## CS598PS - Problem Set 3 solutions

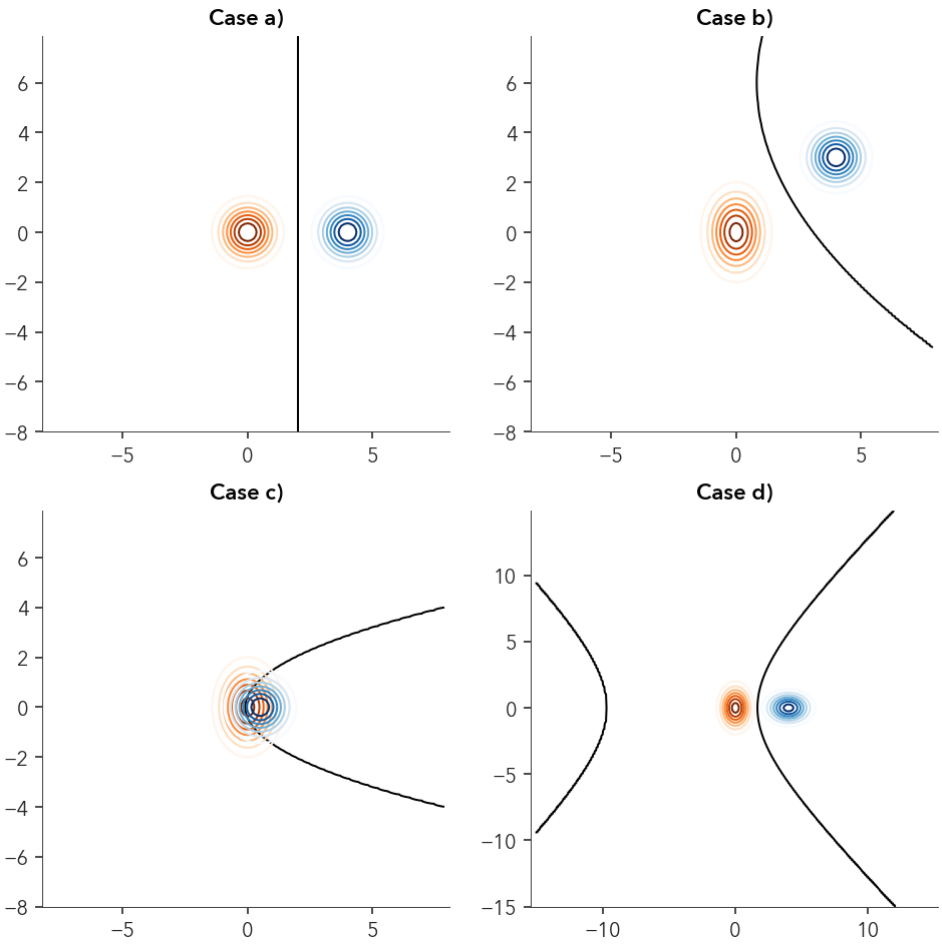
[7]

```
%load_ext autoreload
%autoreload 2
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import numpy
from numpy import *
from matplotlib.pylab import *

# Some beutification
rcParams['figure.figsize'] = (8,8)
rcParams['lines.linewidth'] = 1
rcParams['image.cmap'] = 'Greys'
rcParams['axes.spines.right'] = False
rcParams['axes.spines.top'] = False
rcParams['font.family'] = 'Avenir Next LT Pro'
rcParams['font.weight'] = 400
rcParams['xtick.color'] = '#222222'
rcParams['ytick.color'] = '#222222'
rcParams['grid.color'] = '#dddddd'
rcParams['grid.linestyle'] = '-'
rcParams['grid.linewidth'] = 0.5
rcParams['axes.titlesize'] = 11
rcParams['axes.titleweight'] = 600
rcParams['axes.labelsize'] = 10
rcParams['axes.labelweight'] = 400
rcParams['axes.linewidth'] = 0.5
rcParams['axes.edgecolor'] = [.25,.25,.25]
datapath = '/Users/paris/Dropbox/Classes/CS598PS/Problem Sets/data/'

# Perform PCA by diagonalizing the covariance, return transform parameters
def pca( x, k):
    # Remove data mean
    m = mean( x, axis=1, keepdims=True)
    xm = x - m

    # Get covariance estimate
    c = xm.dot( xm.T) / (xm.shape[1]-1)

    # Get top k PCA covariance eigenvectors/values
    v,u = scipy.sparse.linalg.eigsh( c, k=k)

    # Get overall transform
    w = diag( 1./sqrt(v)).dot( u.T)

    return w,m,v


# A simple Gaussian classifier
def gclass( x, g=None, diagonal=False):
    # If no model is provided, then train
    if g is None:
        # Get class mean
        m = mean( x, axis=1, keepdims=True)

        # Get class covariance
        c = (x-m).dot( (x-m).T) / (x.shape[1]-1)
        if diagonal:
            c = diag( diag( c))

        # Pack into the model, use inverse covariance so we don't invert every time
        return {'mean':m, 'icov':linalg.inv( c)}

    # If a model is provided just compute log likelihoods
    else:
        # Remove mean
        xm = x-g['mean']

        # Get exponent
        ds = sum( xm * g['icov'].dot( xm), axis=0)

        # Compute overall likelihood
        return -.5*(-log( det( g['icov'])) + ds + x.shape[0]*log( 2*pi))
```

# Problem 1

For this problem you had to solve for equality in the likelihood of the two Gaussians and then plot the resulting discriminant functions. Here are the four plots you should get:

