HOMEWORK 4

CS178

Jeremy Parnell (jmparnel)

In [8]:
```
from __future__ import division
import numpy as np
import mltools as ml
import math

%matplotlib inline
import matplotlib.pyplot as plt

np.random.seed(0)
```

1 Setting up the data

In [9]:
```
X = np.genfromtxt('data/X_train.txt', delimiter=None)
Y = np.genfromtxt('data/Y_train.txt', delimiter=None)
X,Y = ml.shuffleData(X,Y)
```

Problem 1.1

In [10]:
```
for num in range(0,14):
    print("Feature {}:  Min: {}  Max: {}  Mean: {}  Variance: {}".format
(num+1,np.amin(X[:,num]),np.amax(X[:,num]),np.mean(X[:,num]),np.var(X[:,
num])))
```

```
Feature 1:  Min: 193.5  Max: 253.0  Mean: 241.6011037  Variance: 83.499
1711498
Feature 2:  Min: 152.5  Max: 249.0  Mean: 227.3765713  Variance: 92.625
593125
Feature 3:  Min: 214.25  Max: 252.5  Mean: 241.5541505  Variance: 35.28
63398033
Feature 4:  Min: 152.5  Max: 252.5  Mean: 232.82676815  Variance: 97.62
57317486
Feature 5:  Min: 10.0  Max: 31048.0  Mean: 3089.923365  Variance: 15651
513.7564
Feature 6:  Min: 0.0  Max: 13630.0  Mean: 928.25902  Variance: 3081761.
81695
Feature 7:  Min: 0.0  Max: 9238.0  Mean: 138.09383  Variance: 443951.74
6446
Feature 8:  Min: 0.0  Max: 125.17  Mean: 3.2485793303  Variance: 8.2194
8502491
Feature 9:  Min: 0.87589  Max: 19.167  Mean: 6.49865290275  Variance:
6.40504819136
Feature 10:  Min: 0.0  Max: 13.23  Mean: 2.09713912048  Variance: 4.363
44047061
Feature 11:  Min: 0.0  Max: 66.761  Mean: 4.21766040935  Variance: 4.08
637188423
Feature 12:  Min: 0.0  Max: 73.902  Mean: 2.69171845215  Variance: 2.19
877847436
Feature 13:  Min: 0.99049  Max: 975.04  Mean: 10.2715904759  Variance:
404.646245041
Feature 14:  Min: -999.9  Max: 797.2  Mean: 5.7814805  Variance: 3406.5
2055098
```

Problem 1.2

In [11]:
```
Xtr, Xva, Ytr, Yva = ml.splitData(X, Y)
Xt, Yt = Xtr[:5000], Ytr[:5000] # subsample for efficiency (you can go h
igher)
XtS, params = ml.rescale(Xt) # Normalize the features
XvS, _ = ml.rescale(Xva, params) # Normalize the features
```

In [12]:
```
print("X-training:")
for num in range(0,14):
    print("Feature {}:  Min: {}  Max: {}  Mean: {}  Variance: {}".format
(num+1,np.amin(XtS[:,num]),np.amax(XtS[:,num]),np.mean(XtS[:,num]),np.va
r(XtS[:,num])))
print("\nX-validation:")
for num in range(0,14):
    print("Feature {}:  Min: {}  Max: {}  Mean: {}  Variance: {}".format
(num+1,np.amin(XvS[:,num]),np.amax(XvS[:,num]),np.mean(XvS[:,num]),np.va
r(XvS[:,num])))
```

```
X-training:
Feature 1:  Min: -4.42216573151  Max: 1.24467764141  Mean: 1.0618350643
2e-14  Variance: 1.0
Feature 2:  Min: -3.83799540084  Max: 1.8142505917  Mean: 8.15703060653
e-16  Variance: 1.0
Feature 3:  Min: -4.59918459307  Max: 1.80668179413  Mean: -3.058886477
45e-14  Variance: 1.0
Feature 4:  Min: -2.91081642991  Max: 1.95449774254  Mean: -1.167990149
04e-14  Variance: 1.0
Feature 5:  Min: -0.779511378114  Max: 7.30095388843  Mean: -3.19744231
092e-17  Variance: 1.0
Feature 6:  Min: -0.516235100982  Max: 7.37342139706  Mean: 7.105427357
6e-18  Variance: 1.0
Feature 7:  Min: -0.20010710502  Max: 13.7671968271  Mean: -4.263256414
56e-18  Variance: 1.0
Feature 8:  Min: -1.13819869133  Max: 7.35307846764  Mean: 1.5816681298
e-15  Variance: 1.0
Feature 9:  Min: -2.10058928482  Max: 4.72658990213  Mean: -1.890043677
12e-15  Variance: 1.0
Feature 10:  Min: -0.989591373122  Max: 5.43214474257  Mean: -1.8900436
7712e-15  Variance: 1.0
Feature 11:  Min: -2.10536921641  Max: 7.41739991327  Mean: 1.364242052
66e-15  Variance: 1.0
Feature 12:  Min: -1.94981434836  Max: 6.11287976902  Mean: 3.182520913
47e-15  Variance: 1.0
Feature 13:  Min: -0.375997776162  Max: 37.4187664809  Mean: -2.0889956
4313e-16  Variance: 1.0
Feature 14:  Min: -16.3042146041  Max: 12.7847476615  Mean: -1.31450406
116e-17  Variance: 1.0

X-validation:
Feature 1:  Min: -5.16853046843  Max: 1.24467764141  Mean: -0.010605234
7626  Variance: 1.01024379009
Feature 2:  Min: -3.94266662292  Max: 2.23293548003  Mean: -0.020506610
139  Variance: 1.0035701127
Feature 3:  Min: -4.4782240948  Max: 1.82684187717  Mean: -0.0021482668
4922  Variance: 0.986731039437
Feature 4:  Min: -2.91081642991  Max: 1.9666711725  Mean: -0.0196728968
419  Variance: 0.994919186428
Feature 5:  Min: -0.779511378114  Max: 7.30095388843  Mean: 0.022502187
4675  Variance: 1.08000814085
Feature 6:  Min: -0.516235100982  Max: 7.37342139706  Mean: 0.021938165
2465  Variance: 1.07093091244
Feature 7:  Min: -0.20010710502  Max: 13.7671968271  Mean: 0.0063063027
3877  Variance: 1.01268888581
Feature 8:  Min: -1.13819869133  Max: 8.79005323252  Mean: -0.013491928
0669  Variance: 0.983746400605
Feature 9:  Min: -2.20331072131  Max: 4.63570002556  Mean: 0.0200222407
51  Variance: 0.997988030777
Feature 10:  Min: -0.989591373122  Max: 5.43214474257  Mean: 0.01896850
95889  Variance: 1.01208973842
Feature 11:  Min: -2.10536921641  Max: 31.1851785066  Mean: 0.001453886
91939  Variance: 1.04616651754
Feature 12:  Min: -1.94981434836  Max: 51.875770174  Mean: 0.0075750868
4929  Variance: 1.173569622
Feature 13:  Min: -0.378031482006  Max: 37.4187664809  Mean: -0.0176734
760563  Variance: 0.538504761236
```

