2. Setting up the data

In [1]:

```
import numpy as np
import matplotlib.pyplot as pt
import mltools as ml
import random

random.seed(0)
X = np.genfromtxt("data/X_train.txt")
Y = np.genfromtxt("data/Y_train.txt")

X, Y = ml.shuffleData(X, Y)
```

2.1 Print the minimum, maximum, mean, and the variance of all of the features.

In [2]:

```
minimum = [
    min(X[:, feature])
    for feature in range(X.shape[1])
]

maximum = [
    max(X[:, feature])
    for feature in range(X.shape[1])
]

mean = [
    np.mean(X[:, feature])
    for feature in range(X.shape[1])
]

variance = [
    np.var(X[:, feature])
    for feature in range(X.shape[1])
]

print("Minimum of the featurs: \n{}\n".format(minimum))
print("Maximum of the featurs: \n{}\n".format(maximum))
print("Mean of the featurs: \n{}\n".format(mean))
print("Variance of the featurs: \n{}\n".format(variance))
```

```
Minimum of the featurs:
[193.0, 190.0, 214.97, 205.42, 10.0, 0.0, 0.0, 0.0, 0.68146, 0.0, 0.
0, 0.0, 1.0074, -999.9]

Maximum of the featurs:
[253.0, 250.5, 252.5, 252.5, 17130.0, 12338.0, 9238.0, 35.796, 19.89
9, 11.368, 21.466, 14.745, 278.71, 782.5]

Mean of the featurs:
[241.7972204, 228.22826005000007, 241.79629755, 233.64929865000002,
2867.97959, 884.073295, 173.553355, 3.04719571745, 6.35196721804999
9, 1.92523231921, 4.2937934887, 2.8094717789999994, 10.3679146455,
7.8733445]

Variance of the featurs:
[82.69456190782383, 90.95739454607401, 35.72557959436399, 95.2608539
1860819, 10619418.044443432, 3257029.8456128435, 740656.1336232441,
7.422442772290731, 6.332299131939854, 4.284487034670786, 4.046840868
867377, 1.9821830277466965, 166.67925177399366, 1410.79679273432]
```

2.2 Split the dataset, and rescale each into training and validation.

In [3]:

```
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

minimum = [
    min(XtS[:, feature])
    for feature in range(XtS.shape[1])
]

maximum = [
    max(XtS[:, feature])
    for feature in range(XtS.shape[1])
]

mean = [
    np.mean(XtS[:, feature])
    for feature in range(XtS.shape[1])
]

variance = [
    np.var(XtS[:, feature])
    for feature in range(XtS.shape[1])
]

print("Minimum of the featurs: \n{}\n".format(minimum))
print("Maximum of the featurs: \n{}\n".format(maximum))
print("Mean of the featurs: \n{}\n".format(mean))
print("Variance of the featurs: \n{}\n".format(variance))
```

```
Minimum of the featurs:
[-5.077115407043245, -4.089845800502515, -4.522141152449949, -2.9190
402562630764, -0.8638950971668569, -0.48369608840868406, -0.19596529
660814596, -1.1310751887585495, -1.9844393103846658, -0.930012323034
9963, -2.1762866861845973, -2.00362357654661, -0.7564309749550104, -
26.843406161933267]

Maximum of the featurs:
[1.219983613968418, 1.7081774342764569, 1.6731483057441807, 1.841905
6220904828, 4.386010812620294, 6.501875584521141, 11.61273065945613
9, 8.56749743356835, 4.763551494747096, 4.632622320223143, 6.4826169
80472011, 5.845184126624507, 15.559371529596728, 19.77483470817384]

Mean of the featurs:
[-1.1357315088389442e-14, 1.3180567748349858e-15, -2.161186785087920
6e-14, -6.508855676656821e-14, 1.7053025658242403e-17, 8.52651282912
1201e-18, 1.8474111129762604e-17, 7.332801033044234e-16, -3.95345978
17691975e-15, -8.540723683836404e-16, -8.427036846114788e-16, -1.000
7994433181011e-15, 1.0615508472255897e-15, 1.7621459846850484e-16]

Variance of the featurs:
[0.9999999999999974, 0.9999999999999846, 1.0000000000000024, 0.99999
99999999958, 1.0000000000000007, 0.9999999999999746, 0.9999999999999
729, 1.0000000000000016, 0.9999999999999977, 0.9999999999999698, 0.9
999999999999982, 1.0000000000000024, 0.9999999999999701, 1.000000000
0000648]
```

3. Linear Classifiers

In [4]:

```
learner = ml.linearC.linearClassify()
learner.train(XtS, Yt, reg=0.0, initStep=0.5, stopTol=1e-6, stopIter=100)
learner.auc(XtS, Yt)  # train AUC
```

Out[4]:

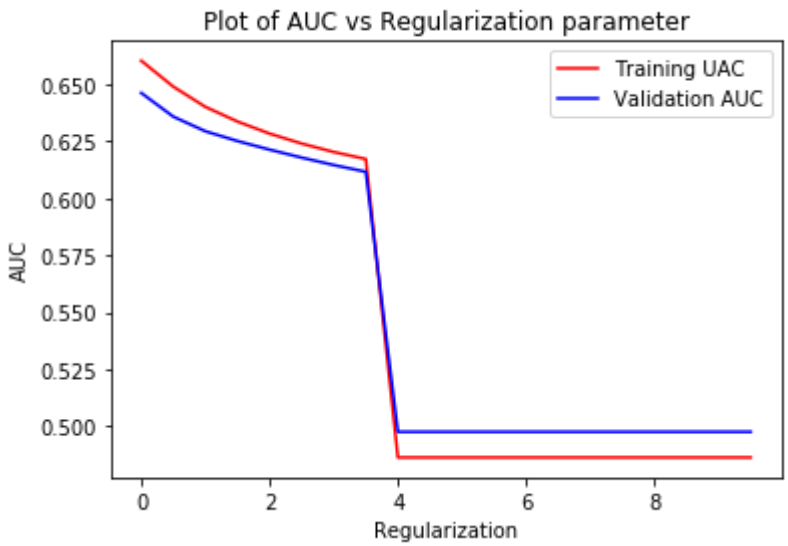
```
0.6602730107947675
```

3.1 Regularization

In [5]:

```
reg = np.arange(0.0, 10.0, 0.5)
auc_val, auc_tr = [], []

for r in reg:
    learner.train(XtS, Yt, reg=r, initStep=0.5, stopTol=1e-6, stopIter=100)
    auc_tr.append(learner.auc(XtS, Yt))
    auc_val.append(learner.auc(XvS, Yva))

pt.plot(reg, auc_tr, "r", label="Training UAC")
pt.plot(reg, auc_val, "b", label="Validation AUC")
pt.xlabel("Regularization")
pt.ylabel("AUC")
pt.legend()
pt.title("Plot of AUC vs Regularization parameter")
pt.show()
```

```
/Users/pranav/School/ml/hw4/mltools/linearC.py:122: RuntimeWarning:
overflow encountered in exp
  sigx  = np.exp(respi) / (1.0+np.exp(respi))
/Users/pranav/School/ml/hw4/mltools/linearC.py:122: RuntimeWarning:
invalid value encountered in true_divide
  sigx  = np.exp(respi) / (1.0+np.exp(respi))
```



3.2 Adding a 2nd degree polynomial

Total number of features before = 14.

