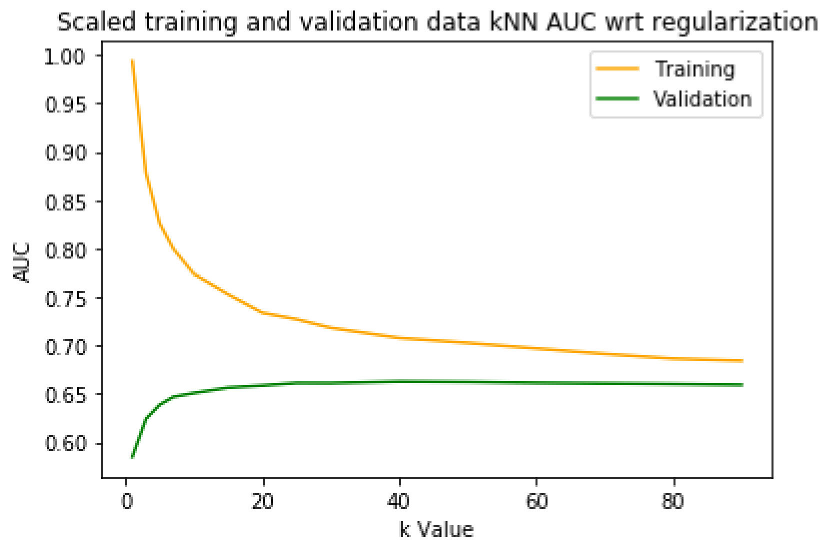
Soln 4.1

Scaled Data

In [17]:
```
kVal = []                                   # Very quick training

for i in range (1,8,2):
    kVal.append(i)
for i in range (10,26,5):
    kVal.append(i)
for i in range(30,100,10):
    kVal.append(i)


tS_AUCkNN = [None] * len(kVal)
vS_AUCkNN = [None] * len(kVal)


for i,k in enumerate(kVal):
    learnerkNN = ml.knn.knnClassify()
    learnerkNN.train(XtS, Yt, K = k, alpha = 0.0)
    tS_AUCkNN[i] = learnerkNN.auc(XtS, Yt)
    vS_AUCkNN[i] = learnerkNN.auc(XvS, Yva)
```

In [18]:

```
plt.plot(kVal,tS_AUCkNN,'orange',label = 'Training')
plt.plot(kVal,vS_AUCkNN,'green',label = 'Validation')

plt.title('Scaled training and validation data kNN AUC wrt regularization')
plt.xlabel('k Value')
plt.ylabel('AUC')
plt.legend()
plt.show()
```



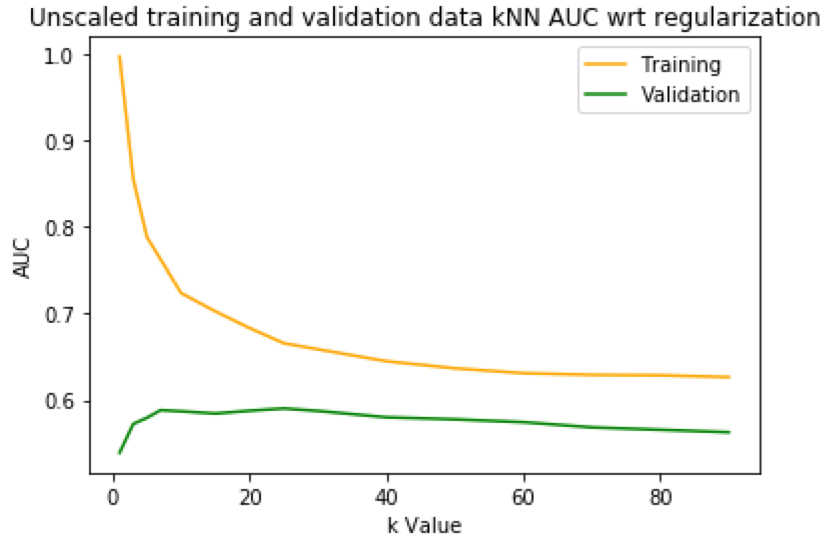
Soln 4.2

Unscaled data

In [106]:

```
kVal = []                                      # Very quick training

for i in range (1,8,2):
    kVal.append(i)
for i in range (10,26,5):
    kVal.append(i)
for i in range(30,100,10):
    kVal.append(i)


t_AUCkNN = [None] * len(kVal)
v_AUCkNN = [None] * len(kVal)


for i,k in enumerate(kVal):
    learnerkNN = ml.knn.knnClassify()
    learnerkNN.train(Xt, Yt, K = k, alpha = 0.0)
    t_AUCkNN[i] = learnerkNN.auc(Xt, Yt)
    v_AUCkNN[i] = learnerkNN.auc(Xva, Yva)
```

In [110]:
```
plt.plot(kVal,t_AUCkNN,'orange',label = 'Training')
plt.plot(kVal,v_AUCkNN,'green',label = 'Validation')

plt.title('Unscaled training and validation data kNN AUC wrt regularization')
plt.xlabel('k Value')
plt.ylabel('AUC')
plt.legend()
plt.show()
```