[8]
```
# Define a Gaussian
def gp( x, m, c):
    return exp( -(x-m).T.dot( linalg.inv( c)).dot( x-m)) / sqrt( linalg.det( 2*pi*array(c)))

# Show me the two Gaussians and the boundary, sneakily offload the discriminant computation to countc
def plotem( m1, c1, m2, c2, l=8):
    x = arange( -l, l, .125)
    g1 = array( [[gp(array([x[i],x[j]]), m1, c1) for i in range( len( x))] for j in range( len( x))]
    g2 = array( [[gp(array([x[i],x[j]]), m2, c2) for i in range( len( x))] for j in range( len( x))]
    contour( x, x, g1-g2, [0], colors='k')
    contour( x, x, g1, cmap='Oranges')
    contour( x, x, g2, cmap='Blues')
    axis( 'equal')

# Draw the boundaries
subplot( 2, 2, 1), plotem( [0,0], [[1,0],[0,1]], [4,0], [[1,0],[0,1]]), title( 'Case a)')
subplot( 2, 2, 2), plotem( [0,0], [[1,0],[0,2]], [4,3], [[1,0],[0,1]]), title( 'Case b)')
subplot( 2, 2, 3), plotem( [0,0], [[1,0],[0,2]], [.5,0], [[1,0],[0,1]]), title( 'Case c)')
subplot( 2, 2, 4), plotem( [0,0], [[1,0],[0,2]], [4,0], [[2,0],[0,1]], 15), title( 'Case d)');
```




Note that in d) we have two solutions to the equal likelihood case and that implies two discriminant boundaries. Since the right Gaussian is wider along the horizontal axis, it will eventually become more probable

than the left Gaussian as we move towards the left. The second discriminant curve takes care of that situation.

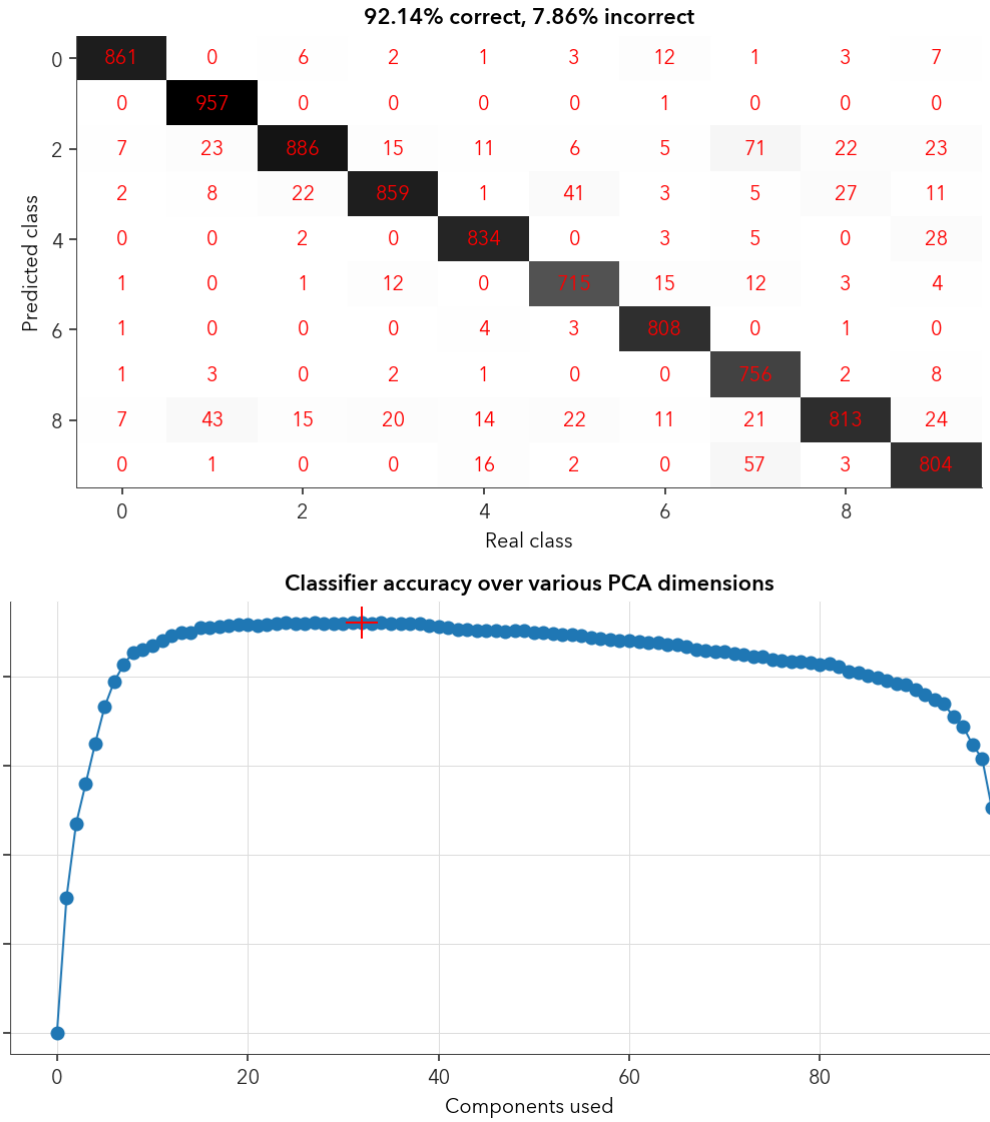
## Problem 2

In this problem we had to make a digit classifier using PCA and a Gaussian model. Doing this across different number of dimensions for PCA we get the following results. On the top the confusion matrix shown is for 30 components which, for the randomly chosen training set, achieves the best recognition at 90.9%. At the bottom we see the performance as estimated for various dimensionality reduction values.