The new features will consist of all these 14 old features.

Additionally, we will now have all combinations of these features in our second degree polynomial as well.

Hence, $+^{14}C_2 = 91$ features.

Finally, we also have $+14$ new features where each feature is a square of the corresponding old feature.

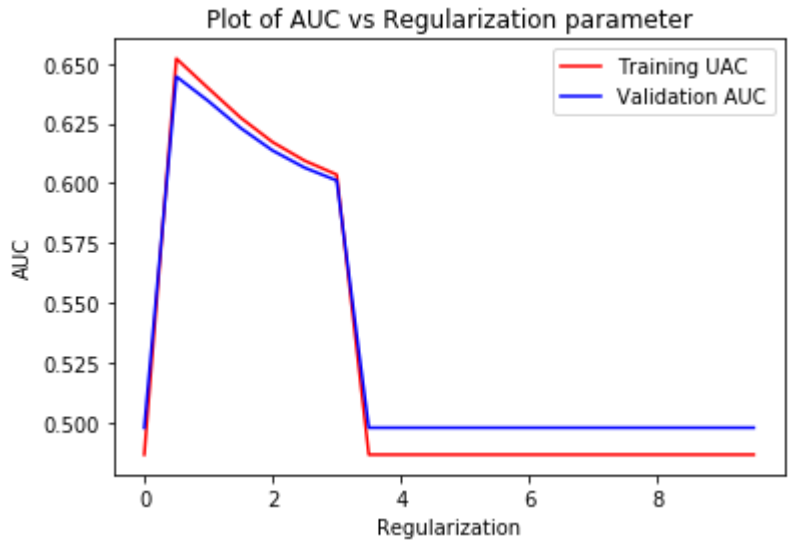
Total new features = $14 + 91 + 14 = 119 features$

3.3 AUC performance with second degree polynomial.

In [6]:

```
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)

auc_val, auc_tr = [], []

for r in reg:
    learner = ml.linearC.linearClassify()
    learner.train(XtPS, Yt, reg=r, initStep=0.5, stopTol=1e-6, stopIter=100)
    auc_tr.append(learner.auc(XtPS, Yt))
    auc_val.append(learner.auc(XvPS, Yva))

pt.plot(reg, auc_tr, "r", label="Training UAC")
pt.plot(reg, auc_val, "b", label="Validation AUC")
pt.xlabel("Regularization")
pt.ylabel("AUC")
pt.legend()
pt.title("Plot of AUC vs Regularization parameter")
pt.show()
```

```
/Users/pranav/School/ml/hw4/mltools/base.py:96: RuntimeWarning: divi
de by zero encountered in log
  return - np.mean( np.log( P[ np.arange(M), Y ] ) ) # evaluate
/Users/pranav/School/ml/hw4/mltools/linearC.py:134: RuntimeWarning:
invalid value encountered in double_scalars
  done = (it > stopIter) or ( (it>1) and (abs(Jsur[-1]-Jsur[-2])<sto
pTol) )
```



4. Nearest Neighbors

In [7]:

```
learner = ml.knn.knnClassify()
learner.train(XtS, Yt, K=1, alpha=0.0)
learner.auc(XtS, Yt)  # train AUC
```

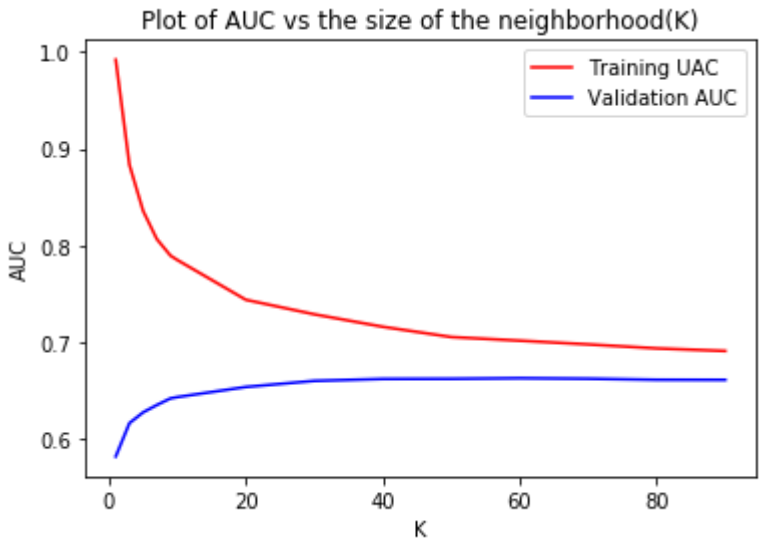
Out[7]:

```
0.992002018529519
```

4.1 Plot of the training and validation performance for an appropriately wide range of K, with α = 0.

In [8]:

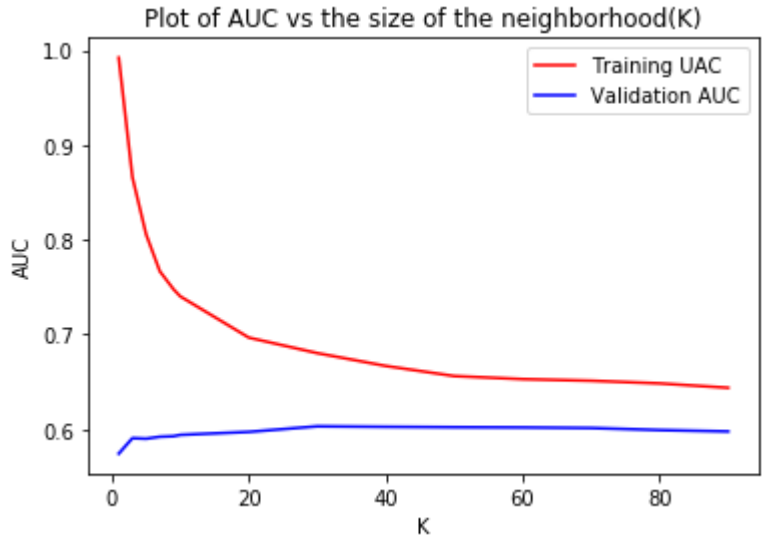
```
K = list(range(1, 10, 2)) + list(range(10, 100, 10))
auc_val, auc_tr = [], []

for k in K:
    learner = ml.knn.knnClassify()
    learner.train(XtS, Yt, K=int(k), alpha=0.0)
    auc_tr.append(learner.auc(XtS, Yt))
    auc_val.append(learner.auc(XvS, Yva))

pt.plot(K, auc_tr, "r", label="Training UAC")
pt.plot(K, auc_val, "b", label="Validation AUC")
pt.xlabel("K")
pt.ylabel("AUC")
pt.legend()
pt.title("Plot of AUC vs the size of the neighborhood(K)")
pt.show()
```



