Assignment 3, Methods 3, 2021, autumn semester

Aleksander Moeslund Wael, 09/12/2021

Importing packages

In [ ]:
```
from matplotlib.colors import Colormap
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 200 # HIGH DPI PLOTS PLEASE
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

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

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

In [ ]:
```
data = np.load("megmag_data.npy")
```

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?

In [ ]:
```
data.shape
```

Out[ ]:
```
(682, 102, 251)
```

There are 682 repetitions, 102 sensors and 251 time samples.

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.

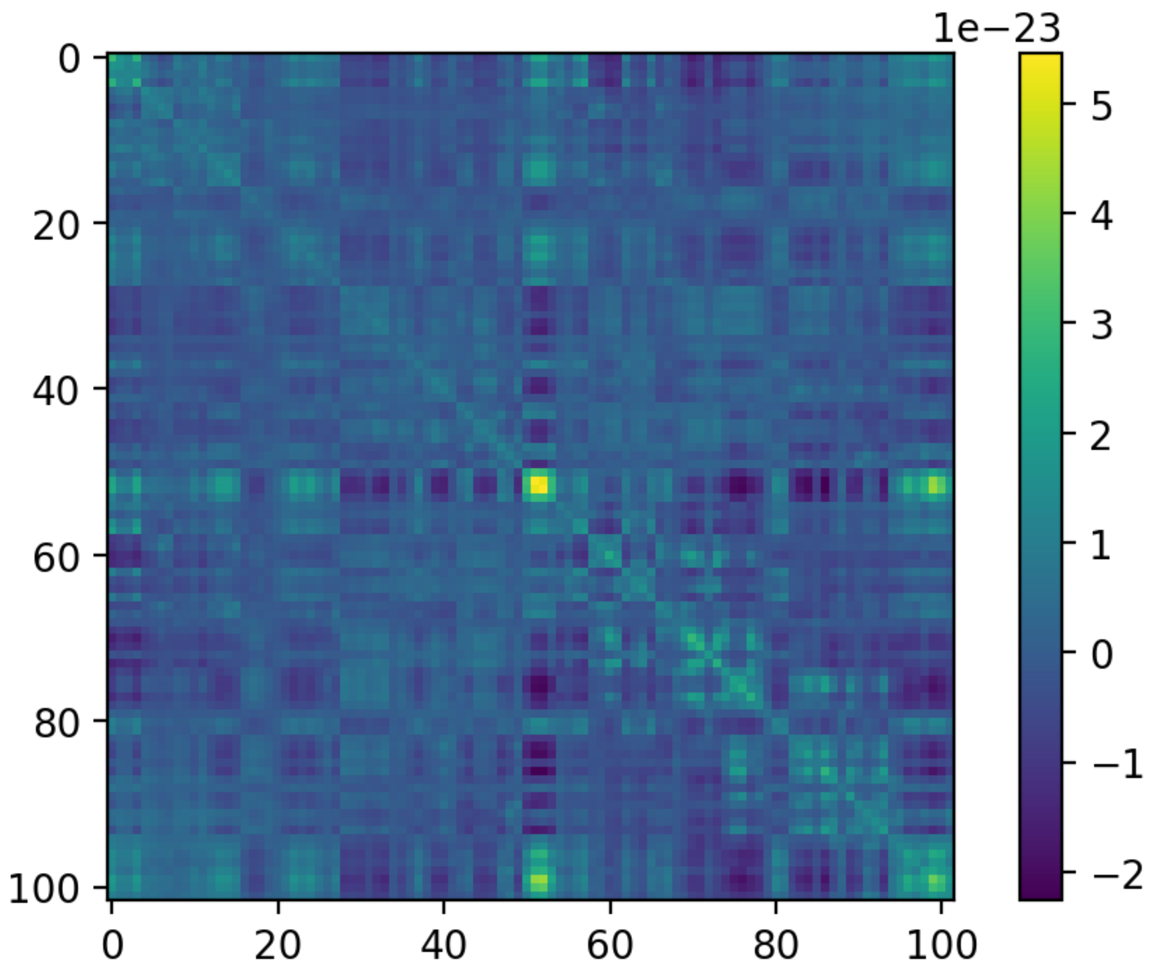
In [ ]:
```
times = np.arange(-200, 804, 4)
```

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)

In [ ]:
```
covar = np.zeros(shape = (102,102))

