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 #01

Names:

This exercise sheet refers to lecture #2. The dataset is taken from an ERP Speller experiment, similar to the one discussed in the lecture. The first two tasks are about basic visualizations of Event-Related Potentials (ERPs), and the third one is the implementation of a univariate measure of separability and its visualization.

In [1]:

```
% matplotlib inline

import numpy as np
import scipy as sp
from matplotlib import pyplot as plt
plt.style.use('classic')

import bci_minitoolbox as bci
```

Preparation: Loading Data

In [2]:

```
fname = 'erp_hexVPsag.npz'
cnt, fs, clab, mnt, mrk_pos, mrk_class, mrk_className = bci.load_data(fname)
```

In [3]:

```
print (cnt.shape) # 2D array of multi-channel timeseries (channels x samples), unit [uV]
print (fs) #sampling frequency, unit Hz
print (clab) #a 1D LIST of channel names  (channels) // 55 channels
print (mnt.shape) # a 2D array of channel coordinates
print (mrk_pos) #a 1D array of marker positions (in samples)
print (mrk_class) #a 1D array that assigns markers to classes (0, 1) / (1200,)
print (mrk_className) # a list that assigns class names to classes
```

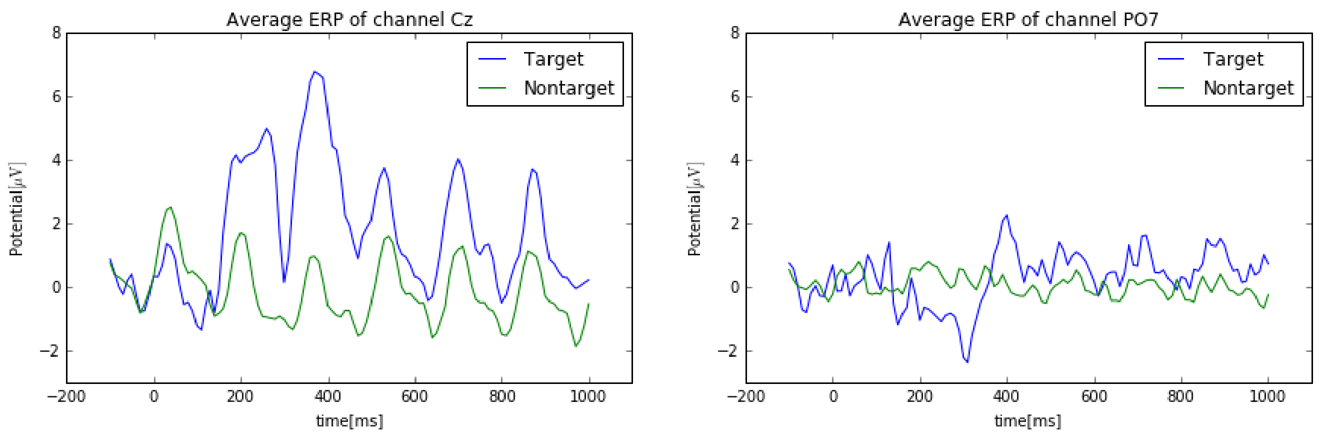
```
(55, 38710)
100
['F7', 'F5', 'F3', 'F1', 'Fz', 'F2', 'F4', 'F6', 'F8', 'FT7', 'FC5', 'FC3', 'FC1', 'FCz', 'FC2',
'FC4', 'FC6', 'FT8', 'T7', 'C5', 'C3', 'C1', 'Cz', 'C2', 'C4', 'C6', 'T8', 'TP7', 'CP5', 'CP3',
'CP1', 'CPz', 'CP2', 'CP4', 'CP6', 'TP8', 'P9', 'P7', 'P5', 'P3', 'P1', 'Pz', 'P2', 'P4', 'P6',
'P8', 'P10', 'PO7', 'PO3', 'POz', 'PO4', 'PO8', 'O1', 'Oz', 'O2']
(55, 2)
[  957   974   991 ... 38376 38393 38410]
[1 1 1 ... 1 0 1]
['target' 'non-target']
```