4.2 Unscaled/original data

In [9]:

```
K = list(range(1, 10, 2)) + list(range(10, 100, 10))
auc_val, auc_tr = [], []

for k in K:
    learner = ml.knn.knnClassify()
    learner.train(Xt, Yt, K=k, alpha=0.0)
    auc_tr.append(learner.auc(Xt, Yt))
    auc_val.append(learner.auc(Xva, Yva))

pt.plot(K, auc_tr, "r", label="Training UAC")
pt.plot(K, auc_val, "b", label="Validation AUC")
pt.xlabel("K")
pt.ylabel("AUC")
pt.legend()
pt.title("Plot of AUC vs the size of the neighborhood(K)")
pt.show()
```



4.3 Select both the value of K and α

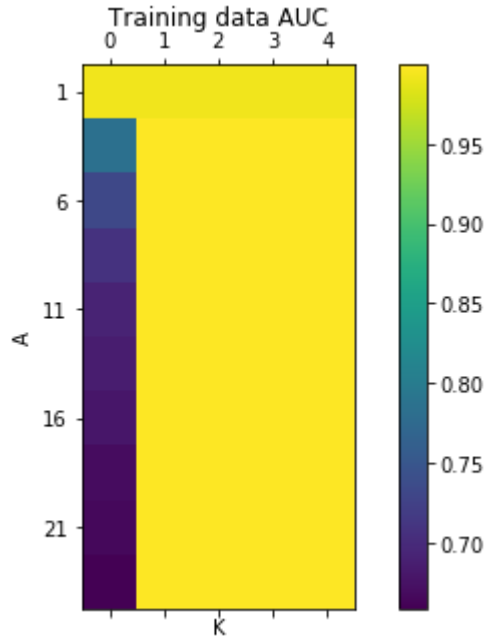
In [10]:

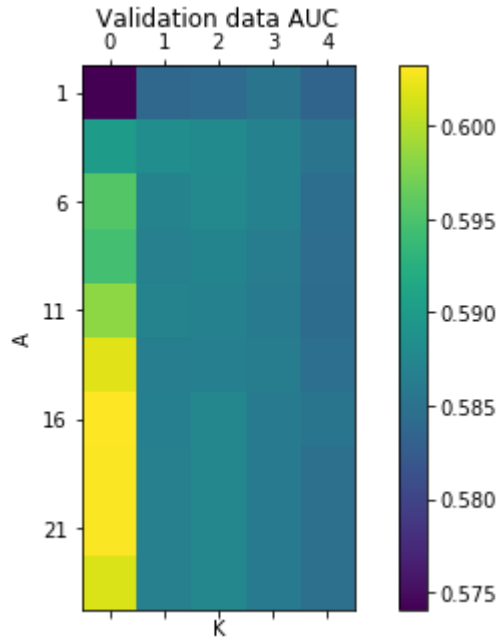
```
K = range(1, 50, 5)
A = range(0, 5, 1)
tr_auc = np.zeros((len(K), len(A)))
va_auc = np.zeros((len(K), len(A)))

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

```
/Users/pranav/School/ml/hw4/mltools/knn.py:103: RuntimeWarning: inva
lid value encountered in true_divide
  prob[i,:] = count / count.sum()    # save (soft) results
```

In [11]:

```
f, ax = pt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(tr_auc, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+list(A))
ax.set_yticklabels(['']+list(K))
pt.ylabel("A")
pt.xlabel("K")
pt.title("Training data AUC")
pt.show()

f, ax = pt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(va_auc, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+list(A))
ax.set_yticklabels(['']+list(K))
pt.ylabel("A")
pt.xlabel("K")
pt.title("Validation data AUC")
pt.show()
```





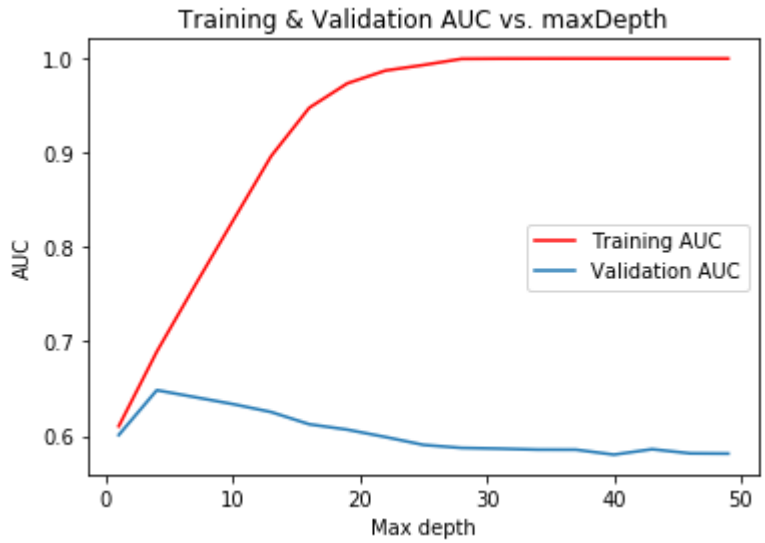
Looking at the plots above K=1 and A=4 looks like optimal values.

5. Decision Trees

5.1 Vary maxDepth to a range of your choosing, and plot the training and validation

In [12]:

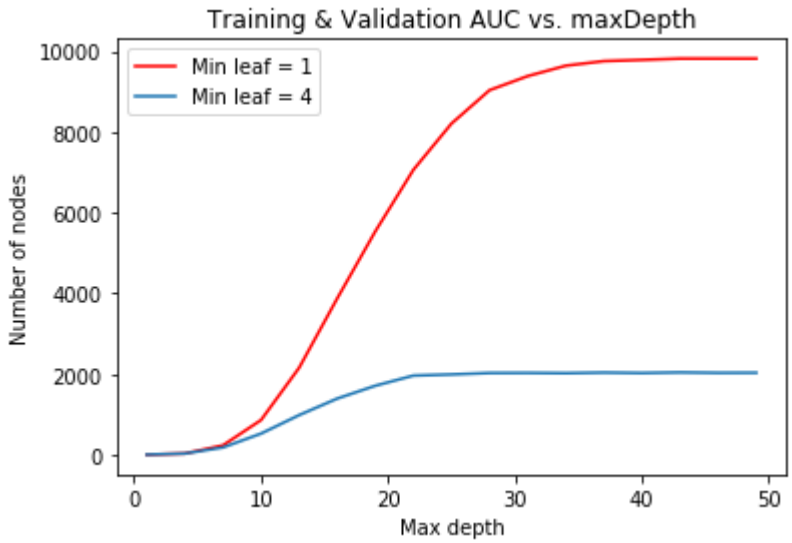
```
max_depths = list(range(1, 50, 3))
tr_auc, va_auc = [], []