for i in range(682):
    X = data[i]
    covar = covar + X @ np.transpose(X)

covar = (1/682)*covar

plt.close("all")
plt.figure()
plt.imshow(covar)
plt.colorbar()
plt.show()
```



There is indeed covariance between the sensors, but it is unclear why this is. It might be that sensors pick up the same signals, OR they pick up different signals but with similar values.

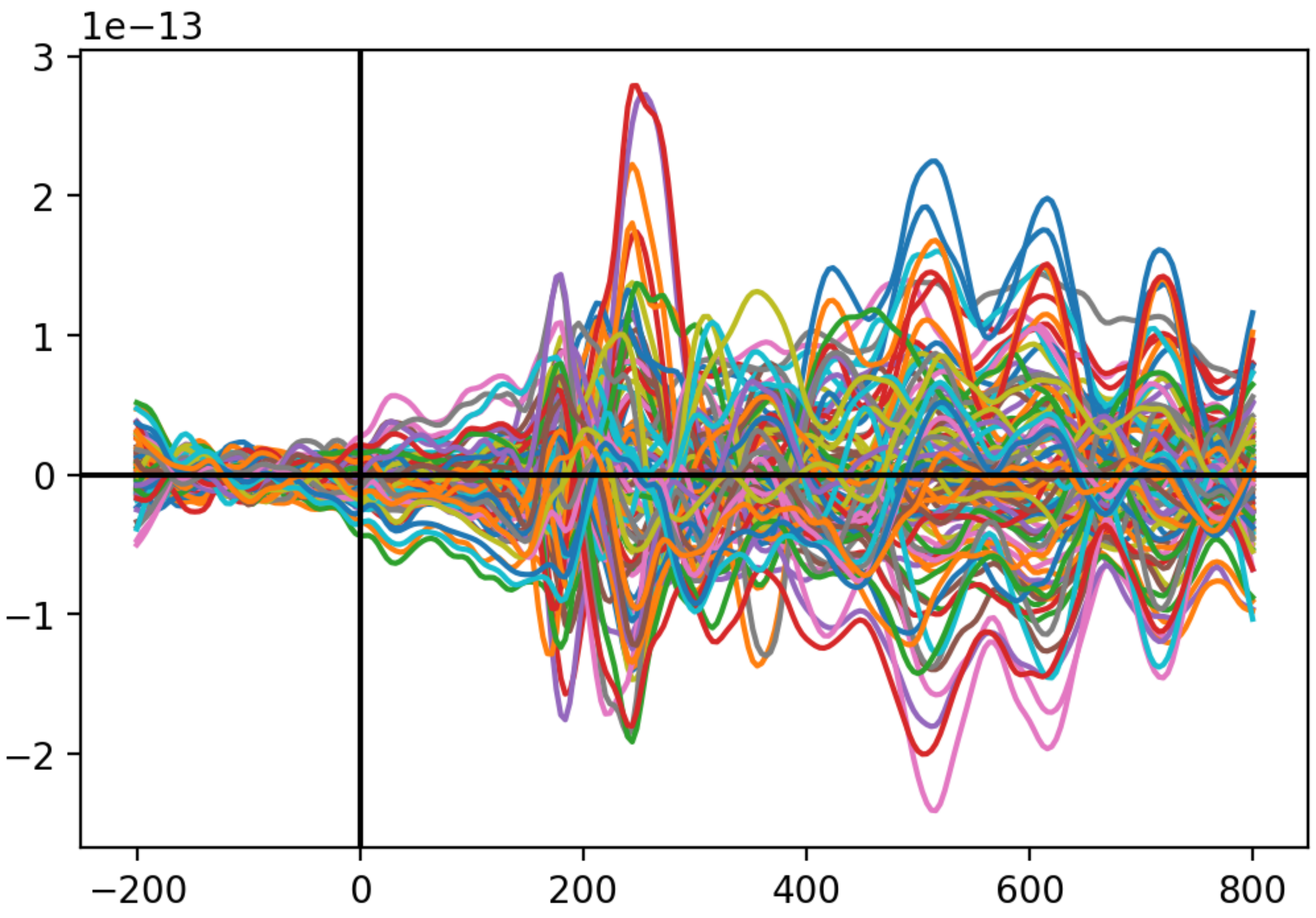
iv. Make an average over the repetition dimension using `np.mean` - use the `axis` argument. (The resulting array should have two dimensions ~~with time as the first and magnetic field as the second~~ with sensor as the first and time as the second)

In [ ]:
```python
datamean = data.mean(axis=0)
datamean.shape
```

Out[ ]:
```
(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`

In [ ]:
```python
plt.close("all")
plt.figure()
plt.plot(times, datamean.T)
plt.axvline(x = 0, color = "black")
plt.axhline(y = 0, color = "black")
plt.show()
```



vi. Find the maximal magnetic field in the average. Then use `np.argmax` and `np.unravel_index` to find the sensor that has the maximal magnetic field.

