Homework4

February 23, 2021

[24]:
```
import numpy as np
import mltools as ml
import matplotlib
import matplotlib.pyplot as plt
from IPython.display import display
import pandas as pd
plt.style.use('/Users/brookeryan/Developer/CS273A Homework/seawitch.mplstyle')
```

1 CS 273A Homework 4

1.1 Machine Learning, Winter 2021

1.1.1 Brooke Ryan

[25]:
```
# Data Loading
X = np.genfromtxt('data/X_train.txt', delimiter=None)
Y = np.genfromtxt('data/Y_train.txt', delimiter=None)
X, Y = ml.shuffleData(X, Y)
```

2 Problem 1

2.1 Setting up the data and Linear Classifier

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

[26]:
```
print('--------------------------')
print('          X               ')
print('--------------------------')
print('Min:', np.min(X, axis=0))
print('Max:', np.max(X, axis=0))
print('Mean:', np.mean(X, axis=0))
print('Var:', np.var(X,axis=0))
```

```
--------------------------
          X
--------------------------
Min: [ 1.9350e+02  1.5250e+02  2.1425e+02  1.5250e+02  1.0000e+01  0.0000e+00
  0.0000e+00  0.0000e+00  8.7589e-01  0.0000e+00  0.0000e+00  0.0000e+00
```

```
  9.9049e-01 -9.9990e+02]
Max: [2.5300e+02 2.4900e+02 2.5250e+02 2.5250e+02 3.1048e+04 1.3630e+04
 9.2380e+03 1.2517e+02 1.9167e+01 1.3230e+01 6.6761e+01 7.3902e+01
 9.7504e+02 7.9720e+02]
Mean: [2.41601104e+02 2.27376571e+02 2.41554150e+02 2.32826768e+02
 3.08992337e+03 9.28259020e+02 1.38093830e+02 3.24857933e+00
 6.49865290e+00 2.09713912e+00 4.21766041e+00 2.69171845e+00
 1.02715905e+01 5.78148050e+00]
Var: [8.34991711e+01 9.26255931e+01 3.52863398e+01 9.76257317e+01
 1.56515138e+07 3.08176182e+06 4.43951746e+05 8.21948502e+00
 6.40504819e+00 4.36344047e+00 4.08637188e+00 2.19877847e+00
 4.04646245e+02 3.40652055e+03]
```

2.1.2 2. Split the dataset, and rescale each into training and validation. Print the min, max, mean, and variance of the rescaled features.

[27]:
```
Xtr, Xva, Ytr, Yva = ml.splitData(X, Y)
Xt, Yt = Xtr[:5000], Ytr[:5000]
XtS, params = ml.rescale(Xt)
XvS, _ = ml.rescale(Xva, params)

print('--------------------------')
print('          XtS             ')
print('--------------------------')
print('Min:', np.min(XtS, axis=0))
print('Max:', np.max(XtS, axis=0))
print('Mean:', np.mean(XtS, axis=0))
print('Var:', np.var(XtS,axis=0))

print('--------------------------')
print('          XvS             ')
print('--------------------------')
print('Min:', np.min(XvS, axis=0))
print('Max:', np.max(XvS, axis=0))
print('Mean:', np.mean(XvS, axis=0))
print('Var:', np.var(XvS,axis=0))
```

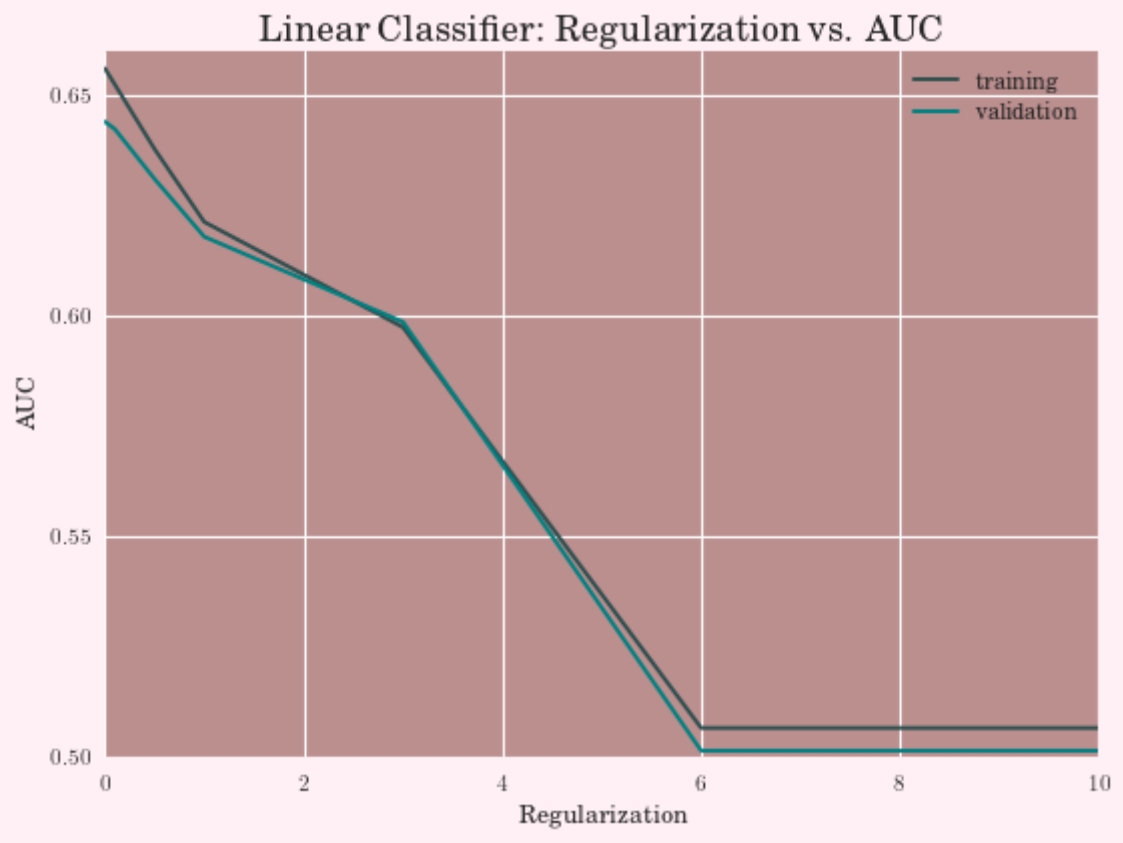
```
--------------------------
          XtS
--------------------------
Min: [ -4.95461163  -3.77377426  -4.32020667  -2.72309574  -0.80079029
  -0.53013724  -0.21359787  -1.13208597  -1.86949978  -0.99881312
  -2.04150726  -1.76459435  -0.56616293 -17.83750767]
Max: [ 1.25802831  1.82829827  1.7623924   1.89756632  7.35822512  7.27787363
 12.20948006  7.66054885  4.44531405  4.27556024  8.51935491 28.50291881
 29.21021596 13.9093136 ]
Mean: [-1.49759316e-14  1.24051880e-15 -5.27949240e-14 -1.72764025e-14
 -3.14193116e-17 -2.56850097e-17 -3.99831557e-16  1.16933130e-15
```

```
 -2.27000641e-15  1.88613569e-15 -3.31152883e-15  2.25992558e-15
 -6.67177424e-16  2.60541588e-17]
Var: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
--------------------------
          XvS
--------------------------
Min: [ -4.84749715  -3.77377426  -4.47140881  -2.82287251  -0.80210486
  -0.53013724  -0.21359787  -1.13208597  -2.18042818  -0.99881312
  -2.04150726  -1.76459435  -0.56755109 -17.83750767]
Max: [ 1.25802831  2.20852944  1.81498445  1.97439443  7.35822512  7.27787363
 12.20948006  8.11187516  4.44531405  5.19999292 11.06332139 34.24050622
 59.52517649 13.81548761]
Mean: [ 0.04499673  0.020381    0.02279997  0.0159113   0.00788919  0.00058467
 -0.03482025  0.01659666 -0.02328468 -0.019657    -0.006209    -0.01831332
 -0.00035561 -0.04281895]
Var: [0.9497013  0.9448252  0.94372135 0.96972101 1.08628933 1.00665293
 0.7556602  1.00937045 0.93525385 0.94766453 0.94872618 0.92641444
 1.27705585 1.20503815]
```

2.1.3 3. Vary the amount of regularization, reg, in a wide enough range and plot the training and validation AUC as the regularization weight is varied. Show the plot.

[28]:
```
reg = [0.0, 0.1, 0.5, 1.0, 3.0, 6.0, 8.0, 10.0]

tr_auc = np.zeros(len(reg))
va_auc = np.zeros(len(reg))

for i,r in enumerate(reg):
    learner = ml.linearC.linearClassify()
    learner.train(XtS, Yt, reg=r, initStep=0.5, stopTol=1e-6, stopIter=100)
    tr_auc[i] = learner.auc(XtS, Yt) # train AUC
    va_auc[i] = learner.auc(XvS, Yva)

plt.plot(reg, tr_auc, label='training')
plt.plot(reg, va_auc, label='validation')
plt.xlabel("Regularization")
plt.ylabel("AUC")
plt.title("Linear Classifier: Regularization vs. AUC")
plt.legend()
plt.show()
```

```
/Users/brookeryan/Developer/CS273A Homework/Homework4/mltools/linearC.py:122:
RuntimeWarning: overflow encountered in exp
  sigx  = np.exp(respi) / (1.0+np.exp(respi))
/Users/brookeryan/Developer/CS273A Homework/Homework4/mltools/linearC.py:122:
RuntimeWarning: invalid value encountered in true_divide
  sigx  = np.exp(respi) / (1.0+np.exp(respi))
```



2.1.4 4. Add *degree* 2 polynomial features, print out the number of features, and explain why it is what it is.

[29]:
```
# Add degree 2 polynomial features
XtP = ml.transforms.fpoly(Xt, 2, bias=False)
XtP, params = ml.transforms.rescale(XtP)
XvP = ml.transforms.rescale(ml.transforms.fpoly(Xva,2, bias=False), params)[0]

print("Number of Features (Xt):",Xt.shape[1])
print("Number of Features (XtP):",XtP.shape[1])
```

```
Number of Features (Xt): 14
Number of Features (XtP): 119
```

When our classifier is linear, the number of features is equal to 14.

As we can see from the transformation, the number of features has increased to 119 for a polynomial classifier.

This is because there will be 14 quadratic terms of features.

