General rules:

- For all figures that you generate, remember to add meaningful labels to the axes (including units), and provide a legend and colorbar, if applicable.
- Do not hard code constants, like number of samples, number of channels, etc in your program. These values should always be determined from the given data. This way, you can easily use the code to analyse other data sets.
- Do not use high-level functions from toolboxes like scikit-learn.
- Before submitting, check your code by executing: Kernel -> Restart & run all.
- Replace *Template* by your *FirstnameLastname* in the filename, or by *Lastname1Lastname2* if you work in pairs.

BCI-IL - Exercise Sheet #07

11/15

Name:

In [1]:
```
% matplotlib inline

import numpy as np
import scipy as sp
import scipy.signal
from matplotlib import pyplot as plt

import bci_minitoolbox as bci
import bci_classifiers as cfy
import bci_classifiers2 as cfy2
```

In [2]:
```
def proc_spatialFilter(cnt, clab, chan, neighbors='*'):
    '''
    Usage:
        cnt_sf = proc_spatialFilter(cnt, clab, chan, neighbors='*')
    Parameters:
        cnt:       a 2D array of multi-channel timeseries (size: channels x samples),
        clab:      a 1D array of channel names  (size: channels)
        chan:      channel of center location
        neighbors: labels of channels that are to be subtracted
    Returns:
        cnt_sf:    timeseries of spatially filtered channel (size: 1 x samples)
    Examples:
        cnt_c4_bip = proc_spatialFilter(cnt, clab, 'C4', 'CP4')
        cnt_c4_lap = proc_spatialFilter(cnt, clab, 'C4', ['C2','C6','FC4','CP4'])
        cnt_c4_car = proc_spatialFilter(cnt, clab, 'C4', '*')
    '''
    cidx= clab.index(chan)
    if isinstance(neighbors, list):
        nidx = [clab.index(cc) for cc in neighbors]
    elif neighbors == '*':
        nidx = range(len(clab))    # Common Average Reference (CAR)
    else:
        nidx = [clab.index(neighbors)]
    cnt_sf = cnt[[cidx],:] - np.mean(cnt[nidx,:], axis=0)
    return cnt_sf
```

Preparation: Load data

In [3]:
```
fname = 'imagVPaw.npz'
cnt, fs, clab, mnt, mrk_pos, mrk_class, mrk_className = bci.load_data(fname)
```

Exercise 1: Determining a Frequency Band (6 points)

4/6

Calculate the classwise averaged power spectral density (PSD) at scalp locations C3 and C4 in the data set `imagVPaw`. For each motor imagery condition, you may use the interval 1000-5000 ms. Determine a frequency band that seems useful to discriminate the two moto imagery conditions. **Note:** To take into account what was said in the lecture about spectra and spatial filtering, use a bipolar filter for C3 and a Laplace filter for C4. To calculate the average spectra over single trials you can use

```
>>> f, psd = sp.signal.welch(X.flatten('F'), fs=100)
```

assuming the single trials of one channel to be the columns of `X` and sampled at 100Hz.