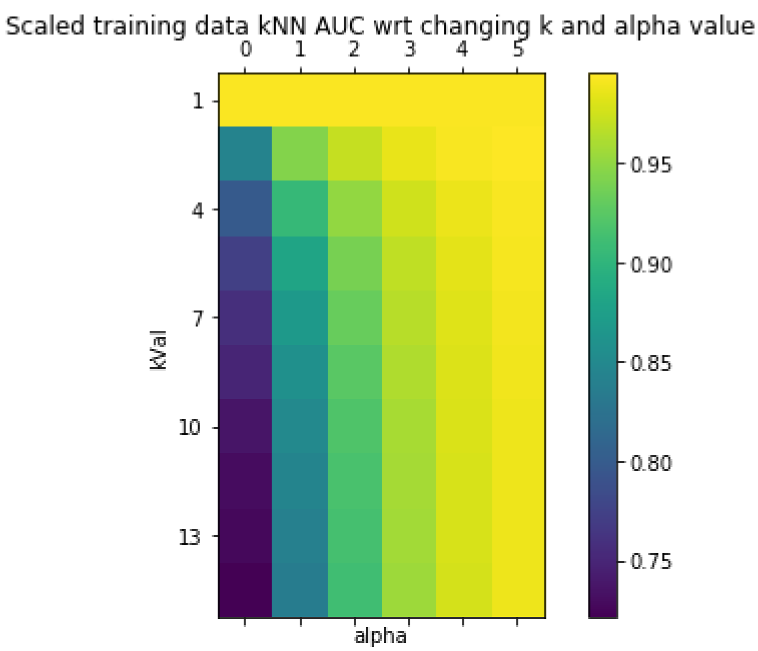


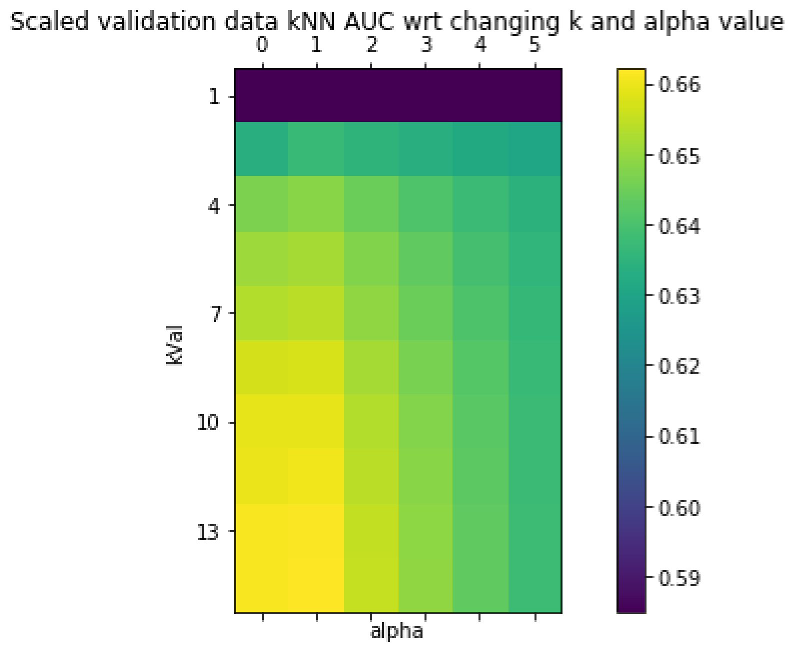
Soln 4.3

In [20]:
```
K = range(1,30,3)
A = range(0,5,1)
tr_auc = np.zeros((len(K),len(A)))
va_auc = np.zeros((len(K),len(A)))

for i,k in enumerate(K):
    for j,a in enumerate(A):
        learnerkNN = ml.knn.knnClassify()
        learnerkNN.train(XtS,Yt, K = k, alpha = a)
        tr_auc[i][j] = learnerkNN.auc(XtS,Yt)
        va_auc[i][j] = learnerkNN.auc(XvS,Yva)
```

In [21]:

```
# Now plot it
f, ax = plt.subplots(1, 1, figsize=(8, 5))

plt.xlabel('alpha')
plt.ylabel('kVal')
plt.title('Scaled training data kNN AUC wrt changing k and alpha value')

cax = ax.matshow(tr_auc, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+list(A))
ax.set_yticklabels(['']+list(K))

plt.show()


f, ax = plt.subplots(1, 1, figsize=(10, 5))

plt.xlabel('alpha')
plt.ylabel('kVal')
plt.title('Scaled validation data kNN AUC wrt changing k and alpha value')

cax = ax.matshow(va_auc, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+list(A))
ax.set_yticklabels(['']+list(K))

plt.show()
```





For a good model, we want to maximize Area Under the ROC(or any other curve) by changing the hyperparameters(which we can control).

For the scaled validation data, when running kNN for different values of k and alpha, we find that AUC is maximum for a fixed value of k and alpha , which is respectively, k = 13 and alpha = 0