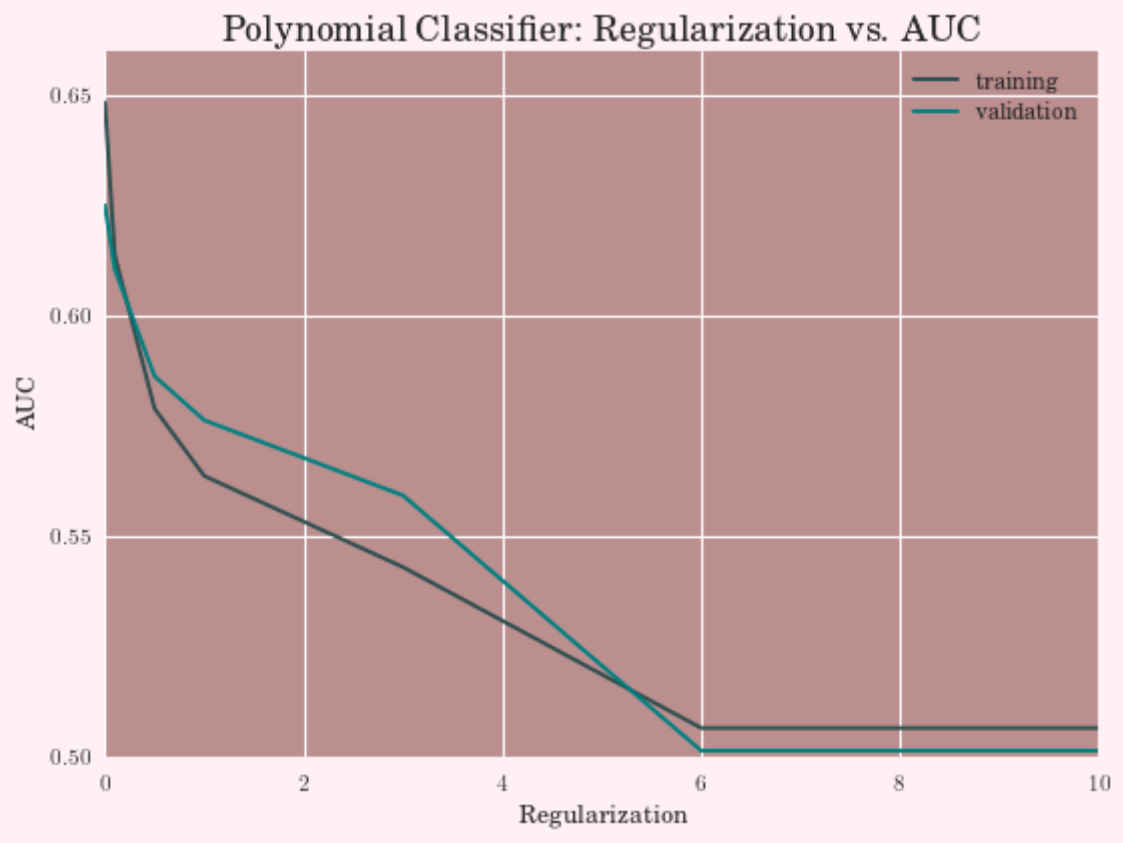
$C_{14}^2 = \frac{14*13}{2}$ terms of cross product of any two features add to the original 14 features.

As a result, $14 + 14 + (14 * 13)/2 = 119$, which is mathematically how we arrive at 119 features.

2.1.5 5. Reuse your code that varied regularization to compute the training and validation performance (AUC) for this transformed data. Show the plot.

[30]:
```
# Reset tr and va AUC values
tr_auc = np.zeros(len(reg))
va_auc = np.zeros(len(reg))

for i,r in enumerate(reg):
    learner = ml.linearC.linearClassify()
    learner.train(XtP, Yt, reg=r, initStep=0.5, stopTol=1e-6, stopIter=100)
    tr_auc[i] = learner.auc(XtP, Yt) # train AUC
    va_auc[i] = learner.auc(XvP, Yva)

plt.plot(reg, tr_auc, label='training')
plt.plot(reg, va_auc, label='validation')
plt.xlabel("Regularization")
plt.ylabel("AUC")
plt.title("Polynomial Classifier: Regularization vs. AUC")
plt.legend()
plt.show()
```

```
/Users/brookeryan/Developer/CS273A Homework/Homework4/mltools/base.py:97:
RuntimeWarning: divide by zero encountered in log
  return - np.mean( np.log( P[ np.arange(M), Y ] ) ) # evaluate
/Users/brookeryan/Developer/CS273A Homework/Homework4/mltools/linearC.py:134:
RuntimeWarning: invalid value encountered in double_scalars
  done = (it > stopIter) or ( (it>1) and (abs(Jsur[-1]-Jsur[-2])<stopTol) )
/Users/brookeryan/Developer/CS273A Homework/Homework4/mltools/linearC.py:122:
RuntimeWarning: overflow encountered in exp
  sigx  = np.exp(respi) / (1.0+np.exp(respi))
/Users/brookeryan/Developer/CS273A Homework/Homework4/mltools/linearC.py:122:
RuntimeWarning: invalid value encountered in true_divide
  sigx  = np.exp(respi) / (1.0+np.exp(respi))
```



3 Problem 2

3.1 Nearest Neighbors

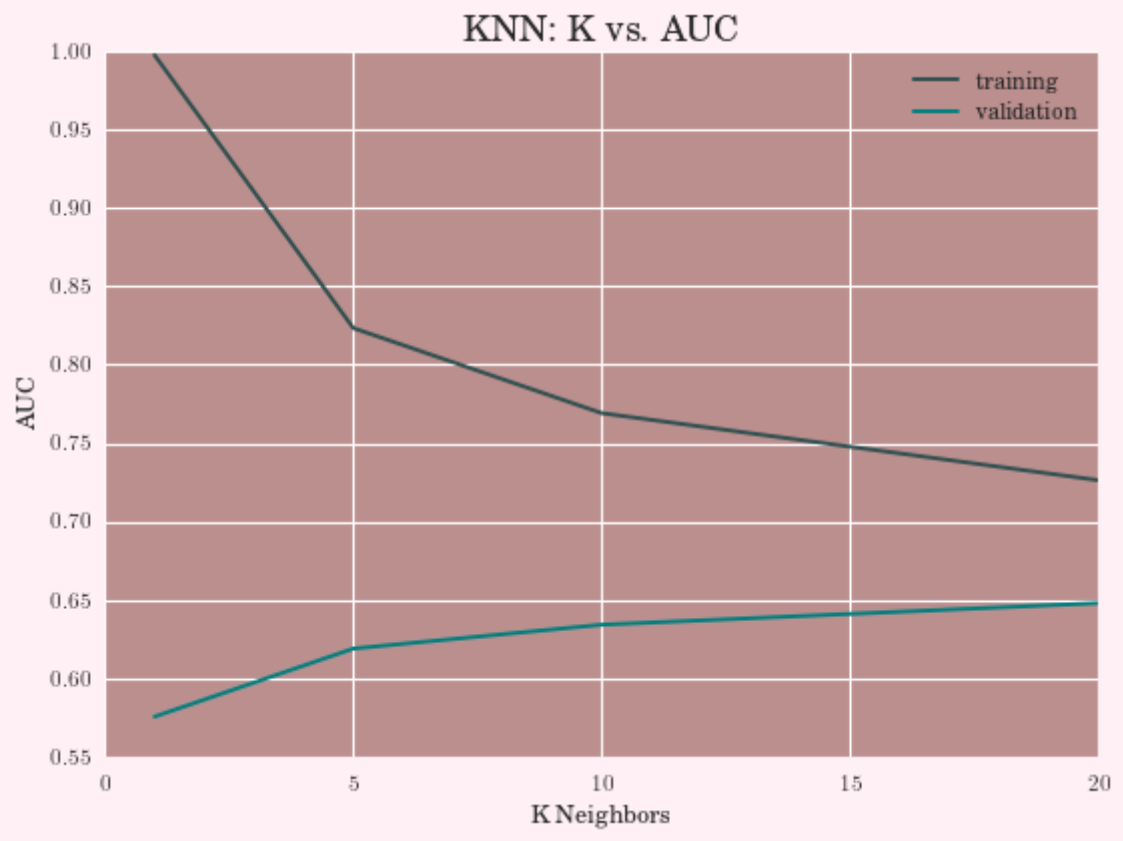
[31]:
```
learner = ml.knn.knnClassify()
learner.train(XtS, Yt, K=1, alpha=0.0)
learner.auc(XtS, Yt)
```

[31]:
```
0.9974687652518592
```

3.1.1 1. Plot of the training and validation performance for an appropriately wide range of K, with $\alpha = 0$.

[32]:
```
K = [1,5,10,20]
tr_auc = np.zeros(len(K))
va_auc = np.zeros(len(K))
learner = ml.knn.knnClassify()

for i,k in enumerate(K):
    learner.train(XtS, Yt, K=k, alpha=0.0)
```

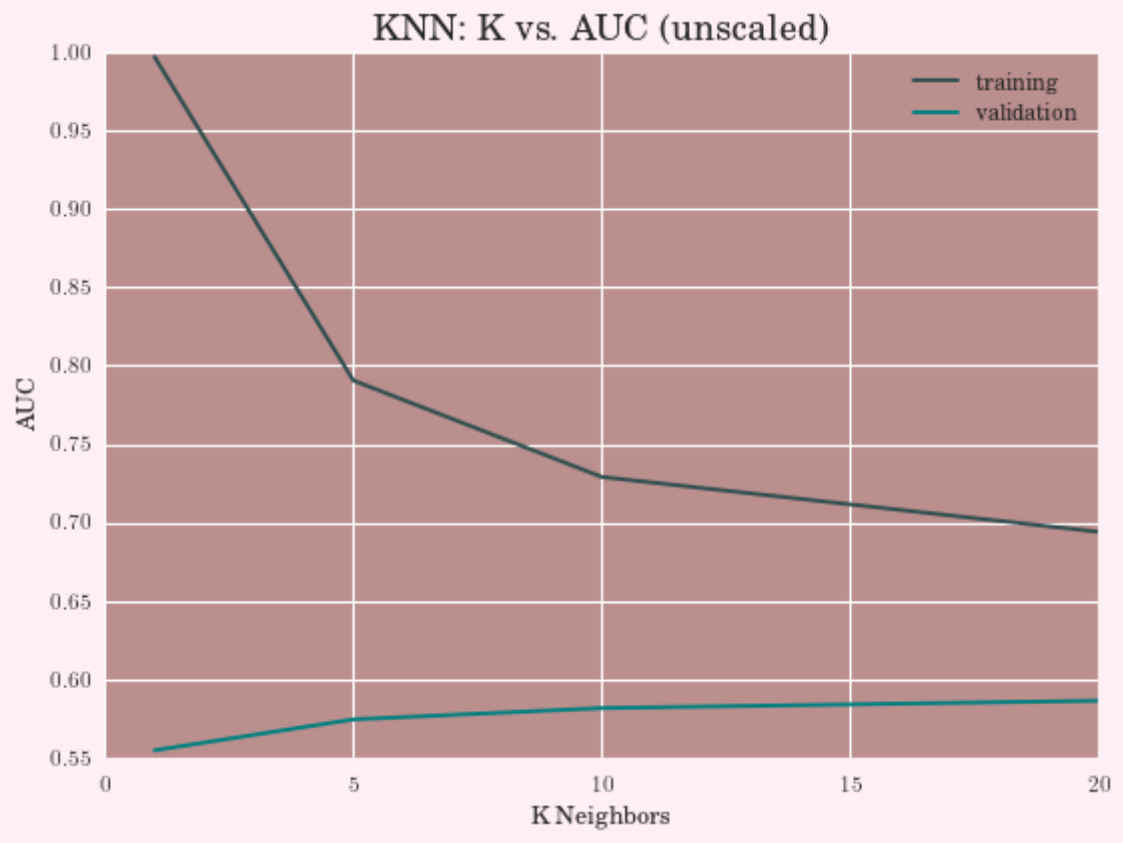
```
    tr_auc[i] = learner.auc(XtS, Yt)
    va_auc[i] = learner.auc(XvS, Yva)

plt.plot(K, tr_auc, label='training')
plt.plot(K, va_auc, label='validation')
plt.xlabel("K Neighbors")
plt.ylabel("AUC")
plt.title("KNN: K vs. AUC")
plt.legend()
plt.show()
```