for max_depth in max_depths:
    learner = ml.dtree.treeClassify(
        XtS, Yt, minParent=2, minLeaf=1, maxDepth=max_depth
    )
    tr_auc.append(learner.auc(XtS, Yt))
    va_auc.append(learner.auc(XvS, Yva))

pt.plot(max_depths, tr_auc, "r", label="Training AUC")
pt.plot(max_depths, va_auc, label="Validation AUC")
pt.xlabel("Max depth")
pt.ylabel("AUC")
pt.title("Training & Validation AUC vs. maxDepth")
pt.legend()
pt.show()
```



5.2 Plot the number of nodes in the tree as maxDepth is varied.

In [13]:

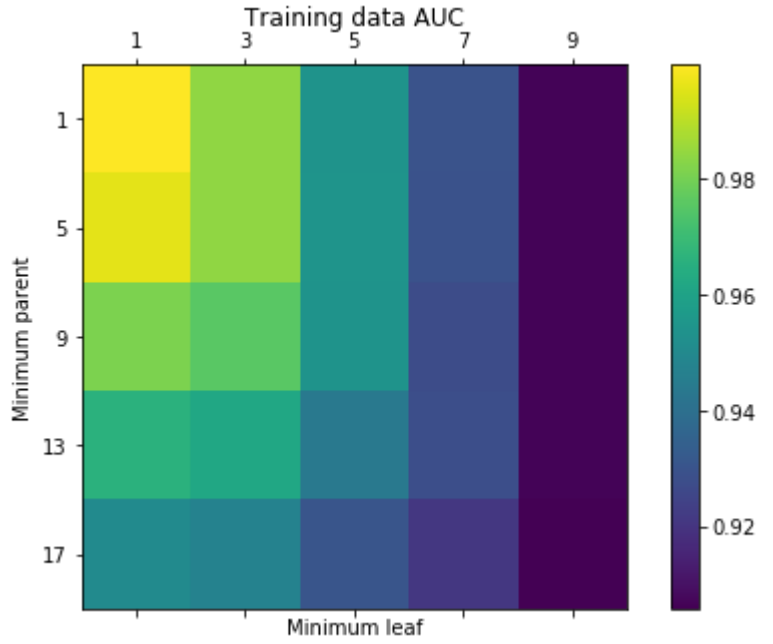
```
l1, l2 = [], []

for max_depth in max_depths:
    learner1 = ml.dtree.treeClassify(
        XtS, Yt, minParent=2, minLeaf=1, maxDepth=max_depth
    )
    learner2 = ml.dtree.treeClassify(
        XtS, Yt, minParent=2, minLeaf=4, maxDepth=max_depth
    )
    l1.append(learner1.sz)
    l2.append(learner2.sz)

pt.plot(max_depths, l1, "r", label="Min leaf = 1")
pt.plot(max_depths, l2, label="Min leaf = 4")
pt.xlabel("Max depth")
pt.ylabel("Number of nodes")
pt.title("Training & Validation AUC vs. maxDepth")
pt.legend()
pt.show()
```

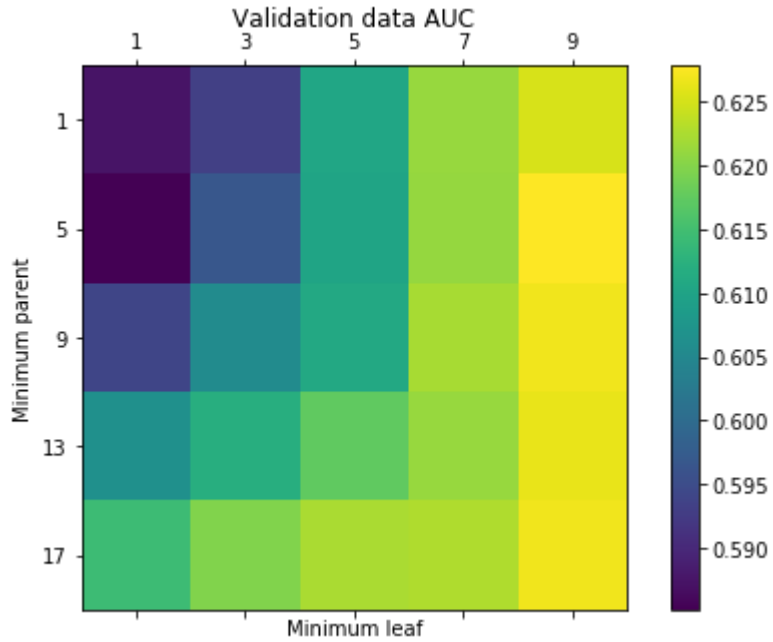


5.3 Recommend a choice for minParent and minLeaf

In [14]:

```
min_parents = range(1, 20, 4)
min_leaves = range(1, 10, 2)
tr_auc = np.zeros((len(min_parents), len(min_leaves)))
va_auc = np.zeros((len(min_parents), len(min_leaves)))

for i,  min_parent in enumerate(min_parents):
    for j, min_leaf in enumerate(min_leaves):
        learner = ml.dtree.treeClassify(
            XtS, Yt,
            minParent=min_parent, minLeaf=min_leaf, maxDepth=30
        )
        tr_auc[i][j] = learner.auc(XtS, Yt)
        va_auc[i][j] = learner.auc(XvS, Yva)

f, ax = pt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(tr_auc, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+list(min_leaves))
ax.set_yticklabels(['']+list(min_parents))
pt.ylabel("Minimum parent")
pt.xlabel("Minimum leaf")
pt.title("Training data AUC")
pt.show()

f, ax = pt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(va_auc, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+list(min_leaves))
ax.set_yticklabels(['']+list(min_parents))
pt.ylabel("Minimum parent")
pt.xlabel("Minimum leaf")
pt.title("Validation data AUC")
pt.show()
```





I would recommend Minimum parent as 1 and Minimum leaf as 7.

6. Neural Networks

6.1 Vary the number of hidden layers and the nodes in each layer

In [15]:

