practical_exercise_8 , Methods 3, 2021, autumn semester

Rikke Uldbæk

December 1st 2021

```
#packages
pacman::p_load(reticulate, Rcpp)

#set wd
setwd("~/Desktop/Cognitive Science/3rd semester/Methods 3/github_methods_3/week_08")
```

Exercises and objectives

1) Load the magnetoencephalographic recordings and do some initial plots to understand the data

2) Do logistic regression to classify pairs of PAS-ratings

3) Do a Support Vector Machine Classification on all four PAS-ratings

REMEMBER: In your report, make sure to include code that can reproduce the answers requested in the exercises below (**MAKE A KNITTED VERSION**)
REMEMBER: This is Assignment 3 and will be part of your final portfolio

EXERCISE 1 - Load the magnetoencephalographic recordings and do some initial plots to understand the data

The files `megmag_data.npy` and `pas_vector.npy` can be downloaded here (http://laumollerandersen.org/data_methods_3/megmag_data.npy) and here (http://laumollerandersen.org/data_methods_3/pas_vector.npy)

1) Load `megmag_data.npy` and call it `data` using `np.load`. You can use `join`, which can be imported from `os.path`, to create paths from different string segments

```
#packages in python
import numpy as np
import pandas as pd
import matplotlib as plt
plt.rcParams['figure.dpi'] = 300 # HIGH DPI PLOTS PLEASE

data = np.load("megmag_data.npy")
data.shape #inspect the data (number of dimensions = 3)
#we have 682 = number of repetitions of a visual stimulus, 102 = the second dimension is the number of
```

```
## (682, 102, 251)
```

i. The data is a 3-dimensional array. The first dimension is number of repetitions of a visual stimulus , the second dimension is the number of sensors that record magnetic fields (in Tesla) that stem from neurons activating in the brain, and the third dimension is the number of time samples. How many repetitions, sensors and time samples are there?

ii. The time range is from (and including) -200 ms to (and including) 800 ms with a sample recorded every 4 ms. At time 0, the visual stimulus was briefly presented. Create a 1-dimensional array called `times` that represents this.

```
#make a vector fra -200 til 800, den har en værdi hvert 4. milisekund
#dvs et spænd på 1000 (fra -200 til 800, med en optagelse hver 4 sek = )
times = np.arange(-200, 801, 4)
```

iii. Create the sensor covariance matrix Σ_{XX}:

$$\Sigma_{XX} = \frac{1}{N}\sum_{i=1}^{N} XX^T$$

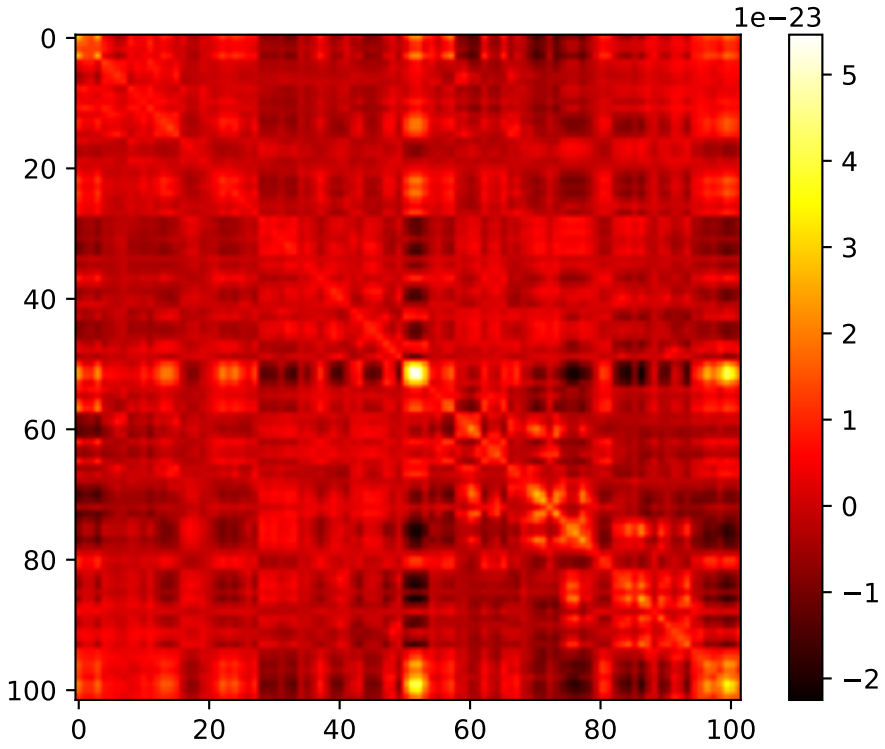
N is the number of repetitions and X has s rows and t columns (sensors and time), thus the shape is $X_{s\times t}$. Do the sensors pick up independent signals? (Use `plt.imshow` to plot the sensor covariance matrix)

We see some covariance sporadically distributed in the sensor covariance matrix, this suggests that they do not really pick up independent signals.

```
#     iii. Create the sensor covariance matrix
#calculating the dot product for all rows i using all datapoints in the dimensions
covariance_matrix = []
n = 682

for i in range(n):
  covariance_matrix.append(data[i,:,:] @ data[i,:,:].T)

#out of the loop the dot product of the the matrices for each i is summed and divided by n.
covariance_matrix = sum(covariance_matrix)/n

#plotting the covariance matrix
import matplotlib.pyplot as plt

plt.figure()
plt.imshow(covariance_matrix,cmap = "hot")
plt.colorbar()
```

```
## <matplotlib.colorbar.Colorbar object at 0x1644d74c0>
```

```
plt.show()
```



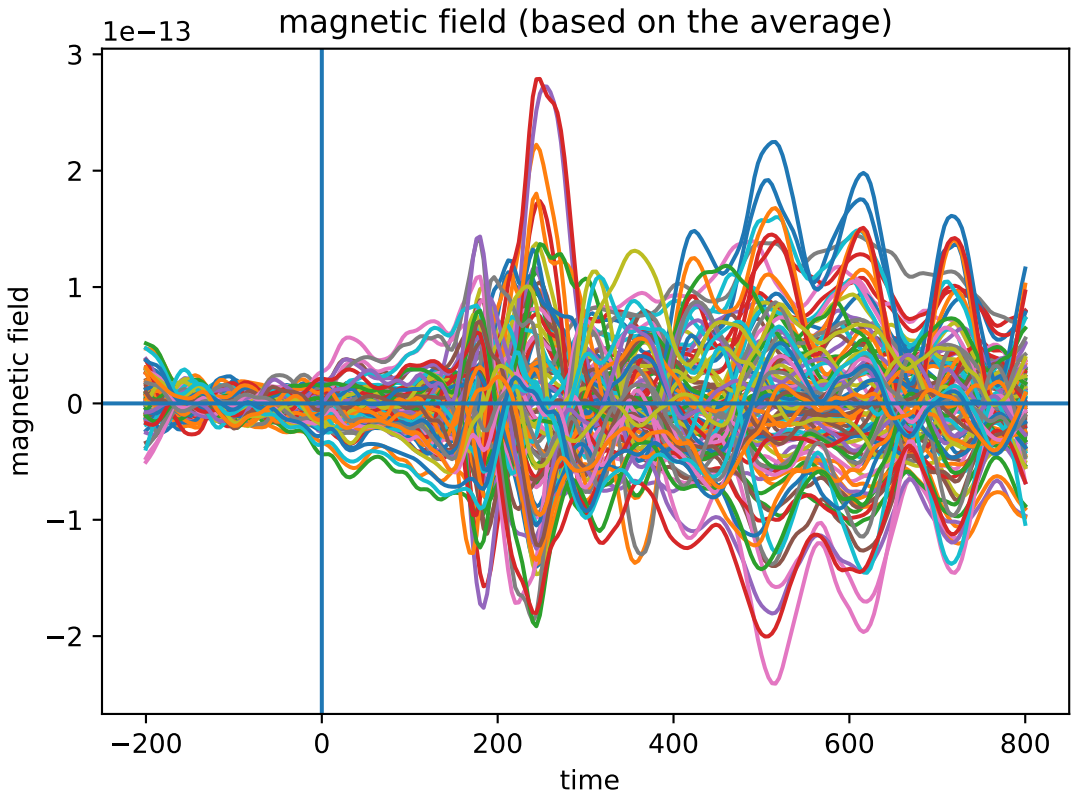
iv. Make an average over the repetition dimension using `np.mean` - use the `axis` argument. (The resulting array should have two dimensions with time as the first and magnetic field as the second)

```
avr_rep = np.mean(data, axis=0) # repetition of visual stimulus :))
avr_rep.shape #inspect, vi har 102 elementer (avrg repetition across sensors) i col 1 og 251 elementer
```

```
## (102, 251)
```