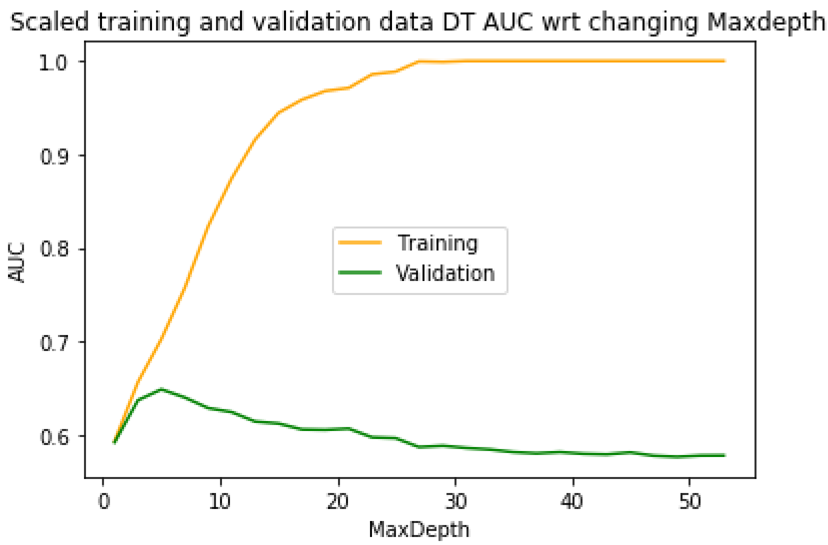
Prob 5

Soln 5.1

In [24]:

```
minParent = 2
minLeaf = 1
maxDepth = range(1,55,2)
tS_AUCdt = []
vS_AUCdt = []


for cMDepth in maxDepth:
    learnerdt=ml.dtree.treeClassify(XtS,Yt,maxDepth=cMDepth,
                                    minLeaf=minLeaf,minParent=minParent)
    tS_AUCdt.append(learnerdt.auc(XtS,Yt))
    vS_AUCdt.append(learnerdt.auc(XvS,Yva))
```

In [25]:
```
plt.plot(maxDepth, tS_AUCdt, 'orange', label = 'Training')
plt.plot(maxDepth, vS_AUCdt, 'green', label = 'Validation')

plt.title('Scaled training and validation data DT AUC wrt changing Maxdepth')
plt.xlabel('MaxDepth')
plt.ylabel('AUC')
plt.legend(loc = 'center')
plt.show()
```

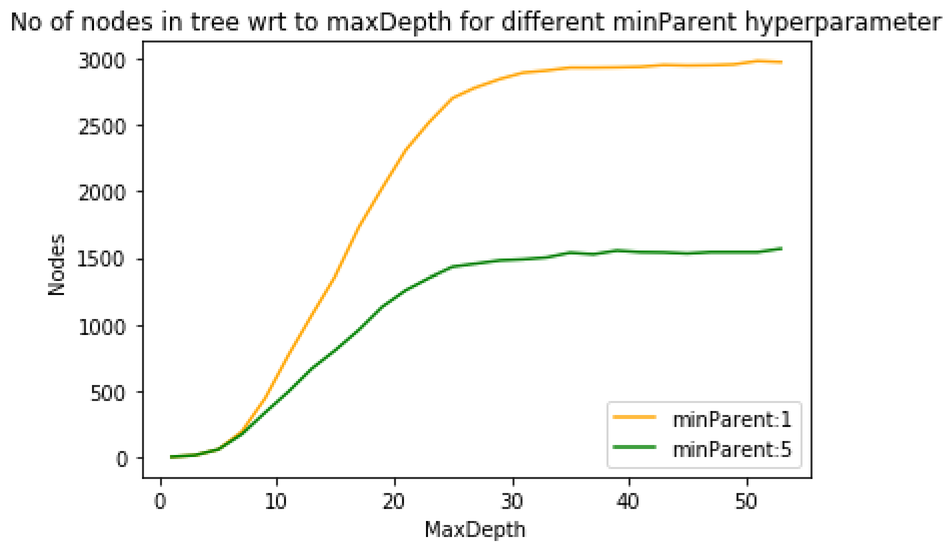


Soln 5.2

In [26]:
```
maxDepth2 = range(1,55,2)
nodesNo1 = []
nodesNo2 = []

for cMDepth in maxDepth2:
    learnerdt=ml.dtree.treeClassify(XtS,Yt,maxDepth=cMDepth,
                                    minLeaf=minLeaf,minParent=minParent)
    learnerdt2=ml.dtree.treeClassify(XtS,Yt,maxDepth=cMDepth,
                                     minLeaf=minLeaf,minParent=5)

    nodesNo1.append(learnerdt.sz)
    nodesNo2.append(learnerdt2.sz)
```

In [27]:

```
plt.plot(maxDepth2,nodesNo1,'orange',label = 'minParent:1')
plt.plot(maxDepth2,nodesNo2,'green',label = 'minParent:5')

plt.title('No of nodes in tree wrt to maxDepth for different minParent hyperp
plt.xlabel('MaxDepth')
plt.ylabel('Nodes')
plt.legend(loc = 'lower right')
plt.show()
```