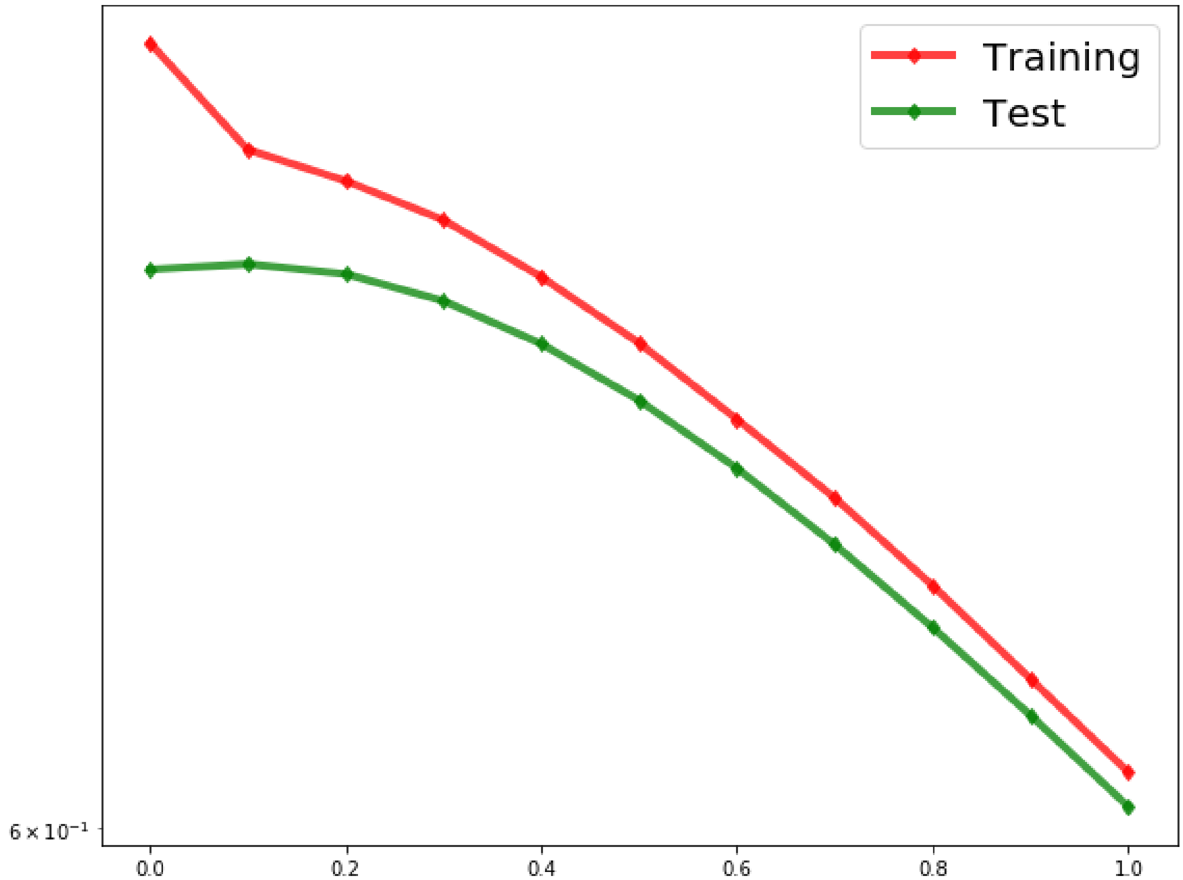
```
Feature 14:  Min: -16.3042146041  Max: 12.8463871288  Mean: 0.010115077
1271  Variance: 0.865691072192
```

2 Linear Classifiers

Problem 2.1

In [6]:
```
reg = [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
trAUC = np.zeros(11)
valAUC = np.zeros(11)

for i,r in enumerate(reg):
    learner = ml.linearC.linearClassify()
    learner.train(XtS, Yt, reg=r, initStep=0.5, stopTol=1e-6, stopIter=1
00)
    trAUC[i] = learner.auc(XtS, Yt)
    valAUC[i] = learner.auc(XvS, Yva)
```

In [7]:
```
fig,ax=plt.subplots(1,1, figsize=(10, 8))
ax.semilogy(reg,trAUC,'r-',lw=4, marker='d', alpha=0.75, label='Trainin
g')
ax.semilogy(reg,valAUC,'g-',lw=4, marker='d', alpha=0.75, label='Test')
ax.legend(fontsize=20, loc=0)
plt.show()
```



Problem 2.2

In [13]:
```
XtP = ml.transforms.fpoly(Xt,2, bias=False)
XtP,params = ml.transforms.rescale(XtP)
XvP,_ = ml.transforms.rescale( ml.transforms.fpoly(XvS,2,False), params)
```