3.1.2 2. Do the same with unscaled/ original data, and show the plots.

[33]:
```
tr_auc = np.zeros(len(K))
va_auc = np.zeros(len(K))

for i,k in enumerate(K):
    learner.train(Xt, Yt, K=k, alpha=0.0)
    tr_auc[i] = learner.auc(Xt, Yt)
    va_auc[i] = learner.auc(Xva, Yva)
```

```
plt.plot(K, tr_auc, label='training')
plt.plot(K, va_auc, label='validation')
plt.xlabel("K Neighbors")
plt.ylabel("AUC")
plt.title("KNN: K vs. AUC (unscaled)")
plt.legend()
plt.show()
```



3.1.3 3. For a range of both K and α, compute the training and validation AUC and plot them in a two-dimensional plot. Show both plots, and reccommend a choice of K and α based on those results.

In both examples, the general trend of the data seems to be very similar.

The kNN algorithm relies on majority voting, typically with Euclidean distance, which also appears to be the implementation used in the `mltools` library.

Non-normalized data can result in a skewed graph representation of the data, and cause the kNN classifier to completely ignore features that are indeed relevant to the model.

The implementation for normalization in the `mltools` library shifts and scales data to be zero

mean, unit variance in each dimension. The effect this would have on our cats vs. dogs dataset should result in a graphical representation in which all features are considered equally.

Without knowing more about the application or goal beyond binary classification with this dataset, I would probably recommend a *scaled* representation of the data. However the great thing with *machine* learning is that its really not a big deal to run some tests on both, so I'd feel unburdened to switch to an unscaled representation if we were seeing better results for the particular context, which is important to consider as well.

[37]:
```
K = [1,2,3,5,10]
A = range(0,4,1)
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.train(XtS, Yt, K=k, alpha=a)
        tr_auc[i][j] = learner.auc(XtS, Yt)
        va_auc[i][j] = learner.auc(XvS, Yva)
```

```
/Users/brookeryan/Developer/CS273A Homework/Homework4/mltools/knn.py:103:
RuntimeWarning: invalid value encountered in true_divide
  prob[i,:] = count / count.sum()    # save (soft) results
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-37-d7e642eab715> in <module>
      8         learner.train(XtS, Yt, K=k, alpha=a)
      9         tr_auc[i][j] = learner.auc(XtS, Yt)
---> 10         va_auc[i][j] = learner.auc(XvS, Yva)
     11
     12

~/Developer/CS273A Homework/Homework4/mltools/base.py in auc(self, X, Y)
    115
    116     try:                    # compute 'response' (soft binary␣
 ↪classification score)
--> 117         soft = self.predictSoft(X)[:,1]  # p(class = 2nd)
    118     except (AttributeError, IndexError):  # or we can use 'hard' binary␣
 ↪prediction if soft is unavailable
    119         soft = self.predict(X)

~/Developer/CS273A Homework/Homework4/mltools/knn.py in predictSoft(self, X)
     93         for i in range(mte):                    # for each test example…
     94             # …compute sum of squared differences…
---> 95             dist = np.sum(np.power(self.Xtr - arr(X)[i,:], 2), axis=1)
     96             # …find nearest neighbors over training data and keep␣
 ↪nearest K data points
```

```
     97             indices = np.argsort(dist, axis=0)[0:K]

KeyboardInterrupt:
```

[ ]:
```
f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax1 = ax.matshow(tr_auc, interpolation='nearest')
f.colorbar(cax1)
ax.set_xticklabels(['']+list(A))
ax.set_yticklabels(['']+list(K))
ax.set_xlabel("A")
ax.set_ylabel("K")
ax.set_title("KNN Training AUC (unscaled)")
plt.show()
```

[ ]:
```
f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax2 = ax.matshow(va_auc, interpolation='nearest')
f.colorbar(cax2)
ax.set_xticklabels(['']+list(A))
ax.set_yticklabels(['']+list(K))
ax.set_xlabel("A")
ax.set_ylabel("K")
ax.set_title("KNN validation AUC (unscaled)")
plt.show()
```

To be honest, I was not able to generate a graph for this. I believe this is due to the fact that I tried to do this homework on my local machine, and it does not have enough computational power to generate these graphs. I waited for 30 minutes.

However, based on a theoretical examination of the concepts from class as well as from the sources listed in the Sources Acknowledgement section, I recommend $K = 10$ and $a = 0$.

A higher K tends towards majority voting, and is less biased and localized compared to smaller values of K.

4 Problem 3

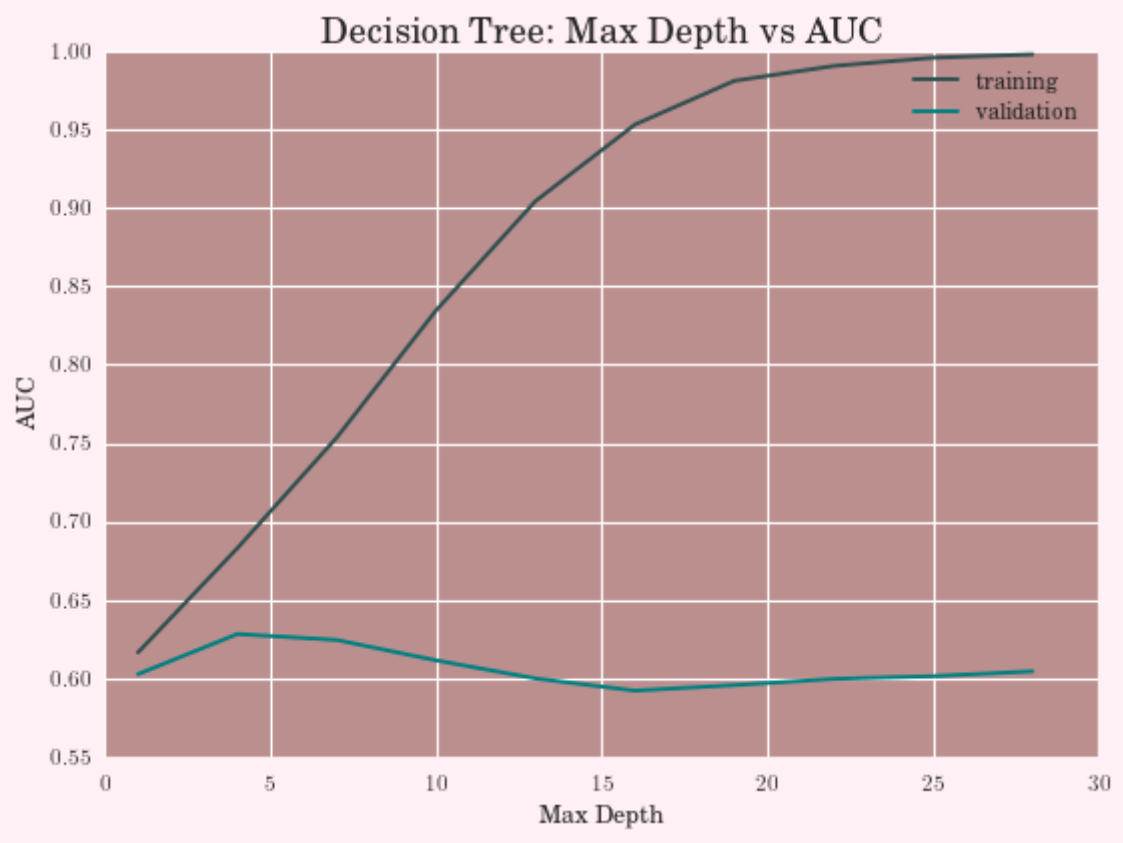
4.1 Decision trees

[39]:
```
learner = ml.dtree.treeClassify(Xt, Yt, maxDepth=15)
```

4.1.1 1. Keeping `minParent=2` and `minLeaf=1`, vary `maxDepth` to a range of your choosing, and plot the training and validation AUC.