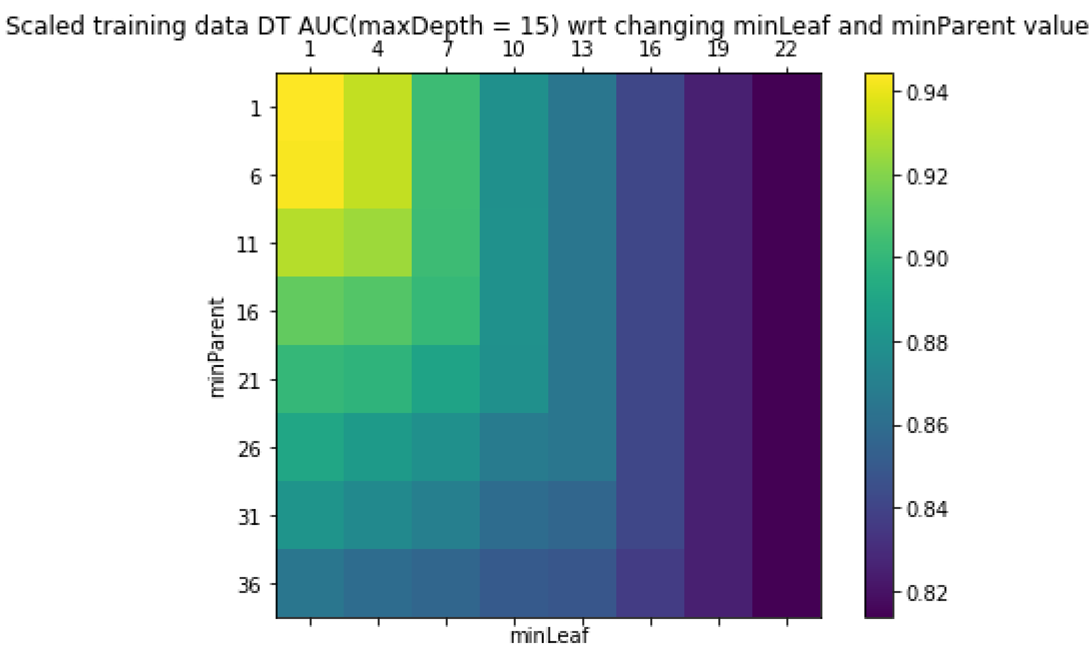


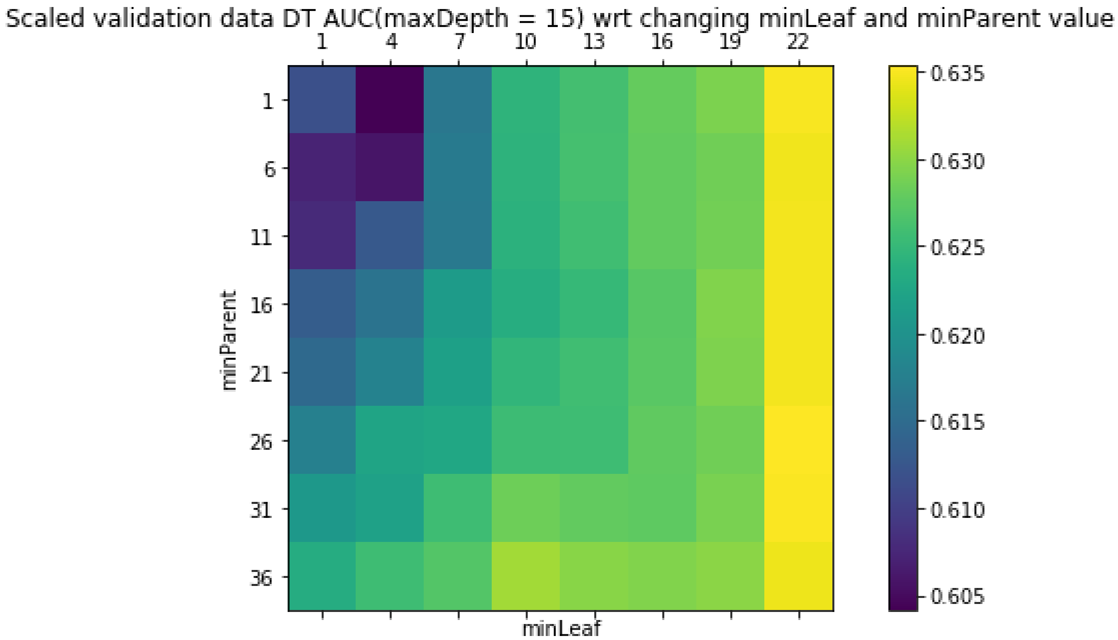
Soln 5.3

In [29]:

```
maxDepth = 15

minParent = range(1,40,5)
minLeaf = range(1,25,3)

tS_AUCdtParLea = np.zeros((len(minParent), len(minLeaf)))
vS_AUCdtParLea = np.zeros((len(minParent), len(minLeaf)))

for i, minPar in enumerate(minParent):
    for j, minLea in enumerate(minLeaf):
        learnerdt=ml.dtree.treeClassify(XtS,Yt,maxDepth=15,
                                        minParent=minPar,minLeaf=minLea)
        tS_AUCdtParLea[i][j] = learnerdt.auc(XtS, Yt)
        vS_AUCdtParLea[i][j] = learnerdt.auc(XvS, Yva)
```

In [32]:
```
# Now plot it
f, ax = plt.subplots(1, 1, figsize=(8, 5))

plt.xlabel('minLeaf')
plt.ylabel('minParent')
plt.title('Scaled training data DT AUC(maxDepth = 15) wrt changing minLeaf ar

cax = ax.matshow(tS_AUCdtParLea, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+list(minLeaf))
ax.set_yticklabels(['']+list(minParent))

plt.show()


f, ax = plt.subplots(1, 1, figsize=(10, 5))

plt.xlabel('minLeaf')
plt.ylabel('minParent')
plt.title('Scaled validation data DT AUC(maxDepth = 15) wrt changing minLeaf

cax = ax.matshow(vS_AUCdtParLea, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+list(minLeaf))
ax.set_yticklabels(['']+list(minParent))

plt.show()
```





Based on the validation plot, I would recommend minLeaf value as 22 and minParent as 1(when maxDepth = 15)