```
layers, nodes = range(1, 6), range(2, 8)
tr_auc = np.zeros((len(layers), len(nodes)))
va_auc = np.zeros((len(layers), len(nodes)))
for i, layer in enumerate(layers):
    for j, node in enumerate(nodes):
        nn = ml.nnet.nnetClassify()
        nn.init_weights([XtS.shape[1], 5, 2], 'random', XtS, Yt)
        nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
        tr_auc[i][j] = nn.auc(XtS, Yt)
        va_auc[i][j] = nn.auc(XvS, Yva)


f, ax = pt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(tr_auc, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+list(nodes))
ax.set_yticklabels(['']+list(layers))
pt.ylabel("Layer")
pt.xlabel("Node")
pt.title("Training data AUC")
pt.show()

f, ax = pt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(va_auc, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+list(nodes))
ax.set_yticklabels(['']+list(layers))
pt.ylabel("Layer")
pt.xlabel("Node")
pt.title("Validation data AUC")
pt.show()
```

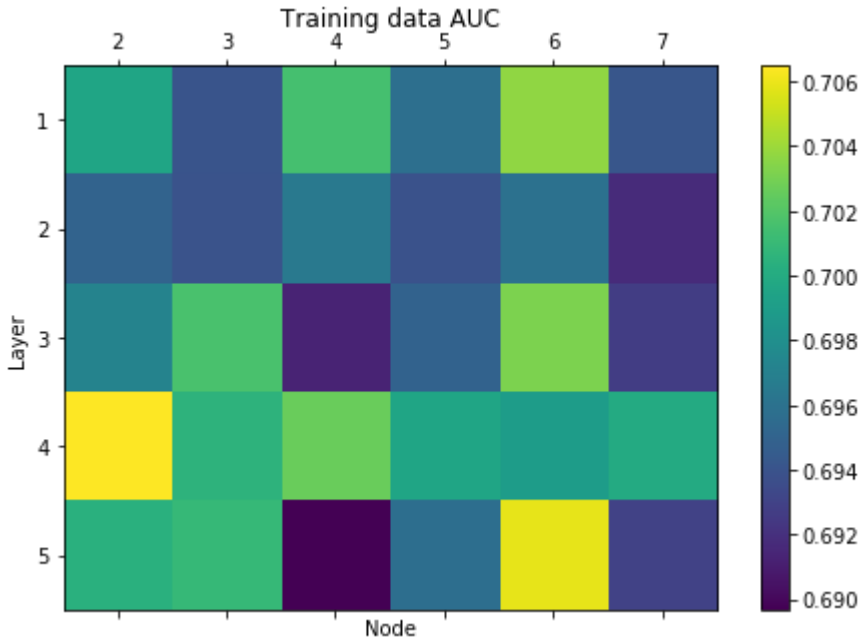
```
it 1 : Jsur = 0.4250057207838684, J01 = 0.3296
it 2 : Jsur = 0.4235300506762318, J01 = 0.3252
it 4 : Jsur = 0.42043357014266375, J01 = 0.3232
it 8 : Jsur = 0.4155975725277318, J01 = 0.316
it 16 : Jsur = 0.4120429554714452, J01 = 0.3118
it 32 : Jsur = 0.4099953006812022, J01 = 0.311
it 64 : Jsur = 0.4099099283366171, J01 = 0.3108
it 128 : Jsur = 0.40915837285817264, J01 = 0.3118
it 256 : Jsur = 0.40557397635352466, J01 = 0.3084
it 1 : Jsur = 0.427242663323677, J01 = 0.3306
it 2 : Jsur = 0.4252191839360672, J01 = 0.3294
it 4 : Jsur = 0.42206261956867075, J01 = 0.3254
it 8 : Jsur = 0.4172652735348476, J01 = 0.318
it 16 : Jsur = 0.4126520309544064, J01 = 0.3154
it 32 : Jsur = 0.4099101746059325, J01 = 0.3132
it 64 : Jsur = 0.4082884890365469, J01 = 0.3092
it 128 : Jsur = 0.40740768550254897, J01 = 0.3062
it 256 : Jsur = 0.4065420483502329, J01 = 0.3064
it 1 : Jsur = 0.42660464609267384, J01 = 0.3306
it 2 : Jsur = 0.4243151569940635, J01 = 0.327
it 4 : Jsur = 0.42154694550083566, J01 = 0.3252
it 8 : Jsur = 0.41646445800160325, J01 = 0.3184
it 16 : Jsur = 0.412449697772209, J01 = 0.3158
it 32 : Jsur = 0.4101290815853877, J01 = 0.3146
it 64 : Jsur = 0.4082077184474498, J01 = 0.31
it 128 : Jsur = 0.40630596223625265, J01 = 0.3082
it 256 : Jsur = 0.40211421725517404, J01 = 0.3036
it 1 : Jsur = 0.4267795974389006, J01 = 0.3246
it 2 : Jsur = 0.423478962432299, J01 = 0.3252
it 4 : Jsur = 0.42102156824931136, J01 = 0.326
it 8 : Jsur = 0.4183848174101244, J01 = 0.3248
it 16 : Jsur = 0.41422294708165563, J01 = 0.3192
it 32 : Jsur = 0.4098361339558871, J01 = 0.315
it 64 : Jsur = 0.4074281369961581, J01 = 0.3112
it 128 : Jsur = 0.4055881545011775, J01 = 0.3054
it 256 : Jsur = 0.40441658228273125, J01 = 0.3054
it 1 : Jsur = 0.42549747967623863, J01 = 0.3284
it 2 : Jsur = 0.42329218158550813, J01 = 0.3262
it 4 : Jsur = 0.41955278779200345, J01 = 0.3234
it 8 : Jsur = 0.4160653722924964, J01 = 0.3162
it 16 : Jsur = 0.4126737371824983, J01 = 0.3138
it 32 : Jsur = 0.4097339125760505, J01 = 0.3132
it 64 : Jsur = 0.4068191944070662, J01 = 0.3146
it 128 : Jsur = 0.40396924652209654, J01 = 0.3078
it 256 : Jsur = 0.4010177917351617, J01 = 0.3024
it 1 : Jsur = 0.4260694641807957, J01 = 0.3282
it 2 : Jsur = 0.42411452608171757, J01 = 0.3282
it 4 : Jsur = 0.4217470132078119, J01 = 0.3272
it 8 : Jsur = 0.4175417242192153, J01 = 0.3226
it 16 : Jsur = 0.4124782055968579, J01 = 0.3166
it 32 : Jsur = 0.4093133772397141, J01 = 0.312
it 64 : Jsur = 0.4073461192906983, J01 = 0.3066
it 128 : Jsur = 0.40593519838955583, J01 = 0.3056
it 256 : Jsur = 0.40496143790969524, J01 = 0.3028
it 1 : Jsur = 0.4251295239066036, J01 = 0.3266
it 2 : Jsur = 0.42394722633728277, J01 = 0.3274
it 4 : Jsur = 0.42056868096853506, J01 = 0.3222
it 8 : Jsur = 0.4164611203194074, J01 = 0.317
it 16 : Jsur = 0.4126605048108959, J01 = 0.3112
it 32 : Jsur = 0.40913062683318424, J01 = 0.3062
it 64 : Jsur = 0.4084039449018O3, J01 = 0.3058
```

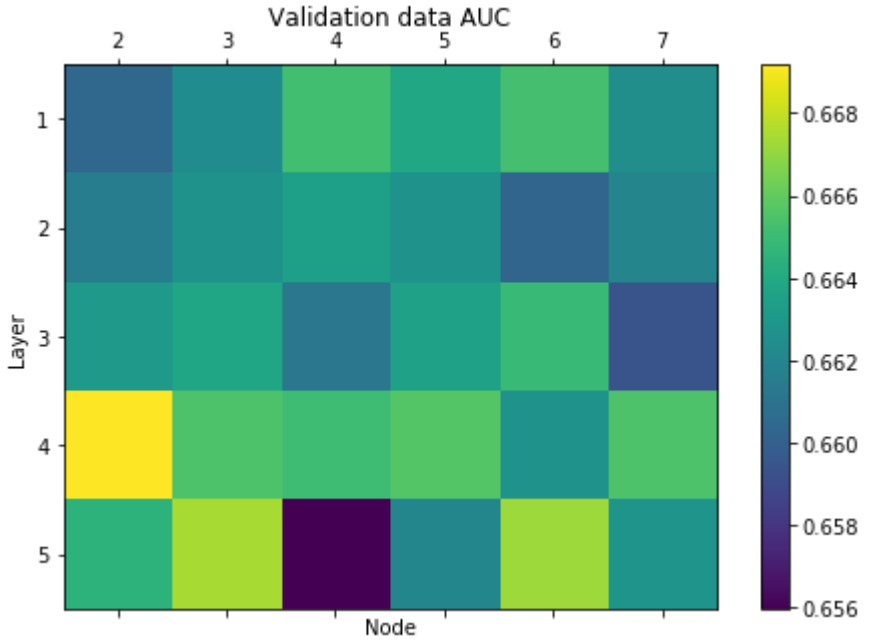
it 128 : Jsur = 0.4087479713091212, J01 = 0.306
it 256 : Jsur = 0.40790485168899876, J01 = 0.3122
it 1 : Jsur = 0.4270043060458601, J01 = 0.3308
it 2 : Jsur = 0.424585157591223, J01 = 0.3276
it 4 : Jsur = 0.4217612142336606, J01 = 0.3248
it 8 : Jsur = 0.4168597086699634, J01 = 0.3194
it 16 : Jsur = 0.4124714868644874, J01 = 0.3146
it 32 : Jsur = 0.4097452774919404, J01 = 0.3144
it 64 : Jsur = 0.40789053112361073, J01 = 0.308
it 128 : Jsur = 0.4064416132062082, J01 = 0.306
it 256 : Jsur = 0.40532462267449976, J01 = 0.3026
it 1 : Jsur = 0.4248346970109329, J01 = 0.3286
it 2 : Jsur = 0.42361178456262544, J01 = 0.3278
it 4 : Jsur = 0.42015069729624027, J01 = 0.3218
it 8 : Jsur = 0.4157013262676246, J01 = 0.316
it 16 : Jsur = 0.4123335298799962, J01 = 0.3112
it 32 : Jsur = 0.41055735233734925, J01 = 0.3096
it 64 : Jsur = 0.41019531902667605, J01 = 0.3068
it 128 : Jsur = 0.41046366077496504, J01 = 0.3134
it 256 : Jsur = 0.40798695324478323, J01 = 0.3148
it 1 : Jsur = 0.42759754748911655, J01 = 0.333
it 2 : Jsur = 0.4242493647596546, J01 = 0.3264
it 4 : Jsur = 0.4209954850512977, J01 = 0.3228
it 8 : Jsur = 0.4181395939600561, J01 = 0.3212
it 16 : Jsur = 0.4144375826900261, J01 = 0.3136
it 32 : Jsur = 0.4110640954415394, J01 = 0.3112
it 64 : Jsur = 0.40832701378213554, J01 = 0.3074
it 128 : Jsur = 0.40650033376942285, J01 = 0.3062
it 256 : Jsur = 0.40466464301896055, J01 = 0.3034
it 1 : Jsur = 0.42596326412902036, J01 = 0.3304
it 2 : Jsur = 0.42396928966865727, J01 = 0.3276
it 4 : Jsur = 0.42095839224098575, J01 = 0.3238
it 8 : Jsur = 0.4178544120901744, J01 = 0.317
it 16 : Jsur = 0.41496979847823584, J01 = 0.3146
it 32 : Jsur = 0.4112956315324529, J01 = 0.3104
it 64 : Jsur = 0.40725096628484736, J01 = 0.3062
it 128 : Jsur = 0.4051566475211389, J01 = 0.3054
it 256 : Jsur = 0.40336312659357043, J01 = 0.304
it 1 : Jsur = 0.42614124129848413, J01 = 0.3292
it 2 : Jsur = 0.4248209636546649, J01 = 0.3312
it 4 : Jsur = 0.4226860744573363, J01 = 0.3274
it 8 : Jsur = 0.4186626881276776, J01 = 0.3204
it 16 : Jsur = 0.4136004426126394, J01 = 0.3144
it 32 : Jsur = 0.41013914365764303, J01 = 0.3134
it 64 : Jsur = 0.40809833361612874, J01 = 0.309
it 128 : Jsur = 0.40677645832965764, J01 = 0.3056
it 256 : Jsur = 0.4058516530497365, J01 = 0.306
it 1 : Jsur = 0.4265775398218121, J01 = 0.3302
it 2 : Jsur = 0.4235306926737607, J01 = 0.3262
it 4 : Jsur = 0.41965831118048746, J01 = 0.3208
it 8 : Jsur = 0.4152818258044463, J01 = 0.3174
it 16 : Jsur = 0.4120583296637366, J01 = 0.3112
it 32 : Jsur = 0.4090236317524091, J01 = 0.3108
it 64 : Jsur = 0.40702117834465973, J01 = 0.3118
it 128 : Jsur = 0.4057330473481154, J01 = 0.3106
it 256 : Jsur = 0.405066153615057, J01 = 0.311
it 1 : Jsur = 0.4262577924776786, J01 = 0.3296
it 2 : Jsur = 0.42437280744685485, J01 = 0.3278
it 4 : Jsur = 0.4210069669441371, J01 = 0.3238
it 8 : Jsur = 0.41648256530739425, J01 = 0.3176
it 16 : Jsur = 0.4127592010297257, J01 = 0.3132