[40]:
```
max_depth = range(1,30,3)
tr_auc = np.zeros(len(max_depth))
va_auc = np.zeros(len(max_depth))
```

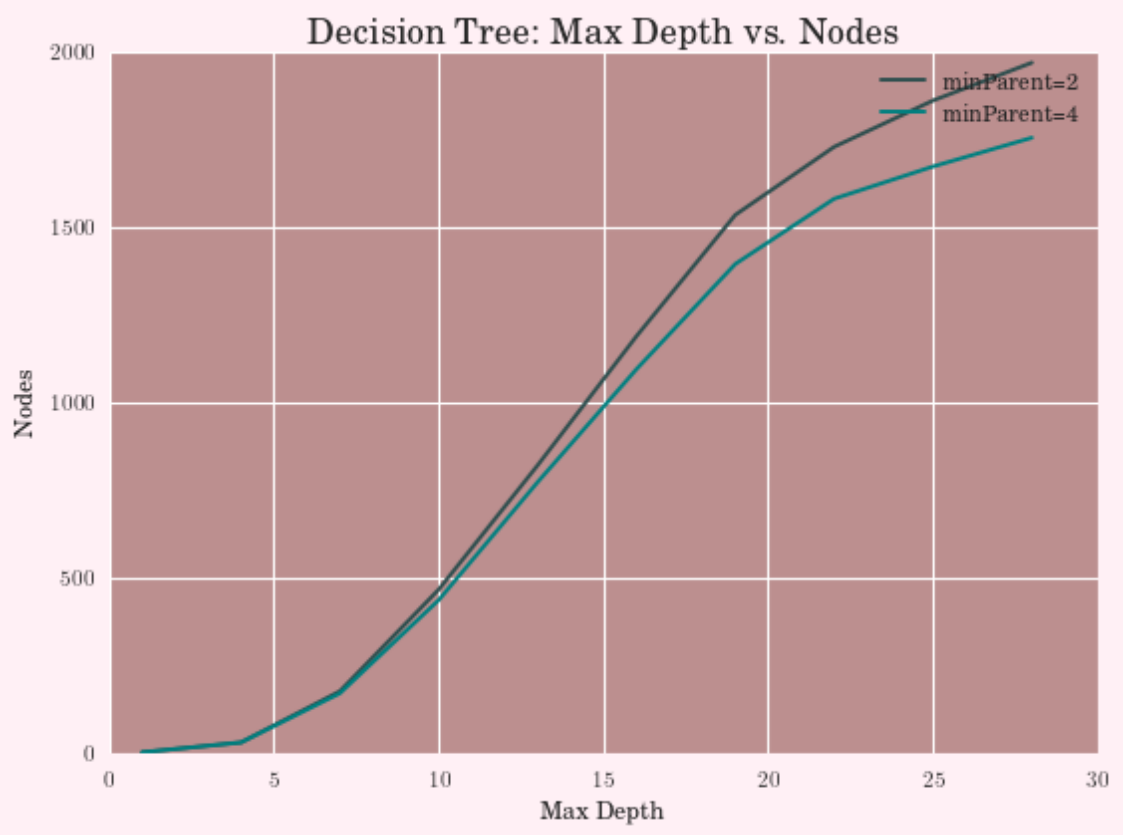
```
# Question 4.2
sz1 = np.zeros(len(max_depth))
sz2 = np.zeros(len(max_depth))

learner = ml.dtree.treeClassify()

for i,d in enumerate(max_depth):
    learner.train(Xt, Yt, maxDepth=d, minParent=2, minLeaf=1)
    tr_auc[i] = learner.auc(Xt, Yt)
    va_auc[i] = learner.auc(Xva, Yva)
    sz1[i] = learner.sz

    learner.train(Xt, Yt, maxDepth=d, minParent=4, minLeaf=1)
    sz2[i] = learner.sz

plt.plot(max_depth, tr_auc, label='training')
plt.plot(max_depth, va_auc, label='validation')
plt.xlabel("Max Depth")
plt.ylabel("AUC")
plt.title("Decision Tree: Max Depth vs AUC")
plt.legend()
plt.show()
```



4.1.2 2. Plot the number of nodes in the tree as `maxDepth` is varied. Plot another line in this plot by increasing either `minParent` or `minLeaf`.

In the code above, `sz1` represents the line where `minParent=2`, and `sz2` represents the line where `minParent` is increased to 4 for this test.

[41]:
```
plt.plot(max_depth, sz1, label='minParent=2')
plt.plot(max_depth, sz2, label='minParent=4')
plt.xlabel("Max Depth")
plt.ylabel("Nodes")
plt.title("Decision Tree: Max Depth vs. Nodes")
plt.legend()
plt.show()
```



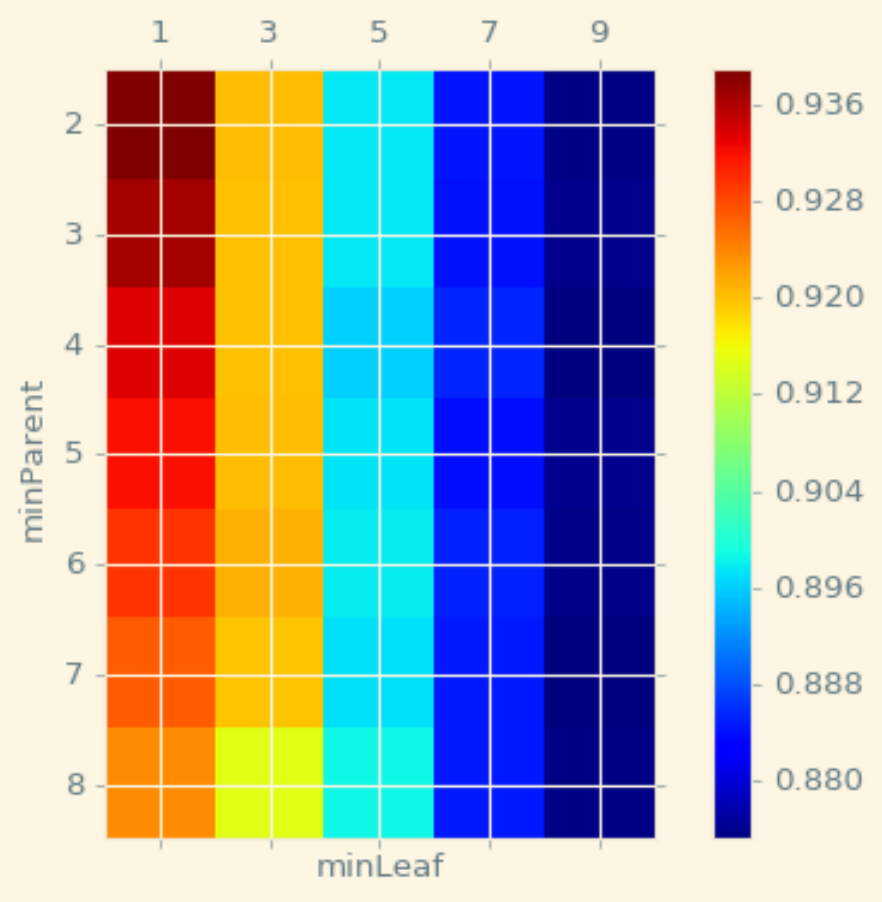
4.1.3 3. Set `maxDepth` to a fixed value, and plot the training and validation performance of the other two hyperparameters in an appropriate range, using the same 2D plot for nearest-neighbors. Show the plots, and reccommend a choice for `minParent` and `minLeaf` based on these results.

[43]:
```
# Set maxDepth = 15
mPar = range(2,9,1)
mLea = range(1,11,2)
tr_auc = np.zeros((len(mPar),len(mLea)))
va_auc = np.zeros((len(mPar),len(mLea)))
for i,p in enumerate(mPar):
    for j,l in enumerate(mLea):
        learner.train(Xt, Yt, maxDepth=15, \
                      minParent=p, minLeaf=l)
        tr_auc[i][j] = learner.auc(Xt, Yt)
        va_auc[i][j] = learner.auc(Xva, Yva)
```

4.1.4 Training

[60]:
```
plt.style.use('Solarize_Light2')    #this style works better for heat map graphs

f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax1 = ax.matshow(tr_auc, interpolation='nearest')
f.colorbar(cax1)
ax.set_xticklabels(['']+list(mLea))
ax.set_yticklabels(['']+list(mPar))
ax.set_xlabel("minLeaf")
ax.set_ylabel("minParent")
plt.show()
```

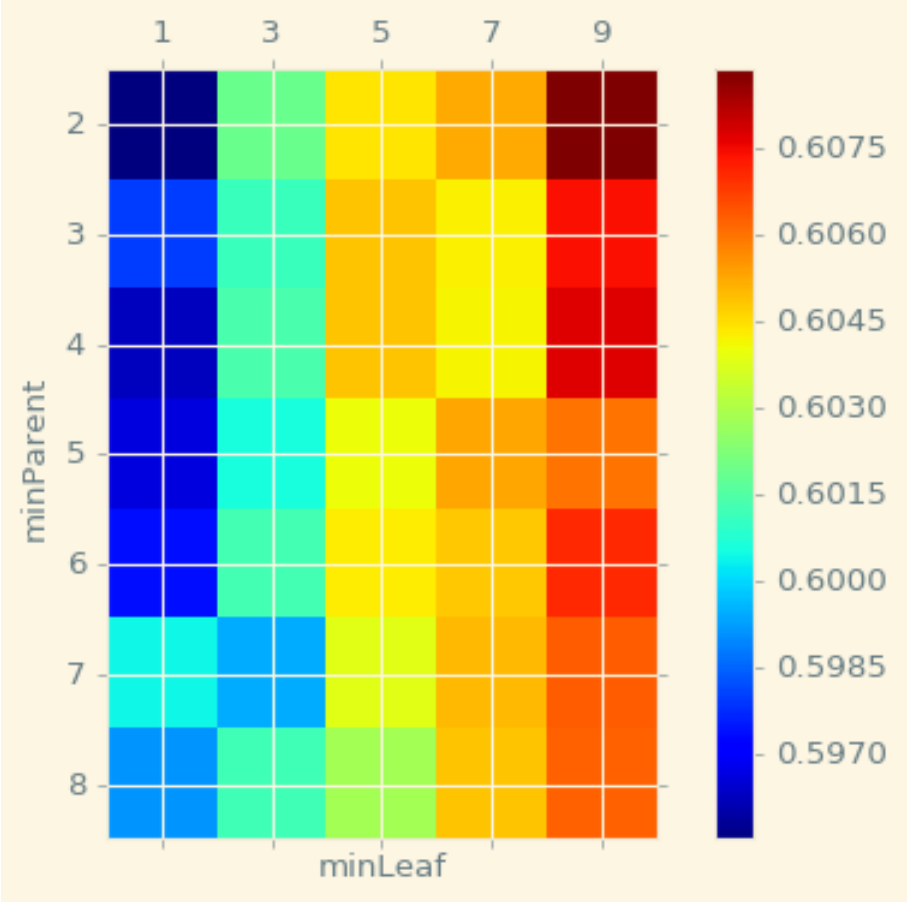
```
<ipython-input-60-88f05062f184>:6: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(['']+list(mLea))
<ipython-input-60-88f05062f184>:7: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_yticklabels(['']+list(mPar))
```



4.1.5 Validation

[61]:
```
plt.style.use('Solarize_Light2')    #this style works better for heat map graphs

f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax2 = ax.matshow(va_auc, interpolation='nearest')
f.colorbar(cax2)
ax.set_xticklabels(['']+list(mLea))
ax.set_yticklabels(['']+list(mPar))
ax.set_xlabel("minLeaf")
ax.set_ylabel("minParent")
plt.show()
```

```
<ipython-input-61-8bd62ded197a>:6: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(['']+list(mLea))
<ipython-input-61-8bd62ded197a>:7: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_yticklabels(['']+list(mPar))
```



As we can see from the graphs, AUC validation is the highest when minParent is equal to 8, and minLeaf is equal to 9.