In [14]:
```
XtP.shape[1]
```

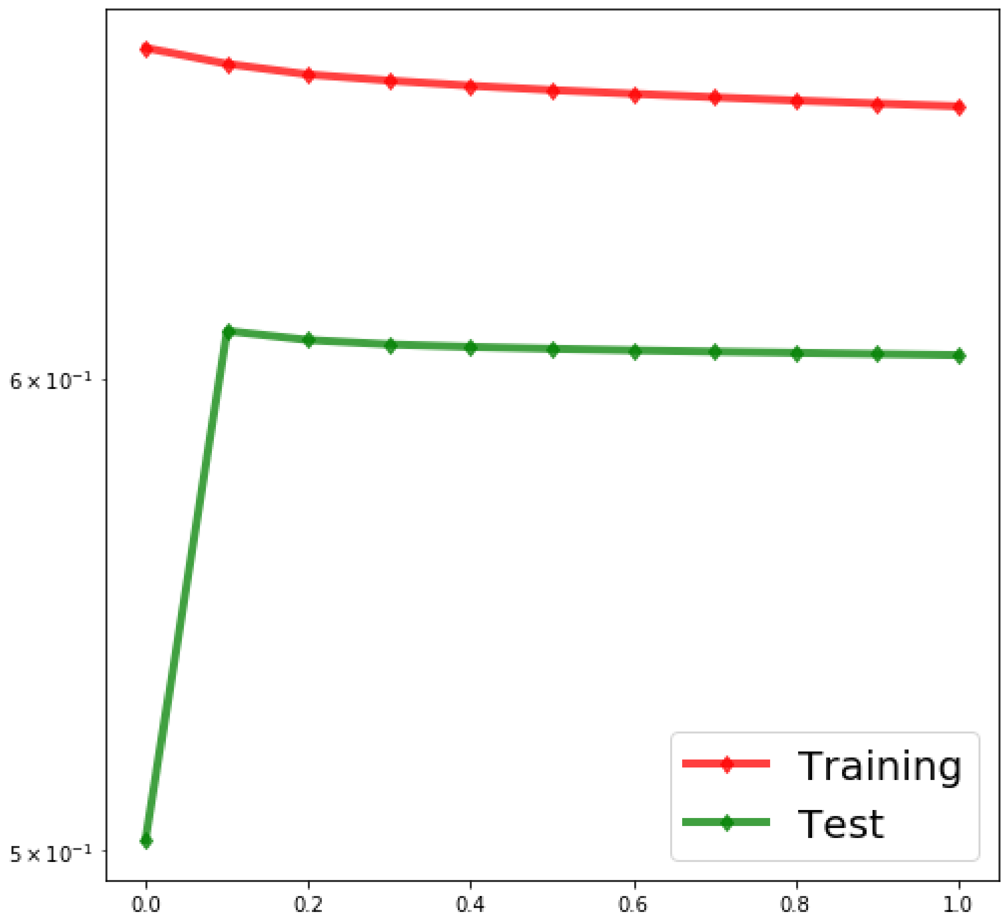
Out[14]: 119

The number of features increased to 119 because you are adding up all the possible combinations of two features out of the 14 given, while also including the possibilites that a feature could also be chosen twice when choosing the two out of the 14. For example X1X1, X2X2...etc.

Problem 2.3

In [10]:
```
reg = [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
trAUC = np.zeros(11)
valAUC = np.zeros(11)

for i,r in enumerate(reg):
    learner = ml.linearC.linearClassify()
    learner.train(XtP, Yt, reg=r, initStep=0.01, stopTol=1e-6, stopIter=
100)
    trAUC[i] = learner.auc(XtP, Yt)
    valAUC[i] = learner.auc(XvP, Yva)
```

In [11]:
```
fig,ax=plt.subplots(1,1, figsize=(8, 8))
ax.semilogy(reg,trAUC,'r-',lw=4, marker='d', alpha=0.75, label='Trainin
g')
ax.semilogy(reg,valAUC,'g-',lw=4, marker='d', alpha=0.75, label='Test')
ax.legend(fontsize=20, loc=0)
plt.show()
```

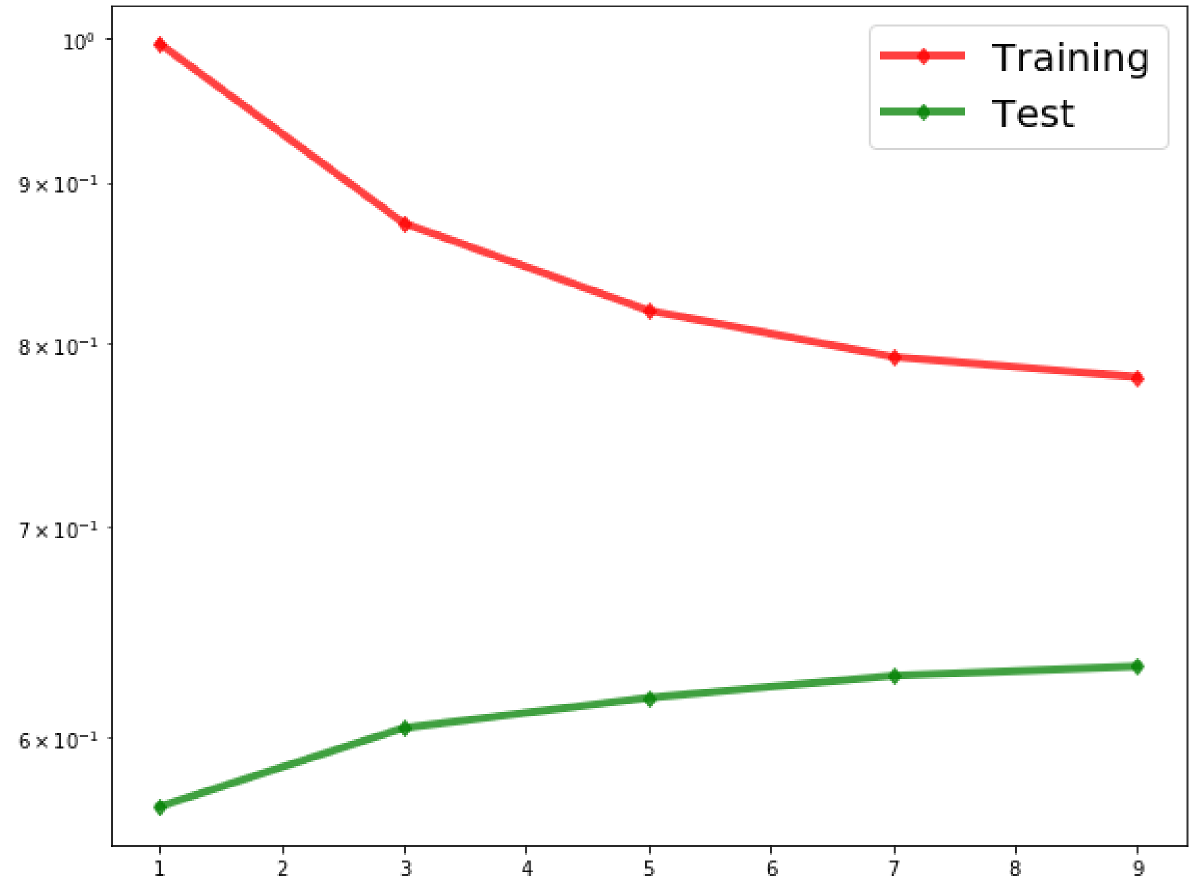


3 Nearest Neighbors

Problem 3.1

In [12]:
```
kvals = [1,3,5,7,9]
trAUC = np.zeros(5)
valAUC = np.zeros(5)

for i, k in enumerate(kvals):
    learner = ml.knn.knnClassify()
    learner.train(XtS, Yt, K=k, alpha=0.0)
    trAUC[i] = learner.auc(XtS, Yt) # train AUC
    valAUC[i] = learner.auc(XvS, Yva)
```