```
it 32 : Jsur = 0.4097308080181325, J01 = 0.3114
it 64 : Jsur = 0.40690598234444053, J01 = 0.3098
it 128 : Jsur = 0.4052954012069591, J01 = 0.3128
it 256 : Jsur = 0.40293839903196627, J01 = 0.31
it 1 : Jsur = 0.42725212978701765, J01 = 0.3302
it 2 : Jsur = 0.42514744605534344, J01 = 0.3292
it 4 : Jsur = 0.42270961170486443, J01 = 0.3274
it 8 : Jsur = 0.41918956777818184, J01 = 0.3206
it 16 : Jsur = 0.41498859690872614, J01 = 0.3134
it 32 : Jsur = 0.4114112858906664, J01 = 0.3124
it 64 : Jsur = 0.4086339388125525, J01 = 0.3096
it 128 : Jsur = 0.40715967325506247, J01 = 0.3068
it 256 : Jsur = 0.40621737790664797, J01 = 0.305
it 1 : Jsur = 0.42670071304327356, J01 = 0.3296
it 2 : Jsur = 0.4244714934281834, J01 = 0.328
it 4 : Jsur = 0.4216716967220747, J01 = 0.3256
it 8 : Jsur = 0.4167539823725679, J01 = 0.3182
it 16 : Jsur = 0.41259943133540417, J01 = 0.3142
it 32 : Jsur = 0.40972732858191996, J01 = 0.3146
it 64 : Jsur = 0.4079318640764337, J01 = 0.308
it 128 : Jsur = 0.4064442749628626, J01 = 0.3056
it 256 : Jsur = 0.40503983564440965, J01 = 0.303
it 1 : Jsur = 0.42726390670552344, J01 = 0.3318
it 2 : Jsur = 0.4241219688112531, J01 = 0.3262
it 4 : Jsur = 0.4212991856512435, J01 = 0.3258
it 8 : Jsur = 0.4164531081648594, J01 = 0.3196
it 16 : Jsur = 0.41201430904677766, J01 = 0.3166
it 32 : Jsur = 0.4083574919874236, J01 = 0.3116
it 64 : Jsur = 0.40499616360481405, J01 = 0.3058
it 128 : Jsur = 0.4024215709795134, J01 = 0.3072
it 256 : Jsur = 0.40160297494948416, J01 = 0.308
it 1 : Jsur = 0.42553170197104556, J01 = 0.3252
it 2 : Jsur = 0.4243558106552624, J01 = 0.3296
it 4 : Jsur = 0.4215911161097078, J01 = 0.323
it 8 : Jsur = 0.41657535281875163, J01 = 0.3164
it 16 : Jsur = 0.41216059909033154, J01 = 0.3096
it 32 : Jsur = 0.40878897478585224, J01 = 0.3064
it 64 : Jsur = 0.40670311787954866, J01 = 0.3102
it 128 : Jsur = 0.40678135489684425, J01 = 0.3072
it 1 : Jsur = 0.42699401595830677, J01 = 0.326
it 2 : Jsur = 0.42537562608564583, J01 = 0.3304
it 4 : Jsur = 0.4225771991513913, J01 = 0.3266
it 8 : Jsur = 0.41844081658506677, J01 = 0.3202
it 16 : Jsur = 0.4136781416661217, J01 = 0.3152
it 32 : Jsur = 0.4099597293256471, J01 = 0.3082
it 64 : Jsur = 0.4061830903053118, J01 = 0.3076
it 128 : Jsur = 0.40151919819749193, J01 = 0.3042
it 256 : Jsur = 0.3990690229070184, J01 = 0.304
it 1 : Jsur = 0.4264736598325439, J01 = 0.3322
it 2 : Jsur = 0.4241799795763214, J01 = 0.3278
it 4 : Jsur = 0.4217350068117772, J01 = 0.3258
it 8 : Jsur = 0.4181464243265 17, J01 = 0.3218
it 16 : Jsur = 0.4127997744118 54, J01 = 0.316
it 32 : Jsur = 0.4097213789111488, J01 = 0.3124
it 64 : Jsur = 0.40764727433829334, J01 = 0.3098
it 128 : Jsur = 0.40548537438306936, J01 = 0.3038
it 256 : Jsur = 0.4030468643448386, J01 = 0.301
it 1 : Jsur = 0.42740283046867733, J01 = 0.3318
it 2 : Jsur = 0.42414451010541887, J01 = 0.3248
it 4 : Jsur = 0.4213584457604756, J01 = 0.3254
it 8 : Jsur = 0.4179253985949239, J01 = 0.3216
```