Exercise 1: Plotting ERPs (5 points)

Plot the average ERPs corresponding to *target* and *nontarget* stimuli in the time interval -100 to 1000 ms for the channels Cz and PO7. This means that the average ERPs are class and channel-wise.
In order to make the curves look reasonable, a so called 'baseline correction' should be performed: for each channel and trial, calculate the average across the prestimulus time interval -100 to 0 ms. This results in one value per channel and trial. Subtract this 'baseline' value from each (single channel/single trial) time course. The function `baseline` is provided in the `bci_minitoolbox`.
It is recommended to have a look at the code of `bci.baseline`! It is three lines only, and shows, e.g., how to get the indices within an epoch that correspond to a given time interval. This will prove useful in future exercises.

In [4]:

```
# Store given information in variables. Subsequent code should only refer to these variables and n
ot
# contain the constants.
ival= [-100, 1000]
ref_ival= [-100, 0]
chans = ['Cz', 'PO7']
chan_pos = []
for i in range(len(chans)):
    chan_pos.append(clab.index(chans[i]))

# Segment continuous data into epochs:
epo, epo_t = bci.makeepochs(cnt, fs, mrk_pos, ival)

# Baseline correction:
epo = bci.baseline(epo, epo_t, ref_ival)

# Now it is your turn to continue ...
for i in range(len(chan_pos)):
    plt.subplot(1,2,i+1)
    plt.plot(epo_t, epo[:,chan_pos[i],mrk_class==0].mean(axis=1),label = 'Target')
    plt.plot(epo_t, epo[:,chan_pos[i],mrk_class==1].mean(axis=1), label = 'Nontarget')
    plt.xlim(-200, 1100)
    plt.ylim(-3, 8)

    plt.title('Average ERP of channel ' + chans[i])
    plt.xlabel("time[ms]")
    plt.ylabel("Potential" + '$[\mathrm{\mu V}]$')
    plt.legend()

plt.subplots_adjust(left  = 0.1, right = 2, bottom = 0.1, top = 0.9, wspace = 0.2, hspace = 0.2)
```

