Cross-validation

One possibility to evaluate classifier performance is to use cross-validation procedures (description below). This is very useful when the amount of data for training the classifier model is limited (such as typically in BCI). You can use your own LDA code implemented in unit #2 or - if you have issues with your code - you can use the provided LDA code.

Exercise 1.1: 10-times 10-fold cross validation on artificial training data

Cross-validation procedure:

- Divide your artificial training data into ten different parts by random. Each part should contain class 1 and class 2 trials. Use Matlab `crossvalind` to generate the parts.
- Use 90% (nine parts) of your data for training and 10% (one part) for testing.
- Re-iterate until each of the parts was used for testing once.
- Repeat the procedure described above 10 times.
- Compute the average accuracy you get (average over 100 accuracies (10 x 10 folds)).

Task:

- Implement the cross-validation procedure described above
- Which accuracy do you get? Is the cross-validation accuracy similar to the accuracy computed with your test data?

Exercise 1.2: 10-times 10-fold cross validation on real AEP data

Task:

- Repeat Exercise 1 but use the real AEP data (unit #1 and #2)
- What accuracy do you get? Is the cross-validation accuracy similar to the accuracy computed with your test data?

Trial averaging

Exercise 2.1: Data preparation

EPs have smaller amplitudes compared to the ongoing spontaneous EEG. Since the ongoing EEG is not correlated with the Eps, averaging over trials will improve the classification accuracy of EP's, as the impact of the uncorrelated spontaneous EEG is reduced.

Task:

- Compute a new data sets by using the AEP data set and average over trials. Use different amounts of trials. Compute a set with 3, 5 and 10 averaged trials. Use sequential trials. Example for 3 averaged trials: New first trial consists of original trial 1, 2, 3; new second trial consists of original trial 4, 5, 6; etc. Do not mix classes during the averaging! The new amount of trials is reduced by the factor of averaging. Do not split into training and testing set.

Exercise 2.2: Cross-validation on averaged trials

Task:

- Compute a 10-times 10-fold cross-validation using the averaged trials (exercise 2.1).
- What is the cross-validation accuracy you get in comparison to the accuracies computed in the LDA exercise of unit #2?
- What is the cross-validation accuracy you get in comparison to the accuracies computed in the cross-validation exercise 1 above?

Feature extraction: Fourier transform - SSEP

Exercise 3.1: Implementation of a Discrete Fourier Transformation (DFT)

Exercise 3.1.1: Implement a custom DFT function (`custom_dft`) using the given equation:

$$\hat{a}_k = \sum_{j=0}^{N-1} e^{-2\pi i \cdot \frac{jk}{N}} \cdot a_j$$

with $\hat{a} = (\hat{a}_0,\ \hat{a}_1, \dots, \hat{a}_k, \dots, \hat{a}_{N-1})$ as the discrete Fourier transformed of the time signal $a = \left(a_0,\ a_1, \dots, a_j, \dots, a_{N-1}\right)$.

Task:

- Write the function `y = custom_dft(x)`. Where `x` is a time signal and `y` are the Fourier coefficients of `x`. (This function should transform a time signal into the frequency domain)

Exercise 3.1.2.: Implement a custom inverse DFT function (custom_idft()) using the given equation:

$$a_k = \frac{1}{N} \sum_{j=0}^{N-1} e^{2\pi i \cdot \frac{jk}{N}} \cdot \hat{a}_j$$

where $a = \left(a_0,\ a_1, \dots, a_j, \dots, a_{N-1}\right)$ is the time signal and $\hat{a} = (\hat{a}_0,\ \hat{a}_1, \dots, \hat{a}_k, \dots, \hat{a}_{N-1})$ are the discrete Fourier coefficient.

Task:

- Write the function `x = custom_idft(y)`. Where `y` are Fourier coefficients and `x` is the time signal built with this coefficients. (This function should transform a signal from the frequency domain back to the time domain)

Exercise 3.2: Build a test signal and test your custom_dft() and custom_idft() functions

Construct a test signal with a sampling rate of 100 Hz. The signal should consist of 3 summed sine waves (amplitudes: 1; frequency: 11 Hz, 27 Hz, 46 Hz; no phase delay; length of signal: 100 s).

Task:

- Use your `custom_dft` function to calculate the spectrum of the test signal.
- Use your `custom_idft` function to transform your signal back into the time domain.
- Document all your results (plots)

Exercise 3.3: Aliasing

Down sample your test signal by a factor of two (use the MATLAB function `downsample`).

Task:
- Use your `custom_dft` function to calculate the spectrum of the down sampled signal.
- Use your `custom_idft` function to transform your signal back into the time domain.
- What do you notice in frequency and time domain? Document your results (plots).

Exercise 3.4: Anti-Aliasing filter

Design a FIR filter of order 25 with a cut off frequency at 40 Hz. Use MATLAB function `fir1`. Filter your test signal with the MATLAB function `filter`. Down sample your test signal by a factor of two (use the MATLAB function `downsample`).

Task:
- Use your `custom_dft` function to calculate the spectrum of the down sampled signal.
- Use your `custom_idft` function to transform your signal back into the time domain.
- What do you notice in frequency and time domain? Document your results (plots).

Exercise 3.5: Resample

Use the MATLAB function `resample` to perform the down sampling.

Task:
- Do you notice aliasing?
- What does the `resample` function do?

Pre-submission via TeachCenter at the end of the unit; final submission 21.04.2019