```
it 16 : Jsur = 0.4134095017764703, J01 = 0.3166
it 32 : Jsur = 0.4095893251655786, J01 = 0.3134
it 64 : Jsur = 0.4070404050003309, J01 = 0.3092
it 128 : Jsur = 0.40459500292352024, J01 = 0.3068
it 256 : Jsur = 0.40174714665469247, J01 = 0.302
it 1 : Jsur = 0.4261857653604928, J01 = 0.331
it 2 : Jsur = 0.4229588051302634, J01 = 0.3238
it 4 : Jsur = 0.4189897483549241, J01 = 0.3224
it 8 : Jsur = 0.415953026149407, J01 = 0.3192
it 16 : Jsur = 0.4121706289976578, J01 = 0.3134
it 32 : Jsur = 0.40942786076164567, J01 = 0.312
it 64 : Jsur = 0.40662953495315524, J01 = 0.3126
it 128 : Jsur = 0.4046654749040218, J01 = 0.308
it 256 : Jsur = 0.40312434767743915, J01 = 0.3098
it 1 : Jsur = 0.42570919991933076, J01 = 0.3304
it 2 : Jsur = 0.4234399467684419, J01 = 0.3262
it 4 : Jsur = 0.4203829069599953, J01 = 0.3232
it 8 : Jsur = 0.4186155974510435, J01 = 0.3194
it 16 : Jsur = 0.4162645229063246, J01 = 0.3176
it 32 : Jsur = 0.41381088944142314, J01 = 0.3154
it 64 : Jsur = 0.41065049819313343, J01 = 0.311
it 128 : Jsur = 0.4070899704916656, J01 = 0.3032
it 256 : Jsur = 0.4039107555692006, J01 = 0.302
it 1 : Jsur = 0.42721229972727903, J01 = 0.3266
it 2 : Jsur = 0.42357797341618697, J01 = 0.325
it 4 : Jsur = 0.4211445508230371, J01 = 0.3252
it 8 : Jsur = 0.4183989491022674, J01 = 0.3234
it 16 : Jsur = 0.41416562883268254, J01 = 0.3182
it 32 : Jsur = 0.41056198918469816, J01 = 0.317
it 64 : Jsur = 0.40736419965177945, J01 = 0.3106
it 128 : Jsur = 0.40493932833763613, J01 = 0.3082
it 256 : Jsur = 0.40316414083047813, J01 = 0.302
it 1 : Jsur = 0.42622325963605673, J01 = 0.3306
it 2 : Jsur = 0.4241993322157714, J01 = 0.3276
it 4 : Jsur = 0.42160683772167995, J01 = 0.3262
it 8 : Jsur = 0.4165382705080177, J01 = 0.3184
it 16 : Jsur = 0.41232794655840044, J01 = 0.3154
it 32 : Jsur = 0.4095893084618057, J01 = 0.3122
it 64 : Jsur = 0.40755854321305485, J01 = 0.31
it 128 : Jsur = 0.40589922994651767, J01 = 0.3066
it 256 : Jsur = 0.40305730967479103, J01 = 0.3022
it 1 : Jsur = 0.4265476697813002, J01 = 0.3286
it 2 : Jsur = 0.4240659069863212, J01 = 0.3262
it 4 : Jsur = 0.42140401757041634, J01 = 0.3264
it 8 : Jsur = 0.41673527142317796, J01 = 0.321
it 16 : Jsur = 0.412133012146952, J01 = 0.3172
it 32 : Jsur = 0.4087514952788932, J01 = 0.3104
it 64 : Jsur = 0.4063896967013605, J01 = 0.3072
it 128 : Jsur = 0.40484562396515705, J01 = 0.306
it 256 : Jsur = 0.4027912400459407, J01 = 0.3052
it 1 : Jsur = 0.4255234798426214, J01 = 0.3272
it 2 : Jsur = 0.42377322011865953, J01 = 0.3274
it 4 : Jsur = 0.42094249930992644, J01 = 0.3228
it 8 : Jsur = 0.4160669407897138, J01 = 0.3156
it 16 : Jsur = 0.41255103324924364, J01 = 0.3098
it 32 : Jsur = 0.40953789361161, J01 = 0.3062
it 64 : Jsur = 0.40774580842016883, J01 = 0.3084
it 128 : Jsur = 0.40827250527805053, J01 = 0.3042
it 256 : Jsur = 0.40915192697188535, J01 = 0.3096
it 1 : Jsur = 0.42688775288366176, J01 = 0.326
it 2 : Jsur = 0.423863709716833, J01 = 0.3256
```