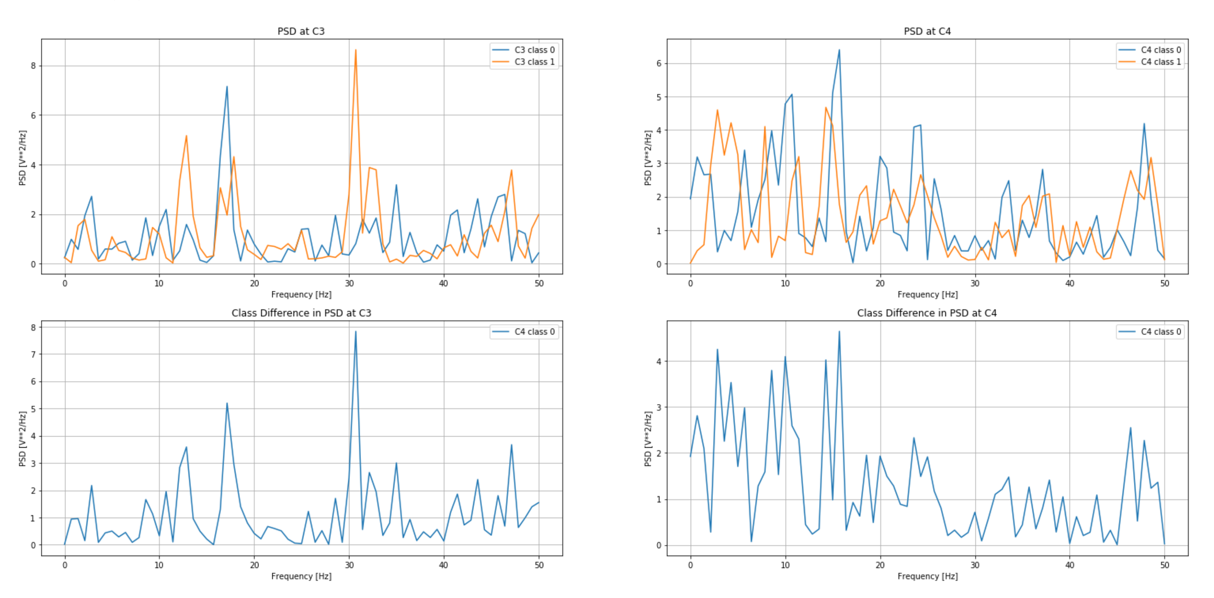
In [4]:

-2

```
cnt_c3_bip = proc_spatialFilter(cnt, clab, 'C3', 'CP3')
cnt_c4_lap = proc_spatialFilter(cnt, clab, 'C4', ['C2','C6','FC4','CP4'])

c3_0 = cnt_c3_bip[0][mrk_pos][mrk_class==0]
c3_1 = cnt_c3_bip[0][mrk_pos][mrk_class==1]

c4_0 = cnt_c4_lap[0][mrk_pos][mrk_class==0]
c4_1 = cnt_c4_lap[0][mrk_pos][mrk_class==1]

f_c3_0, psd_c3_0 = sp.signal.welch(c3_0.flatten('F'), fs=100)
f_c3_1, psd_c3_1 = sp.signal.welch(c3_1.flatten('F'), fs=100)

f_c4_0, psd_c4_0 = sp.signal.welch(c4_0.flatten('F'), fs=100)
f_c4_1, psd_c4_1 = sp.signal.welch(c4_1.flatten('F'), fs=100)

#qpsd_c3_0 = 10 * np.log10(psd_c3_0)
#qpsd_c3_1 = 10 * np.log10(psd_c3_1)
#qpsd_c4_0 = 10 * np.log10(psd_c4_0)
#qpsd_c4_1 = 10 * np.log10(psd_c4_1)

plt.figure(figsize=(26,12))
plt.subplot(2,2,1)
plt.plot(f_c3_0, psd_c3_0, label= 'C3 class 0')
plt.plot(f_c3_1, psd_c3_1, label= 'C3 class 1')

plt.xlabel('Frequency [Hz]')
plt.ylabel('PSD [V**2/Hz]')
plt.grid()
plt.legend()
plt.title('PSD at C3')

plt.subplot(2,2,2)
plt.plot(f_c4_0, psd_c4_0, label= 'C4 class 0')
plt.plot(f_c4_1, psd_c4_1, label= 'C4 class 1')

plt.xlabel('Frequency [Hz]')
plt.ylabel('PSD [V**2/Hz]')
plt.grid()
plt.legend()
plt.title('PSD at C4')

plt.subplot(2,2,3)
plt.plot(f_c4_0, np.abs(psd_c3_0 - psd_c3_1), label= 'C4 class 0')
plt.xlabel('Frequency [Hz]')
plt.ylabel('PSD [V**2/Hz]')
plt.grid()
plt.legend()
plt.title('Class Difference in PSD at C3')

plt.subplot(2,2,4)
plt.plot(f_c4_0, np.abs(psd_c4_0 - psd_c4_1), label= 'C4 class 0')
plt.xlabel('Frequency [Hz]')
plt.ylabel('PSD [V**2/Hz]')
plt.grid()
plt.legend()
plt.title('Class Difference in PSD at C4')
```

```
C:\zzz My Shit\Pthon\lib\site-packages\scipy\signal\spectral.py:1785: UserWarning: nperseg = 256
is greater than input length  = 140, using nperseg = 140
  .format(nperseg, input_length))
```