Prob 6

Soln 6.1

In [33]:

```
nnLayers = range(1,5,1)
nnNodesPerLayer = range(3,8,1)

tS_AUCnn = np.zeros((len(nnLayers), len(nnNodesPerLayer)))
vS_AUCnn = np.zeros((len(nnLayers), len(nnNodesPerLayer)))

for i, noOfLayers in enumerate(nnLayers):
    for j, noOfNodesPerLayer in enumerate(nnNodesPerLayer):
        learnernn = ml.nnet.nnetClassify()
        size = [XtS.shape[1]]
        for k in range(noOfLayers):                                 # Adding
            size.append(noOfNodesPerLayer)
        size.append(2)                                              # No of classe
        learnernn.init_weights(size, 'random', XtS, Yt)
        learnernn.train(XtS,Yt,stopTol=1e-8,stepsize=.25,stopIter=300)

        tS_AUCnn[i][j] = learnernn.auc(XtS, Yt)
        vS_AUCnn[i][j] = learnernn.auc(XvS, Yva)
```

```
it 1 : Jsur = 0.43554648781868743, J01 = 0.3342
it 2 : Jsur = 0.4342008550775996, J01 = 0.3386
it 4 : Jsur = 0.4281714577218307, J01 = 0.3284
it 8 : Jsur = 0.42284986443747075, J01 = 0.3198
it 16 : Jsur = 0.4203974502776625, J01 = 0.3198
it 32 : Jsur = 0.41940072378813165, J01 = 0.3176
it 64 : Jsur = 0.4182749694803877, J01 = 0.3162
it 128 : Jsur = 0.4168916944075869, J01 = 0.3132
it 256 : Jsur = 0.4160140608156071, J01 = 0.3128
it 1 : Jsur = 0.43685236167312086, J01 = 0.3406
it 2 : Jsur = 0.4333175356729476, J01 = 0.3346
it 4 : Jsur = 0.4294920394682661, J01 = 0.3304
it 8 : Jsur = 0.42629707690245866, J01 = 0.3264
it 16 : Jsur = 0.42304950972424177, J01 = 0.3186
it 32 : Jsur = 0.4198499243655027, J01 = 0.3178
it 64 : Jsur = 0.4164534762619982, J01 = 0.314
it 128 : Jsur = 0.4131141078177063, J01 = 0.3114
it 256 : Jsur = 0.41083719895939796, J01 = 0.3108
it 1 : Jsur = 0.43493557413057754, J01 = 0.334
it 2 : Jsur = 0.43365272344625383, J01 = 0.3372
it 4 : Jsur = 0.42859488357103953, J01 = 0.3278
it 8 : Jsur = 0.422614869025599, J01 = 0.3192
it 16 : Jsur = 0.4190998426336746, J01 = 0.3188
it 32 : Jsur = 0.4166714982394336, J01 = 0.3144
it 64 : Jsur = 0.4145330623084441, J01 = 0.3134
it 128 : Jsur = 0.4120410538956162, J01 = 0.3112
it 256 : Jsur = 0.40899921988708127, J01 = 0.3092
it 1 : Jsur = 0.4350701553731623, J01 = 0.3362
it 2 : Jsur = 0.4340960603957648, J01 = 0.3364
it 4 : Jsur = 0.42908030648487366, J01 = 0.3296
it 8 : Jsur = 0.424215600567906, J01 = 0.32
it 16 : Jsur = 0.42050319973731776, J01 = 0.3188
it 32 : Jsur = 0.41842453976782873, J01 = 0.316
it 64 : Jsur = 0.41581779483696957, J01 = 0.3134
it 128 : Jsur = 0.41236824309652403, J01 = 0.3112
it 256 : Jsur = 0.40944306677115744, J01 = 0.3064
it 1 : Jsur = 0.43705669421678106, J01 = 0.3384
it 2 : Jsur = 0.4335658141723138, J01 = 0.3366
```