[10]
```
# Load the data
import scipy.io
d = scipy.io.loadmat( datapath+'digits-labels.mat')
l = d['l']
d = d['d']

# Split data in training and testing sets
for i in range( 10):
    # Get samples of the i'th digit and permute their order
    j = find( l == i)
    j = numpy.random.permutation( j)

    # Training set is 100 random samples of each digit
    if i == 0:
        rx = d[:,j[:100]]
        ry = l[0,j[:100]]
        sx = d[:,j[100:]]
        sy = l[0,j[100:]]
    else:
        rx = append( rx, d[:,j[:100]], axis=1)
        ry = append( ry, l[0,j[:100]], axis=0)
        sx = append( sx, d[:,j[100:]], axis=1)
        sy = append( sy, l[0,j[100:]], axis=0)


# Scan through multiple number of components
E = zeros( 100)
for c in range( 1, 100):

    # Perform dimensionality reduction with PCA
    pw,pm,_ = pca( rx, c)

    # Get low-dimensional versions of both training and testing data
    rxl = pw.dot(rx-pm)
    sxl = pw.dot(sx-pm)

    # Learn a Gaussian classifier for each class
    G = [gclass( rxl[:,ry==i]) for i in range( 10)]

    # Get data likelihood over all models
    ll = [gclass( sxl, g) for g in G]

    # Pick the class that has the highest likelihood for each data point
    j = argmax( ll, axis=0)

    # Compute and show the confusion matrix
    C = histogram2d( j, sy, bins=10)[0]

    # Remember results
    E[c] = 100*trace( C)/len(sy)

    # Show me the best confusion so far
    if E[c] == amax( E):
        subplot( 2, 1, 1), cla(), imshow( C, aspect=1/2)
        for i in range( 10):
            for j in range( 10):
                text( i, j, int( C[j,i]), color='r',
                      horizontalalignment='center', verticalalignment='center')
        xlabel( 'Real class'), ylabel( 'Predicted class')

        # Compute accuracy
        title( '%.2f%% correct, %.2f%% incorrect' %
```

```
                (100*trace( C)/len(sy), 100*(sum( C)-trace(C))/len(sy)))


# Show me what we found out
i = argmax( E)
subplot( 2, 1, 2), plot( E, '-o')
plot( i, E[i], 'r+', markersize=16), grid( 'on')
xlabel( 'Components used'), ylabel( '% correct classification')
title( 'Classifier accuracy over various PCA dimensions')
print( 'Best result is at %d dimensions' % i)
tight_layout()
```

```
Best result is at 32 dimensions
```







Note how because of the small training set (100 samples per class), we observe overfitting when we use too many components. This is because the Gaussian covariance estimate ends up being unreliable due to too few samples. If we used a larger training set the curve would instead stabilize as dimensions increased.

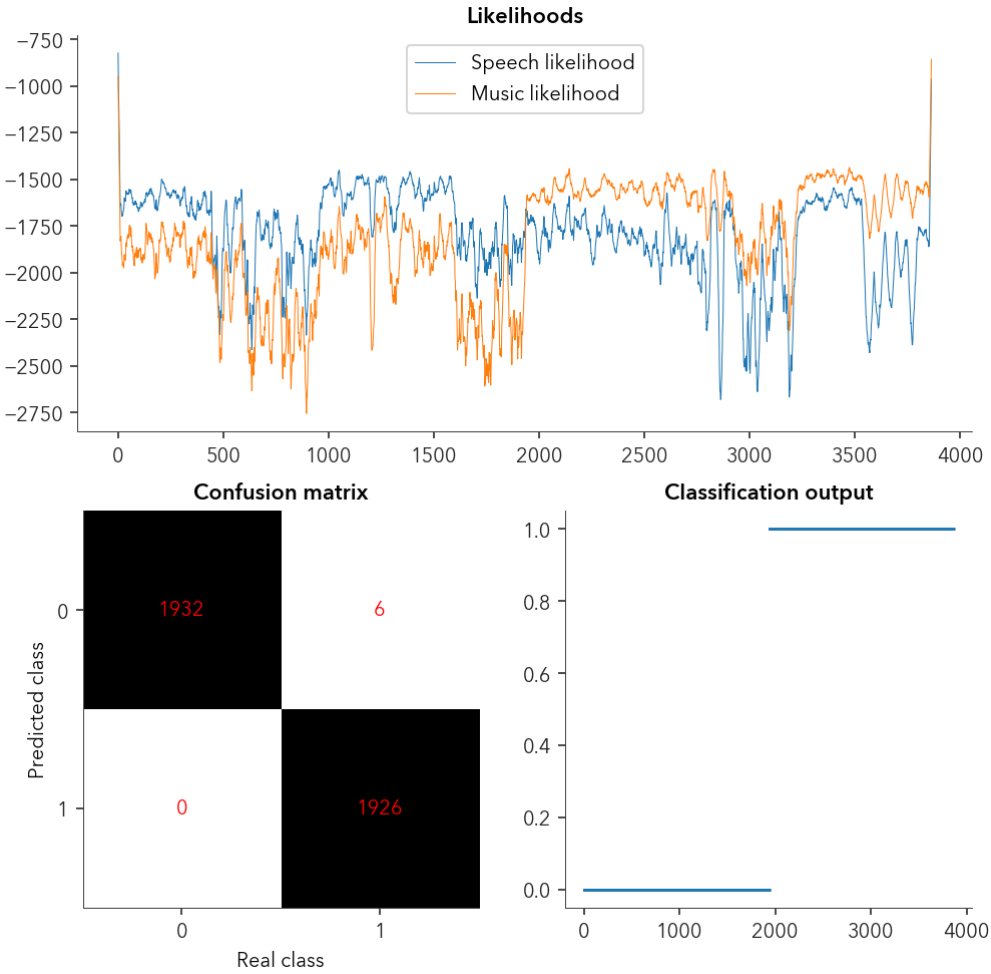
## Problem 3

In this problem we have to design a speech/no-speech classifier. We can do so by using PCA for dimensionality reduction, and employing a simple Gaussian model for each class. In order to calculate the likelihood for each 1

sec segment we sum the log likelihoods of all the included frames to obtain a total likelihood value. Following is the code for this model. The likelihoods, confusion matrix and classification output are shown further down.