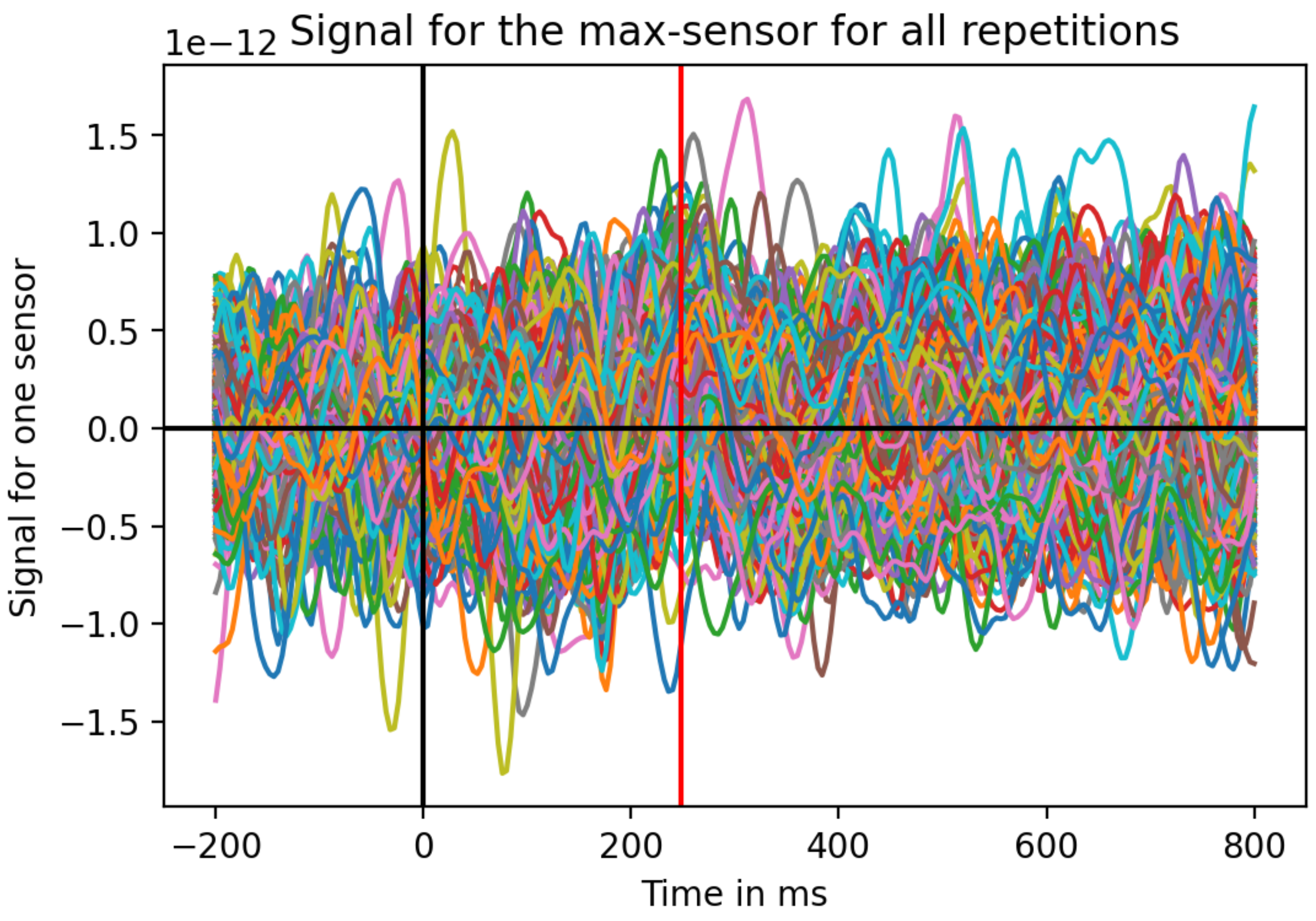
In [ ]:
```
# finding the biggest value
print(np.amax(datamean))

# finding the flat index for the biggest value
print(np.argmax(datamean, axis = 1))

# converting flat index into tuple so i know which sensor it is
print(np.unravel_index(np.argmax(datamean), shape = (102, 251))) # shape = the space where it searches for the index.
```

```
2.7886216843591933e-13
[127  95   5 133  94  94  94 111  96  95  95  95 214 208 207 201  96  95
 246  96 238 112 207 222   0 145 139   0 113 113 112 111 111 112 113 111
 111 143 111 167 142 112 111 203 111 110 172 202 212 151 179 180 179 180
 142 112 143 136 111 200 177 111 144 112 113 143 201 203 157 179 179 111
 112 112  98  95  96  99 128 129 141 141  94  94  94  95  95  95 139 139
  94  95  95  94  95  95 181 206 180 180 180   0]
(73, 112)
```