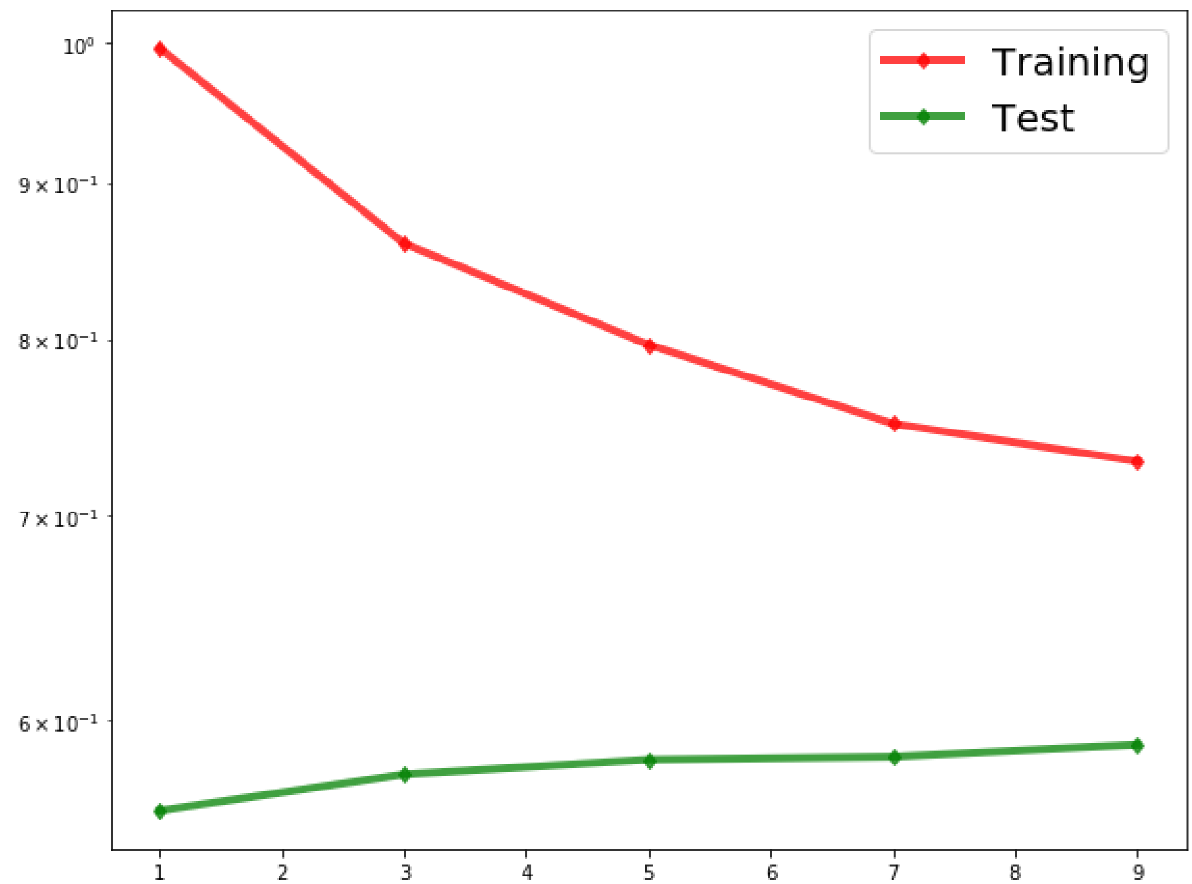
In [13]:
```
fig,ax=plt.subplots(1,1, figsize=(10, 8))
ax.semilogy(kvals,trAUC,'r-',lw=4, marker='d', alpha=0.75, label='Traini
ng')
ax.semilogy(kvals,valAUC,'g-',lw=4, marker='d', alpha=0.75, label='Test'
)
ax.legend(fontsize=20, loc=0)
plt.show()
```



Problem 3.2

In [14]:
```
kvals = [1,3,5,7,9]
trAUC = np.zeros(5)
valAUC = np.zeros(5)

for i, k in enumerate(kvals):
    learner = ml.knn.knnClassify()
    learner.train(Xt, Yt, K=k, alpha=0.0)
    trAUC[i] = learner.auc(Xt, Yt) # train AUC
    valAUC[i] = learner.auc(Xva, Yva)
```

In [15]:
```
fig,ax=plt.subplots(1,1, figsize=(10, 8))
ax.semilogy(kvals,trAUC,'r-',lw=4, marker='d', alpha=0.75, label='Traini
ng')
ax.semilogy(kvals,valAUC,'g-',lw=4, marker='d', alpha=0.75, label='Test'
)
ax.legend(fontsize=20, loc=0)
plt.show()
```



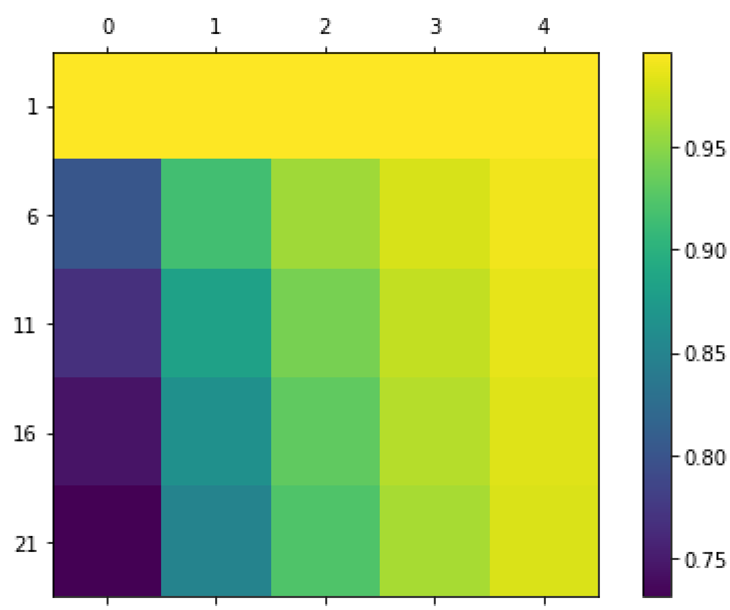
Problem 3.3

In [16]:
```
K = range(1,25,5) # Or something else
A = range(0,5,1) # Or something else
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(XtS, Yt, K=k, alpha=a)
        tr_auc[i][j] = learner.auc(XtS, Yt) # train learner using k and
 a
        va_auc[i][j] = learner.auc(XvS, Yva)
```

```
mltools\knn.py:103: RuntimeWarning: invalid value encountered in divide
  prob[i,:] = count / count.sum()    # save (soft) results
```

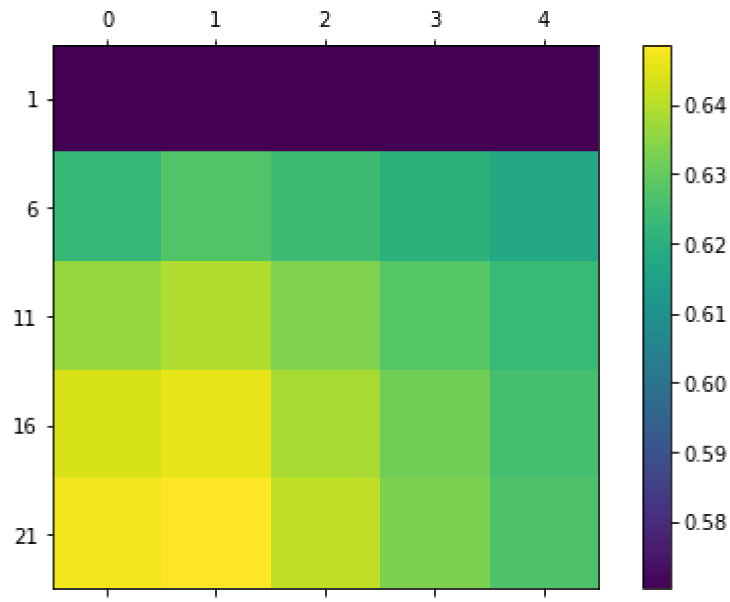
In [17]:
```
# Now plot it
print("TRAINING")
f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(tr_auc, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+A)
ax.set_yticklabels(['']+K)
plt.show()
```

```
TRAINING
```