v. Plot the magnetic field (based on the average) as it evolves over time for each of the sensors (a line for each) (time on the x-axis and magnetic field on the y-axis). Add a horizontal line at $y = 0$ and a vertical line at $x = 0$ using `plt.axvline` and `plt.axhline`

```
import matplotlib.pyplot as plt

#Plot
plt.figure()
plt.plot(times, avr_rep.T)
plt.axvline()
plt.axhline()
plt.xlabel(" time")
plt.ylabel("magnetic field")
plt.title("magnetic field (based on the average)")
plt.show()
```



vi. Find the maximal magnetic field in the average. Then use `np.argmax` (Returns the indices of the maximum values along an axis.) and `np.unravel_index` to find the sensor that has the maximal magnetic field.

```
#the maximal magnetic field in the average
maxiking = np.unravel_index(np.argmax(avr_rep), avr_rep.shape)
print(maxiking) # sensor nr 73 at repetition 112 got the maximum magnetic field
```

```
## (73, 112)
```

```
print(avr_rep[73,112]) #2.7886216843591933e-13 # is the same as below (SAME)
```

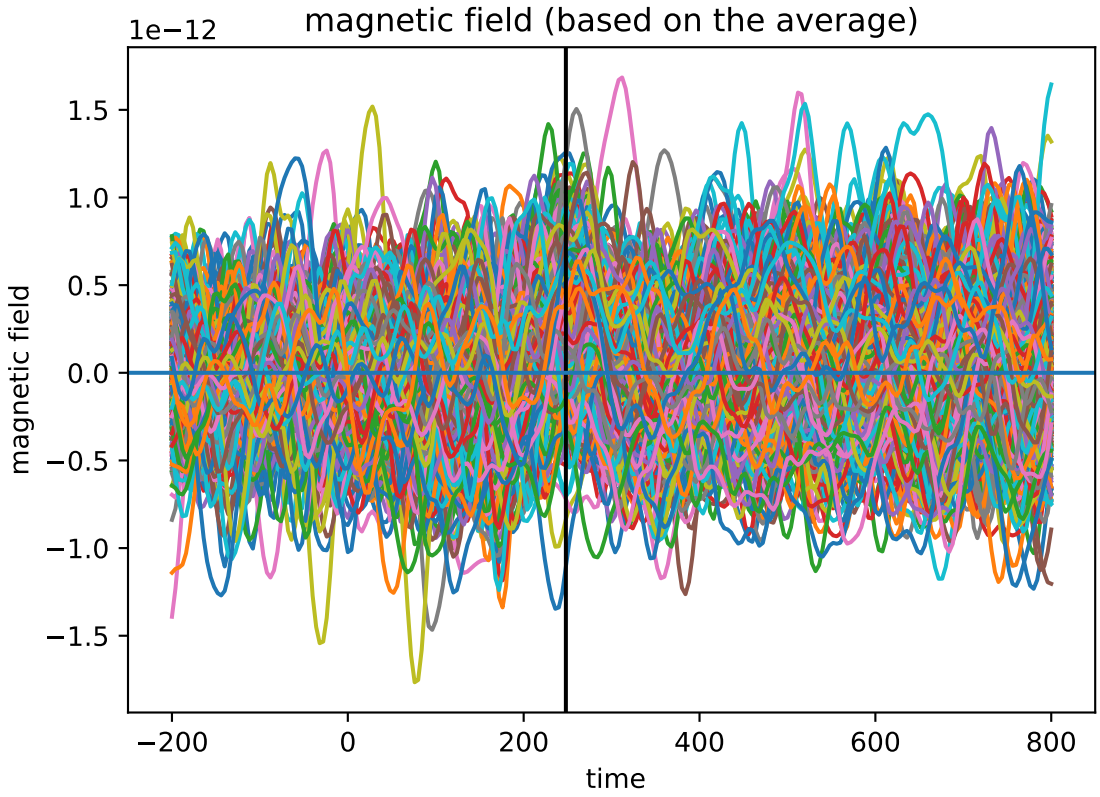
```
## 2.7886216843591933e-13
```

```
np.amax(avr_rep) #2.7886216843591933e-13 # the maximum value in the average (SAME)
```

```
## 2.7886216843591933e-13
```

vii. Plot the magnetic field for each of the repetitions (a line for each) for the sensor that has the maximal magnetic field (sensor 73). Highlight the time point with the maximal magnetic field in the average (as found in 1.1.v) using `plt.axvline`

```
#Plot the magnetic field for each of the repetitions (a line for each) for the sensor that has the maxi
plt.figure()
plt.plot(times, data[:,73,:].T)
plt.axvline((times[112]), color= "black") #Highlight the time point with the maximal magnetic field in
plt.axhline()
plt.xlabel(" time")
plt.ylabel("magnetic field")
plt.title("magnetic field (based on the average)")
plt.show()


#time in ms
```




```
times[112]
```

```
## 248
```

viii. Describe in your own words how the response found in the average is represented in the single repetitions. But do make sure to use the concepts *signal* and *noise* and comment on any differences on the range of values on the y-axis

In the plot above we see each single repetition of visual stimuli (682 repetitions) for sensor 73. The individual repetitions contains a lot of noise (I assume that the noise is the squiggles, but honestly I don't know much about MEG data). The response found in the average plot (2.1 v (Plot the magnetic field (based on the

average))) where we see a large peak around time 112/248ms (marked with a black line), is also visualized in the plot above where we see a little concave curve around time 112/248ms as well.I would say the plot in exercise 2.1 v (Plot the magnetic field (based on the average)) is more informative since the signals are based on an average per sensor, which makes it easier to distinguish the between them.

2) Now load `pas_vector.npy` (call it `y`). PAS is the same as in Assignment 2, describing the clarity of the subjective experience the subject reported after seeing the briefly presented stimulus

i. Which dimension in the `data` array does it have the same length as?

The new y value we just imported, which reflects PAS-ratings (perceptual awareness scale), has the same length as number of repetitions of a visual stimulus. Meaning that for each visual stimulus there is a PAS-rating.

```
#load data
y = np.load("pas_vector.npy")
len(y)
```

```
## 682
```

ii. Now make four averages (As in Exercise 1.1.iii), one for each PAS rating, and plot the four time courses (one for each PAS rating) for the sensor found in Exercise 1.1.v

```
#subset sensor 73
sensor_73 = data[:,73,:]

len(y)
# Now make four averages (As in Exercise 1.1.iii), one for each PAS rating
#PAS 1
```

```
## 682
```

```
PAS_1= np.where(y == 1) #finding out where pas ratings are
avg_rep_PAS_1 = np.mean(sensor_73[PAS_1], axis= 0)
sensor_73[PAS_1].shape #inspect - we've got 99 pas 1 ratings

#PAS 2
```

```
## (99, 251)
```

```
PAS_2= np.where(y == 2) #finding out where pas ratings are
avg_rep_PAS_2 = np.mean(sensor_73[PAS_2], axis= 0)
sensor_73[PAS_2].shape #inspect - we've got 115 pas 2 ratings


#PAS 3
```

```
## (115, 251)
```

```
PAS_3= np.where(y == 3) #finding out where pas ratings are
avg_rep_PAS_3 = np.mean(sensor_73[PAS_3], axis= 0)
sensor_73[PAS_3].shape #inspect - we've got 208 pas 3 ratings

#PAS 4
```

```
## (208, 251)
```