Out[4]: Text(0.5,1,'Class Difference in PSD at C4')



In [5]:
```
# based on the Class Difference plots we can determine the frequency bands 30-35 Hz (for C3) and 14-
17 Hz (for C4)
# as intervals that can be used to discriminate between two moto imagery conditions.
```

Exercise 2: Visualizing ERD/ERS curves (4 points)

3/4

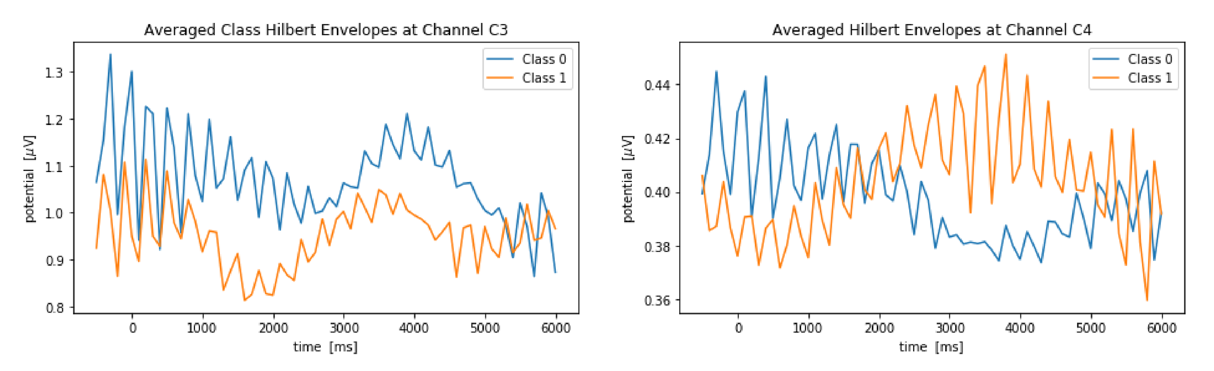
Design a band-pass filter with the frequency band that was selected in exercises #1 (use the band `[11. 16.]` if you did not succeed with that, but note that this band may be suboptimal). For the same channels (and spatial filters) as in exercise #1, calculate and display the classwise averaged ERD/ERS curves with respect to the determined frequency band for the time interval that encompasses a prestimulus interval of 500 ms and extends to 6000 ms poststimulus.

In [6]:

