2019ws-BCIIL-Sheet04-Solution

November 19, 2019

0.0.1 General rules:

- For all figures that you generate, remember to add meaningful labels to the axes, and make a legend, 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.
- Replace *Template* by your *FirstnameLastname* in the filename, or by *Lastname1Lastname2* if you work in pairs.

1 BCI-IL WS - Exercise Sheet #04

Name:

[1]:
```
import numpy as np
import scipy as sp
from matplotlib import pyplot as plt

import bci_minitoolbox as bci
import bci_classifiers as cfy
```

1.1 Exercise 1: Implementation of Shrinkage for Covariance Estimation (7 points)

Implement a function `cov_shrink` that estimates the covariance matrix of data using shrinkage with the analytic method of determining the shrinkage coefficient as presented in the lecture. Input and output of that function should be as in the function `numpy.cov`.

If you cannot succeed with this task, you may import the function `train_LDAshrink` from `bci_helper_sheet04_pythonPV.pyc` (available at the moodle page) with PV being your python version (27,35,36,37) for the subsequent exercises.

[2]:
```
def cov_shrink(X):
    '''
    Estimate covariance of given data using shrinkage estimator.
    Synopsis:
    C= cov_shrink(X)
    Argument:
    X: data matrix (features x samples)
    Output:
```

```
    C: estimated covariance matrix
    '''
    Xc = X - np.mean(X, axis=1, keepdims=True)
    d, K = Xc.shape
    Cemp= Xc.dot(Xc.T)/(K-1)
    # memory saving way:
    sumVarCij = 0
    for ii in range(d):
        for jj in range(d):
            varCij = np.var(Xc[ii,:]*Xc[jj,:])
            sumVarCij += varCij

    nu = np.mean(np.diag(Cemp))
    gamma = K/(K-1.)**2 * sumVarCij / np.sum((Cemp-nu*np.eye(d,d))**2)
    S= (1-gamma)*Cemp + gamma*nu*np.eye(d,d)
    return S
```

1.2 Exercise 2: Implementation of LDA with Shrinkage (3 points)

Implement a function `train_LDAshrink` that calculates the LDA classifier in which the estimation of the covariance matrices is enhanced by shrinkage. Input and output should be the same as for `train_LDA` from sheet #03. As for LDA, use the pseudo inverse (`numpy.linalg.pinv`) instead of the usual matrix inversion.

If you cannot succeed with this task, you may import the function `train_LDAshrink` from `bci_helper_sheet04_pythonPV.pyc` (available at the moodle page) with PV being your python version (27,35,36,37) for the subsequent exercises.

[3]:
```
def train_LDAshrink(X, y):
    '''
    Synopsis:
        w, b= train_LDAshrink(X, y)
    Arguments:
        X: data matrix (features X samples)
        y: labels with values 0 and 1 (1 x samples)
    Output:
        w: LDA weight vector
        b: bias term
    '''
    mu1 = np.mean(X[:, y==0], axis=1)
    mu2 = np.mean(X[:, y==1], axis=1)
    # pool centered features to estimate covariance on samples of both classes␣
 ↪at once
    Xpool = np.concatenate((X[:, y==0]-mu1[:,np.newaxis], X[:, y==1]-mu2[:,np.
 ↪newaxis]), axis=1)
    C = cov_shrink(Xpool)
    w = np.linalg.pinv(C).dot(mu2-mu1)
    b = w.T.dot((mu1 + mu2) / 2.)
```

```
    return w, b
```

1.3 Preparation: Load data

[4]:
```
fname = 'erp_hexVPsag.npz'
cnt, fs, clab, mnt, mrk_pos, mrk_class, mrk_className = bci.load_data(fname)
```

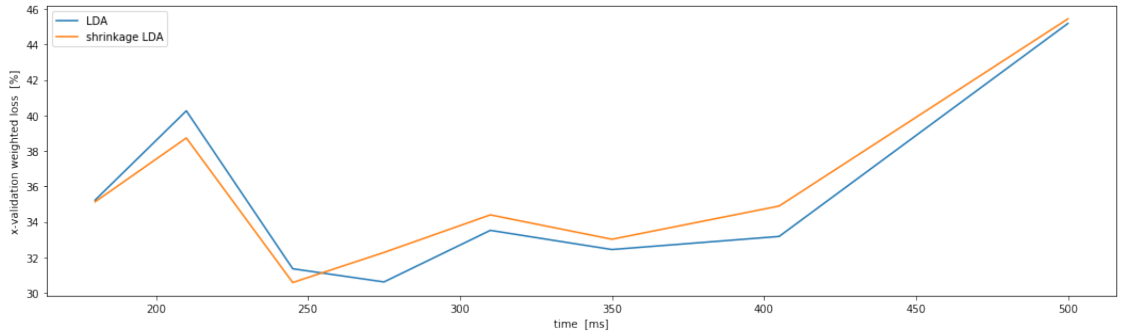
1.4 Exercise 3: Classification of Spatio-Temporal Features (5 points)

First, use the time-averaged potential in each of the intervals [ms]: 160-200, 200-220, 230-260, 260-290, 300-320, 330-370, 380-430, and 480-520 as feature vector (dimension 55 x 1) for each trial. For each interval calculate the 3-fold cross-validation error using LDA with and without shrinkage.

In a second step, concatenate the eight feature vectors, that were calcualted for the eight specified intervals into one feature vector (dimension 440 x 1) for each trial. Again, determine the 3-fold cross-validation error using LDA with and without shrinkage.

[5]:
```
ival_list = np.array([[160, 200], [200, 220], [230, 260], [260, 290], [300,␣
 ↪320], [330, 370], [380, 430], [480, 520]])
t_prestim = 100
t_end = np.max(ival_list)
epo, epo_t = bci.makeepochs(cnt, 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
print("Results on spatial features (separately for each a single component):")
fv= np.zeros((nIvals, nChans, nEpochs))
loss=np.zeros((len(ival_list)))
loss_shrink=np.zeros((len(ival_list)))
for k, ival in enumerate(ival_list):
    tidx = (ival[0] <= epo_t) & (epo_t <= ival[1])
    fv[k] = np.mean(epo[tidx, :, :], axis=0)
    loss[k], _=cfy.crossvalidation(cfy.train_LDA, fv[k], mrk_class, folds=3)
    loss_shrink[k], _=cfy.crossvalidation(train_LDAshrink, fv[k], mrk_class,␣
 ↪folds=3)
plt.figure(figsize=[18, 5])
plt.plot(np.mean(ival_list,axis=1), loss, label='LDA')
plt.plot(np.mean(ival_list,axis=1), loss_shrink, label='shrinkage LDA')
plt.xlabel('time  [ms]')
plt.ylabel('x-validation weighted loss  [%]')
plt.legend()
plt.show()
```

```
Results on spatial features (separately for each a single component):
```



[6]:
```
print("Results on spatio-temporal features:")
fvcat = np.vstack(fv)
cfy.crossvalidation(cfy.train_LDA, fvcat, mrk_class, folds=3, verbose=True)
cfy.crossvalidation(train_LDAshrink, fvcat, mrk_class, folds=3, verbose=True)
```

```
Results on spatio-temporal features:
 24.9 +/- 0.4  (training:  1.0 +/- 0.8)  [using train_LDA]
 16.2 +/- 3.4  (training:  2.9 +/- 0.5)  [using train_LDAshrink]
```

[6]:
```
(16.23462093039527, 2.904869937739422)
```