In [18]:
```
print("TEST")
f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(va_auc, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+A)
ax.set_yticklabels(['']+K)
plt.show()
```

```
TEST
```



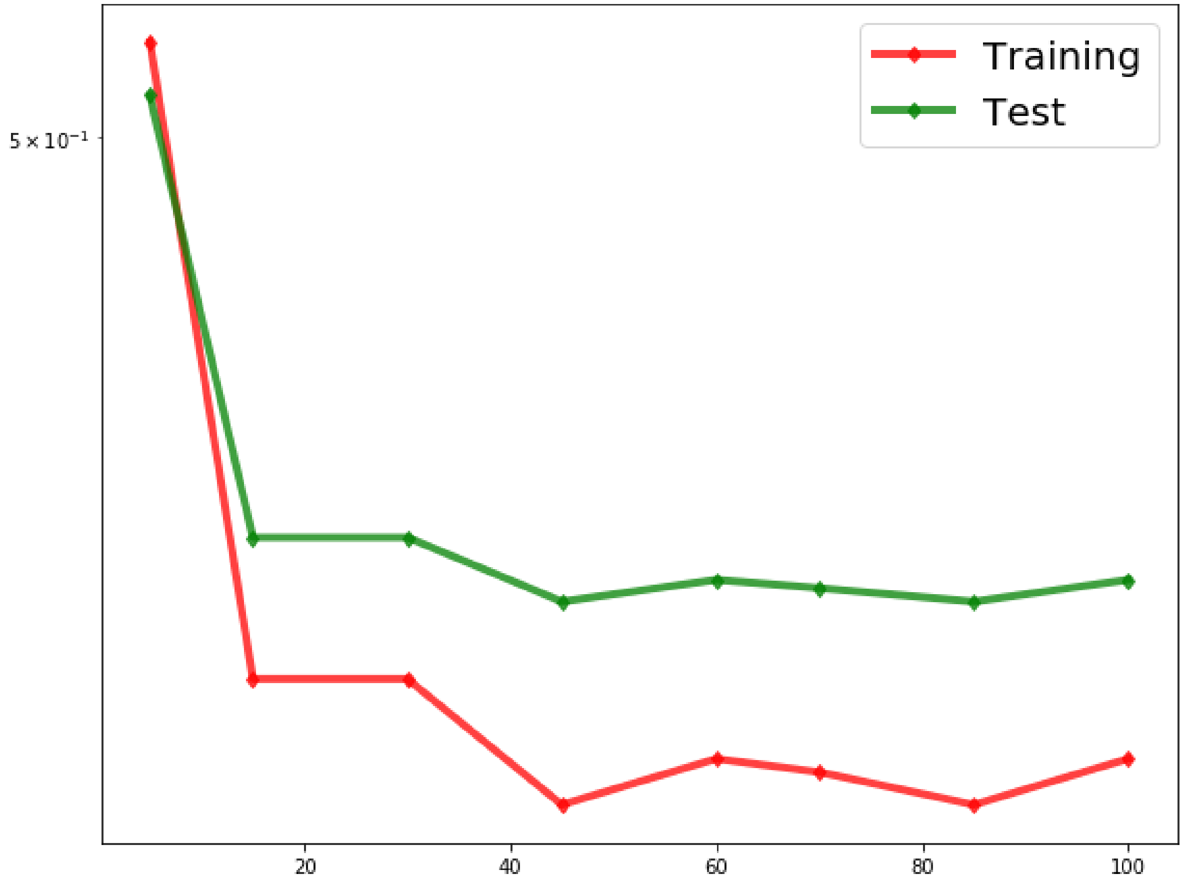
I would recommend a K value of ~21 with an alpha value of 1

4 Decision Trees

Problem 4.1

In [19]:
```
mDepths = [5,15,30,45,60,70,85,100]
trAUC = np.zeros(8)
valAUC = np.zeros(8)

for i, depth in enumerate(mDepths):
    learner = ml.dtree.treeClassify(Xt, Yt, minLeaf = 1, maxDepth=depth,
 minParent = 2)
    trAUC[i] = learner.auc(XtS, Yt) # train AUC
    valAUC[i] = learner.auc(XvS, Yva)
```