[11]
```
import glob
import scipy.io.wavfile
from random import shuffle

# Define the feature function
def feats( x):
    # Get frames
    sz = 1024
    l = len( x)//sz
    x = reshape( x[:l*sz], (l,-1)).T

    # Window and FFT them
    w = hanning( sz)[:,None]
    f = numpy.fft.rfft( w*x, axis=0)

    # Return log magnitude
    return log( abs( f) + 1e-5)


# Load the data and get feature representation
p = datapath+'SpeechMusic/'
S = [feats( scipy.io.wavfile.read( i)[1]) for i in glob.glob( p+'speech/*.wav')]
M = [feats( scipy.io.wavfile.read( i)[1]) for i in glob.glob( p+'music/*.wav')]

# Split inputs into training and testing data
trs = hstack( S[6:])
trm = hstack( M[6:])
tss = hstack( S[:6])
tsm = hstack( M[:6])

# Get PCA from training data
w,m,_ = pca( hstack( (trs, trm)), 60)

# Learn models
g1 = gclass( w.dot( trs-m))
g2 = gclass( w.dot( trm-m))

# Evaluate on test data
tsx = w.dot( hstack( (tss, tsm))-m)
l1 = gclass( tsx, g1)
l2 = gclass( tsx, g2)

# Get classification output
r = (l1 < l2).astype( int)

# Figure out ground-truth labels
tsl = hstack( (zeros( tss.shape[1]), ones( tsm.shape[1])))

# Get frame-by-frame error
print( 'Frame-level error: %.2f%%' % (100*sum( r[tsl==1]≠tsl[tsl==1]) / len( tsl[tsl==1])) )

# Get second-long likelihood averages (21 frames in a second)
l1 = convolve( l1, ones( 21), 'same')
l2 = convolve( l2, ones( 21), 'same')

# Get classification result again
r = (l1 < l2).astype( int)

# Show me what's going on
subplot( 2, 1, 1)
plot( l1, lw=.5, label='Speech likelihood')
plot( l2, lw=.5, label='Music likelihood')
legend(), title( 'Likelihoods')
subplot( 2, 2, 4), plot( r, '.', ms=.5), title( 'Classification output')
subplot( 2, 2, 3)
C = histogram2d( r, tsl, bins=2)[0]
imshow( C), xticks([0,1]), yticks([0,1])
for i in range( 2):
    for j in range( 2):
        text( i, j, int( C[j,i]), color='r',
              horizontalalignment='center', verticalalignment='center')
xlabel( 'Real class'), ylabel( 'Predicted class')
title( 'Confusion matrix')

# Show classification percentages
print( 'Seconds-level errors: %.2f%%' % (100*sum( r[tsl==1]≠tsl[tsl==1]) / len( tsl[tsl==1])) )
```

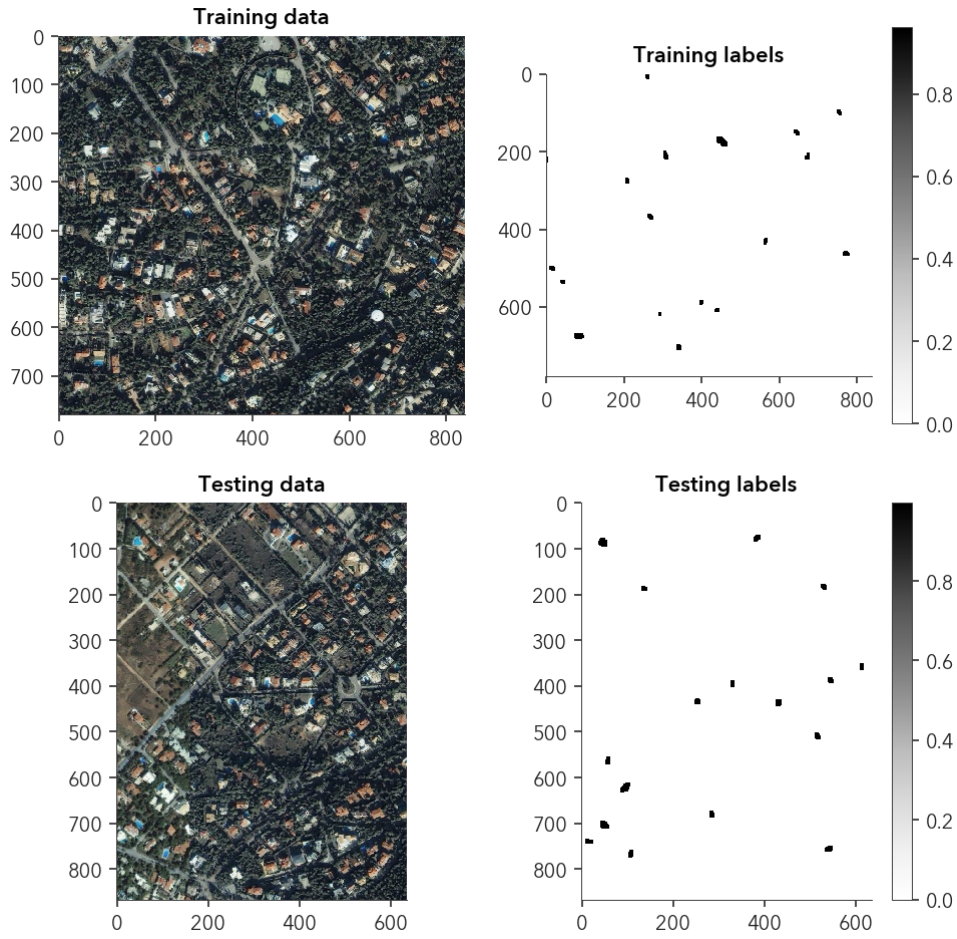
```
Frame-level error: 4.19%
Seconds-level errors: 0.31%
```




# Problem 4

We need to find if there's a pool at any pixel in an image. To start with we need labelled data. To get these I made a set of image masks, which are 0 for pixels that aren't part of a pool and 1 otherwise. These correspond to the ground truth for both the training and testing. Having these we can make a classifier. In the following code I load the data and the class labels and I plot them.