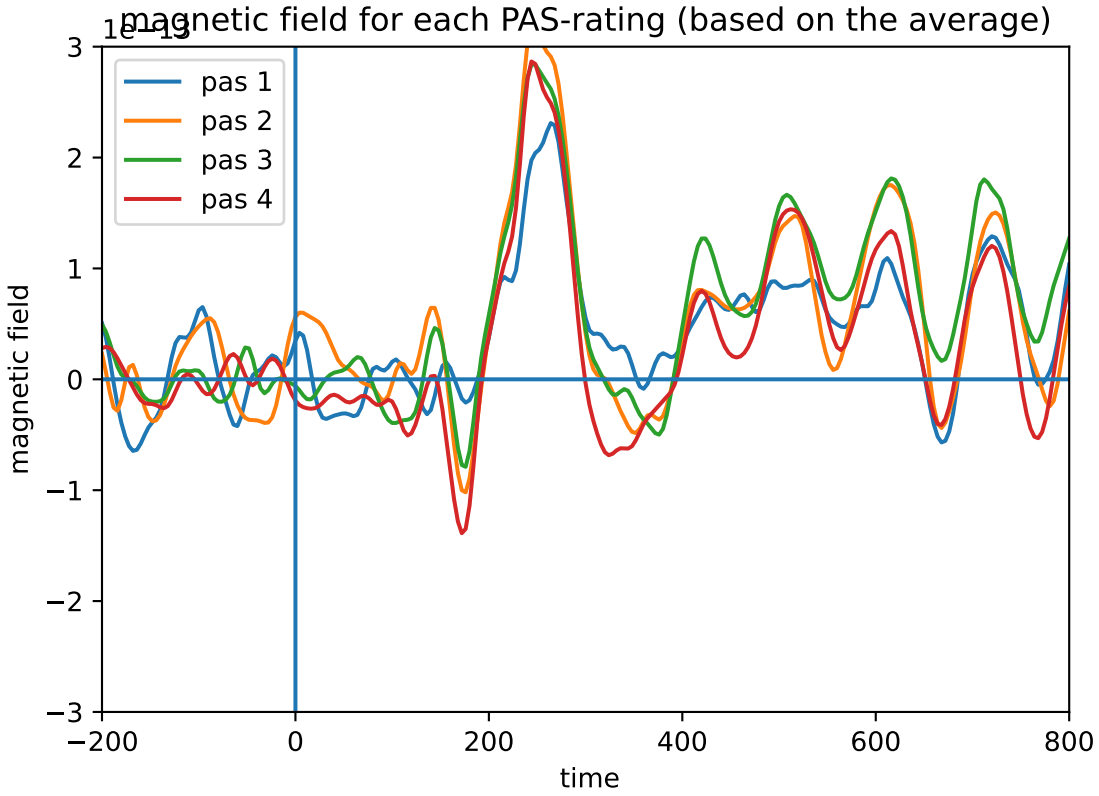
```
PAS_4= np.where(y == 4) #finding out where pas ratings are
avg_rep_PAS_4 = np.mean(sensor_73[PAS_4], axis= 0)
sensor_73[PAS_4].shape #inspect - we've got 260 pas 4 ratings


#Plot the four time courses (one for each PAS rating) for the sensor found in Exercise 1.1.v
#plot
```

```
## (260, 251)
```

```
plt.figure()
plt.plot(times, avg_rep_PAS_1)
plt.plot(times, avg_rep_PAS_2)
plt.plot(times, avg_rep_PAS_3)
plt.plot(times, avg_rep_PAS_4)
plt.axis([-200,800,-3e-13,3e-13])
```

```
## (-200.0, 800.0, -3e-13, 3e-13)
```

```
plt.axvline()
plt.axhline()
plt.xlabel(" time")
plt.ylabel("magnetic field")
plt.title("magnetic field for each PAS-rating (based on the average)")
plt.legend(["pas 1", "pas 2", "pas 3", "pas 4"])
plt.show()
```



iii. Notice that there are two early peaks (measuring visual activity from the brain), one before 200 ms and one around 250 ms. Describe how the amplitudes of responses are related to the four PAS-scores. Does PAS 2 behave differently than expected?

PAS 2 does behave a little differently than the other PAS ratings, since it deviates a little more than the others, at least from 0 ms to around 250 ms. We see PAS 1 as being quite lower than the other PAS ratings throughout the time interval, which would make sense as it represents "No Experience", thus we could expect PAS 2 ("Weak Glimpse") to be similarly low or only a little higher, but this does not seem to be the case (although it varies a lot).

EXERCISE 2 - Do logistic regression to classify pairs of PAS-ratings

1) Now, we are going to do Logistic Regression with the aim of classifying the PAS-rating given by the subject

   i. We'll start with a binary problem - create a new array called `data_1_2` that only contains PAS responses 1 and 2. Similarly, create a `y_1_2` for the target vector

```
#create a new array called `data_1_2`
data_1_2 = np.concatenate((data[PAS_1], data[PAS_2]), axis=0) # 3D
data_1_2.shape
```

```
## (214, 102, 251)
```

```
data_1_2.ndim

#create a `y_1_2` for the target vector
```

```
## 3
```

```
y_1_2 = []
for i in range(len(y)):
    if y[i] == 1:
        y_1_2.append(1) #append real numbers and not indices (if u put in 'i' then they append the numb
    if y[i] == 2:
        y_1_2.append(2) #append real numbers and not indices (if u put in 'i' then they append the numb
```

ii. Scikit-learn expects our observations (`data_1_2`) to be in a 2d-array, which has samples (repetitions) on dimension 1 and features (predictor variables) on dimension 2. Our `data_1_2` is a three-dimensional array. Our strategy will be to collapse our two last dimensions (sensors and time) into one dimension, while keeping the first dimension as it is (repetitions). Use `np.reshape` to create a variable `X_1_2` that fulfils these criteria.

```
#collapse our two last dimensions (sensors and time) into one dimension, while keeping the first dimens

#dim 1: repetition[dim 0]
# dim 2: sensor & time [dim 1-2]
X_1_2= data_1_2.reshape(data_1_2.shape[0],(data_1_2.shape[1]*data_1_2.shape[2]))
X_1_2.shape  #we actually just reduced dim 1 & 2 together by multiplying them :))
```

```
## (214, 25602)
```

iii. Import the `StandardScaler` and scale `X_1_2`

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_X_1_2 = scaler.fit_transform(X_1_2)
scaled_X_1_2.shape
```

```
## (214, 25602)
```

iv. Do a standard `LogisticRegression` - can be imported from `sklearn.linear_model` - make sure there is no `penalty` applied (DEFAULT IS 'L2')

v. Use the `score` method of `LogisticRegression` to find out how many labels were classified correctly. Are we overfitting? Besides the score, what would make you suspect that we are overfitting?

We could suspect that we are overfitting, because the score is perfect with an prediction accuracy of 100%. Also, we test the model on the data we fitted it with. In order to evaluate the model, we would have to see how good then model predicts new data.

```
# Do a standard `LogisticRegression`
from sklearn.linear_model import LogisticRegression #importing
logR = LogisticRegression(penalty ='none') #specifying stuff in model
logR.fit(scaled_X_1_2, y_1_2) #actually fitting the model

#Use the `score` method of `LogisticRegression` to find out how many labels were classified correctly.
```

```
## LogisticRegression(penalty='none')
```

```
logR.score(scaled_X_1_2, y_1_2) # 1.0
```

```
## 1.0
```

vi. Now apply the *L1* penalty instead - how many of the coefficients (`.coef_`) are non-zero after this?

After applying L1 penalty 217 of my coefficients are non-zero.