```
it 4 : Jsur = 0.42132148175235545, J01 = 0.3254
it 8 : Jsur = 0.41821340455266237, J01 = 0.3224
it 16 : Jsur = 0.4132464871968848, J01 = 0.3174
it 32 : Jsur = 0.409569991217195, J01 = 0.3142
it 64 : Jsur = 0.4074768433727495, J01 = 0.3088
it 128 : Jsur = 0.40597885641823855, J01 = 0.308
it 256 : Jsur = 0.40473903335143235, J01 = 0.3054
it 1 : Jsur = 0.42538164596584976, J01 = 0.3278
it 2 : Jsur = 0.4239418866825503, J01 = 0.3274
it 4 : Jsur = 0.4208177310236667, J01 = 0.3226
it 8 : Jsur = 0.41669941951048933, J01 = 0.317
it 16 : Jsur = 0.41263993010133926, J01 = 0.3138
it 32 : Jsur = 0.40750076181548844, J01 = 0.308
it 64 : Jsur = 0.4031882966241822, J01 = 0.3066
it 128 : Jsur = 0.40040493703754765, J01 = 0.304
it 256 : Jsur = 0.3994422611480746, J01 = 0.3042
it 1 : Jsur = 0.42726679409443713, J01 = 0.3324
it 2 : Jsur = 0.4251389421703898, J01 = 0.3308
it 4 : Jsur = 0.4222571007096969, J01 = 0.3254
it 8 : Jsur = 0.4171363201902457, J01 = 0.3186
it 16 : Jsur = 0.412712630410663, J01 = 0.3144
it 32 : Jsur = 0.40999160281400293, J01 = 0.312
it 64 : Jsur = 0.40818074827388245, J01 = 0.308
it 128 : Jsur = 0.40693388908085704, J01 = 0.3062
it 256 : Jsur = 0.4060464121912763, J01 = 0.307
```





6.2 Implement a custom activation function

In [16]:

```
def sig(x):
    return np.atleast_2d(np.exp(x**2/2))

def dsig(x):
    return np.atleast_2d(-x*np.exp(x**2/2))

nn = ml.nnet.nnetClassify()
nn.init_weights([XtS.shape[1], 5, 2], 'random', XtS, Yt)
nn.setActivation('custom', sig, dsig)
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
print("Gaussian Training AUC: {}".format(nn.auc(XtS, Yt)))
print("Gaussian Validation AUC: {}".format(nn.auc(XvS, Yva)))


nn = ml.nnet.nnetClassify()
nn.init_weights([XtS.shape[1], 5, 2], 'random', XtS, Yt)
nn.setActivation('logistic')
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
print("Logistic Training AUC: {}".format(nn.auc(XtS, Yt)))
print("Logistic Validation AUC: {}".format(nn.auc(XvS, Yva)))

nn = ml.nnet.nnetClassify()
nn.init_weights([XtS.shape[1], 5, 2], 'random', XtS, Yt)
nn.setActivation('htangent')
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
print("HTangent Training AUC: {}".format(nn.auc(XtS, Yt)))
print("HTangent Validation AUC: {}".format(nn.auc(XvS, Yva)))
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site
-packages/ipykernel_launcher.py:5: RuntimeWarning: overflow encounte
red in exp
  """
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site
-packages/ipykernel_launcher.py:2: RuntimeWarning: overflow encounte
red in exp
```

```
it 1 : Jsur = nan, J01 = 0.3618
it 2 : Jsur = nan, J01 = 0.3618
it 4 : Jsur = nan, J01 = 0.3618
it 8 : Jsur = nan, J01 = 0.3618
it 16 : Jsur = nan, J01 = 0.3618
it 32 : Jsur = nan, J01 = 0.3618
it 64 : Jsur = nan, J01 = 0.3618
it 128 : Jsur = nan, J01 = 0.3618
it 256 : Jsur = nan, J01 = 0.3618
Gaussian Training AUC: 0.4864403218075159
Gaussian Validation AUC: 0.4976982014111342
it 1 : Jsur = 0.4294068329110018, J01 = 0.3306
it 2 : Jsur = 0.42845278346567595, J01 = 0.3324
it 4 : Jsur = 0.4280002609976649, J01 = 0.3334
it 8 : Jsur = 0.4276746793339877, J01 = 0.3314
it 16 : Jsur = 0.42762070190020096, J01 = 0.33
it 32 : Jsur = 0.4276826701052916, J01 = 0.3306
it 64 : Jsur = 0.4277852126577249, J01 = 0.331
it 128 : Jsur = 0.4279363665792282, J01 = 0.331
it 256 : Jsur = 0.4281157152087887, J01 = 0.3324
Logistic Training AUC: 0.6620277040231484
Logistic Validation AUC: 0.6441042042654319
it 1 : Jsur = 0.4254252553921908, J01 = 0.3272
it 2 : Jsur = 0.423323276879804, J01 = 0.3266
it 4 : Jsur = 0.4196362956893101, J01 = 0.3224
it 8 : Jsur = 0.416189989345554, J01 = 0.3166
it 16 : Jsur = 0.4126088807891817, J01 = 0.3128
it 32 : Jsur = 0.40933750863196167, J01 = 0.3128
it 64 : Jsur = 0.40690396673713863, J01 = 0.3152
it 128 : Jsur = 0.4048479475787328, J01 = 0.3154
it 256 : Jsur = 0.4025472690009502, J01 = 0.3068
HTangent Training AUC: 0.7016429915605302
HTangent Validation AUC: 0.663681862860188
```

Gaussian Training AUC: 0.49458635778635773

Gaussian Validation AUC: 0.496823807293988

Logistic Training AUC: 0.649433462033462

Logistic Validation AUC: 0.6483055317573599

HTangent Training AUC: 0.6832237666237666

HTangent Validation AUC: 0.6660746786034655

For both Training and Validation AUC, htangent seems to be the best activation function.

Statement of Collaboration

I have not collaborated with anyone for this homework.