-1

```
band_c3 = [30., 35.]
Wn_c3 = np.array(band_c3) / fs * 2
b_c3, a_c3 = sp.signal.butter(5, Wn_c3, btype='bandpass')

cnt_c3_bip_filtered = sp.signal.lfilter(b_c3, a_c3, cnt_c3_bip)
cnt_c3_hilbert = np.imag(sp.signal.hilbert(cnt_c3_bip_filtered))

list0 = []
for a in mrk_pos[mrk_class==0]:
    list0.append(cnt_c3_hilbert[0][a-5:a+61])

list1 = []
for b in mrk_pos[mrk_class==1]:
    list1.append(cnt_c3_hilbert[0][b-5:b+61])

cnt_c3_0_avg = np.mean(np.abs(np.asarray(list0)),axis=0)
cnt_c3_1_avg = np.mean(np.abs(np.asarray(list1)),axis=0)

timeaxis = np.arange(-500, 6100, fs)

plt.figure(figsize=(16,4))
plt.subplot(1,2,1)
plt.plot(timeaxis, cnt_c3_0_avg, label = 'Class 0')
plt.plot(timeaxis, cnt_c3_1_avg, label = 'Class 1')
plt.title('Averaged Class Hilbert Envelopes at Channel C3')
plt.xlabel('time  [ms]')
plt.ylabel('potential  [$\mu$V]')
plt.legend()

#################################################################################
band_c4 = [14., 17.]
Wn_c4 = np.array(band_c4) / fs * 2
b_c4, a_c4 = sp.signal.butter(5, Wn_c4, btype='bandpass')

cnt_c4_lap_filtered = sp.signal.lfilter(b_c4, a_c4, cnt_c4_lap)
cnt_c4_hilbert = np.imag(sp.signal.hilbert(cnt_c4_lap_filtered))

qlist0 = []
for c in mrk_pos[mrk_class==0]:
    qlist0.append(cnt_c4_hilbert[0][c-5:c+61])

qlist1 = []
for d in mrk_pos[mrk_class==1]:
    qlist1.append(cnt_c4_hilbert[0][d-5:d+61])

cnt_c4_0_avg = np.mean(np.abs(np.asarray(qlist0)),axis=0)
cnt_c4_1_avg = np.mean(np.abs(np.asarray(qlist1)),axis=0)

plt.subplot(1,2,2)
plt.plot(timeaxis, cnt_c4_0_avg, label = 'Class 0')
plt.plot(timeaxis, cnt_c4_1_avg, label = 'Class 1')
plt.title('Averaged Hilbert Envelopes at Channel C4')
plt.xlabel('time  [ms]')
plt.ylabel('potential  [$\mu$V]')
plt.legend()
```

Out[6]: <matplotlib.legend.Legend at 0x1f195adca58>



Exercise 3: Classification of single-trial ERD/ERS curves (5 points)

4/5

Subsample the band-pass filtered and rectified epochs of the interval 1000 ms to 5000 ms down to 5 Hz by calculating the average of every consequtive window of 200 ms. Perform crossvalidation of those features separately for each single channel and display the result as scalp map. (In this case, do not use a spatial filter.) Furthermore, perform a 3-fold crossvalidation for the joint feature vector (dimensionality is 20 [time points] $\times$ 51 [channels]). **Note:** Don't be disappointed if the results are not good. On the next sheet you will implement a powerful method for this case.

In [25]:

-1

```
# based on the difference graphs from exercise 1, we chose a frequency band [16,18] to use for the e
ntire dataset
band=[16. ,18.]
b,a = scipy.signal.butter(5,(band[0]/fs*2,band[1]/fs*2), btype = 'bandpass')
cnt_filtered = scipy.signal.lfilter(b,a,cnt)
#cnt_hilbert = np.imag(sp.signal.hilbert(cnt_filtered))

ival = []
for i in np.arange(1000,5200,200):
    ival.append([i,i+200])
ival.pop(-1)
#print(ival)
ival_list = np.array(ival)
#print(ival_list.shape)

t_prestim = 100
t_end = np.max(ival_list)
epo, epo_t = bci.makeepochs(cnt_filtered, fs, mrk_pos, [-t_prestim, t_end])
epo = bci.baseline(epo, epo_t, [-t_prestim, 0])

nIvals = len(ival_list)
T, nChans, nEpochs = epo.shape

fv= np.zeros((nIvals, nChans, nEpochs))

loss=np.zeros((len(clab)))
#print(loss.shape)

for k, ival in enumerate(ival_list):
    tidx = (ival[0] <= epo_t) & (epo_t <= ival[1])
    fv[k] = np.mean(epo[tidx, :, :], axis=0)

for i in range(len(clab)):
    loss[i], _=cfy.crossvalidation(cfy.train_LDA, fv[:,i,:], mrk_class, folds=10)

bci.scalpmap(mnt, loss, clim='minmax', cb_label='10-validation weighted loss  [%]')

fvcat = np.vstack(fv)
#print(fvcat.shape)
cfy.crossvalidation(cfy.train_LDA, fvcat, mrk_class, folds=3, verbose=True)
```

```
 45.9 +/- 6.1  (training:  9.0 +/- 3.1)  [using train_LDA]
```

Out[25]: (45.87122026990544, 9.035390448187067)