The sensor with maximal magnetic field (on average) is no. 73

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

In [ ]:
```
plt.figure()
plt.plot(times, data[:,73,:].T)
plt.axvline(times[112], color = "red")
plt.axvline(x = 0, color = "black")
plt.axhline(y = 0, color = "black")
plt.xlabel("Time in ms")
plt.ylabel("Signal for one sensor")
plt.title("Signal for the max-sensor for all repetitions")
plt.show()
```



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

- As can be seen by the plot, there is some diffeerence between repetitions. These can be due to hidden variables or noise in the data, which i'm not interested in modelling. Taking the mean of repetitions is a way of isolating the signal which is what matters for analysis.

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

In [ ]:
```
y = np.load("pas_vector.npy")
```

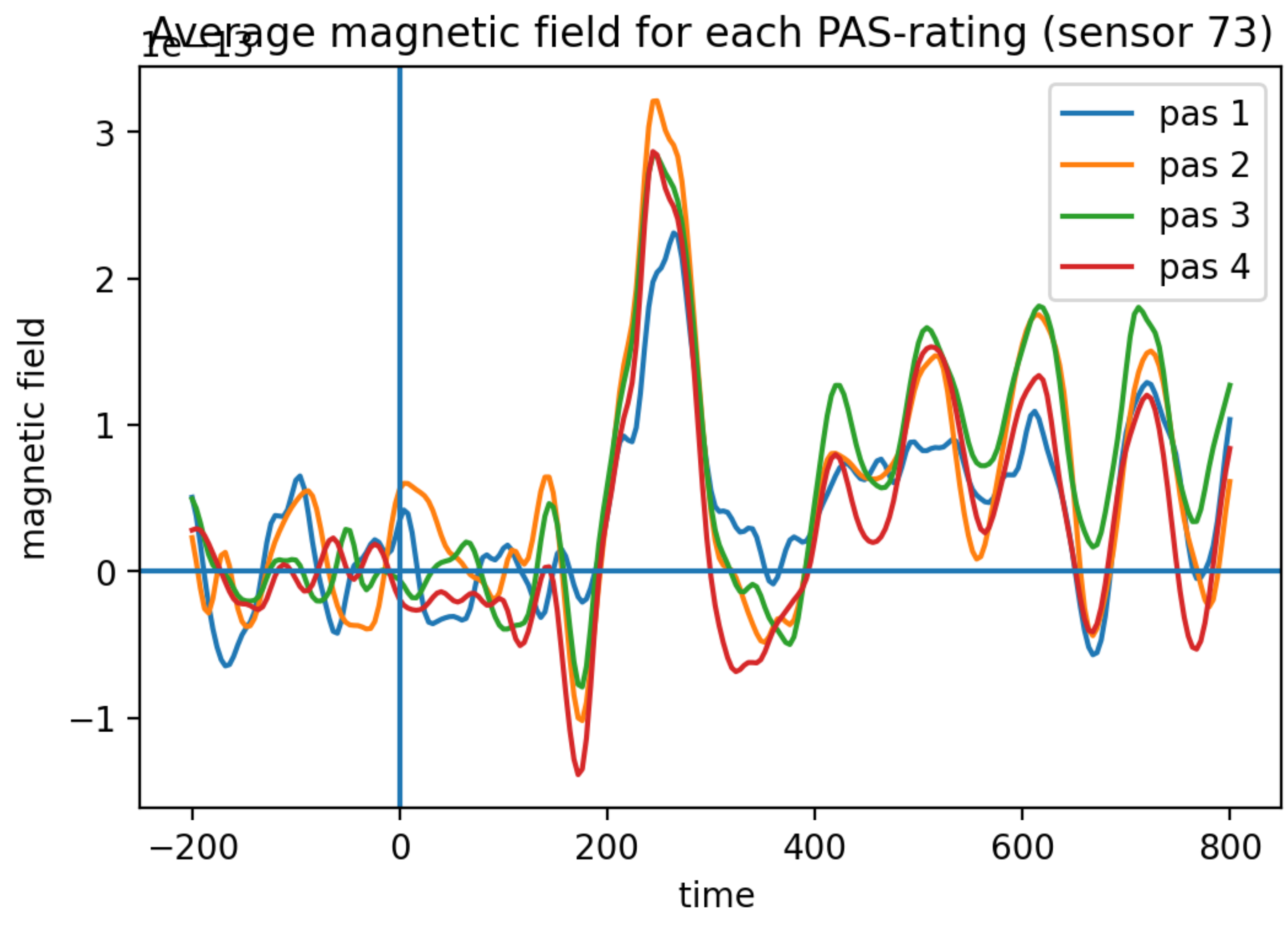
i. Which dimension in the `data` array does it have the same length as?

