13/15 points

2019ws-BCIIL-Sheet02-Karastoyanov,Mohan,Al-Asadi

November 5, 2019

1 BCI-IL - Exercise Sheet #02

1.0.1 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.

Name(s):

[1]:
```
%matplotlib inline

import numpy as np
import scipy as sp
from matplotlib import pyplot as plt
from matplotlib.patches import Ellipse
import matplotlib.transforms as transforms

import bci_minitoolbox as bci
```

1.1 Preparation: Loading Data

[2]:
```
fname = 'erp_hexVPsag.npz'
cnt, fs, clab, mnt, mrk_pos, mrk_class, mrk_className = bci.load_data(fname)
```

[3]:
```
print (clab)
```

```
['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']
```

1.2 Exercise 1: Scatter Plot of 2D Features (8 points)

7/8

Make a scatter plot of the two distributions - *targets* and *nontargets*, one dot for each trial. On the x-axis, plot the value of channel Cz at t = 380 ms, and at the y-axis the value of PO3 at t = 300 ms. *(You may refer to the results of sheet 01 for the reason of this choice.)* Draw for both distributions the two principal axes, with the lengths of the axes being the standard deviation of the data in that direction. Draw also the corresponding ellipses. **Hint:** You can get that with a transformation of the unit circle as on the slide *Illustration of Multiplication* ....

[7]:
```
# Follow the same approach as exercise 2 to get the epoch of a continuoussignal␣
↪which will contain the data points
# for which the potential needs to be extracted

ival= [-100, 1000]
ref_ival= [-100, 0]

channels = ['Cz','PO3']
chans = [ clab.index(x) for x in channels]

time_points = [380,300]

# 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)

Cz_380_t = epo[list(epo_t).index(time_points[0]),chans[0],mrk_class==0]
PO3_300_t = epo[list(epo_t).index(time_points[1]),chans[1],mrk_class==0]

Cz_380_nt = epo[list(epo_t).index(time_points[0]),chans[0],mrk_class==1]
PO3_300_nt = epo[list(epo_t).index(time_points[1]),chans[1],mrk_class==1]

Y = np.vstack((Cz_380_t, PO3_300_t))      #stack the target arrays together
Z = np.vstack((Cz_380_nt, PO3_300_nt))    #stack the non-target arrays together

mean_x = np.mean(Y[0,:])                  # get the mean of of all the targets␣
↪in Cz
mean_y = np.mean(Y[1,:])                  # get the mean of of all the targets␣
↪in PO3
mean_vector = np.array([[mean_x],[mean_y]]).reshape(2,)  #concatenat the mean␣
↪vectors and stace then as two rows
cov_mat = np.cov(Y)                       # covariance matrix of targets
eig_vals, eig_vecs = np.linalg.eig(cov_mat)

mean_x_nt = np.mean(Z[0,:])
mean_y_nt = np.mean(Z[1,:])
mean_vector_nt = np.array([[mean_x_nt],[mean_y_nt]]).reshape(2,)
```

```
cov_mat_nt = np.cov(Z)
eig_vals_nt, eig_vecs_nt = np.linalg.eig(cov_mat_nt)

print('Mean Vector:\n', mean_vector) # pca.mean_
print('Covariance Matrix:\n',cov_mat) #pca.get_covariance
print('Eigenvalues:\n',eig_vals) # pca.explained_variance_
print('Eigenvectors:\n',eig_vecs) #pca.components_

####################################################
def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0,
                    shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)

plt.scatter(Y[0,:], Y[1,:], alpha=0.5, label = 'Target')
plt.scatter(Z[0,:], Z[1,:], alpha=0.1, label = 'Nontarget')

for length, vector in zip(eig_vals,eig_vecs):
    v = vector * np.sqrt(length)
    draw_vector(mean_vector.T,mean_vector.T + v)
    plt.axis('equal');

for length, vector in zip(eig_vals_nt,eig_vecs_nt):
    v = vector * np.sqrt(length)
    draw_vector(mean_vector_nt.T,mean_vector_nt.T + v)
    plt.axis('equal');

def eigsorted(cov):
    vals, vecs = np.linalg.eigh(cov)
    order = vals.argsort()[::-1]
    return vals[order], vecs[:,order]

nstd = 2              # Took the confidence value for two standard deviations
# ax = plt.subplot(111)

vals, vecs = eigsorted(cov_mat)
theta = np.degrees(np.arctan2(*vecs[:,0][::-1]))
w, h = 2 * ~~nstd~~ * np.sqrt(vals)
ell = Ellipse(xy=(mean_x, mean_y),
              width=w, height=h,
              angle=theta, color='black')
ell.set_facecolor('none')
plt.gca().add_patch(ell)
```