[12]
```
import scipy.misc

# Load the files and class labels
tr = scipy.misc.imread( datapath+'ekalismall.png', mode='RGB')/256
mr = scipy.misc.imread( datapath+'ekalismall_mask4.png', mode='L')/265
ts = scipy.misc.imread( datapath+'ekalismall2.png', mode='RGB')/256
ms = scipy.misc.imread( datapath+'ekalismall2_mask4.png', mode='L')/256

# Show me
subplot( 2, 2, 1), imshow( tr), title( 'Training data')
subplot( 2, 2, 2), imshow( mr), title( 'Training labels'), colorbar()
subplot( 2, 2, 3), imshow( ts), title( 'Testing data')
subplot( 2, 2, 4), imshow( ms), title( 'Testing labels'), colorbar();
```

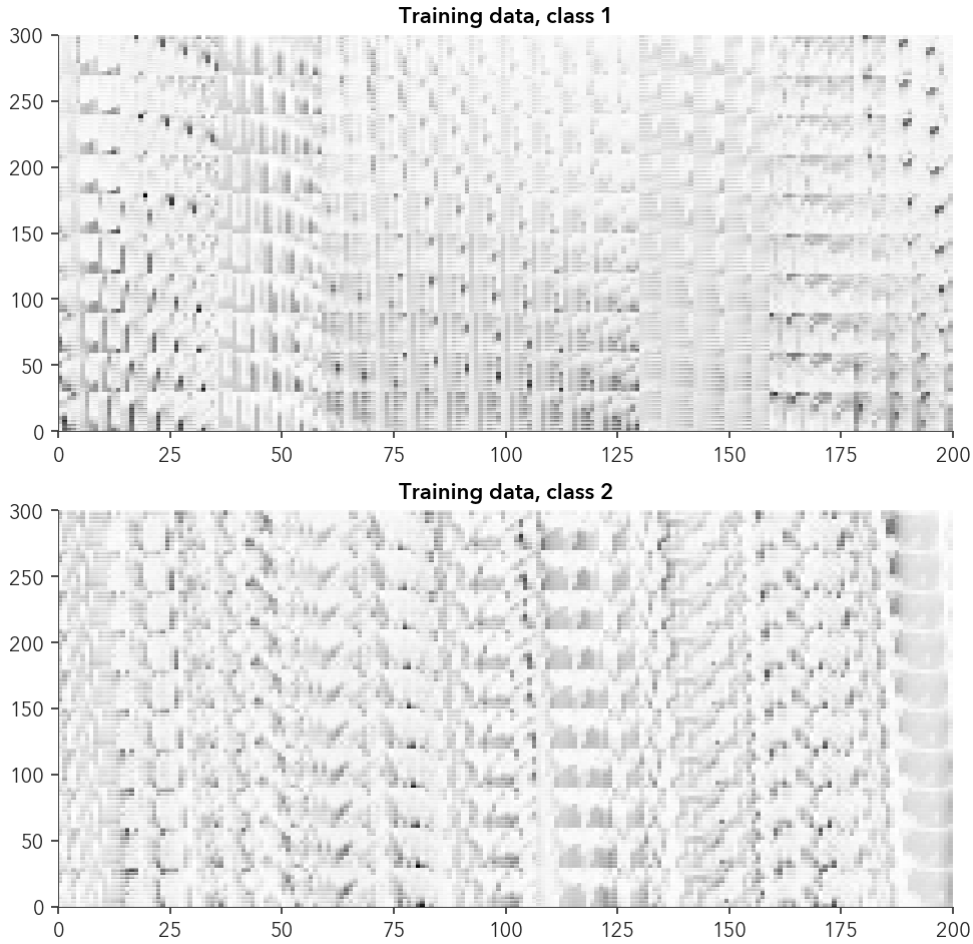



Next we chop up the image files into small patches (10 by 10 pixels), and vectorize them. For each patch I look at the corresponding values in the ground truth matrices which indicate if there is a pool in that patch of not. This way each bit of the image is represented as a vector. I also remove their mean and normalize them to sum to 1 since I want to be brightness invariant.

[19

```
# Get patch features
def patchfeat( p):
    # Bring up the greens and blues a bit
    p = p ** [1,1.2,1.5]

    # Normalize to get some brightness invariance
    p = p / sum( p)

    # Return as a vector
    return reshape( p, (-1,1))


# Get vectorized patches, corresponding labels and locations
def patchit( x, m, b=5, hp=2):
    # Make patch indices
    i,j = meshgrid( range( b, x.shape[0]-b, hp), range( b, x.shape[1]-b, hp))
    pos = hstack((reshape(i,(-1,1)),reshape(j,((-1,1)))))

    inp = hstack( [patchfeat( x[k[0]-b:k[0]+b,k[1]-b:k[1]+b]) for k in pos])

    # Get labels
    lab = array([mean( m[k[0]-b:k[0]+b,k[1]-b:k[1]+b]) > .5 for k in pos]).astype( int)


    return inp,lab,pos


# Get training data
trb,yr,tri = patchit( tr, mr)

# Get testing data
tsb,ys,tsi = patchit( ts, ms)
```

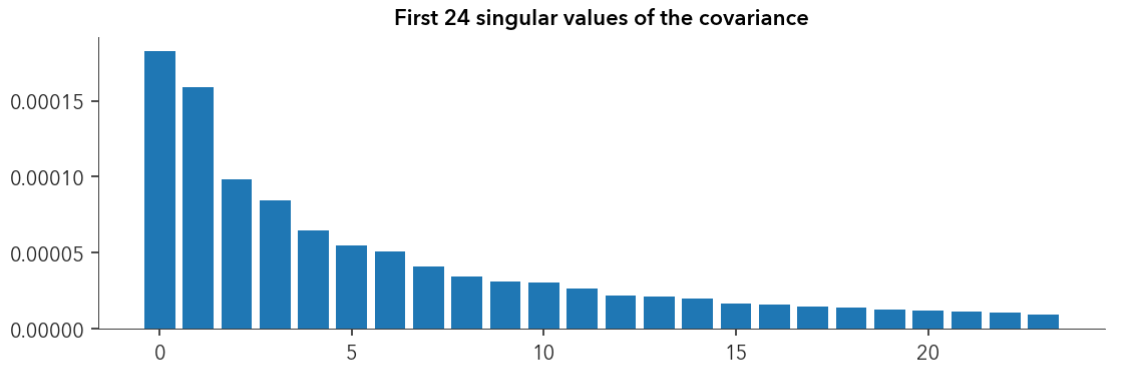
```
# Show me
subplot( 2, 1, 1), pcolormesh( trb[:,yr==1][:,:200])
title( 'Training data, class 1')
subplot( 2, 1, 2), pcolormesh( trb[:,yr==0][:,:200])
title( 'Training data, class 2');
```