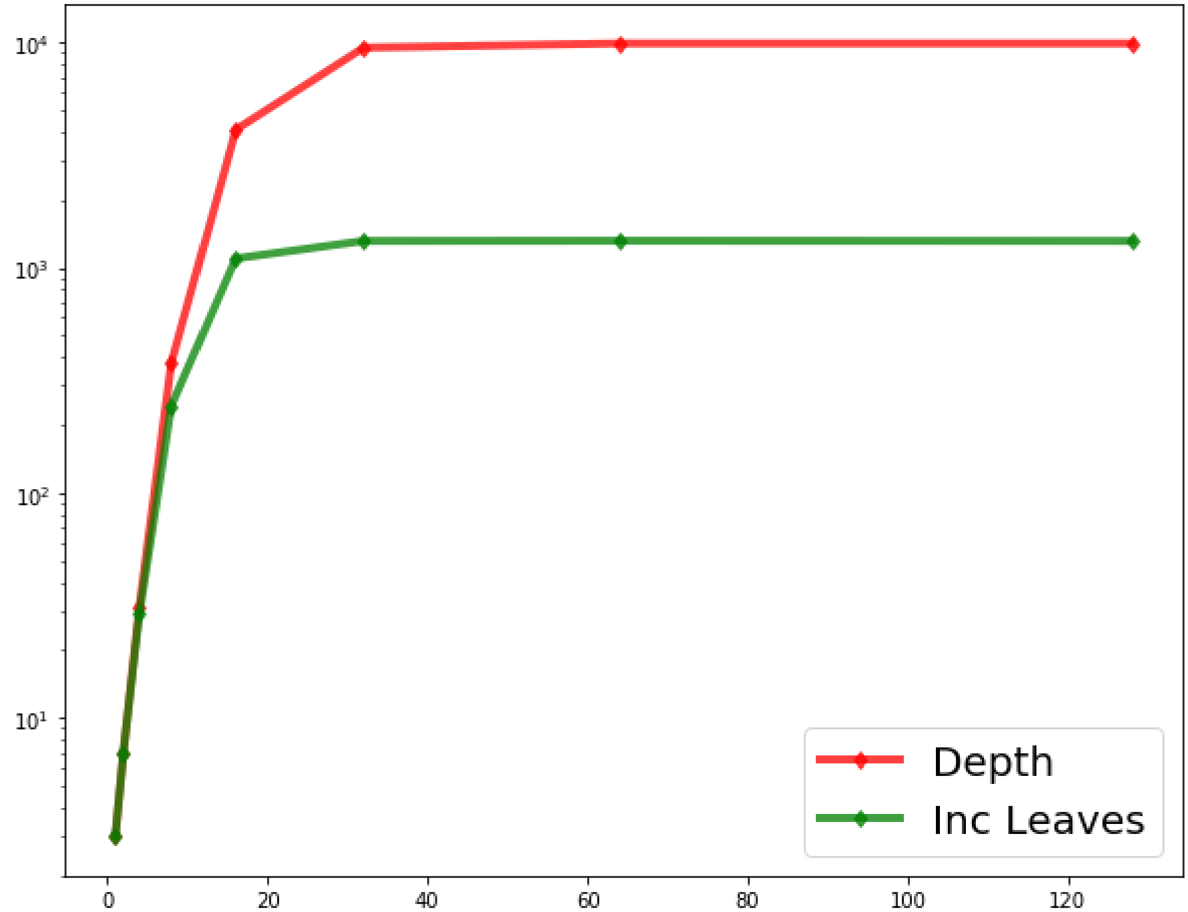
In [20]:
```
fig,ax=plt.subplots(1,1, figsize=(10, 8))
ax.semilogy(mDepths,trAUC,'r-',lw=4, marker='d', alpha=0.75, label='Trai
ning')
ax.semilogy(mDepths,valAUC,'g-',lw=4, marker='d', alpha=0.75, label='Tes
t')
ax.legend(fontsize=20, loc=0)
plt.show()
```



Problem 4.2

In [21]:
```
mDepths = [1,2,4,8,16,32,64,128] # Or something else
num_nodes1 = np.zeros(len(mDepths))
num_nodes2 = np.zeros(len(mDepths))

for i, depth in enumerate(mDepths):
    learner = ml.dtree.treeClassify(Xt, Yt, minLeaf = 1, maxDepth = dept
h, minParent = 2)
    num_nodes1[i] = learner.sz

for j, depth in enumerate(mDepths):
    learner = ml.dtree.treeClassify(Xt, Yt, minLeaf = 6, maxDepth = dept
h, minParent = 2)
    num_nodes2[j] = learner.sz
```

In [22]:
```
fig,ax=plt.subplots(1,1, figsize=(10, 8))
ax.semilogy(mDepths,num_nodes1,'r-',lw=4, marker='d', alpha=0.75, label=
'Depth')
ax.semilogy(mDepths,num_nodes2,'g-',lw=4, marker='d', alpha=0.75, label=
'Inc Leaves')
ax.legend(fontsize=20, loc=0)
plt.show()
```

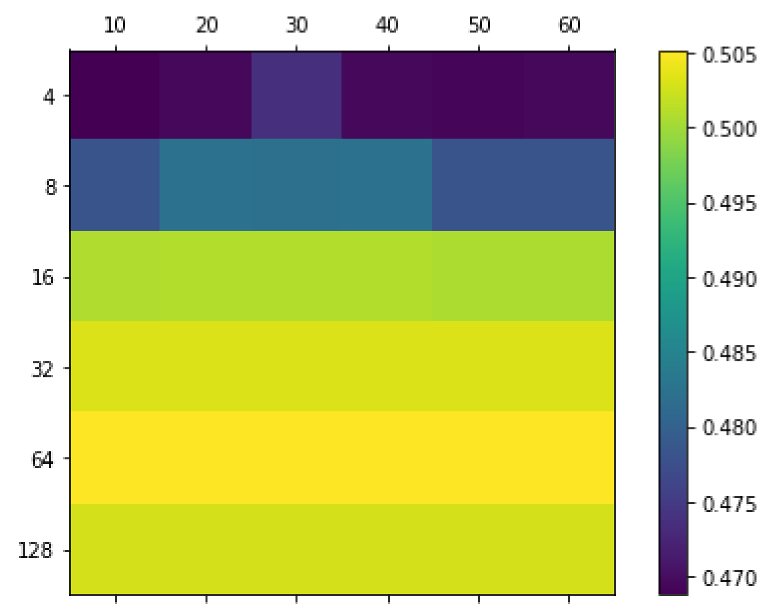


Problem 4.3

In [23]:
```
mLeaf = [4,8,16,32,64,128] # Or something else
mPar = [10,20,30,40,50,60] # Or something else
tr_auc = np.zeros((len(mLeaf),len(mPar)))
va_auc = np.zeros((len(mLeaf),len(mPar)))

for i, leaf in enumerate(mLeaf):
    for j, par in enumerate(mPar):
        learner = ml.dtree.treeClassify(Xt, Yt, minParent = par, maxDept
h=15, minLeaf = leaf)
        tr_auc[i][j] = learner.auc(XtS, Yt) # train learner using k and
 a
        va_auc[i][j] = learner.auc(XvS, Yva)
```

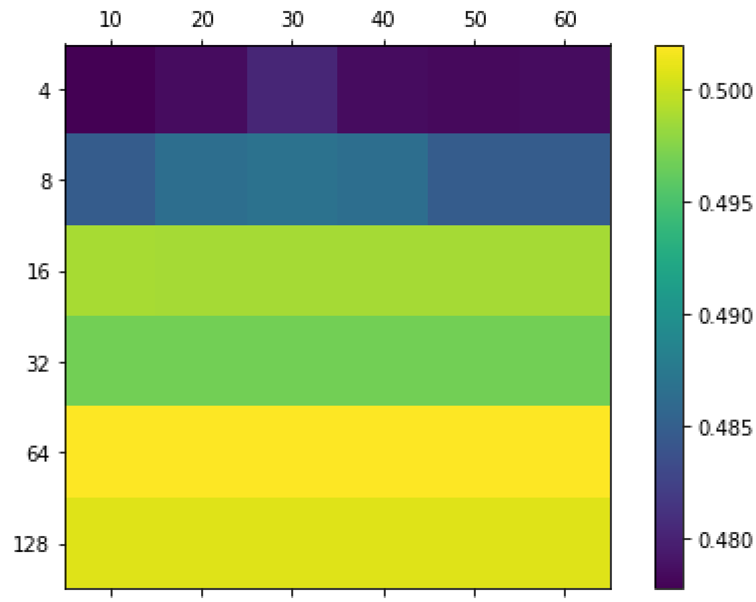
In [24]:
```
# Now plot it
print("TRAINING")
f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(tr_auc, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+mPar)
ax.set_yticklabels(['']+mLeaf)
plt.show()
```