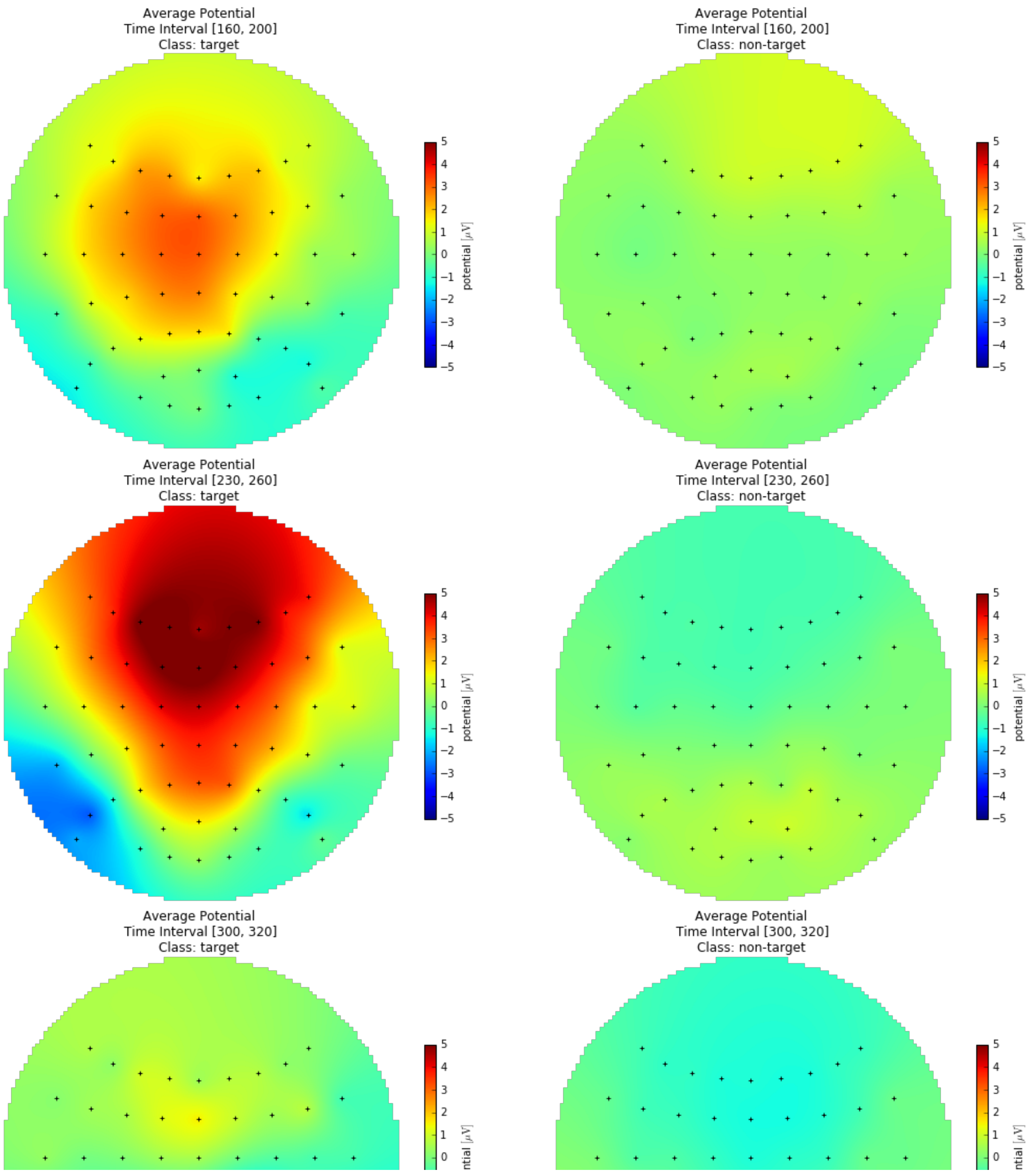


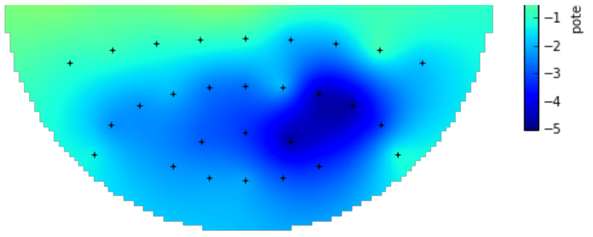
Exercise 2: Scalp Topographies of ERPs (5 points)

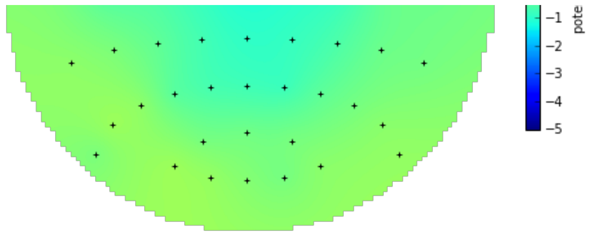
Calculate the average potential separately for the classes *target* and *non-target* in the following time intervals [ms]: [160 200], [230 260], [300 320], and [380 430]. (This involves averaging over all trials of the respective class and averaging over all sample points in the respective time interval.) Visualize the result as scalp topographies using the function `bci.scalpmap`, i.e., 4 maps for each class.
Note: In order to make the maps look reasonable, do a *baseline correction* as in the previous task. To make the maps comparable, use a common scale for the colorbar (see help of `scalpmap`).