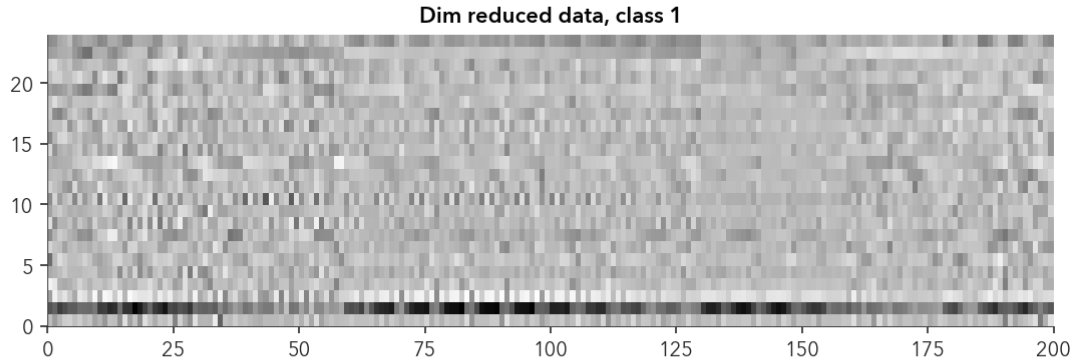


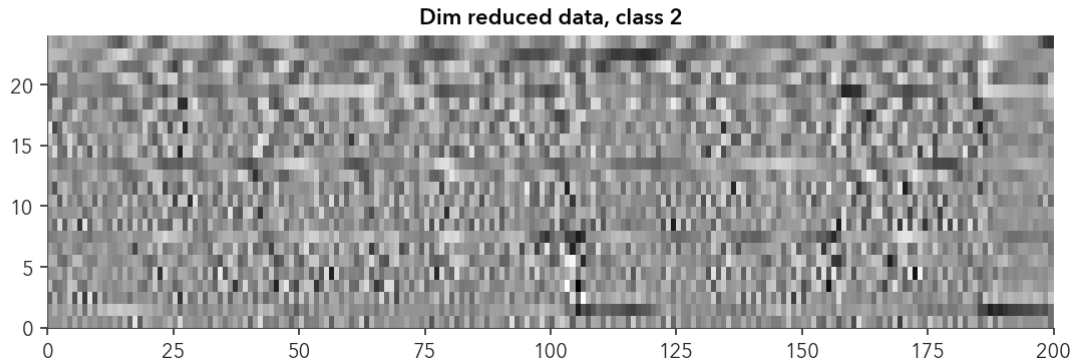

Looking at the matrices that contain each class' data points we see that they are quite dissimilar, so there's light in the end of the tunnel. But 300 dimensions is a little too much so I should bring these down to a reasonable dimension first. We can do that with PCA.

[20

```
# Do PCA
w,m,v = pca( trb, 24)
subplot( 3, 1, 1), bar( range( len( v)), v[::-1])
title( 'First 24 singular values of the covariance')

# Project training and test data to new space
trb = w.dot( trb-m)
tsb = w.dot( tsb-m)

# Show me
subplot( 3, 1, 2), pcolormesh( trb[:,yr==1][:,:200])
title( 'Dim reduced data, class 1')
subplot( 3, 1, 3), pcolormesh( trb[:,yr==0][:,:200])
title( 'Dim reduced data, class 2')
tight_layout()
```