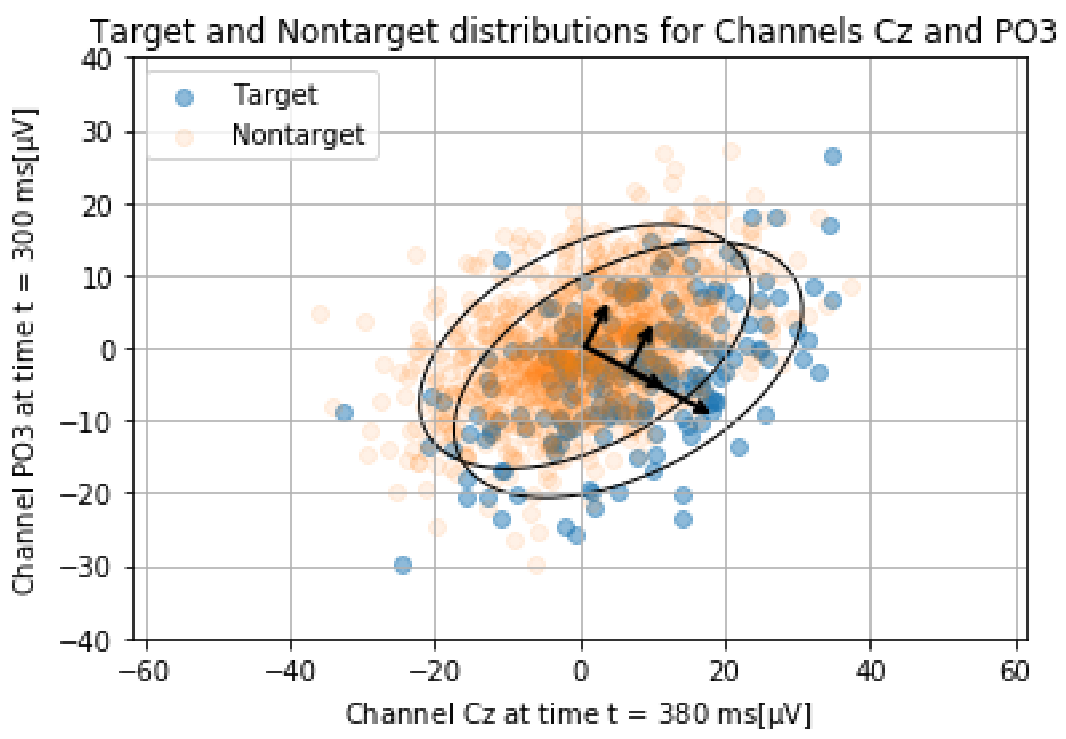
-0.5

-0.5

```
vals_nt, vecs_nt = eigsorted(cov_mat_nt)
theta = np.degrees(np.arctan2(*vecs[:,0][::-1]))
w, h = 2 * nstd * np.sqrt(vals_nt)
ell = Ellipse(xy=(mean_x_nt, mean_y_nt),
              width=w, height=h,
              angle=theta, color='black')
ell.set_facecolor('none')
plt.gca().add_patch(ell)

plt.xlim(-40, 40)
plt.ylim(-40, 40)
plt.title('Target and Nontarget distributions for Channels Cz and PO3 ')
plt.xlabel("Channel Cz at time t = 380 ms" + '$[\mathrm{\mu V}]$')
plt.ylabel("Channel PO3 at time t = 300 ms" + '$[\mathrm{\mu V}]$')
plt.grid()
plt.legend()
```

```
Mean Vector:
 [ 6.69664551 -3.02509904]
Covariance Matrix:
 [[144.39289434  49.99331452]
 [ 49.99331452  77.58950833]]
Eigenvalues:
 [171.11610959  50.86629308]
Eigenvectors:
 [[ 0.88191223 -0.47141364]
 [ 0.47141364  0.88191223]]
```

```
[7]: <matplotlib.legend.Legend at 0xb1c41d898>
```