In [5]:

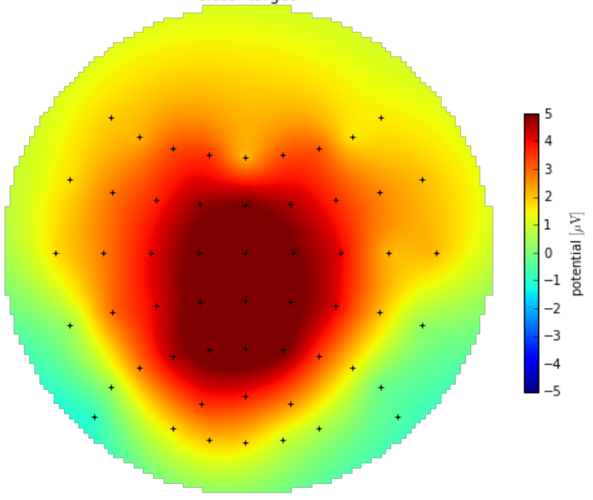
```
ival = [[160, 200], [230, 260], [300, 320], [380, 430]]

#epo: a 3D array of segmented signals (samples x channels x epochs)
#epo_t: a 1D array of time points of epochs relative to marker (in ms)
scalp_potential = []
plt.figure(figsize=(16,30))

for i in range(len(ival)): # for each interval
    start = list(epo_t).index(ival[i][0]) #index of beginning of intervals in epo_t
    end = list(epo_t).index(ival[i][1]) #index of ends of intervals in epo_t

    for k in range(len(mrk_className)): # for each class
        scalp_potential.append([])
        scalp_potential[k].append(epo[start:end,:,mrk_class == k].mean(axis=2))
        scalp_potential[k][i] = np.mean(scalp_potential[k][i], axis=0)

        plt.subplot(len(ival),len(mrk_className),2*i+1 + k)
        bci.scalpmap(mnt, scalp_potential[k][i],clim=[-5,5], cb_label='potential '+'$[\mathrm{\mu V
}]}$')
        plt.title('Average Potential\n Time Interval '+str(ival[i])+'\nClass: '+mrk_className[k]);

plt.subplots_adjust(left  = 0.125, right = 1, bottom = 0.1, top = 0.9, wspace = 0.1, hspace = 0.001)
```



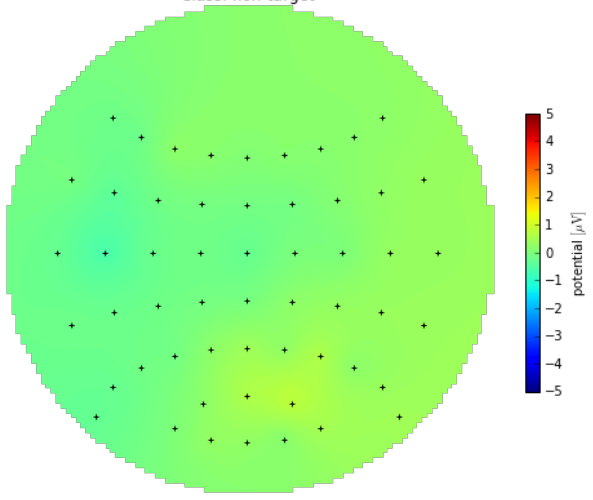




Average Potential
Time Interval [380, 430]
Class: target



Average Potential
Time Interval [380, 430]
Class: non-target



Exercise 3: Visualization with the Biserial Correlation Coefficient (5 points)

Implement a function for the calculation of the signed `r^2`-value (see point-biserial correlation coefficient in BCI lecture #02). From the given data set, extract epochs for the time interval [-100 600] ms relativ to each stimulus presentation. Calculate for each channel and each point in time the signed `r^2`-value wrt. classes *target* vs. *non-target* and visualize this (channel `x` time points) matrix (`pl.imshow`). Again, use 100 ms prestimulus for baseline correction.