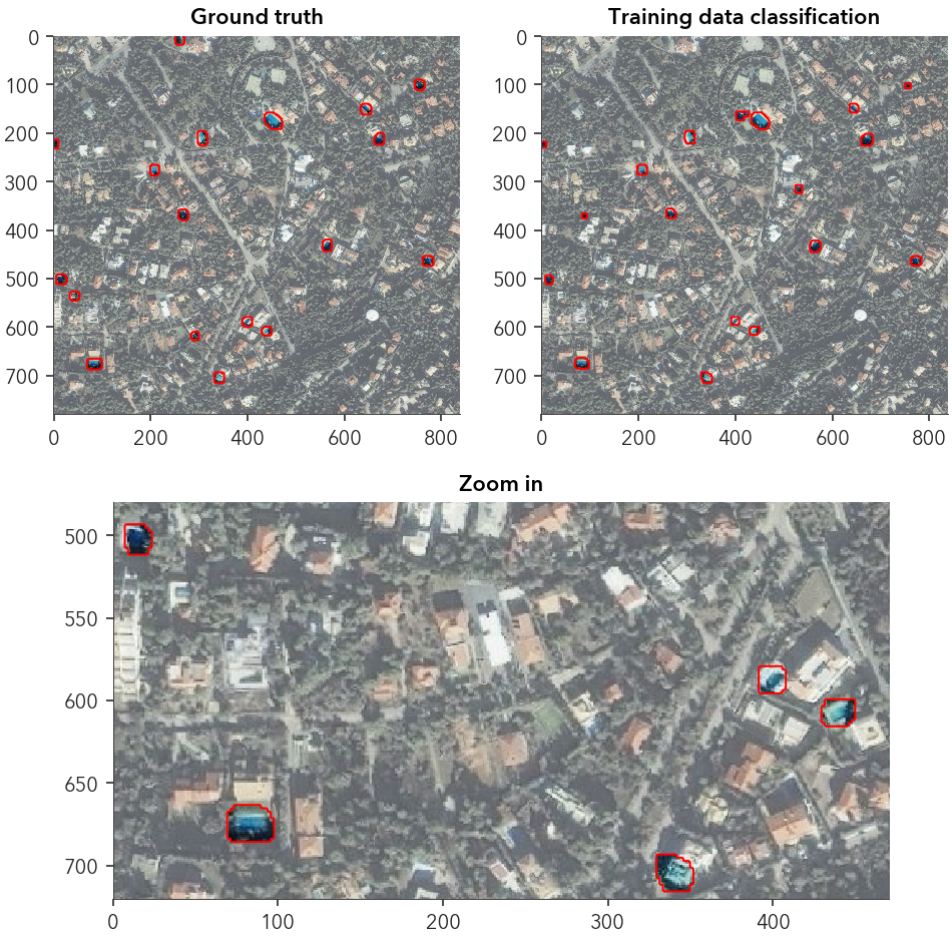




Looks good, the classes look rather different. Now we can learn a classifier. There are a ton of options, since I don't feel particularly fancy I'll simply use a Gaussian classifier.

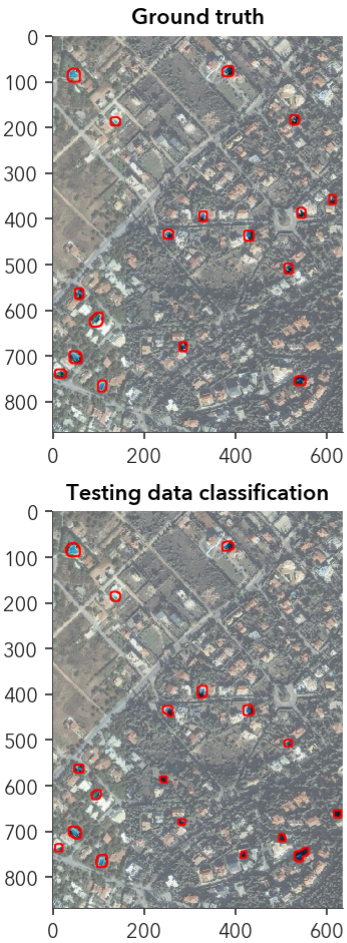
[26

```
# Get simple Gaussian classifiers for both classes
g1 = gclass( trb[:,yr==1], diagonal=True)
g2 = gclass( trb[:,yr==0], diagonal=True)

# Get model likelihoods from both models
l1 = gclass( trb, g1) - 10 # the subtracted offset adds a prior to boost class 1 likelihoods
l2 = gclass( trb, g2)

# Get class labels based on likelihoods
r = (l1 > l2).astype( int)

# Show me the positive hits
def maskshow( x, r, ti):
    xa = .65*ones( (x.shape[0],x.shape[1],4))
    xa[:,:,:3] = x
    for k in find( r == 1):
        i = int( ti[k,0])
        j = int( ti[k,1])
        xa[(i-5):(i+5),(j-5):(j+5),3] = 1
    imshow( xa)
    contour( xa[:,:,-1], levels=[.75], colors='r', linewidths=1)

subplot( 2, 2, 1, facecolor='white'), maskshow( tr, yr, tri), title( 'Ground truth')
subplot( 2, 2, 2, facecolor='white'), maskshow( tr, r, tri), title( 'Training data classification')
subplot( 2, 1, 2, facecolor='white'), maskshow( tr, r, tri), title( 'Zoom in')
axis( [0, 470, 720, 480]);
```




Looks good, we seem to have something that's working ok. It did miss a couple of pools, but it also detected a couple of pools that were not marked in the training data. Let's try this on the test data.

[31]
```
# Get model likelihoods from both models
l1 = gclass( tsb, g1) - 10
l2 = gclass( tsb, g2)

# Get class labels based on comparing the likelihoods
r = (l1 > l2).astype( int)

# Show me the hits
subplot( 2, 2, 1, facecolor='white'), maskshow( ts, ys, tsi)
gca().facecolor = '#FFF'
title( 'Ground truth')

subplot( 2, 2, 3, facecolor='white'), maskshow( ts, r, tsi)
title( 'Testing data classification')

subplot( 1, 2, 2, facecolor='white'), maskshow( ts, r, tsi)
axis( [0, 150, 850, 550])
title( 'Zoom in');
```





Same as before, we missed a couple, but we also found a couple that weren't marked. Can we do better? Sure, we can use more sophisticated classifiers and features (e.g. a GMM or a Neural Net which can drop the error dramatically), but you don't want to see all that code here!

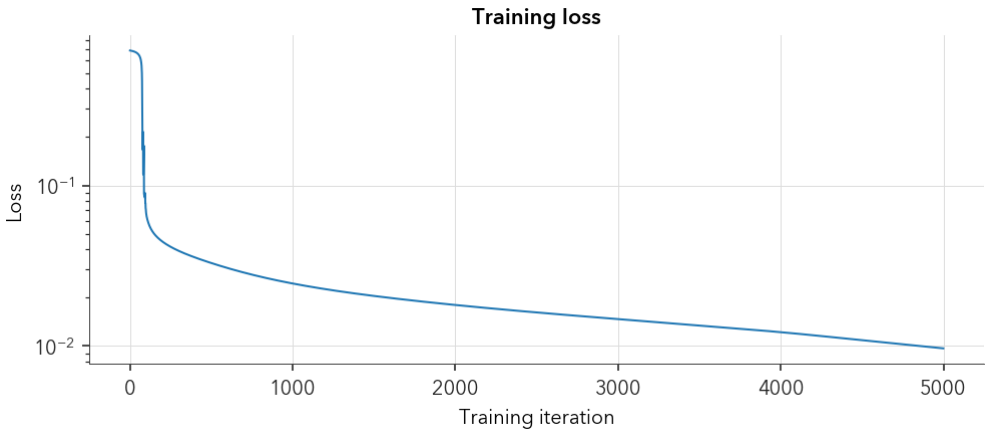
## An alternative classifier

Now, let's use a simple 2-layer neural net and see how we do.