6/7

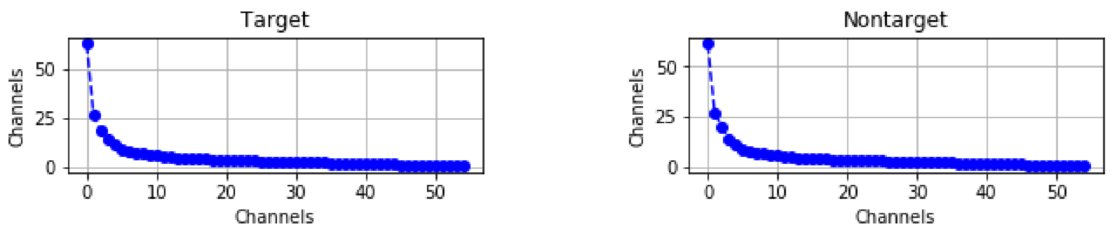
1.3 Exercise 2: Covariances and Eigenvalues (7 points)

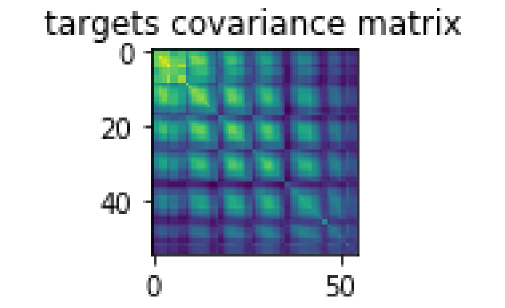
Calculate the channelwise covariance matrices (channel x channel) of the data for time point 380 ms for both classes (`np.cov`) and visualize them (`pl.imshow`). Perform an Eigenvalue decomposition (`np.linalg.eigh`) of the covariance matrices and plot (again class-wise) the eigenvalue spectrum. Then determine the four principle components (Eigenvectors) for each class that correspond to the largest Eigenvalues and display them as scalp maps (function `scalpmap` provided in the `bci_minitoolbox`).

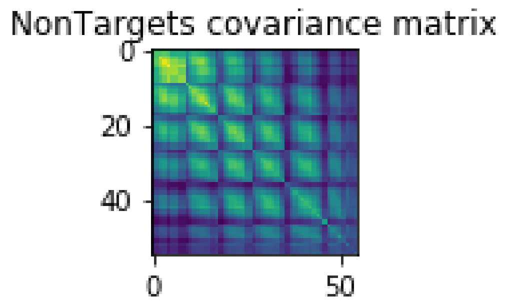
[5]:
```
# print (Y)
arr = []
t = np.empty((55,55))     # numpy array for targets
nt = np.empty((55,55))    # numpy array for non-targets

# stack up the potentials for all channels for target class
for x in clab:
    temp = epo[list(epo_t).index(time_points[0]),clab.index(x),mrk_class==0]
    arr.append(temp)

t = np.vstack(arr)    # Stacked array of target values

# print (t.shape)

arr =[]
```

```
for x in clab:
    temp = epo[list(epo_t).index(time_points[0]),clab.index(x),mrk_class==1]
    arr.append(temp)

nt = np.vstack(arr)   # Stacked array of non-target values

cov_t = np.cov(t)     # covariance matrix for targets
cov_nt = np.cov(nt)   # covariance matrix for non-targets

lambda_t, E_t = np.linalg.eig(cov_t)
lambda_nt, E_nt = np.linalg.eig(cov_nt)

i1 = lambda_t.argsort()[::-1]
EigenValuesTarget = lambda_t[i1]
EigenVectoresTarget = E_t[:,i1]

plt.subplot(2, 2, 1)
plt.title('Target')
plt.xlabel("Channels")
plt.ylabel("Channels")
plt.grid()

plt.plot(np.sqrt(EigenValuesTarget), "bo--")
plt.subplots_adjust(left = 0.125, right = 1.5, bottom = 0.1, top = 0.9, wspace␣
 ↪= 0.5, hspace = 1)

i2 = lambda_nt.argsort()[::-1]
EigenValuesNonTarget = lambda_nt[i2]
EigenVectoresNonTarget = E_nt[:,i2]

plt.subplot(2, 2, 2)
plt.title('Nontarget')
plt.xlabel("Channels")
plt.ylabel("Channels")
plt.grid()
plt.plot(np.sqrt(EigenValuesNonTarget), "bo--")

fig, axs = plt.subplots(2)
axs[0].set_title('targets covariance matrix')
axs[0].imshow(cov_t)
plt.subplots_adjust(left = 0.125, right = 1.5, bottom = 0.1, top = 0.9, wspace␣
 ↪= 0.1, hspace = 1)
axs[1].set_title('NonTargets covariance matrix')
axs[1].imshow(cov_nt)
```