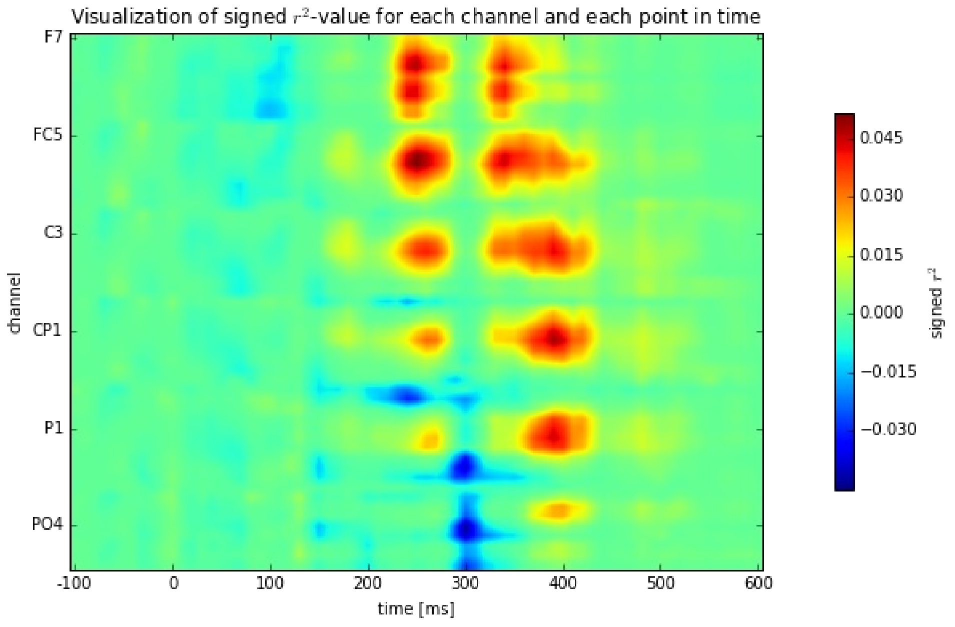
In [6]:

```
def signed_r_square(epo, y):
    '''
    Synopsis:
        epo_r = signed_r_square(epo, y)
    Arguments:
        epo:    3D array of segmented signals (time x channels x epochs),
                see makeepochs
        y:      labels with values 0 and 1 (1 x epochs)
    Output:
        epo_r:  2D array of signed r^2 values (time x channels)
    '''
    a = epo[:,:,y==0].mean(axis = 2)
    b = epo[:,:,y==1].mean(axis = 2)
    v = np.var(epo,axis=2)
    n1 = epo[:,:,y==0].shape[2];
    n2 = epo[:,:,y==1].shape[2];

    epo_r = np.sign(a-b)*(n1*n2/np.power(n1+n2,2))*(np.power(a-b,2)/v);
    return epo_r
```

In [7]:

```
ival= [-100, 600]
ref_ival= [-100, 0] # 100ms prestimilus

# Segment continuous data into epochs:
epo, epo_t = bci.makeepochs(cnt, fs, mrk_pos, ival)

# Baseline correction:
epo = bci.baseline(epo, epo_t, ref_ival)
```

```
epo_r = signed_r_square(epo, mrk_class)

plt.figure(1,figsize=(16,6))
im = plt.imshow(epo_r.transpose())

xlocs, xlabels = plt.xticks()
ylocs, ylabels = plt.yticks()
plt.xticks(xlocs[1:-1], [-100, 0, 100, 200, 300, 400, 500, 600])
plt.yticks(ylocs[1:-1],[clab[0],clab[10],clab[20],clab[30],clab[40],clab[50],clab[54]])

plt.title('Visualization of signed ' + '$r^2$' + '-value for each channel and each point in time')

plt.xlabel('time [ms]')
plt.ylabel('channel')

cbar = plt.colorbar(im, shrink=0.7)
cbar.set_label('signed '+'$r^2$')
```


Visualization of signed r^2-value for each channel and each point in time
F7
FC5
C3
CP1
P1
PO4
channel
time [ms]
signed r^2