[35

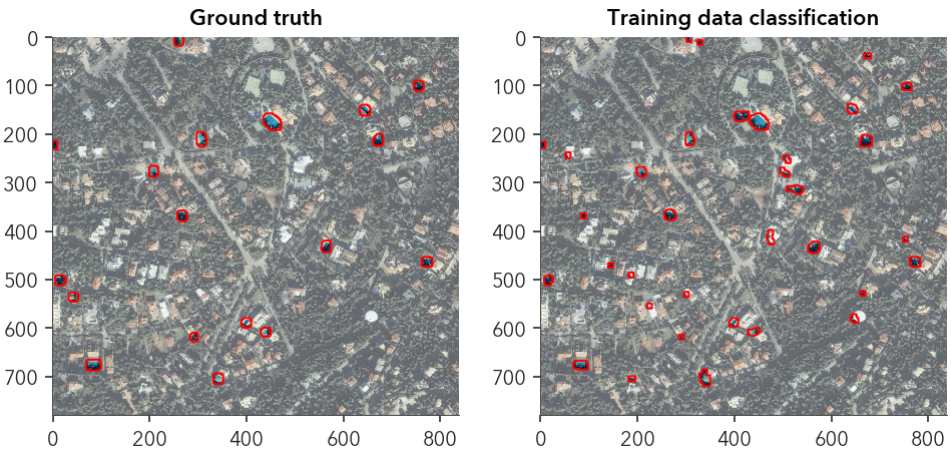
```
# Simple neural net feedforward pass
def ffnet( l0, net):
    l1 = 1/(1 + exp( -l0.dot( net['w1']) - net['b1']))
    l2 = 1/(1 + exp( -l1.dot( net['w2']) - net['b2']))
    return l2,l1

# Training pass for a simple neural net
def trnet( x, y, net):
    # Forward pass
    l2,l1 = ffnet( x, net)

    # Get gradients
    l2d = (y-l2)*(l2*(1-l2))
    l1d = l2d.dot( net['w2'].T) * (l1*(1-l1))

    # Update weights
    h = .001
```

```
        net['w2'] += h*l1.T.dot( l2d)
        net['w1'] += h*x.T.dot( l1d)
        net['b2'] += h*ones( l1.shape[0]).T.dot( l2d)
        net['b1'] += h*ones( x.shape[0]).T.dot( l1d)
        return net,mean( (y-l2)**2)

    # Make training data, just use the first 1600 samples of each class
    x = vstack( (trb[:,yr==1].T,trb[:,yr==0].T[:1600]))
    y = hstack( (yr[yr==1].T,yr[yr==0].T[:1600]))[:,None]

    # Initialize network
    net = {'w1': random( (x.shape[1],50)), 'w2': random( (50,1)),
           'b1': random( 50),   'b2': random( 1)}

    # Train it for 100 epochs
    loss = zeros( 5000)
    for i in range( len( loss)):
        net,loss[i] = trnet( x, y, net)
    figure(figsize=(8,3))
    semilogy( loss), grid( 'on'), title( 'Training loss')
    ylabel( 'Loss'), xlabel( 'Training iteration');
```

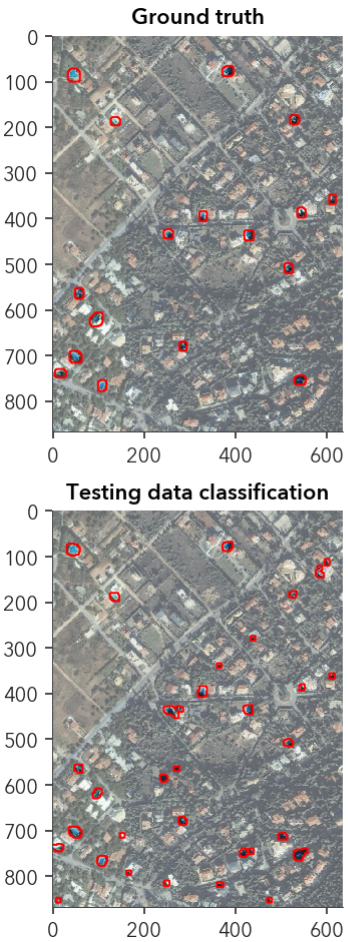



The loss looks low enough. We can now try it out on the training and testing data to get a sense of how well it works. During classification I set a threshold of 0.99, over which the input is definitely a pool. By adjusting that threshold I can make the classifier more strict (fewer false positives, more false negatives), or not (more false positives, fewer false negatives)

[42]
```
    # Evaluate
    r,_ = ffnet( trb.T, net)
    r = r > .99

    # Show me
    subplot( 2, 2, 1, facecolor='white'), maskshow( tr, yr, tri), title( 'Ground truth')
    subplot( 2, 2, 2, facecolor='white'), maskshow( tr, r, tri), title( 'Training data classification')
    subplot( 2, 1, 2, facecolor='white'), maskshow( tr, r, tri), title( 'Zoom in')
    axis( [0, 470, 720, 480]);

    # Evaluate on test data
    r,_ = ffnet( tsb.T, net)
    r = r > .99

    # Show me
    figure()
    subplot( 2, 2, 1, facecolor='white'), maskshow( ts, ys, tsi), title( 'Ground truth')
    subplot( 2, 2, 3, facecolor='white'), maskshow( ts, r, tsi), title( 'Testing data classification')
    subplot( 1, 2, 2, facecolor='white'), maskshow( ts, r, tsi), title( 'Zoom in')
    axis( [0, 140, 860, 550]);
```


Ground truth
Training data classification


Zoom in


Ground truth
Testing data classification


Zoom in