```
# Do a standard `LogisticRegression` - this time use penalty 'L1' (LASSO)
from sklearn.linear_model import LogisticRegression #importing
logR_L1 = LogisticRegression(penalty ='l1', solver = "liblinear", random_state= 1) #specifying stuff in
logR_L1.fit(scaled_X_1_2, y_1_2)  #actually fitting the model

#Use the `score` method of `LogisticRegression` to find out how many labels were classified correctly.
```

```
## LogisticRegression(penalty='l1', random_state=1, solver='liblinear')
```

```
logR_L1.score(scaled_X_1_2, y_1_2) # still 1.0

#coefs being 0?
```

```
## 1.0
```

```
print(np.sum(logR_L1.coef_ !=0))
```

```
## 217
```

vii. Create a new reduced X that only includes the non-zero coefficients - show the covariance of the non-zero features (two covariance matrices can be made; $X_{reduced}X_{reduced}^T$ or $X_{reduced}^T X_{reduced}$ (you choose the right one)) . Plot the covariance of the features using `plt.imshow`. Compared to the plot from 1.1.iii, do we see less covariance?

It looks like there is bit more covariance in this covariance matrix compared to the plot from 1.1.iii. (?)

```
#### Create a new reduced $X$ that only includes the non-zero coefficients
coefs = logR_L1.coef_.flatten() #makes 2D to list
non_zero = coefs!= 0 #isolate coefficients that are  non-zero
X_reduced = scaled_X_1_2[:,non_zero] #making the reduced X
X_reduced.shape
```

```
## (214, 217)
```

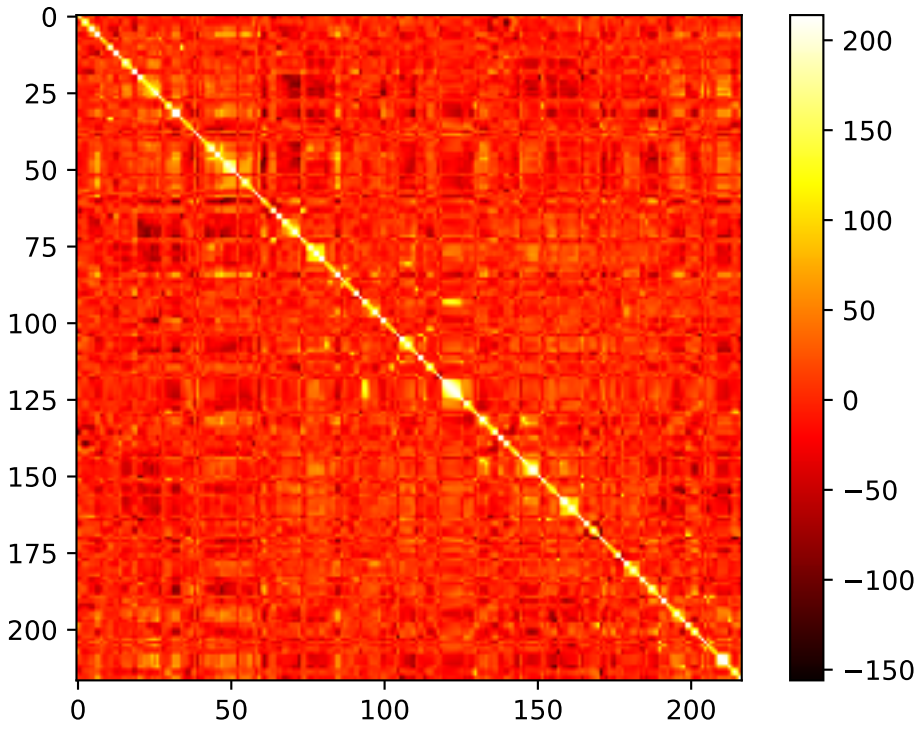
```
covariance_matrix_1 = X_reduced.T @ X_reduced # matrix multiplication
covariance_matrix_1.shape #now its (217,217)

#plot the covariance_matrix_1
```

```
## (217, 217)
```

```
import matplotlib.pyplot as plt
plt.figure()
plt.imshow(covariance_matrix_1, cmap = "hot")
plt.colorbar()
```

```
## <matplotlib.colorbar.Colorbar object at 0x1644e0130>
```

```
plt.show()
```



2) Now, we are going to build better (more predictive) models by using cross-validation as an outcome measure

   i. Import `cross_val_score` and `StratifiedKFold` from `sklearn.model_selection`

```
# importing functions
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
```

   ii. To make sure that our training data sets are not biased to one target (PAS) or the other, create `y_1_2_equal`, which should have an equal number of each target (i.e same number of PAS 1 ratings and PAS 2 ratings, and not 99 and 115). Create a similar `X_1_2_equal`. The function `equalize_targets_binary` in the code chunk associated with Exercise 2.2.ii can be used. Remember to scale `X_1_2_equal`!

```
# Create `y_1_2_equal`, which should have an equal number of each target
#vi skal have 107 PAS 1 og 107 PAS 2
y_1_2= np.array(y_1_2) #make y_1_2 into an array in order to make the function below run

# Exercise 2.2.ii
def equalize_targets_binary(data, y):
    np.random.seed(7)
    targets = np.unique(y) ## find the number of targets
    if len(targets) > 2:
        raise NameError("can't have more than two targets")
    counts = list()
    indices = list()
    for target in targets:
        counts.append(np.sum(y == target)) ## find the number of each target
        indices.append(np.where(y == target)[0]) ## find their indices
    min_count = np.min(counts)
    # randomly choose trials
    first_choice = np.random.choice(indices[0], size=min_count, replace=False)
    second_choice = np.random.choice(indices[1], size=min_count,replace=False)

    # create the new data sets
    new_indices = np.concatenate((first_choice, second_choice))
    new_y = y[new_indices]
    new_data = data[new_indices, :, :]

    return new_data, new_y


data_1_2_equal, y_1_2_equal = equalize_targets_binary(data_1_2, y_1_2) #make that code spinnnnn
y_1_2_equal.shape #check that dope data out (1 dim)
```

```
## (198,)
```

```
data_1_2_equal.shape #check that dope data out (3 dim)

#rehape data_1_2_equa from 3D to X_1_2_equal 2D
```

```
## (198, 102, 251)
```

```
X_1_2_equal= data_1_2_equal.reshape(198,-1) # now it's 2D (doing -1 takes everything else than )
X_1_2_equal.ndim

#Scale X_1_2_equal
```

```
## 2
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_X_1_2_equal = scaler.fit_transform(X_1_2_equal)
scaled_X_1_2_equal.shape
```

```
## (198, 25602)
```

iii. Do cross-validation with 5 stratified folds doing standard `LogisticRegression` (See Exercise 2.1.iv)

```
#first make standard logistic regression
from sklearn.linear_model import LogisticRegression #importing
logR_standard = LogisticRegression(penalty ='none') #specifying stuff in model
logR_standard.fit(scaled_X_1_2_equal, y_1_2_equal) #actually fitting the model

#import packages
```

```
## LogisticRegression(penalty='none')
```

```
from sklearn.model_selection import cross_val_score, StratifiedKFold
cv = StratifiedKFold(n_splits=5) #abbreviation for simplicity and defining number of k-folds (default=5)

scaled_X_1_2_equal.shape # our X (standardized)
```

```
## (198, 25602)
```

```
y_1_2_equal.shape # our y
```

```
## (198,)
```

```
scores = cross_val_score(logR_standard, scaled_X_1_2_equal, y_1_2_equal, cv=cv) #cv = StratiffiedKfold
print(np.mean(scores)) #0.4746153846153846
```

```
## 0.4746153846153846
```

iv. Do L2-regularisation with the following `Cs= [1e5, 1e1, 1e-5]`. Use the same kind of cross-validation as in Exercise 2.2.iii. In the best-scoring of these models, how many more/fewer predictions are correct (on average)?

The best model has a C = 1e-5.