TRAINING



In [25]:
```
print("TEST")
f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(va_auc, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+mPar)
ax.set_yticklabels(['']+mLeaf)
plt.show()
```

TEST



According to my validation plot, it looks as though you get the best AUC with a minLeaf value of ~64 no matter the size of minParent. Therefore I would recommend a minLeaf size of 64 and a minParent value of 20

5 Neural Networks

Problem 5.1

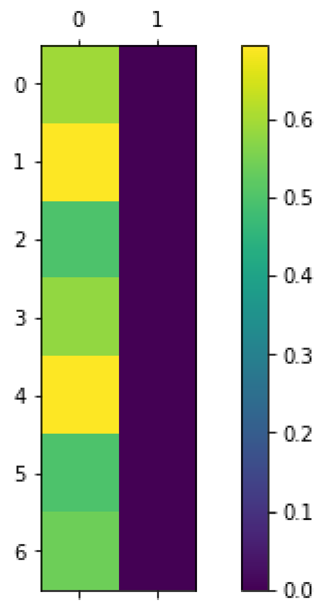
In [40]:
```
trnn_auc = np.zeros((7,2))
vann_auc = np.zeros((7,2))

nn = ml.nnet.nnetClassify()
nn.init_weights([XtS.shape[1], 6,6,6, 2], 'random', XtS, Yt) # as many l
ayers nodes you want
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
trnn_auc[0][0] = nn.auc(XtS,Yt)
vann_auc[0][1] = nn.auc(XvS, Yva)


nn.init_weights([XtS.shape[1], 8, 2], 'random', XtS, Yt) # as many layer
s nodes you want
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
trnn_auc[1][0] = nn.auc(XtS,Yt)
vann_auc[1][1] = nn.auc(XvS, Yva)


nn.init_weights([XtS.shape[1], 1,1,1,1,1,1, 2], 'random', XtS, Yt) # as
 many layers nodes you want
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
trnn_auc[2][0] = nn.auc(XtS,Yt)
vann_auc[2][1] = nn.auc(XvS, Yva)


nn.init_weights([XtS.shape[1], 3,3,3,3, 2], 'random', XtS, Yt) # as many
 layers nodes you want
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
trnn_auc[3][0] = nn.auc(XtS,Yt)
vann_auc[3][1] = nn.auc(XvS, Yva)


nn.init_weights([XtS.shape[1], 9,9, 2], 'random', XtS, Yt) # as many lay
ers nodes you want
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
trnn_auc[4][0] = nn.auc(XtS,Yt)
vann_auc[4][1] = nn.auc(XvS, Yva)


nn.init_weights([XtS.shape[1], 11,11,11,11,11, 2], 'random', XtS, Yt) #
 as many layers nodes you want
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
trnn_auc[5][0] = nn.auc(XtS,Yt)
vann_auc[5][1] = nn.auc(XvS, Yva)


nn.init_weights([XtS.shape[1], 7,7,7,7,7,7, 2], 'random', XtS, Yt) # as
 many layers nodes you want
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
trnn_auc[6][0] = nn.auc(XtS,Yt)
vann_auc[6][1] = nn.auc(XvS, Yva)
```

```
it 1 : Jsur = 0.458377048022, J01 = 0.3358
it 2 : Jsur = 0.451994865405, J01 = 0.3358
it 4 : Jsur = 0.448187339906, J01 = 0.3358
it 8 : Jsur = 0.446727377114, J01 = 0.3358
it 16 : Jsur = 0.44621736512, J01 = 0.3358
it 32 : Jsur = 0.446098371863, J01 = 0.3358
it 64 : Jsur = 0.446080002652, J01 = 0.3358
it 128 : Jsur = 0.446077293816, J01 = 0.3358
it 1 : Jsur = 0.426144848162, J01 = 0.3278
it 2 : Jsur = 0.417511875933, J01 = 0.3218
it 4 : Jsur = 0.41263595587, J01 = 0.316
it 8 : Jsur = 0.408965360036, J01 = 0.3096
it 16 : Jsur = 0.40612571949, J01 = 0.3036
it 32 : Jsur = 0.403637976118, J01 = 0.3004
it 64 : Jsur = 0.400833358079, J01 = 0.299
it 128 : Jsur = 0.398018900313, J01 = 0.3
it 256 : Jsur = 0.395218610031, J01 = 0.2974
it 1 : Jsur = 0.45837704704, J01 = 0.3358
it 2 : Jsur = 0.451994865327, J01 = 0.3358
it 4 : Jsur = 0.448187339889, J01 = 0.3358
it 8 : Jsur = 0.446727377108, J01 = 0.3358
it 16 : Jsur = 0.446217365116, J01 = 0.3358
it 32 : Jsur = 0.446098371861, J01 = 0.3358
it 64 : Jsur = 0.446080002651, J01 = 0.3358
it 128 : Jsur = 0.446077293815, J01 = 0.3358
it 1 : Jsur = 0.458377047166, J01 = 0.3358
it 2 : Jsur = 0.451994865331, J01 = 0.3358
it 4 : Jsur = 0.448187339896, J01 = 0.3358
it 8 : Jsur = 0.44672737711, J01 = 0.3358
it 16 : Jsur = 0.446217365117, J01 = 0.3358
it 32 : Jsur = 0.446098371862, J01 = 0.3358
it 64 : Jsur = 0.446080002651, J01 = 0.3358
it 128 : Jsur = 0.446077293815, J01 = 0.3358
it 1 : Jsur = 0.458377057753, J01 = 0.3358
it 2 : Jsur = 0.451994889567, J01 = 0.3358
it 4 : Jsur = 0.448187343422, J01 = 0.3358
it 8 : Jsur = 0.446727167326, J01 = 0.3358
it 16 : Jsur = 0.446201578972, J01 = 0.3358
it 32 : Jsur = 0.407790261555, J01 = 0.304
it 64 : Jsur = 0.402905135391, J01 = 0.3006
it 128 : Jsur = 0.397963506857, J01 = 0.2994
it 256 : Jsur = 0.394033352582, J01 = 0.293
it 1 : Jsur = 0.45837704959, J01 = 0.3358
it 2 : Jsur = 0.451994865522, J01 = 0.3358
it 4 : Jsur = 0.448187339933, J01 = 0.3358
it 8 : Jsur = 0.446727377124, J01 = 0.3358
it 16 : Jsur = 0.446217365123, J01 = 0.3358
it 32 : Jsur = 0.446098371865, J01 = 0.3358
it 64 : Jsur = 0.446080002651, J01 = 0.3358
it 128 : Jsur = 0.446077293815, J01 = 0.3358
it 1 : Jsur = 0.458377048934, J01 = 0.3358
it 2 : Jsur = 0.451994865509, J01 = 0.3358
it 4 : Jsur = 0.448187339911, J01 = 0.3358
it 8 : Jsur = 0.446727377115, J01 = 0.3358
it 16 : Jsur = 0.446217365118, J01 = 0.3358
it 32 : Jsur = 0.446098371863, J01 = 0.3358
```

```
it 64 : Jsur = 0.446080002651, J01 = 0.3358
it 128 : Jsur = 0.446077293815, J01 = 0.3358
```