Therefore based on this test and trying to maximize AUC on validation, I would recommend those parameters be used.

5 Problem 4

5.1 Neural Networks and Conclusion

5.1.1 1. Vary the number of hidden layers and the nodes in each layer (we will assume each layer has the same number of nodes), and compute the training and validation performance. Show 2D plots, like for decision trees and K-NN classifiers, and recommend a network size based on the above.

[63]:
```
nlayers = range(1,4,1)
nnodes = range(2,32,2)

tr_auc = np.zeros((len(nlayers),len(nnodes)))
va_auc = np.zeros((len(nlayers),len(nnodes)))
```

```
for i, layer in enumerate(nlayers):
    for j, node in enumerate(nnodes):
        nn = ml.nnet.nnetClassify()
        size = [XtS.shape[1]] + [node]*layer + [2]
        nn.init_weights(size, 'random', XtS, Yt)
        nn.train(XtS, Yt, stopTol=1e-8, stepConstant=.25, stopIter=300)
        tr_auc[i][j] = nn.auc(XtS, Yt)
        va_auc[i][j] = nn.auc(XvS, Yva)
```

```
it 2 : Jsur = 0.49854766711828286, J01 = 0.3468
it 4 : Jsur = 0.4982167055539039, J01 = 0.3468
it 8 : Jsur = 0.49789980787457533, J01 = 0.3468
it 16 : Jsur = 0.49759095769171197, J01 = 0.3468
it 32 : Jsur = 0.49754332038625787, J01 = 0.3468
it 64 : Jsur = 0.49741118681428886, J01 = 0.3468
it 128 : Jsur = 0.4973620388157623, J01 = 0.3468
it 256 : Jsur = 0.4972514721565907, J01 = 0.3468
it 512 : Jsur = 0.4971671575584511, J01 = 0.3468
it 2 : Jsur = 0.4975441963324283, J01 = 0.3468
it 4 : Jsur = 0.49721803061803294, J01 = 0.3468
it 8 : Jsur = 0.49719181641038357, J01 = 0.3468
it 16 : Jsur = 0.4969571260239447, J01 = 0.3468
it 32 : Jsur = 0.49703111458911553, J01 = 0.3468
it 64 : Jsur = 0.4970084164855507, J01 = 0.3468
it 128 : Jsur = 0.49684242549098306, J01 = 0.3468
it 256 : Jsur = 0.4966970331473934, J01 = 0.3468
it 512 : Jsur = 0.49660712272296115, J01 = 0.3468
it 2 : Jsur = 0.4976873576452492, J01 = 0.3468
it 4 : Jsur = 0.497749778260442, J01 = 0.3468
it 8 : Jsur = 0.49743496989525865, J01 = 0.3468
it 16 : Jsur = 0.49728380831596003, J01 = 0.3468
it 32 : Jsur = 0.497265969239694, J01 = 0.3468
it 64 : Jsur = 0.4971878293615713, J01 = 0.3468
it 128 : Jsur = 0.49708575110229175, J01 = 0.3468
it 256 : Jsur = 0.49700753261283326, J01 = 0.3468
it 512 : Jsur = 0.49690619818287596, J01 = 0.3468
it 2 : Jsur = 0.4988813888582979, J01 = 0.3468
it 4 : Jsur = 0.49854901147825464, J01 = 0.3468
it 8 : Jsur = 0.4983523791342694, J01 = 0.3468
it 16 : Jsur = 0.4983399426836559, J01 = 0.3468
it 32 : Jsur = 0.4983359248885379, J01 = 0.3468
it 64 : Jsur = 0.4982960824029982, J01 = 0.3468
it 128 : Jsur = 0.49818629158224886, J01 = 0.3468
it 256 : Jsur = 0.49807183560499396, J01 = 0.3468
it 512 : Jsur = 0.4979882075027994, J01 = 0.3468
it 2 : Jsur = 0.5023260134626119, J01 = 0.6532
```

```
it 4 : Jsur = 0.5022603609094834, J01 = 0.6532
it 8 : Jsur = 0.5024642832313196, J01 = 0.6532
it 16 : Jsur = 0.5022466909181258, J01 = 0.6532
it 32 : Jsur = 0.5020099264895688, J01 = 0.6532
it 64 : Jsur = 0.5019715551361066, J01 = 0.6532
it 128 : Jsur = 0.5018718394392283, J01 = 0.6532
it 256 : Jsur = 0.5017506933174074, J01 = 0.6532
it 512 : Jsur = 0.501650630071826, J01 = 0.6532
it 2 : Jsur = 0.5019042376523308, J01 = 0.6532
it 4 : Jsur = 0.5015574161188371, J01 = 0.6532
it 8 : Jsur = 0.5013713455818848, J01 = 0.6532
it 16 : Jsur = 0.501145338639738, J01 = 0.6532
it 32 : Jsur = 0.5010390275031056, J01 = 0.6532
it 64 : Jsur = 0.5008993261224012, J01 = 0.6532
it 128 : Jsur = 0.5007203544154954, J01 = 0.6532
it 256 : Jsur = 0.5006388250686866, J01 = 0.6532
it 512 : Jsur = 0.5005212052971613, J01 = 0.6532
it 2 : Jsur = 0.49821074886623684, J01 = 0.3468
it 4 : Jsur = 0.49827295856641074, J01 = 0.3468
it 8 : Jsur = 0.4981630171463814, J01 = 0.3468
it 16 : Jsur = 0.49804887504883916, J01 = 0.3468
it 32 : Jsur = 0.49790348897507547, J01 = 0.3468
it 64 : Jsur = 0.4978161487423478, J01 = 0.3468
it 128 : Jsur = 0.497797830036867, J01 = 0.3468
it 256 : Jsur = 0.4976828554014284, J01 = 0.3468
it 512 : Jsur = 0.49762124851539813, J01 = 0.3468
it 2 : Jsur = 0.49764297748204483, J01 = 0.3468
it 4 : Jsur = 0.49770540891217124, J01 = 0.3468
it 8 : Jsur = 0.4976516938028139, J01 = 0.3468
it 16 : Jsur = 0.4975672482363612, J01 = 0.3468
it 32 : Jsur = 0.49746290951845984, J01 = 0.3468
it 64 : Jsur = 0.4973957530425436, J01 = 0.3468
it 128 : Jsur = 0.4972853263659744, J01 = 0.3468
it 256 : Jsur = 0.4971657624617263, J01 = 0.3468
it 512 : Jsur = 0.49706524767324045, J01 = 0.3468
it 2 : Jsur = 0.49791477371350756, J01 = 0.3468
it 4 : Jsur = 0.4978572063505746, J01 = 0.3468
it 8 : Jsur = 0.4981782561174556, J01 = 0.3468
it 16 : Jsur = 0.49810397951411495, J01 = 0.3468
it 32 : Jsur = 0.49803363293864905, J01 = 0.3468
it 64 : Jsur = 0.4978982341300498, J01 = 0.3468
it 128 : Jsur = 0.49784648929872194, J01 = 0.3468
it 256 : Jsur = 0.49776116353377275, J01 = 0.3468
it 512 : Jsur = 0.4977024735190439, J01 = 0.3468
it 2 : Jsur = 0.4985578514267168, J01 = 0.3468
it 4 : Jsur = 0.4986198864255124, J01 = 0.3468
it 8 : Jsur = 0.4983011567705009, J01 = 0.3468
it 16 : Jsur = 0.49839964541550796, J01 = 0.3468
```

```
it 32 : Jsur = 0.49835497303596404, J01 = 0.3468
it 64 : Jsur = 0.4982992584379321, J01 = 0.3468
it 128 : Jsur = 0.4982063228153216, J01 = 0.3468
it 256 : Jsur = 0.4981770890830594, J01 = 0.3468
it 512 : Jsur = 0.4980781172578803, J01 = 0.3468
it 2 : Jsur = 0.4977042182589976, J01 = 0.3468
it 4 : Jsur = 0.4973771956410019, J01 = 0.3468
it 8 : Jsur = 0.49706402532428057, J01 = 0.3468
it 16 : Jsur = 0.496819678128392, J01 = 0.3468
it 32 : Jsur = 0.4966371376848903, J01 = 0.3468
it 64 : Jsur = 0.4964652730907184, J01 = 0.3468
it 128 : Jsur = 0.4963478500709102, J01 = 0.3468
it 256 : Jsur = 0.49624599732045754, J01 = 0.3468
it 512 : Jsur = 0.49615257532276463, J01 = 0.3468
it 2 : Jsur = 0.5012440212959953, J01 = 0.6532
it 4 : Jsur = 0.500900244136329, J01 = 0.6532
it 8 : Jsur = 0.5005711751951352, J01 = 0.6532
it 16 : Jsur = 0.5003596298623761, J01 = 0.6532
it 32 : Jsur = 0.5002020876442943, J01 = 0.6532
it 64 : Jsur = 0.5001172342231334, J01 = 0.653
it 128 : Jsur = 0.5000315177738219, J01 = 0.6508
it 256 : Jsur = 0.49989560162570235, J01 = 0.3472
it 512 : Jsur = 0.4998085036334269, J01 = 0.3468
it 2 : Jsur = 0.5012654106284781, J01 = 0.6532
it 4 : Jsur = 0.5009213726048019, J01 = 0.6532
it 8 : Jsur = 0.5007370677508772, J01 = 0.6532
it 16 : Jsur = 0.5005594017660057, J01 = 0.6532
it 32 : Jsur = 0.5005823777961376, J01 = 0.6532
it 64 : Jsur = 0.5004868940588009, J01 = 0.6532
it 128 : Jsur = 0.5004429355315277, J01 = 0.6532
it 256 : Jsur = 0.500348036981891, J01 = 0.6532
it 512 : Jsur = 0.5002508654889408, J01 = 0.6532
it 2 : Jsur = 0.498138781864185, J01 = 0.3468
it 4 : Jsur = 0.49780963391721417, J01 = 0.3468
it 8 : Jsur = 0.4976347810184283, J01 = 0.3468
it 16 : Jsur = 0.497473612877887, J01 = 0.3468
it 32 : Jsur = 0.4974131526221281, J01 = 0.3468
it 64 : Jsur = 0.4973265234762841, J01 = 0.3468
it 128 : Jsur = 0.49726931393865653, J01 = 0.3468
it 256 : Jsur = 0.49718928268775897, J01 = 0.3468
it 512 : Jsur = 0.4970926551123807, J01 = 0.3468
it 2 : Jsur = 0.498135846072886, J01 = 0.3468
it 4 : Jsur = 0.49780744802641347, J01 = 0.3468
it 8 : Jsur = 0.49783896221119084, J01 = 0.3468
it 16 : Jsur = 0.4975298937888332, J01 = 0.3468
it 32 : Jsur = 0.49736721816773133, J01 = 0.3468
it 64 : Jsur = 0.4971736119015099, J01 = 0.3468
it 128 : Jsur = 0.49708382285248465, J01 = 0.3468
```

```
it 256 : Jsur = 0.49701285766703057, J01 = 0.3468
it 512 : Jsur = 0.49691537209444686, J01 = 0.3468
it 2 : Jsur = 0.4985317258835111, J01 = 0.3468
it 4 : Jsur = 0.49886690132776257, J01 = 0.3468
it 8 : Jsur = 0.49854714877891804, J01 = 0.3468
it 16 : Jsur = 0.4983021249124733, J01 = 0.3468
it 32 : Jsur = 0.49820888919126527, J01 = 0.3468
it 64 : Jsur = 0.49806449979849277, J01 = 0.3468
it 128 : Jsur = 0.49795636396052095, J01 = 0.3468
it 256 : Jsur = 0.49785674431662336, J01 = 0.3468
it 512 : Jsur = 0.49774243555949615, J01 = 0.3468
it 2 : Jsur = 0.4987503858559126, J01 = 0.3468
it 4 : Jsur = 0.49869113940998105, J01 = 0.3468
it 8 : Jsur = 0.4985136419775203, J01 = 0.3468
it 16 : Jsur = 0.49843341842774463, J01 = 0.3468
it 32 : Jsur = 0.4984832008121662, J01 = 0.3468
it 64 : Jsur = 0.4983941177503919, J01 = 0.3468
it 128 : Jsur = 0.49830261929947645, J01 = 0.3468
it 256 : Jsur = 0.4982016068454183, J01 = 0.3468