```
################ L2 with C=1e5 (10.0000)
logR_L2_1 = LogisticRegression(C=1e5, penalty ='l2', solver = "liblinear", random_state= 1) #specifying
logR_L2_1.fit(scaled_X_1_2_equal, y_1_2_equal)  #actually fitting the model

#Cross val score
```

```
## LogisticRegression(C=100000.0, random_state=1, solver='liblinear')
```

```
scores_L2_1 = cross_val_score(logR_L2_1, scaled_X_1_2_equal, y_1_2_equal, cv=5) #cv = 5 is 5 folds
print(np.mean(scores_L2_1)) #0.46987179487179487
```

```
## 0.46987179487179487
```

```
scores_L2_1 # 5 values cuz we cross validate 5 times



###############  L2 with C=1e1 (10)
```

```
## array([0.475     , 0.4       , 0.5       , 0.46153846, 0.51282051])
```

```
logR_L2_2 = LogisticRegression(C=1e1, penalty ='l2', solver = "liblinear", random_state= 1) #specifying
logR_L2_2.fit(scaled_X_1_2_equal, y_1_2_equal)  #actually fitting the model

#Cross val score
```

```
## LogisticRegression(C=10.0, random_state=1, solver='liblinear')
```

```
scores_L2_2 = cross_val_score(logR_L2_2, scaled_X_1_2_equal, y_1_2_equal, cv=5) #cv = 5 is 5 folds
print(np.mean(scores_L2_2)) #0.47000000000000003
```

```
## 0.47000000000000003
```

```
scores_L2_2 # 5 values cuz we cross validate 5 times



################  L2 with C=1e-5 (0.00001)
```

```
## array([0.475     , 0.375     , 0.5       , 0.43589744, 0.56410256])
```

```
logR_L2_3 = LogisticRegression(C=1e-5, penalty ='l2', solver = "liblinear", random_state= 1) #specifyin
logR_L2_3.fit(scaled_X_1_2_equal, y_1_2_equal)  #actually fitting the model

#Cross val score
```

```
## LogisticRegression(C=1e-05, random_state=1, solver='liblinear')
```

```
scores_L2_3 = cross_val_score(logR_L2_3, scaled_X_1_2_equal, y_1_2_equal, cv=5) #cv = 5 is 5 folds
print(np.mean(scores_L2_3)) #0.4805128205128205
```

```
## 0.4805128205128205
```

```
scores_L2_3 # 5 values cuz we cross validate 5 times


#In the best-scoring of these models, how many more/fewer predictions are correct (on average)?
#best score model = L2 with C=1e-5 (0.00001)
```

```
## array([0.45      , 0.35      , 0.5       , 0.58974359, 0.51282051])
```

```
print("On average the best model predicts with a %0.2f accuracy with a standard deviation of %0.2f" % (s
```

```
## On average the best model predicts with a 0.48 accuracy with a standard deviation of 0.08
```

v. Instead of fitting a model on all `n_sensors * n_samples` (time samples) features, fit a logistic regression (same kind as in Exercise 2.2.iv (use the `C` that resulted in the best prediction)) for **each** time sample and use the same cross-validation as in Exercise 2.2.iii. What are the time points where classification is best? Make a plot with time on the x-axis and classification score on the y-axis with a horizontal line at the chance level (what is the chance level for this analysis?)

When fitting a logistic regression on each time sample, I picked the model using L2 regularization with C=1e-5 (0.00001), which performed best out of all with an accuracy level of 0.48. After performing cross-validation using the aforementioned model on each time sample, the greatest accuracy level reached 0.58, and this accuracy level happens at 408ms (time point of 152), as visualized in the plot. The chance level for this analysis 50%, with two options being PAS 1 and PAS 2(with an equal amount of observations in each).

```
###############  L2 with C=1e-5 (0.00001) is the best model
cv_scores= [] #make an empty list
log_loop= LogisticRegression(C=1e-5, penalty ='l2', solver = "liblinear", random_state= 1) #best model

#logistic regression on 1 time sample
for i in range(251):
  transformed_data = scaler.fit_transform(data_1_2_equal[:,:,i]) # transform the data first
  log_loop.fit(transformed_data, y_1_2_equal) #loop and fit through the transformed data
  scorez= cross_val_score(log_loop, transformed_data, y_1_2_equal, cv=5) #calculate cross validation sc
  cv_scores.append(np.mean(scorez)) #append to the list 'cv_scores'
```

```
#the maximal cross validation score from the logistic regression loop :)
np.amax(cv_scores) #0.5752564102564103


# find the index place of the max time point 0.5752564102564103
```

```
## 0.5752564102564103
```

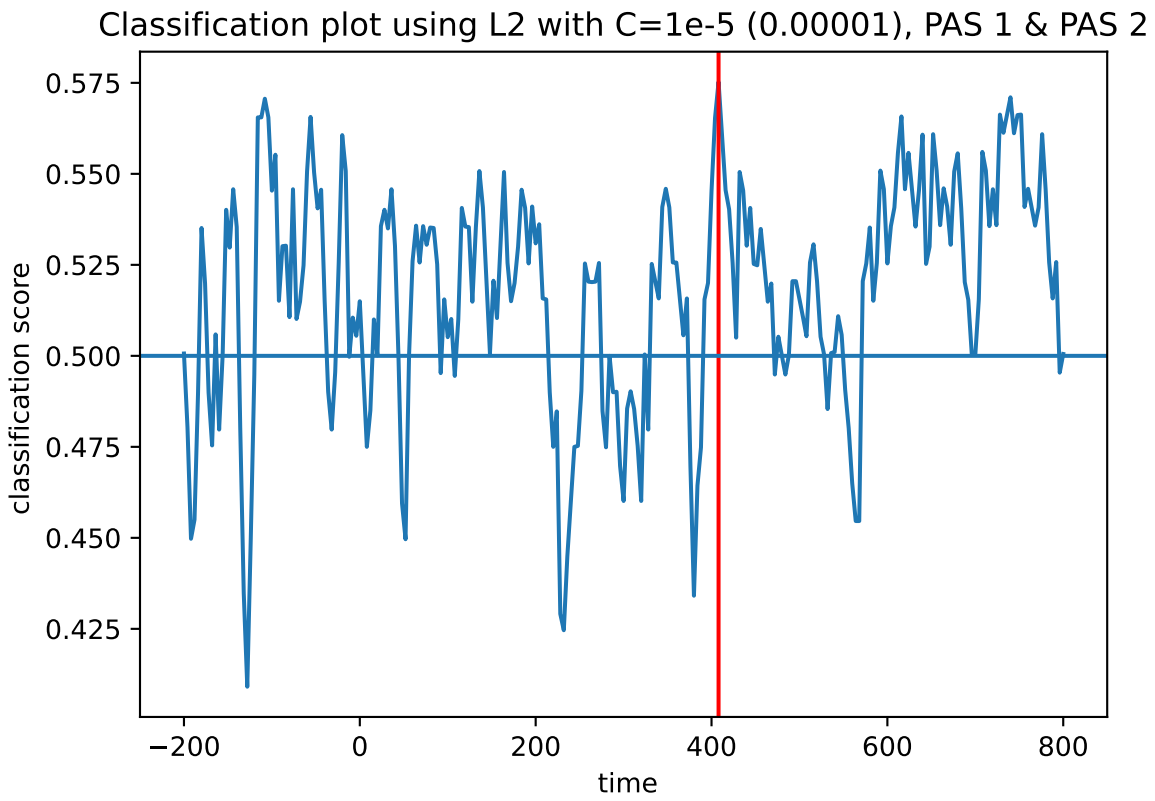
```
np.argmax(cv_scores) # index for maksimale punkt = 152
```

```
## 152
```

```
times[152]


#plot time on the x-axis and classification score on the y-axis with a horizontal line at the chance le
```

```
## 408
```

```
plt.figure()
plt.plot(times, cv_scores)
plt.axvline((times[152]), color= "red") #Highlight of time point where classification is best
plt.axhline(0.5) #chance level of 0.5, since its binary
plt.xlabel(" time")
plt.ylabel("classification score ")
plt.title("Classification plot using L2 with C=1e-5 (0.00001), PAS 1 & PAS 2")
plt.show()
```



vi. Now do the same, but with L1 regression - set `C=1e-1` - what are the time points when classification is best? (make a plot)?

The time point where the classification using L1 with C=1e-1 is best is at -56 ms (time point 36), as visualized in the plot.