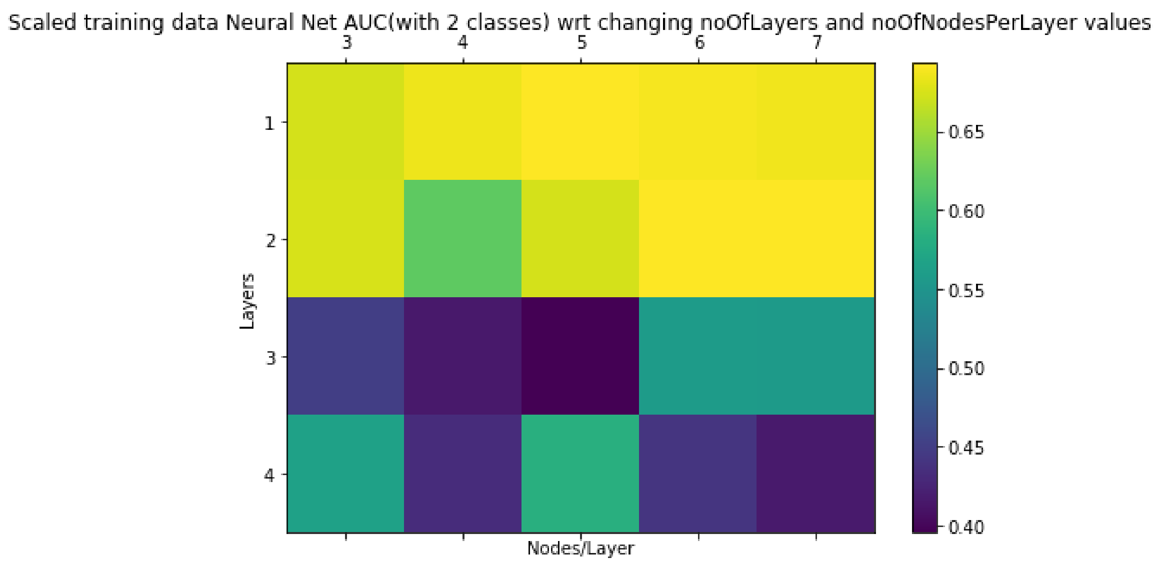
```
it 4 : Jsur = 0.43117858657610375, J01 = 0.3334
it 8 : Jsur = 0.4267302137439556, J01 = 0.3238
it 16 : Jsur = 0.4237800314663891, J01 = 0.3214
it 32 : Jsur = 0.41989619495653535, J01 = 0.3206
it 64 : Jsur = 0.41712410007357764, J01 = 0.3152
it 128 : Jsur = 0.413890676138274, J01 = 0.3118
it 256 : Jsur = 0.410785258396413, J01 = 0.3062
it 1 : Jsur = 0.46549567378355305, J01 = 0.3658
it 2 : Jsur = 0.4642455793193052, J01 = 0.3658
it 4 : Jsur = 0.4640150640562616, J01 = 0.3658
it 8 : Jsur = 0.463993588616971, J01 = 0.3658
it 16 : Jsur = 0.46398924657428414, J01 = 0.3658
it 32 : Jsur = 0.4639850853369466, J01 = 0.3658
it 64 : Jsur = 0.4639807762868383, J01 = 0.3658
it 128 : Jsur = 0.4307559753663578, J01 = 0.3276
it 256 : Jsur = 0.4166540591098814, J01 = 0.3142
it 1 : Jsur = 0.4654956753116894, J01 = 0.3658
it 2 : Jsur = 0.464245582998616, J01 = 0.3658
it 4 : Jsur = 0.4640150717219369, J01 = 0.3658
it 8 : Jsur = 0.4639936047004328, J01 = 0.3658
it 16 : Jsur = 0.46398928710038634, J01 = 0.3658
it 32 : Jsur = 0.46398523535159264, J01 = 0.3658
it 64 : Jsur = 0.46398234283717255, J01 = 0.3658
it 1 : Jsur = 0.4654956747631102, J01 = 0.3658
it 2 : Jsur = 0.46424558145796174, J01 = 0.3658
it 4 : Jsur = 0.46401506771151324, J01 = 0.3658
it 8 : Jsur = 0.46399359509980787, J01 = 0.3658
it 16 : Jsur = 0.46398926052937506, J01 = 0.3658
it 32 : Jsur = 0.4639851369227491, J01 = 0.3658
it 64 : Jsur = 0.46398157736647655, J01 = 0.3658
it 128 : Jsur = 0.46255136679744957, J01 = 0.3658
it 256 : Jsur = 0.4179715248000314, J01 = 0.315
it 1 : Jsur = 0.46549567217417914, J01 = 0.3658
it 2 : Jsur = 0.46424557444044423, J01 = 0.3658
it 4 : Jsur = 0.4640150496223108, J01 = 0.3658
it 8 : Jsur = 0.46399354282202065, J01 = 0.3658
it 16 : Jsur = 0.46398901187187946, J01 = 0.3658
it 32 : Jsur = 0.4639790446111625, J01 = 0.3658
it 64 : Jsur = 0.4252521061275318, J01 = 0.322
it 128 : Jsur = 0.41597242244550464, J01 = 0.3112
it 256 : Jsur = 0.40885041059681054, J01 = 0.3018
it 1 : Jsur = 0.4654956726587474, J01 = 0.3658
it 2 : Jsur = 0.4642455751990403, J01 = 0.3658
it 4 : Jsur = 0.46401505058749454, J01 = 0.3658
it 8 : Jsur = 0.46399354486070554, J01 = 0.3658
it 16 : Jsur = 0.46398902081152354, J01 = 0.3658
it 32 : Jsur = 0.4639794413279862, J01 = 0.3658
it 64 : Jsur = 0.42494450965410485, J01 = 0.321
it 128 : Jsur = 0.4168814368248397, J01 = 0.3118
it 256 : Jsur = 0.40896721396573704, J01 = 0.3054
it 1 : Jsur = 0.4654956755068217, J01 = 0.3658
it 2 : Jsur = 0.46424558338568034, J01 = 0.3658
it 4 : Jsur = 0.4640150725886966, J01 = 0.3658
it 8 : Jsur = 0.46399360624734054, J01 = 0.3658
it 16 : Jsur = 0.4639892898148502, J01 = 0.3658
it 32 : Jsur = 0.4639852402671284, J01 = 0.3658
it 64 : Jsur = 0.4639823524247231, J01 = 0.3658
```