it 512 : Jsur = 0.4980923329789368, J01 = 0.3468
it 2 : Jsur = 0.4973703906283844, J01 = 0.3468
it 4 : Jsur = 0.4970450316283067, J01 = 0.3468
it 8 : Jsur = 0.4970382328686228, J01 = 0.3468
it 16 : Jsur = 0.49687493554194273, J01 = 0.3468
it 32 : Jsur = 0.496814453812115, J01 = 0.3468
it 64 : Jsur = 0.4967024020012752, J01 = 0.3468
it 128 : Jsur = 0.4965685868350728, J01 = 0.3468
it 256 : Jsur = 0.4964774315469835, J01 = 0.3468
it 512 : Jsur = 0.49642253195076136, J01 = 0.3468
it 2 : Jsur = 0.4986144813347255, J01 = 0.3468
it 4 : Jsur = 0.4982832981522695, J01 = 0.3468
it 8 : Jsur = 0.4983969841536509, J01 = 0.3468
it 16 : Jsur = 0.49819276194055817, J01 = 0.3468
it 32 : Jsur = 0.4980514406918896, J01 = 0.3468
it 64 : Jsur = 0.49790047215506866, J01 = 0.3468
it 128 : Jsur = 0.4977762204954364, J01 = 0.3468
it 256 : Jsur = 0.4976946341029906, J01 = 0.3468
it 512 : Jsur = 0.4976146816517743, J01 = 0.3468
it 2 : Jsur = 0.497759640259085, J01 = 0.3468
it 4 : Jsur = 0.4978220783851286, J01 = 0.3468
it 8 : Jsur = 0.4977950243421137, J01 = 0.3468
it 16 : Jsur = 0.49793559850471036, J01 = 0.3468
it 32 : Jsur = 0.4977761890640699, J01 = 0.3468
it 64 : Jsur = 0.4977536676473007, J01 = 0.3468
it 128 : Jsur = 0.4977032582972564, J01 = 0.3468
it 256 : Jsur = 0.4976012000116956, J01 = 0.3468
it 512 : Jsur = 0.49752067602038297, J01 = 0.3468
it 2 : Jsur = 0.5020645836932525, J01 = 0.6532
```

```
it 4 : Jsur = 0.5017170149808345, J01 = 0.6532
it 8 : Jsur = 0.5017736833967056, J01 = 0.6532
it 16 : Jsur = 0.5017907277740059, J01 = 0.6532
it 32 : Jsur = 0.5016238209906246, J01 = 0.6532
it 64 : Jsur = 0.5015186026971314, J01 = 0.6532
it 128 : Jsur = 0.501380293126284, J01 = 0.6532
it 256 : Jsur = 0.5013071759804472, J01 = 0.6532
it 512 : Jsur = 0.5012051116914215, J01 = 0.6532
it 2 : Jsur = 0.4988349756330432, J01 = 0.3468
it 4 : Jsur = 0.49917111227912503, J01 = 0.3468
it 8 : Jsur = 0.49905910972436807, J01 = 0.3468
it 16 : Jsur = 0.4988120455884331, J01 = 0.3468
it 32 : Jsur = 0.49870961778384815, J01 = 0.3468
it 64 : Jsur = 0.49853305326721936, J01 = 0.3468
it 128 : Jsur = 0.49844039497270953, J01 = 0.3468
it 256 : Jsur = 0.4983345019098314, J01 = 0.3468
it 512 : Jsur = 0.498227695136256, J01 = 0.3468
it 2 : Jsur = 0.49767416471434883, J01 = 0.3468
it 4 : Jsur = 0.49761678874906695, J01 = 0.3468
it 8 : Jsur = 0.4975083136676869, J01 = 0.3468
it 16 : Jsur = 0.4972723985432196, J01 = 0.3468
it 32 : Jsur = 0.4971882951374454, J01 = 0.3468
it 64 : Jsur = 0.4971051709235333, J01 = 0.3468
it 128 : Jsur = 0.49702655269135865, J01 = 0.3468
it 256 : Jsur = 0.4968931571150777, J01 = 0.3468
it 512 : Jsur = 0.49682765074414253, J01 = 0.3468
it 2 : Jsur = 0.5014522668931491, J01 = 0.6532
it 4 : Jsur = 0.5013882556297888, J01 = 0.6532
it 8 : Jsur = 0.5014450794739822, J01 = 0.6532
it 16 : Jsur = 0.5011891484693817, J01 = 0.6532
it 32 : Jsur = 0.501033962379215, J01 = 0.6532
it 64 : Jsur = 0.5009067087894433, J01 = 0.6532
it 128 : Jsur = 0.5007659594279801, J01 = 0.6532
it 256 : Jsur = 0.5006590793657127, J01 = 0.6532
it 512 : Jsur = 0.5005403919304124, J01 = 0.6532
it 2 : Jsur = 0.4977156515863082, J01 = 0.3468
it 4 : Jsur = 0.49765821220421796, J01 = 0.3468
it 8 : Jsur = 0.4973438332095524, J01 = 0.3468
it 16 : Jsur = 0.497517345239379, J01 = 0.3468
it 32 : Jsur = 0.4974465083073028, J01 = 0.3468
it 64 : Jsur = 0.4973541141949261, J01 = 0.3468
it 128 : Jsur = 0.49720715274365784, J01 = 0.3468
it 256 : Jsur = 0.497150478942667, J01 = 0.3468
it 512 : Jsur = 0.49703557742623433, J01 = 0.3468
it 2 : Jsur = 0.49776125853541164, J01 = 0.3468
it 4 : Jsur = 0.49770373492501296, J01 = 0.3468
it 8 : Jsur = 0.4976768370269794, J01 = 0.3468
it 16 : Jsur = 0.49752491410995353, J01 = 0.3468
```

```
it 32 : Jsur = 0.49749263177499303, J01 = 0.3468
it 64 : Jsur = 0.4974672094610456, J01 = 0.3468
it 128 : Jsur = 0.4973676734605264, J01 = 0.3468
it 256 : Jsur = 0.4972418161105763, J01 = 0.3468
it 512 : Jsur = 0.49713558817283576, J01 = 0.3468
it 2 : Jsur = 0.5015576719875539, J01 = 0.6532
it 4 : Jsur = 0.5016184333294704, J01 = 0.6532
it 8 : Jsur = 0.5016751424636248, J01 = 0.6532
it 16 : Jsur = 0.5016933997030455, J01 = 0.6532
it 32 : Jsur = 0.5015917485723894, J01 = 0.6532
it 64 : Jsur = 0.5014166947668343, J01 = 0.6532
it 128 : Jsur = 0.5013391008553268, J01 = 0.6532
it 256 : Jsur = 0.5012646302931513, J01 = 0.6532
it 512 : Jsur = 0.5011684280254389, J01 = 0.6532
it 2 : Jsur = 0.4976575621309943, J01 = 0.3468
it 4 : Jsur = 0.49760023455497787, J01 = 0.3468
it 8 : Jsur = 0.49754682126468514, J01 = 0.3468
it 16 : Jsur = 0.4974067361013775, J01 = 0.3468
it 32 : Jsur = 0.4973112446050533, J01 = 0.3468
it 64 : Jsur = 0.49730709432921166, J01 = 0.3468
it 128 : Jsur = 0.4972078009909451, J01 = 0.3468
it 256 : Jsur = 0.49708763274990886, J01 = 0.3468
it 512 : Jsur = 0.4969874811480093, J01 = 0.3468
it 2 : Jsur = 0.4976548706694314, J01 = 0.3468
it 4 : Jsur = 0.49759753028435455, J01 = 0.3468
it 8 : Jsur = 0.49757077602470456, J01 = 0.3468
it 16 : Jsur = 0.4972633080628353, J01 = 0.3468
it 32 : Jsur = 0.497173330034764, J01 = 0.3468
it 64 : Jsur = 0.49713606703599705, J01 = 0.3468
it 128 : Jsur = 0.4970093747802574, J01 = 0.3468
it 256 : Jsur = 0.49691143624508005, J01 = 0.3468
it 512 : Jsur = 0.4968587025425404, J01 = 0.3468
it 2 : Jsur = 0.5021304755872998, J01 = 0.6532
it 4 : Jsur = 0.5021909633390914, J01 = 0.6532
it 8 : Jsur = 0.5020720866044474, J01 = 0.6532
it 16 : Jsur = 0.5020386267217344, J01 = 0.6532
it 32 : Jsur = 0.5019714040557104, J01 = 0.6532
it 64 : Jsur = 0.5020154136368438, J01 = 0.6532
it 128 : Jsur = 0.5019399686897694, J01 = 0.6532
it 256 : Jsur = 0.5018265538698653, J01 = 0.6532
it 512 : Jsur = 0.5017314332387743, J01 = 0.6532
it 2 : Jsur = 0.5017673901508081, J01 = 0.6532
it 4 : Jsur = 0.5014213054945349, J01 = 0.6532
it 8 : Jsur = 0.5012633884975217, J01 = 0.6532
it 16 : Jsur = 0.5013247838222442, J01 = 0.6532
it 32 : Jsur = 0.5011130838090527, J01 = 0.6532
it 64 : Jsur = 0.5009807571097185, J01 = 0.6532
it 128 : Jsur = 0.500959147056947, J01 = 0.6532
```

```
it 256 : Jsur = 0.5008587973653322, J01 = 0.6532
it 512 : Jsur = 0.5007819058833385, J01 = 0.6532
it 2 : Jsur = 0.4976002164950543, J01 = 0.3468
it 4 : Jsur = 0.49793310278813285, J01 = 0.3468
it 8 : Jsur = 0.49790589720553446, J01 = 0.3468
it 16 : Jsur = 0.49767493459937423, J01 = 0.3468
it 32 : Jsur = 0.4975934656913253, J01 = 0.3468
it 64 : Jsur = 0.49761849849043327, J01 = 0.3468
it 128 : Jsur = 0.49748364292209246, J01 = 0.3468
it 256 : Jsur = 0.49739457706384244, J01 = 0.3468
it 512 : Jsur = 0.49728909772985364, J01 = 0.3468
it 2 : Jsur = 0.49779461084477505, J01 = 0.3468
it 4 : Jsur = 0.4974672310981561, J01 = 0.3468
it 8 : Jsur = 0.4972741715499569, J01 = 0.3468
it 16 : Jsur = 0.4970530338209792, J01 = 0.3468
it 32 : Jsur = 0.49692996765315883, J01 = 0.3468
it 64 : Jsur = 0.4968741758472789, J01 = 0.3468
it 128 : Jsur = 0.49675584617917395, J01 = 0.3468
it 256 : Jsur = 0.49667266733874105, J01 = 0.3468
it 512 : Jsur = 0.49661159032765945, J01 = 0.3468
it 2 : Jsur = 0.4979636172706865, J01 = 0.3468
it 4 : Jsur = 0.4980259716484241, J01 = 0.3468
it 8 : Jsur = 0.49791659483065404, J01 = 0.3468
it 16 : Jsur = 0.49816430835901376, J01 = 0.3468
it 32 : Jsur = 0.4980146166036968, J01 = 0.3468
it 64 : Jsur = 0.4978814170912139, J01 = 0.3468
it 128 : Jsur = 0.4977565328342942, J01 = 0.3468
it 256 : Jsur = 0.4976124388504121, J01 = 0.3468
it 512 : Jsur = 0.4975306383682767, J01 = 0.3468
it 2 : Jsur = 0.4981221149468171, J01 = 0.3468
it 4 : Jsur = 0.49779318078417817, J01 = 0.3468
it 8 : Jsur = 0.49747821241429047, J01 = 0.3468
it 16 : Jsur = 0.4973749958491183, J01 = 0.3468
it 32 : Jsur = 0.49728532967524736, J01 = 0.3468
it 64 : Jsur = 0.497281623628325, J01 = 0.3468
it 128 : Jsur = 0.49722930035215707, J01 = 0.3468
it 256 : Jsur = 0.49713577201682074, J01 = 0.3468
it 512 : Jsur = 0.49705320619947524, J01 = 0.3468
it 2 : Jsur = 0.4975691381050541, J01 = 0.3468
it 4 : Jsur = 0.4979019866231698, J01 = 0.3468
it 8 : Jsur = 0.4977268519281729, J01 = 0.3468
it 16 : Jsur = 0.4974901644226324, J01 = 0.3468
it 32 : Jsur = 0.49745796570740275, J01 = 0.3468
it 64 : Jsur = 0.4973169019887963, J01 = 0.3468
it 128 : Jsur = 0.4972626318256131, J01 = 0.3468
it 256 : Jsur = 0.497131557326084, J01 = 0.3468
it 512 : Jsur = 0.4970393465518135, J01 = 0.3468
it 2 : Jsur = 0.4975143063309293, J01 = 0.3468
```