```
###############  L1 with C=1e-1 (0.1)
cv_scores_l1= [] #make an empty list
log_loop_l1= LogisticRegression(C=1e-1, penalty ='l1', solver = "liblinear", random_state= 1) #best mod

#logistic regression on 1 time sample
for i in range(251):
  transformed_data_l1 = scaler.fit_transform(data_1_2_equal[:,:,i]) # transform the data first
  log_loop_l1.fit(transformed_data_l1, y_1_2_equal) #loop and fit through the transformed data
  scorez_l1= cross_val_score(log_loop_l1, transformed_data_l1, y_1_2_equal, cv=5) #calculate cross vali
  cv_scores_l1.append(np.mean(scorez_l1)) #append to the list 'cv_scores'
```

```
#the maximal cross validation score from the logistic regression loop :)
np.amax(cv_scores_l1) #0.6161538461538462


# find the index place of the max time point 0.6161538461538462
```

```
## 0.6161538461538462
```

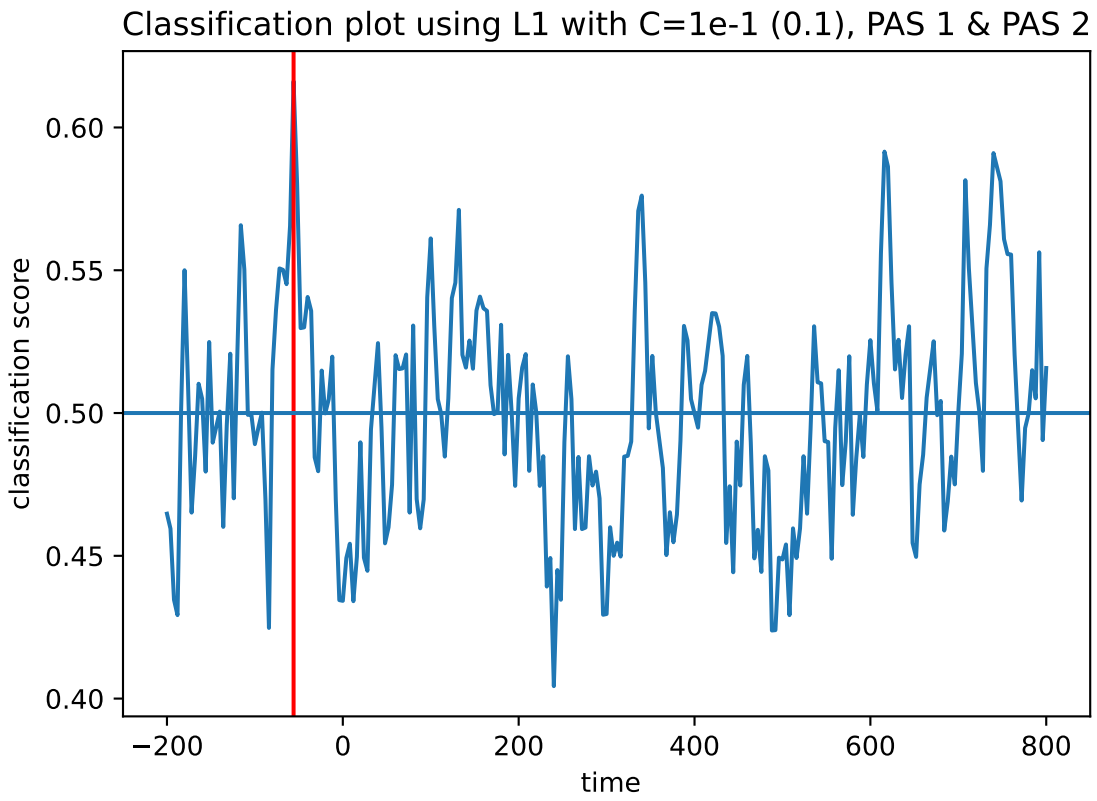
```
np.argmax(cv_scores_l1) # index for maksimale punkt = 36
```

```
## 36
```

```
times[36]


#plot time on the x-axis and classification score on the y-axis with a horizontal line at the chance le
```

```
## -56
```

```
plt.figure()
plt.plot(times, cv_scores_l1)
plt.axvline((times[36]), color= "red") #Highlight of time point where classification is best
plt.axhline(0.5) #chance level of 0.5, since its binary
plt.xlabel(" time")
plt.ylabel("classification score")
plt.title("Classification plot using L1 with C=1e-1 (0.1), PAS 1 & PAS 2")
plt.show()
```



vii. Finally, fit the same models as in Exercise 2.2.vi but now for `data_1_4` and `y_1_4` (create a data set and a target vector that only contains PAS responses 1 and 4). What are the time points when classification is best? Make a plot with time on the x-axis and classification score on the y-axis with a horizontal line at the chance level (what is the chance level for this analysis?)

When fitting logistic regression on each time sample from PAS 1 and PAS 4 ratings with the model using L1 regularization with C=1e-1 (0.1), the classification is best at 112 ms (time point 78). The chance level for this analysis 50%, with two options being PAS 1 and PAS 4 ratings (with an equal amount of observations in each).

```
# create a data set and a target vector that only contains PAS responses 1 and 4 (`data_1_4` and `y_1_4
data_1_4 = np.concatenate((data[PAS_1], data[PAS_4]), axis=0) # 3D
data_1_4.shape
```

```
## (359, 102, 251)
```

```
data_1_4.ndim
```

```
## 3
```

```
y_1_4 = []
for i in range(len(y)):
    if y[i] == 1:
        y_1_4.append(1) #append real numbers and not indices (if u put in 'i' then they append the numb
    if y[i] == 2:
        y_1_4.append(4) #append real numbers and not indices (if u put in 'i' then they append the numb

y_1_4= np.array(y_1_4) #make y_1_4 into an array in order to make the function below run

#### equalize
# Exercise 2.2.ii
def equalize_targets_binary(data, y):
    np.random.seed(7)
    targets = np.unique(y) ## find the number of targets
    if len(targets) > 2:
        raise NameError("can't have more than two targets")
    counts = list()
    indices = list()
    for target in targets:
        counts.append(np.sum(y == target)) ## find the number of each target
        indices.append(np.where(y == target)[0]) ## find their indices
    min_count = np.min(counts)
    # randomly choose trials
    first_choice = np.random.choice(indices[0], size=min_count, replace=False)
    second_choice = np.random.choice(indices[1], size=min_count,replace=False)

    # create the new data sets
    new_indices = np.concatenate((first_choice, second_choice))
    new_y = y[new_indices]
    new_data = data[new_indices, :, :]

    return new_data, new_y


data_1_4_equal, y_1_4_equal = equalize_targets_binary(data_1_4, y_1_4) #make that code spinnnnn
y_1_4_equal.shape #check that dope data out (1 dim)
```

```
## (198,)
```

```
data_1_4_equal.shape #check that dope data out (3 dim)

#rehape data_1_2_equa from 3D to X_1_2_equal 2D
```

```
## (198, 102, 251)
```

```
X_1_4_equal= data_1_4_equal.reshape(198,-1) # now it's 2D (doing -1 takes everything else than )
X_1_4_equal.ndim

#Scale X_1_2_equal
```

```
## 2
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_X_1_4_equal = scaler.fit_transform(X_1_4_equal)
scaled_X_1_4_equal.shape
```

```
## (198, 25602)
```

```
# fit the same models as in Exercise 2.2.vi but now for `data_1_4` and `y_1_4`
###############  L1 with C=1e-1 (0.1)
cv_scores_1_4= [] #make an empty list
log_loop_1_4= LogisticRegression(C=1e-1, penalty ='l1', solver = "liblinear", random_state= 1) #best mo

#logistic regression on 1 time sample
for i in range(251):
  transformed_data_1_4 = scaler.fit_transform(data_1_4_equal[:,:,i]) # transform the data first
  log_loop_1_4.fit(transformed_data_1_4, y_1_4_equal) #loop and fit through the transformed data
  scorez_1_4= cross_val_score(log_loop_1_4, transformed_data_1_4, y_1_4_equal, cv=5) #calculate cross v
  cv_scores_1_4.append(np.mean(scorez_1_4)) #append to the list 'cv_scores'
```

```
#the maximal cross validation score from the logistic regression loop :)
np.amax(cv_scores_1_4) #0.6157692307692308 #almost the same as above