[5]:
```
<matplotlib.image.AxesImage at 0x189473f6588>
```



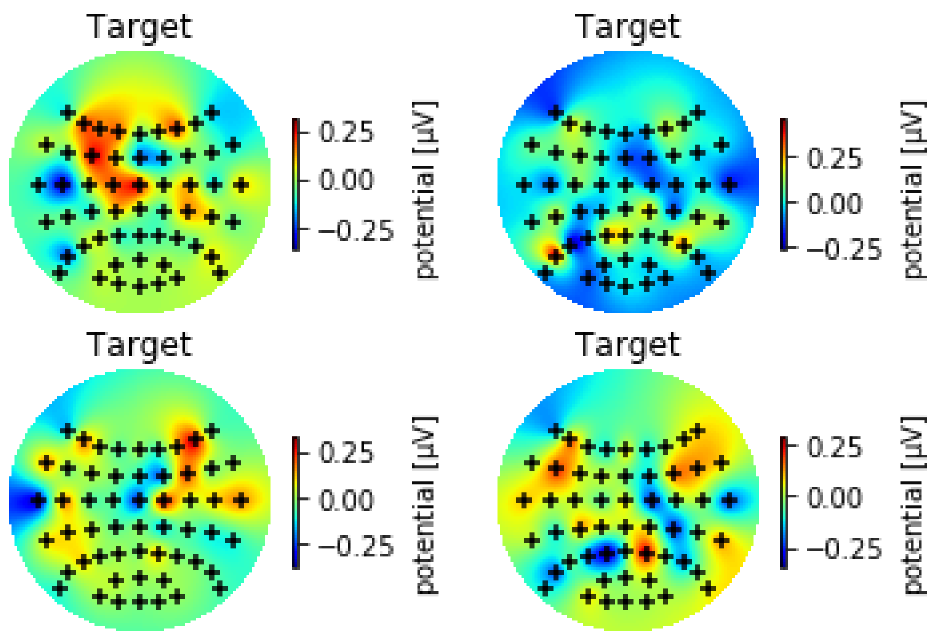




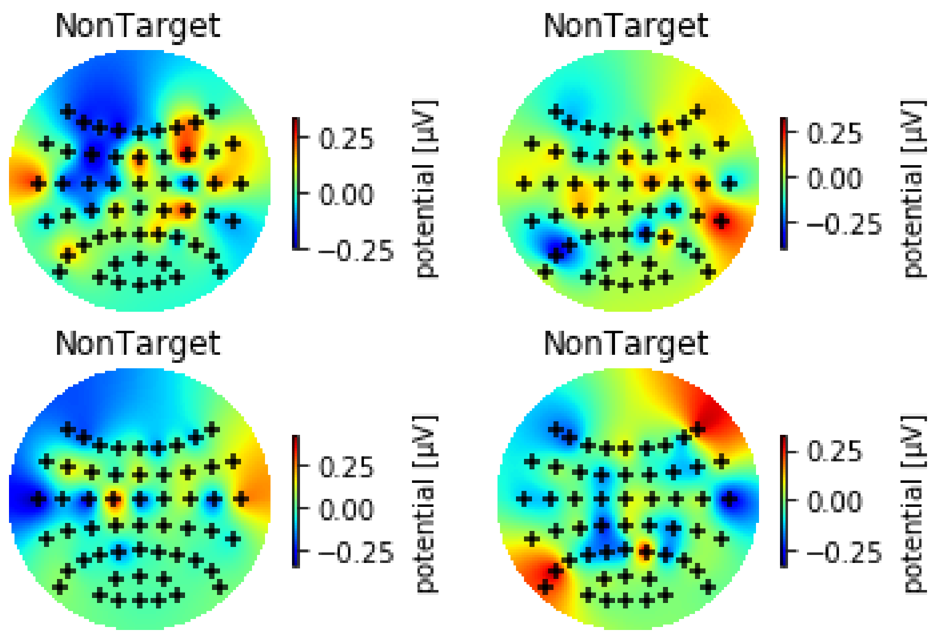
[6]:
```
for i in range(4):
    plt.subplot(2, 2, i+1)
    plt.title('Target')
    bci.scalpmap(mnt, EigenVectoresTarget[i], cb_label='potential␣
↪'+'$[\mathrm{\mu V}]}$')
```

-1

-0.5



[7]:
```
for i in range(4):
    plt.subplot(2, 2, i+1)
    plt.title('NonTarget')
    bci.scalpmap(mnt, EigenVectoresNonTarget[i], cb_label='potential␣
 ↪'+'$[\mathrm{\mu V}]$')
```



[ ]:
```
```

[ ]:
```
```