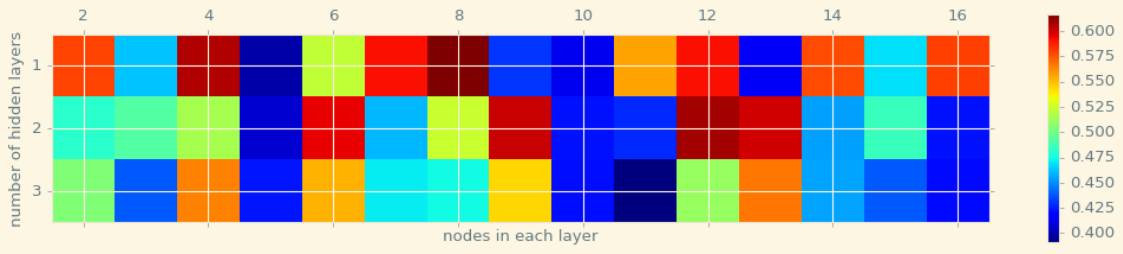
```
it 4 : Jsur = 0.4978470162562357, J01 = 0.3468
it 8 : Jsur = 0.4978199241910197, J01 = 0.3468
it 16 : Jsur = 0.49774919359931846, J01 = 0.3468
it 32 : Jsur = 0.49751746327628255, J01 = 0.3468
it 64 : Jsur = 0.49746105571073784, J01 = 0.3468
it 128 : Jsur = 0.4973745243931719, J01 = 0.3468
it 256 : Jsur = 0.49727592038989826, J01 = 0.3468
it 512 : Jsur = 0.49718512928517555, J01 = 0.3468
it 2 : Jsur = 0.4975019478965379, J01 = 0.3468
it 4 : Jsur = 0.4971759974503806, J01 = 0.3468
it 8 : Jsur = 0.49706856388888737, J01 = 0.3468
it 16 : Jsur = 0.49723423370103026, J01 = 0.3468
it 32 : Jsur = 0.497116572577228, J01 = 0.3468
it 64 : Jsur = 0.497004141080075, J01 = 0.3468
it 128 : Jsur = 0.4969883019586303, J01 = 0.3468
it 256 : Jsur = 0.4968922616697544, J01 = 0.3468
it 512 : Jsur = 0.4968184970070892, J01 = 0.3468
it 2 : Jsur = 0.501451308837694, J01 = 0.6532
it 4 : Jsur = 0.5013873099117746, J01 = 0.6532
it 8 : Jsur = 0.5013275588409158, J01 = 0.6532
it 16 : Jsur = 0.5012304871066424, J01 = 0.6532
it 32 : Jsur = 0.5011842379546486, J01 = 0.6532
it 64 : Jsur = 0.5010762659911759, J01 = 0.6532
it 128 : Jsur = 0.5010416322021569, J01 = 0.6532
it 256 : Jsur = 0.5009497405501001, J01 = 0.6532
it 512 : Jsur = 0.5008360398639692, J01 = 0.6532
it 2 : Jsur = 0.501854484774754, J01 = 0.6532
it 4 : Jsur = 0.50150800268888, J01 = 0.6532
it 8 : Jsur = 0.5013498552802025, J01 = 0.6532
it 16 : Jsur = 0.5011240051807543, J01 = 0.6532
it 32 : Jsur = 0.500951231987189, J01 = 0.6532
it 64 : Jsur = 0.5008926103369881, J01 = 0.6532
it 128 : Jsur = 0.5007456059560329, J01 = 0.6532
it 256 : Jsur = 0.5006481187089117, J01 = 0.6532
it 512 : Jsur = 0.5005647007648882, J01 = 0.6532
it 2 : Jsur = 0.4978688266392436, J01 = 0.3468
it 4 : Jsur = 0.4979312433175424, J01 = 0.3468
it 8 : Jsur = 0.49792342598053174, J01 = 0.3468
it 16 : Jsur = 0.49777079684121234, J01 = 0.3468
it 32 : Jsur = 0.4975137043237595, J01 = 0.3468
it 64 : Jsur = 0.4973512744549045, J01 = 0.3468
it 128 : Jsur = 0.4972826814069025, J01 = 0.3468
it 256 : Jsur = 0.49717299723589115, J01 = 0.3468
it 512 : Jsur = 0.49706307793069315, J01 = 0.3468
it 2 : Jsur = 0.5014186870534324, J01 = 0.6532
it 4 : Jsur = 0.501074195903894, J01 = 0.6532
it 8 : Jsur = 0.5007442938437573, J01 = 0.6532
it 16 : Jsur = 0.5007633377136721, J01 = 0.6532
```

```
it 32 : Jsur = 0.5005222023003408, J01 = 0.6532
it 64 : Jsur = 0.5003812741658418, J01 = 0.6532
it 128 : Jsur = 0.500328892303155, J01 = 0.6532
it 256 : Jsur = 0.5002510198840024, J01 = 0.6532
it 512 : Jsur = 0.5001524901180413, J01 = 0.6532
it 2 : Jsur = 0.5023565731034698, J01 = 0.6532
it 4 : Jsur = 0.5020075933295953, J01 = 0.6532
it 8 : Jsur = 0.5016733841883689, J01 = 0.6532
it 16 : Jsur = 0.5015971772803144, J01 = 0.6532
it 32 : Jsur = 0.5015151681105628, J01 = 0.6532
it 64 : Jsur = 0.5013163600165846, J01 = 0.6532
it 128 : Jsur = 0.501178784210008, J01 = 0.6532
it 256 : Jsur = 0.5011176591349076, J01 = 0.6532
it 512 : Jsur = 0.5010475561264942, J01 = 0.6532
it 2 : Jsur = 0.49810616096361626, J01 = 0.3468
it 4 : Jsur = 0.49777721175525924, J01 = 0.3468
it 8 : Jsur = 0.49746222955550784, J01 = 0.3468
it 16 : Jsur = 0.4974839661016312, J01 = 0.3468
it 32 : Jsur = 0.4973934837444844, J01 = 0.3468
it 64 : Jsur = 0.49722537420322616, J01 = 0.3468
it 128 : Jsur = 0.49716128397861065, J01 = 0.3468
it 256 : Jsur = 0.49710223270210463, J01 = 0.3468
it 512 : Jsur = 0.4970052248453124, J01 = 0.3468
it 2 : Jsur = 0.49773243695007685, J01 = 0.3468
it 4 : Jsur = 0.4977948863621409, J01 = 0.3468
it 8 : Jsur = 0.49747989910427143, J01 = 0.3468
it 16 : Jsur = 0.49741381659221795, J01 = 0.3468
it 32 : Jsur = 0.4973098375187479, J01 = 0.3468
it 64 : Jsur = 0.49720858944799273, J01 = 0.3468
it 128 : Jsur = 0.49716230876191986, J01 = 0.3468
it 256 : Jsur = 0.4970232432958026, J01 = 0.3468
it 512 : Jsur = 0.4969495123430256, J01 = 0.3468
```