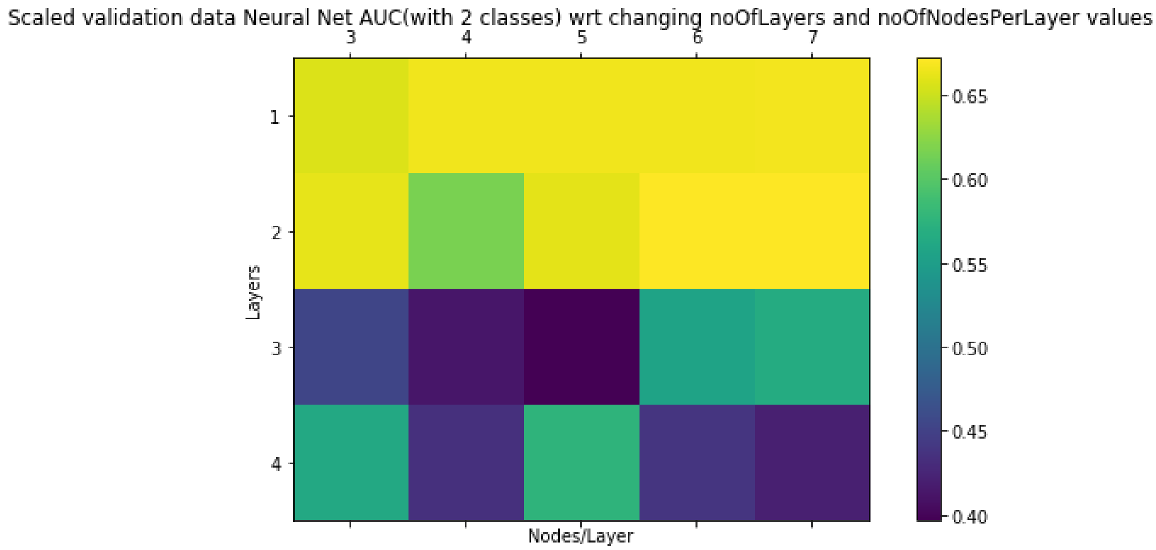
```
it 1 : Jsur = 0.4654956756749493, J01 = 0.3658
it 2 : Jsur = 0.46424558337738464, J01 = 0.3658
it 4 : Jsur = 0.4640150725875367, J01 = 0.3658
it 8 : Jsur = 0.4639936062462657, J01 = 0.3658
it 16 : Jsur = 0.4639892898138474, J01 = 0.3658
it 32 : Jsur = 0.46398524026578836, J01 = 0.3658
it 64 : Jsur = 0.4639823524231853, J01 = 0.3658
it 1 : Jsur = 0.46549567558373217, J01 = 0.3658
it 2 : Jsur = 0.4642455833878854, J01 = 0.3658
it 4 : Jsur = 0.4640150725885774, J01 = 0.3658
it 8 : Jsur = 0.46399360624664987, J01 = 0.3658
it 16 : Jsur = 0.46398928981425763, J01 = 0.3658
it 32 : Jsur = 0.46398524026631094, J01 = 0.3658
it 64 : Jsur = 0.4639823524238237, J01 = 0.3658
it 1 : Jsur = 0.46549567565241784, J01 = 0.3658
it 2 : Jsur = 0.46424558338423083, J01 = 0.3658
it 4 : Jsur = 0.4640150725904233, J01 = 0.3658
it 8 : Jsur = 0.4639936062490592, J01 = 0.3658
it 16 : Jsur = 0.4639892898164324, J01 = 0.3658
it 32 : Jsur = 0.4639852402685106, J01 = 0.3658
it 64 : Jsur = 0.4639823524258854, J01 = 0.3658
it 1 : Jsur = 0.46549567556361815, J01 = 0.3658
it 2 : Jsur = 0.4642455833845193, J01 = 0.3658
it 4 : Jsur = 0.4640150725893554, J01 = 0.3658
it 8 : Jsur = 0.46399360624772223, J01 = 0.3658
it 16 : Jsur = 0.4639892898153028, J01 = 0.3658
it 32 : Jsur = 0.4639852402673807, J01 = 0.3658
it 64 : Jsur = 0.4639823524248538, J01 = 0.3658
it 1 : Jsur = 0.4654956755973315, J01 = 0.3658
it 2 : Jsur = 0.4642455833861364, J01 = 0.3658
it 4 : Jsur = 0.4640150725886888, J01 = 0.3658
it 8 : Jsur = 0.4639936062474576, J01 = 0.3658
it 16 : Jsur = 0.4639892898149553, J01 = 0.3658
it 32 : Jsur = 0.46398524026724164, J01 = 0.3658
it 64 : Jsur = 0.46398235242483454, J01 = 0.3658
it 1 : Jsur = 0.4654956755232304, J01 = 0.3658
it 2 : Jsur = 0.46424558338659017, J01 = 0.3658
it 4 : Jsur = 0.464015072588739, J01 = 0.3658
it 8 : Jsur = 0.46399360624746516, J01 = 0.3658
it 16 : Jsur = 0.46398928981492016, J01 = 0.3658
it 32 : Jsur = 0.46398524026723537, J01 = 0.3658
it 64 : Jsur = 0.4639823524248306, J01 = 0.3658
it 1 : Jsur = 0.4654956755763352, J01 = 0.3658
it 2 : Jsur = 0.46424558338397953, J01 = 0.3658
it 4 : Jsur = 0.46401507258892905, J01 = 0.3658
it 8 : Jsur = 0.46399360624735275, J01 = 0.3658
it 16 : Jsur = 0.463989289815066, J01 = 0.3658
it 32 : Jsur = 0.4639852402672266, J01 = 0.3658
it 64 : Jsur = 0.4639823524248315, J01 = 0.3658
it 1 : Jsur = 0.46549567554197613, J01 = 0.3658
it 2 : Jsur = 0.4642455833840011, J01 = 0.3658
it 4 : Jsur = 0.46401507258911734, J01 = 0.3658
it 8 : Jsur = 0.4639936062472957, J01 = 0.3658
it 16 : Jsur = 0.4639892898150986, J01 = 0.3658
it 32 : Jsur = 0.46398524026720517, J01 = 0.3658
it 64 : Jsur = 0.46398235242481767, J01 = 0.3658
it 1 : Jsur = 0.46549567571242934, J01 = 0.3658
```

```
it 2 : Jsur = 0.46424558338163524, J01 = 0.3658
it 4 : Jsur = 0.4640150725892017, J01 = 0.3658
it 8 : Jsur = 0.4639936062471387, J01 = 0.3658
it 16 : Jsur = 0.4639892898152549, J01 = 0.3658
it 32 : Jsur = 0.46398524026717286, J01 = 0.3658
it 64 : Jsur = 0.46398235242479097, J01 = 0.3658
```