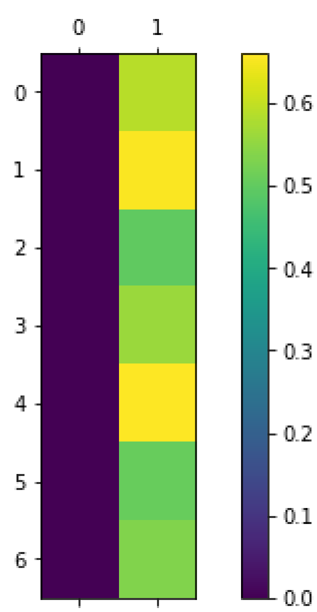
In [41]:
```
print("TRAINING")
f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(trnn_auc, interpolation='nearest')
f.colorbar(cax)
plt.show()
```

```
TRAINING
```



In [42]:
```
print("TEST")
f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(vann_auc, interpolation='nearest')
f.colorbar(cax)
plt.show()
```

```
TEST
```



I recommend an ideal network size to consist of 2 hidden layers of 9 nodes each layer

Problem 5.2

In [71]:
```
nn = ml.nnet.nnetClassify()
sig = lambda z: np.atleast_2d(z/(1 + abs(z)))
dsig = lambda z: np.atleast_2d(1/(1 + abs(z))**2)
nn.setActivation('custom', sig, dsig)

nn.init_weights([XtS.shape[1], 9,9, 2], 'random', XtS, Yt) # as many lay
ers nodes you want
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
print("Custom Activation Function")
print("{0:>15}: {1:.4f}".format('Train AUC',nn.auc(Xt, Yt)))
print("{0:>15}: {1:.4f}".format('Validation AUC', nn.auc(Xva, Yva)))
```

```
it 1 : Jsur = 0.458377052549, J01 = 0.3358
it 2 : Jsur = 0.451994885831, J01 = 0.3358
it 4 : Jsur = 0.448187343134, J01 = 0.3358
it 8 : Jsur = 0.446727175884, J01 = 0.3358
it 16 : Jsur = 0.446206733134, J01 = 0.3358
it 32 : Jsur = 0.408859100998, J01 = 0.3014
it 64 : Jsur = 0.403928242431, J01 = 0.3012
it 128 : Jsur = 0.398614184708, J01 = 0.3
it 256 : Jsur = 0.393933200986, J01 = 0.2948
Custom Activation Function
      Train AUC: 0.5460
 Validation AUC: 0.5502
```

In [72]:
```
nn.init_weights([XtS.shape[1], 9,9, 2], 'random', XtS, Yt) # as many lay
ers nodes you want
nn.setActivation('logistic', sig, dsig)
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
print("Logistic Activation Function")
print("{0:>15}: {1:.4f}".format('Train AUC',nn.auc(Xt, Yt)))
print("{0:>15}: {1:.4f}".format('Validation AUC', nn.auc(Xva, Yva)))
```

```
it 1 : Jsur = 0.437644841169, J01 = 0.3118
it 2 : Jsur = 0.431865749811, J01 = 0.3358
it 4 : Jsur = 0.42781539194, J01 = 0.3358
it 8 : Jsur = 0.426119548244, J01 = 0.3358
it 16 : Jsur = 0.425426808599, J01 = 0.3358
it 32 : Jsur = 0.42526119897, J01 = 0.3358
it 64 : Jsur = 0.425239892151, J01 = 0.3358
it 128 : Jsur = 0.42516013005, J01 = 0.3358
it 256 : Jsur = 0.425081252511, J01 = 0.3358
Logistic Activation Function
      Train AUC: 0.5347
 Validation AUC: 0.5322
```

In [73]:
```
nn.init_weights([XtS.shape[1], 9,9, 2], 'random', XtS, Yt) # as many lay
ers nodes you want
nn.setActivation('htangent', sig, dsig)
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
print("Hyperbolic Tangent Activation Function")
print("{0:>15}: {1:.4f}".format('Train AUC',nn.auc(Xt, Yt)))
print("{0:>15}: {1:.4f}".format('Validation AUC', nn.auc(Xva, Yva)))
```

```
it 1 : Jsur = 0.458377049376, J01 = 0.3358
it 2 : Jsur = 0.451994869403, J01 = 0.3358
it 4 : Jsur = 0.448187341322, J01 = 0.3358
it 8 : Jsur = 0.446727364274, J01 = 0.3358
it 16 : Jsur = 0.446217273415, J01 = 0.3358
it 32 : Jsur = 0.446097264136, J01 = 0.3358
it 64 : Jsur = 0.411325865215, J01 = 0.307
it 128 : Jsur = 0.40315722909, J01 = 0.301
it 256 : Jsur = 0.398150458564, J01 = 0.2952
Hyperbolic Tangent Activation Function
      Train AUC: 0.5752
 Validation AUC: 0.5645
```

My custom activation function performed slightly better than the logistic activation function, but overall the hyperbolic tangent activation function performed the best and gave the highest validation AUC. The difference between all three AUC values were relatively small, however.

6 Conclusions

I believe that the **Decision Tree classifier** will perform the best with a **minLeaf value of 64, a minParent value of 20, and a maxDepth value of 15**. Decision Tree classification resulted in the highest validation AUC out of all the other classifiers, which is the reason I believe it will perform best on the complete data set.

In [5]:
```
Xte = np.genfromtxt('data/X_test.txt', delimiter=None)
learner = ml.dtree.treeClassify(X, Y, minParent = 20, maxDepth=15, minLe
af = 64)
Yte = np.vstack((np.arange(Xte.shape[0]), learner.predictSoft(Xte)[:,1
])).T
np.savetxt('Y_submit.txt', Yte, '%d, %.2f', header='ID,Prob1', comments=
'', delimiter=',')
```

After making my submission to Kaggle, my predictions came out with a score of 0.70380, advancing my team up the leaderboard 119 spots to place at 43rd. My username is jmparnel and my group's name is **The Mean Squares**.

Statement of Collaboration

All the work on this homework was done by me and me alone. Some parts were a little tricky or confusing, and so for that I looked to piazza for tips to help clear up some of the confusion in order to finish this assignment. I also met up with Sergey Kochetov and Chad Lei in person on a couple of occasions to discuss certain aspects of classifiers that I didn't understand that would help me progress through this assignment. I would also ask them questions about some of the provided code that was already given in the homework if I did not understand its functionality. However, there was no sharing of personal code, only discussion about the given code.