[66]:
```
f, ax = plt.subplots(1, 1, figsize=(16, 3))
cax1 = ax.matshow(tr_auc, interpolation='nearest')
f.colorbar(cax1)
ax.set_xticklabels(['']+list(nnodes))
ax.set_yticklabels(['']+list(nlayers))
ax.set_xlabel("nodes in each layer")
ax.set_ylabel("number of hidden layers")
# ax.set_title("Neural Networks Training AUC (rescaled)")
plt.show()
```

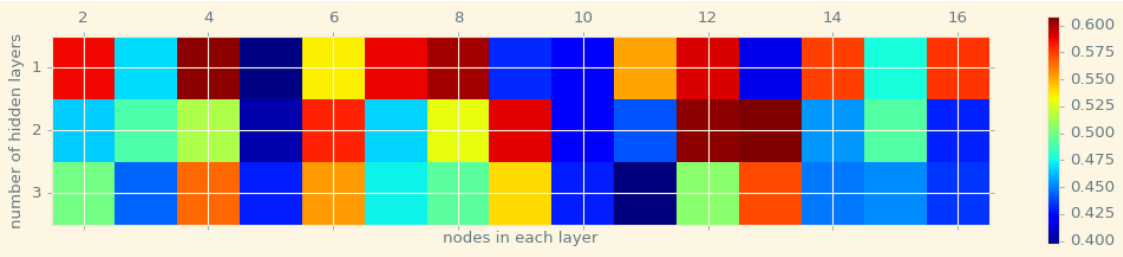
```
<ipython-input-66-c6de7297c027>:4: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(['']+list(nnodes))
<ipython-input-66-c6de7297c027>:5: UserWarning: FixedFormatter should only be
used together with FixedLocator
```

```
  ax.set_yticklabels(['']+list(nlayers))
```



[69]:
```
f, ax = plt.subplots(1, 1, figsize=(16, 3))
cax2 = ax.matshow(va_auc, interpolation='nearest')
f.colorbar(cax2)
ax.set_xticklabels(['']+list(nnodes))
ax.set_yticklabels(['']+list(nlayers))
ax.set_xlabel("nodes in each layer")
ax.set_ylabel("number of hidden layers")
# ax.set_title("Neural Networks validation AUC (rescaled)")
plt.show()
```

```
<ipython-input-69-ad1dfe3c2e09>:4: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_xticklabels(['']+list(nnodes))
<ipython-input-69-ad1dfe3c2e09>:5: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_yticklabels(['']+list(nlayers))
```



[71]:
```
print('--------------------------')
print('    Training AUC          ')
print('--------------------------')
print(tr_auc)

print('--------------------------')
print('    Validation AUC        ')
print('--------------------------')
```

```
print(va_auc)
```

```
--------------------------
    Training AUC
--------------------------
[[0.58019052 0.46274423 0.6062987  0.39976902 0.52231345 0.59193944
  0.61637208 0.43102761 0.4137587  0.55677815 0.59240525 0.41544555
  0.57770273 0.46881725 0.58103862]
 [0.48157143 0.49264829 0.51600699 0.40727585 0.59574786 0.46013071
  0.52481219 0.6021984  0.42342913 0.4288331  0.60929275 0.60052251
  0.45444016 0.48565725 0.42290046]
 [0.50546065 0.43900492 0.56502492 0.42442521 0.55379214 0.47150529
  0.47506155 0.54454788 0.42191198 0.39102915 0.51162028 0.5678227
  0.45612965 0.43891037 0.42161957]]
--------------------------
    Validation AUC
--------------------------
[[0.58734268 0.4691864  0.60550238 0.39783261 0.53504663 0.58763045
  0.60149168 0.4321525  0.42236659 0.55230695 0.5910689  0.41705257
  0.57451469 0.47775508 0.57681961]
 [0.46580592 0.49098413 0.51563419 0.40579273 0.58108816 0.46765785
  0.52977385 0.58966878 0.42081404 0.44126993 0.60516241 0.60809592
  0.45503357 0.49234631 0.43033983]
 [0.50039113 0.44472533 0.56527289 0.42984752 0.55441441 0.47425569
  0.49421147 0.53990536 0.42980602 0.39698683 0.50592117 0.57234565
  0.44867229 0.45253657 0.43439815]]
```

These results are interesting.

We can see from the validation AUC heat map in the second graph, that the dark red areas of the graph show up in a few regions: 1 layer/4 nodes, 1 layer/8 nodes, 2 layers/12 nodes, and 2 layers/ 13 nodes.

If we cross-reference this to the training AUC rates in the first graph, we see that in the 1 layer ones, those AUC rates are also really high. This might be a model that is tending towards over fitting.

Therefore, I might choose the 2 layers/12 nodes configuration. It actually performs slightly better on the validation than it does on the testing, which I think is a little strange.

Based on the results and analysis above, I recommend that the number of hidden layers should be equal to 2, and the nodes in each layer to be equal to 12.

5.1.2 2. Compare the performance of this activation function with logistic and htangent, in terms of the training and validation performance.

[93]:
```
# define Gaussian as the activation function
def sig(z): return np.atleast_2d(np.exp(-z**2/2))
def dsig(z): return np.atleast_2d(-z*np.exp(-z**2/2))
```

[98]:
```
methods = ['logistic' , 'htangent', 'Gaussian']
for i,m in enumerate(methods):
    nn = ml.nnet.nnetClassify()
    nn.init_weights([Xt.shape[1], 5, 2], 'random', XtS, Yt)
    if m == 'Gaussian':
        nn.setActivation('custom', sig, dsig)
    else:
        nn.setActivation(m)
    nn.train(XtS, Yt, stopTol=1e-8, stepConstant=.25, stopIter=100)
    print("{0:>15}: {1:.4f}".format(m+' Train AUC',nn.auc(XtS, Yt)))
    print("{0:>15}: {1:.4f}".format(m+' Validation AUC', nn.auc(XvS, Yva)))
```

```
it 2 : Jsur = 0.4984596288911941, J01 = 0.3468
it 4 : Jsur = 0.4981291737002375, J01 = 0.3468
it 8 : Jsur = 0.49818742689943685, J01 = 0.3468
it 16 : Jsur = 0.498041621421612, J01 = 0.3468
it 32 : Jsur = 0.4978241794015427, J01 = 0.3468
it 64 : Jsur = 0.4976236022568959, J01 = 0.3468
it 128 : Jsur = 0.49755172022994154, J01 = 0.3468
logistic Train AUC: 0.4688
logistic Validation AUC: 0.4761
it 2 : Jsur = 0.5013768899957355, J01 = 0.6532
it 4 : Jsur = 0.5013130318514549, J01 = 0.6532
it 8 : Jsur = 0.5014959028770509, J01 = 0.6532
it 16 : Jsur = 0.5012695668819895, J01 = 0.6532
it 32 : Jsur = 0.5011722896979071, J01 = 0.6532
it 64 : Jsur = 0.5010501113127565, J01 = 0.6532
it 128 : Jsur = 0.5009328189467019, J01 = 0.6532
htangent Train AUC: 0.5632
htangent Validation AUC: 0.5602
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-98-65fc4d2aff23> in <module>
      4     nn.init_weights([Xt.shape[1], 5, 2], 'random', XtS, Yt)
      5     if m == 'Gaussian':
----> 6         nn.setActivation('custom', sig, dsig)
      7     else:
      8         nn.setActivation(m)

~/Developer/CS273A Homework/Homework4/mltools/nnet.py in setActivation(self,␣
 ↪method, sig, sig0)
    233             self.Sig = sig
    234             self.dSig = d_sig
--> 235             if sig_0 is not None:
    236                 self.Sig0 = sig_0
    237             if d_sig_0 is not None:
```

```
NameError: name 'd_sig' is not defined
```

Please note: Although I tried for a very long time to debug this nnetClassify function to pass in our custom function, I cannot get it to work. Via Professor Fox's Piazza post, he said we don't have to edit the code. I also tried posting on Piazza to get advice on how to debug. So, the below analysis is for the only two functions that did work.

As we can see from the output, the `htangent` activation function has the highest training as well as validation for AUC.

I'm not too surprised by this, given that the range of `htangent` lends itself well to binary classification, as is the case with this dogs vs. cats dataset task.

5.1.3 3. Pick the classifier that you think will perform best, mention all of its hyperparameter values, and explain the reason for your choice. Train it on as much data as you can, preferably all of X, submit the predictions on Xtest to Kaggle, and include your Kaggle username and leaderboard AUC in the report.

I think that the Neural Network will perform best for this task, because Neural Networks are good for image classification, and binary image classification is a relatively straightforward problem in the scope of Neural networks. I think that a linear regression approach is too simplistic for this application. Image classification seems like a strange application for decision trees. So, lets try Neural Networks.

Hyperparameters: I will use the same hyperparameters from the above two problems: - Hidden layers = 2 - Nodes in each layer = 12

Although the instructions recommend training on a higher amount of data than 5,000, my machine has limited computing power. I'm using 5,000 for this assignment, so my code is just the same as the above problem #4.

Kaggle Submission information: As noted by a fellow classmate in the Piazza post(https://piazza.com/class/kjdij09z7wx52d?cid=141), we cannot push our scores to the leaderboard as specified in the instructions. Additionally, the Ysubmit file was way too large to include here. So, I have listed my score instead: 0.50142

6 Statement of Collaboration

I had a discussion with my teammates on February 22 helping them debug an issue with graph output. We did not share code but did show each other our graph output to help one of our groupmates diagnose a bug in his code.

I also discussed with my teammates about how long it took for us to each generate our graphs. I confirmed with my teammates that this was a big issue, and one of my other teammates actually was not able to run 2.3 as well.

I abided by the academic integrity standards for UC Irvine.

6.0.1 Acknowledgement of Sources

These sources were used to understand the theoretical context of some of the questions. Including them for my own future reference, as well as for full transparency within the context of academic integrity.

https://stats.stackexchange.com/questions/287425/why-do-you-need-to-scale-data-in-knn

Hal Daumé III, A Course in Machine Learning - Chapter 3, KNN and Geometry - Chapter 5, Practical Issues (Rescaling and Normalization)