In [35]:

```
# Now plot it
f, ax = plt.subplots(1, 1, figsize=(8, 5))

plt.xlabel('Nodes/Layer')
plt.ylabel('Layers')
plt.title('Scaled training data Neural Net AUC(with 2 classes) wrt changing n

cax = ax.matshow(tS_AUCnn, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+list(nnNodesPerLayer))
ax.set_yticklabels(['']+list(nnLayers))

plt.show()


f, ax = plt.subplots(1, 1, figsize=(10, 5))

plt.xlabel('Nodes/Layer')
plt.ylabel('Layers')
plt.title('Scaled validation data Neural Net AUC(with 2 classes) wrt changing

cax = ax.matshow(vS_AUCnn, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+list(nnNodesPerLayer))
ax.set_yticklabels(['']+list(nnLayers))

plt.show()
```





Based on the plots, a network with noOfLayers as 2 and noOfNodesPerLayer as 7 would yield the greatest validation AUC.

Soln 6.2

In [47]:

```
def sig(z): return (np.exp((-z**2) / 2))
def dsig(z): return (-z * (np.exp((-z**2) / 2)))

classes = 2

size = [XtS.shape[1]]
size.append(7)
size.append(7)
size.append(classes)

learnernn = ml.nnet.nnetClassify()
learnernn.init_weights(size, 'random', XtS, Yt)

learnernn.setActivation('custom', sig, dsig)
learnernn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)

print("Training data AUC for custon(Gaussian) activation function"+str(learne
print("Validation data AUC for custom(Gaussian) activation function"+str(lear


learnernn.setActivation('logistic', sig, dsig)
learnernn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)

print("Training data AUC for logistic activation function"+str(learnernn.auc(
print("Validation data AUC for logistic activation function"+str(learnernn.au


learnernn.setActivation('htangent', sig, dsig)
learnernn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)

print("Training data AUC for htangent activation function"+str(learnernn.auc(
print("Validation data AUC for htangent activation function"+str(learnernn.au
```

```
it 1 : Jsur = 0.46620755222342414, J01 = 0.3658
it 2 : Jsur = 0.4649316288633695, J01 = 0.3658
it 4 : Jsur = 0.46407307745370535, J01 = 0.3658
it 8 : Jsur = 0.46397614511012764, J01 = 0.3658
it 16 : Jsur = 0.4638841834457095, J01 = 0.3658
it 32 : Jsur = 0.4638244665672375, J01 = 0.3658
it 64 : Jsur = 0.46346629355559127, J01 = 0.3658
it 128 : Jsur = 0.46593121506271734, J01 = 0.3658
it 256 : Jsur = 0.4640889169511689, J01 = 0.3658
Training data AUC for custon(Gaussian) activation function 0.55107841549967
85
Validation data AUC for custom(Gaussian) activation function 0.555166544832
4423
it 1 : Jsur = 0.46684687108359574, J01 = 0.3658
it 2 : Jsur = 0.4639874328732152, J01 = 0.3658
it 4 : Jsur = 0.44897357360891155, J01 = 0.3658
it 8 : Jsur = 0.4348658554758473, J01 = 0.326
it 16 : Jsur = 0.43358531399743444, J01 = 0.3292
it 32 : Jsur = 0.43388085606426174, J01 = 0.33
it 64 : Jsur = 0.43431692573933733, J01 = 0.3316
it 128 : Jsur = 0.4347667466472021, J01 = 0.3326
it 256 : Jsur = 0.435187835409378, J01 = 0.3332
Training data AUC for logistic activation function 0.6521040788074125
Validation data AUC for logistic activation function 0.6447289920588801
```

```
it 1 : Jsur = 0.44004592943485094, J01 = 0.3576
it 2 : Jsur = 0.4395061412156495, J01 = 0.3566
it 4 : Jsur = 0.4382324553086204, J01 = 0.3504
it 8 : Jsur = 0.4371232691254465, J01 = 0.3426
it 16 : Jsur = 0.43630990186023955, J01 = 0.3404
it 32 : Jsur = 0.4352917851808216, J01 = 0.3378
it 64 : Jsur = 0.4340092286854743, J01 = 0.334
it 128 : Jsur = 0.4323861902252853, J01 = 0.3298
Training data AUC for htangent activation function 0.6493299118118528
Validation data AUC for htangent activation function 0.6420643016051223
```

Custom Activation(Gaussian function)

AUC for training data - 0.5510784154996785, AUC for validation data - 0.5551665448324423

Logistic Activation

AUC for training data - 0.6521040788074125, AUC for validation data - 0.6447289920588801

HTangent Activation

AUC for training data - 0.6493299118118528, AUC for validation data - 0.6420643016051223

This activation function does not perform as well as logistic or htangent function, both in terms of training or validation data. Best one is logistic activation for this data.

Statement of Collaboration

I didn't collaborate with anyone for this assignment. But i used a youtube video https://www.youtube.com/watch?v=tu_TclzJIeM (https://www.youtube.com/watch?v=tu_TclzJIeM) to understand decision tree entropy and information gain.