# find the index place of the max time point 0.6157692307692308
```

```
## 0.6157692307692308
```

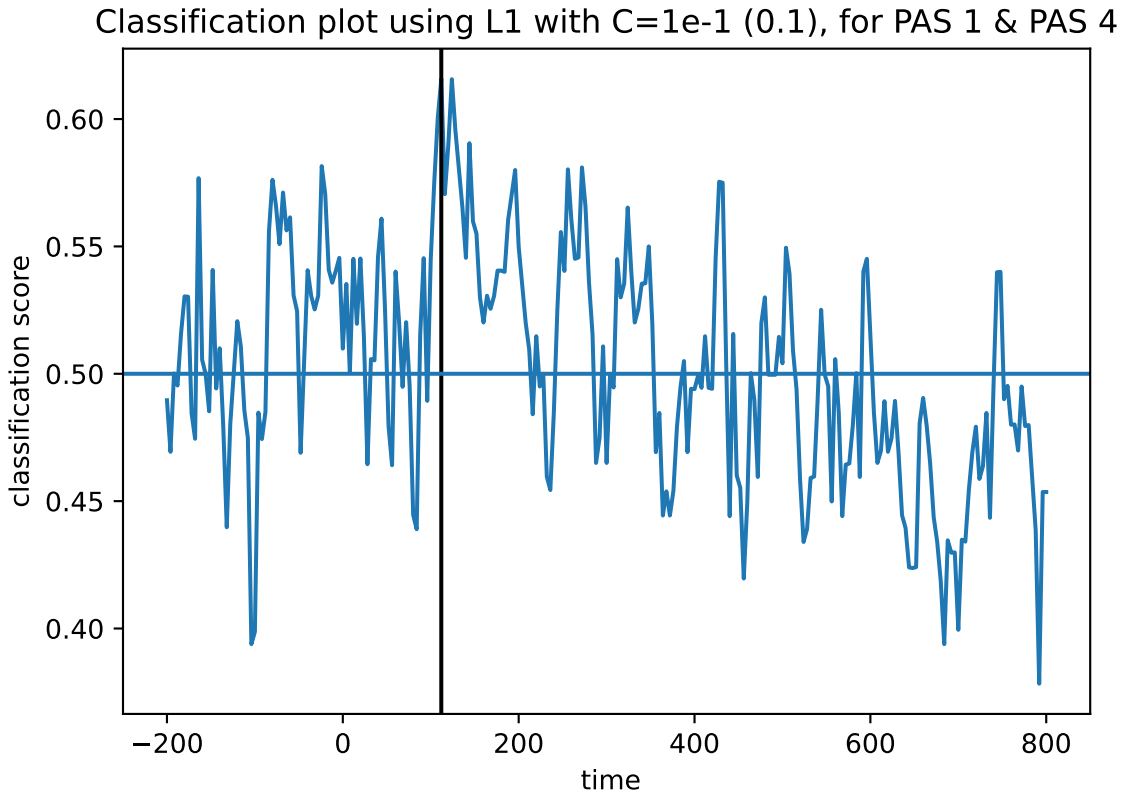
```
np.argmax(cv_scores_1_4) # index for maksimale punkt = 78
```

```
## 78
```

```
times[78]


#plot time on the x-axis and classification score on the y-axis with a horizontal line at the chance le
```

```
## 112
```

```
plt.figure()
plt.plot(times, cv_scores_1_4)
plt.axvline((times[78]), color= "black") #Highlight of time point where classification is best
plt.axhline(0.5) #chance level of 0.5, since its binary
plt.xlabel(" time")
plt.ylabel("classification score")
plt.title("Classification plot using L1 with C=1e-1 (0.1), for PAS 1 & PAS 4")
plt.show()
```



3) Is pairwise classification of subjective experience possible? Any surprises in the classification accuracies, i.e. how does the classification score fore PAS 1 vs 4 compare to the classification score for PAS 1 vs 2?

The models are not really good at classifying PAS ratings, since their scores are very close to the chance level of 50%. We could have expected, that the model would find it easier to classify and distinguish between PAS 1 and PAS 4 compared to PAS 1 and PAS 2, since PAS 1 and PAS 4 are further away from each other on the perceptual awareness scale. Though, it does not seem like the model favors to predict PAS 1 and PAS 4 easier than PAS 1 and 2, because the models performance are equally bad.

EXERCISE 3 - Do a Support Vector Machine Classification on all four PAS-ratings

1) Do a Support Vector Machine Classification

i. First equalize the number of targets using the function associated with each PAS-rating using the function associated with Exercise 3.1.i

```
# First equalize the number of targets using the function associated with the exercise
def equalize_targets(data, y):
    np.random.seed(7)
    targets = np.unique(y)
    counts = list()
    indices = list()
    for target in targets:
        counts.append(np.sum(y == target))
        indices.append(np.where(y == target)[0])
    min_count = np.min(counts)
    first_choice = np.random.choice(indices[0], size=min_count, replace=False)
    second_choice = np.random.choice(indices[1], size=min_count, replace=False)
    third_choice = np.random.choice(indices[2], size=min_count, replace=False)
    fourth_choice = np.random.choice(indices[3], size=min_count, replace=False)

    new_indices = np.concatenate((first_choice, second_choice,
                                  third_choice, fourth_choice))
    new_y = y[new_indices]
    new_data = data[new_indices, :, :]

    return new_data, new_y

#equalizing the data, so I have 98 elements of each PAS rating
data_4, y_4 = equalize_targets(data, y)
data_4.shape #yes I have 98 of each
```

```
## (396, 102, 251)
```

```
y_4.shape #yes I have 98 of each

######prepare data for SVM (scale and reshape)

#rehape data_4 from 3D to X_4 2D
```

```
## (396,)
```

```
X_4= data_4.reshape(396,-1) # now it's 2D (doing -1 takes everything else than )
X_4.ndim #got 2 dimensions now

#Scale X_4 (standardize)
```

```
## 2
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_X_4 = scaler.fit_transform(X_4)
scaled_X_4.shape
```

```
## (396, 25602)
```

ii. Run two classifiers, one with a linear kernel and one with a radial basis (other options should be left at their defaults) - the number of features is the number of sensors multiplied the number of samples. Which one is better predicting the category?

```
# now make a support vector machine (SVM)
from sklearn.svm import SVC

#define the two classes
linear_svm =  SVC(kernel= "linear")
radial_svm = SVC(kernel= "rbf")


#fitting the linear classifier
linear_svm_scores = cross_val_score(linear_svm, scaled_X_4, y_4)
print("The linear basis classifier score is", np.mean(linear_svm_scores))

#fitting the radial classifier
```

```
## The linear basis classifier score is 0.2928164556962025
```

```
radial_svm_scores = cross_val_score(radial_svm, scaled_X_4, y_4)
print("The linear basis classifier score is", np.mean(radial_svm_scores))
```

```
## The linear basis classifier score is 0.3333544303797468
```

iii. Run the sample-by-sample analysis (similar to Exercise 2.2.v) with the best kernel (from Exercise 3.1.ii). Make a plot with time on the x-axis and classification score on the y-axis with a horizontal line at the chance level (what is the chance level for this analysis?)

The radial classifier seemed to perform better than the linear classifier, with a mean score of 0.36. The chance level for this analysis is 25%, with four options being PAS 1, PAS 2, PAS 3 and PAS 4 (with an equal amount of observations in each).

```
# Run the sample-by-sample analysis with the best kernel (radial)
radial_cv_scores= [] #make an empty list
radial_svm = SVC(kernel= "rbf")#best model from above defined before the loop :)

#logistic regression on 1 time sample
for i in range(251):
  transformed_data_radial = scaler.fit_transform(data_4[:,:,i]) # transform the data first
  radial_svm.fit(transformed_data_radial, y_4) #loop and fit through the transformed data
  scorez_4= cross_val_score(radial_svm, transformed_data_radial, y_4, cv=5) #calculate cross validation
  radial_cv_scores.append(np.mean(scorez_4)) #append to the list 'cv_scores'
```

```
#the maximal cross validation score from the radial loop :)
np.amax(radial_cv_scores) #0.3586075949367088


# find the index place of the max time point 0.6161538461538462
```

```
## 0.3586075949367088
```

```
np.argmax(radial_cv_scores) # index for maksimale punkt = 226
```

```
## 226
```

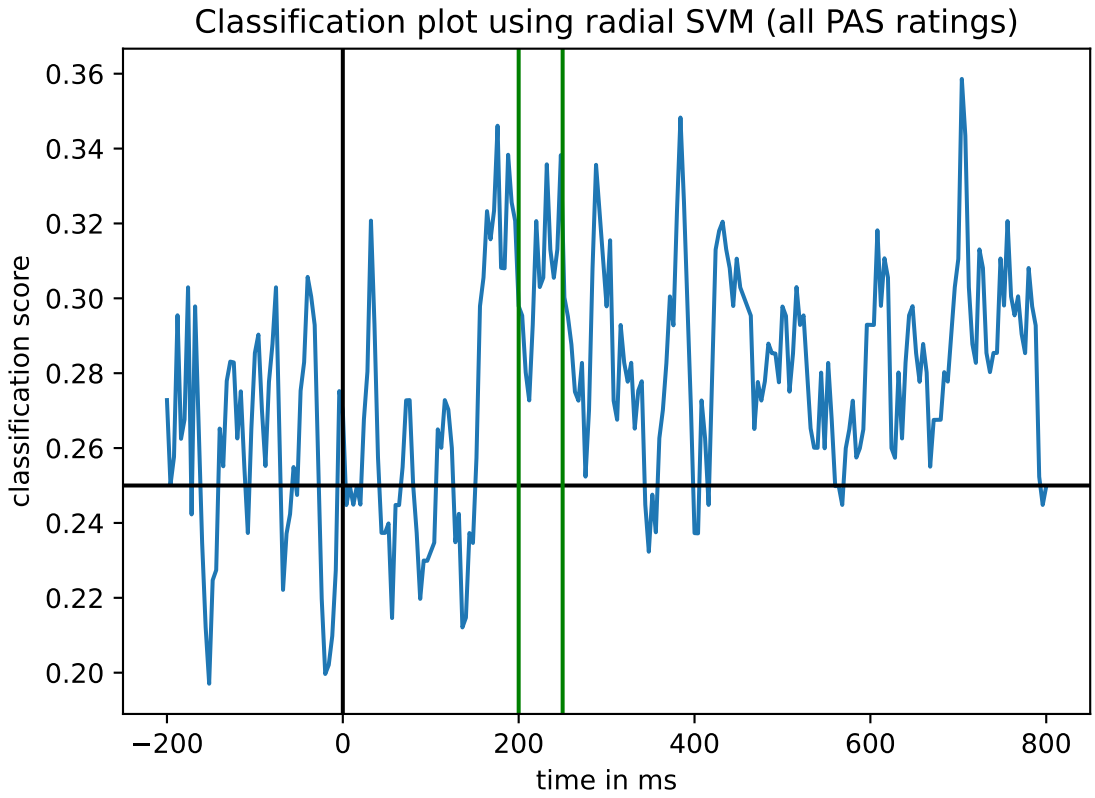
```
times[226]
```

```
## 704
```

```
times[50]


#plot time on the x-axis and classification score on the y-axis with a horizontal line at the chance le
```

```
## 0
```

```
plt.figure()
plt.plot(times, radial_cv_scores)
plt.axvline((times[50]), color= "black") #Highlight of time point where classification is best
plt.axhline(y = 0.25, color = "black")
plt.axvline(x = 200, color = "green")
plt.axvline(x = 250, color = "green")
plt.xlabel("time in ms")
plt.ylabel("classification score")
plt.title("Classification plot using radial SVM (all PAS ratings)")
plt.rcParams['figure.dpi'] = 1000
plt.show()
```



iv. Is classification of subjective experience possible at around 200-250 ms?

Classification of PAS ratings are possible around 200-250 ms, but still only with a low level of accuracy, i.e a classification score between 0.28 and 0.34.

2) Finally, split the equalized data set (with all four ratings) into a training part and test part, where the test part if 30 % of the trials. Use `train_test_split` from `sklearn.model_selection`

```
#importing
from sklearn.model_selection import train_test_split

#split the data
X_train, X_test, y_train, y_test = train_test_split(scaled_X_4, y_4, test_size=0.30)
```

i. Use the kernel that resulted in the best classification in Exercise 3.1.ii and `fit`the training set and `predict` on the test set. This time your features are the number of sensors multiplied by the number of samples.

```
#best classifier from Exercise 3.1.ii
radial_svm = SVC(kernel= "rbf") #best classifier from before

# `fit`the training set with the radial classifier
radial_svm.fit(X_train, y_train)

# `predict` on the test set
```

```
## SVC()
```

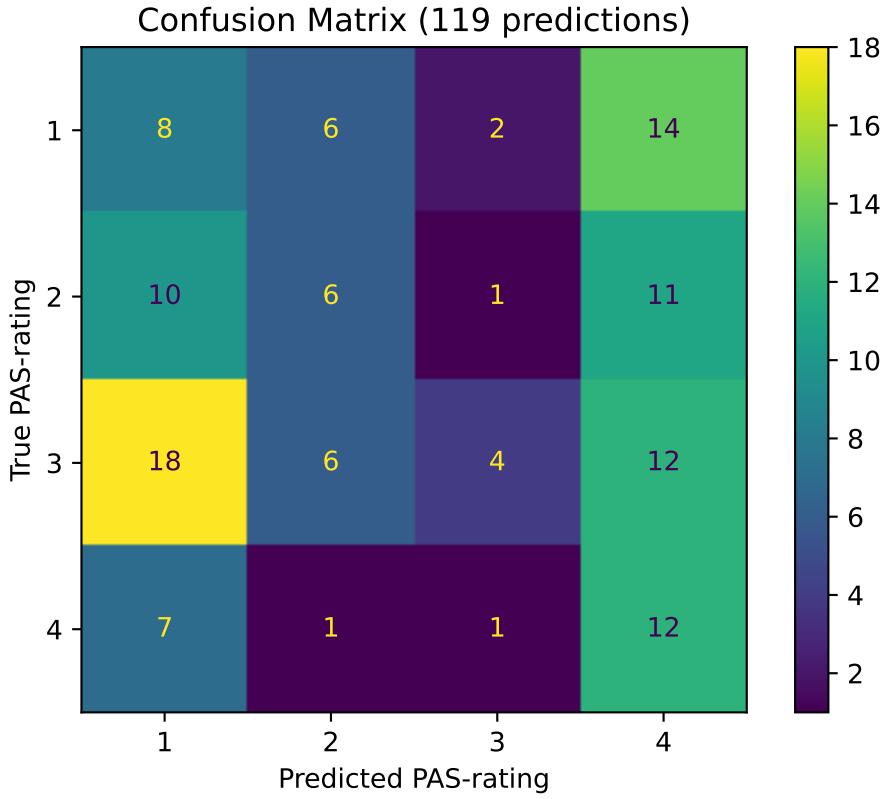
```
radial_predictions = radial_svm.predict(X_test)
```

ii. Create a *confusion matrix*. It is a 4x4 matrix. The row names and the column names are the PAS-scores. There will thus be 16 entries. The PAS1xPAS1 entry will be the number of actual PAS1, y_{pas1} that were predicted as PAS1, $\hat{y}_{pas1}$. The PAS1xPAS2 entry will be the number of actual PAS1, y_{pas1} that were predicted as PAS2, $\hat{y}_{pas2}$ and so on for the remaining 14 entries. Plot the matrix

```
#importing
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay as cmd

#plot
plt.close("all")
cmd.from_estimator(radial_svm, X_test, y_test)
```

```
## <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x161f59e50>
```

```
plt.xlabel("Predicted PAS-rating")
plt.ylabel("True PAS-rating")
plt.title("Confusion Matrix (119 predictions)")
plt.show()
```



iii. Based on the confusion matrix, describe how ratings are misclassified and if that makes sense given that ratings should measure the strength/quality of the subjective experience. Is the classifier biased towards specific ratings?

From the matrix above, it seems like the PAS ratings were to a larger extent classified as either PAS 1 or PAS 4, thus possibly biased towards those two ratings. When the model is more biased towards (mis)classifying PAS 1 and PAS 4, then PAS 2 and PAS 3 receive less classifications (especially PAS 3).