- It has the same length as repetitions, because there is a PAS rating per repitition.

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~~ 1.1.vi

In [ ]:
```
# subsetting the data, so I'm only looking at sensor 73
data_73 = data[:, 73, :]

# in y I find the indices for each different pas rating and assign them to new lists
pas_1 = np.where(y == 1)
pas_2 = np.where(y == 2)
pas_3 = np.where(y == 3)
pas_4 = np.where(y == 4)

# finding the average brain activation in sensor 73 seperated by pas-rating
sens_73_pas1_avg = np.mean(data_73[pas_1], axis = 0)
sens_73_pas2_avg = np.mean(data_73[pas_2], axis = 0)
sens_73_pas3_avg = np.mean(data_73[pas_3], axis = 0)
sens_73_pas4_avg = np.mean(data_73[pas_4], axis = 0)

# plotting this baby
plt.figure
plt.plot(times, sens_73_pas1_avg)
plt.plot(times, sens_73_pas2_avg)
plt.plot(times, sens_73_pas3_avg)
plt.plot(times, sens_73_pas4_avg)
plt.axvline()
plt.axhline()
plt.xlabel(" time")
plt.ylabel("magnetic field")
plt.title("Average magnetic field for each PAS-rating (sensor 73)")
plt.legend(["pas 1", "pas 2", "pas 3", "pas 4"])
plt.show
```

Out[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



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?

- I would expect the peaks at 250 to be ordered by PAS-rating with PAS-4 at the top (largest magnetic field). It seems that PAS-1 is the lowest as expected, by PAS-2 (which is described as a "weak glimpse") has a higher peak than PAS-3 and PAS-4 (which are quite similiar). This pattern is somewhat similar in the other peaks at 450-750 ms, though with some variability.

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

In [ ]:
```python
# subsetting
data_1_2 = data[y < 3,:,:]
y_1_2 = y[y < 3]
```

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.

In [ ]:
```python
# repetition as rows, and sensor and time as columns
X_1_2 = data_1_2.reshape(214, -1)
X_1_2.shape
```

Out[ ]:
```
(214, 25602)
```

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

In [ ]:
```python
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_1_2_scaled = sc.fit_transform(X_1_2)
```

iv. Do a standard `LogisticRegression` - can be imported from `sklearn.linear_model` - make sure there is no `penalty` applied

In [ ]:
```python
from sklearn.linear_model import LogisticRegression
```

```
logR = LogisticRegression(penalty='none') # no regularisation

logR.fit(X_1_2_scaled, y_1_2)
```

Out[ ]:
```
LogisticRegression(penalty='none')
```

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?

In [ ]:
```
print(logR.score(X_1_2_scaled, y_1_2))
```

```
1.0
```

The score was perfect, so it seems we are overfitting. We aren't testing the model on new data, which is probably one of the reasons.

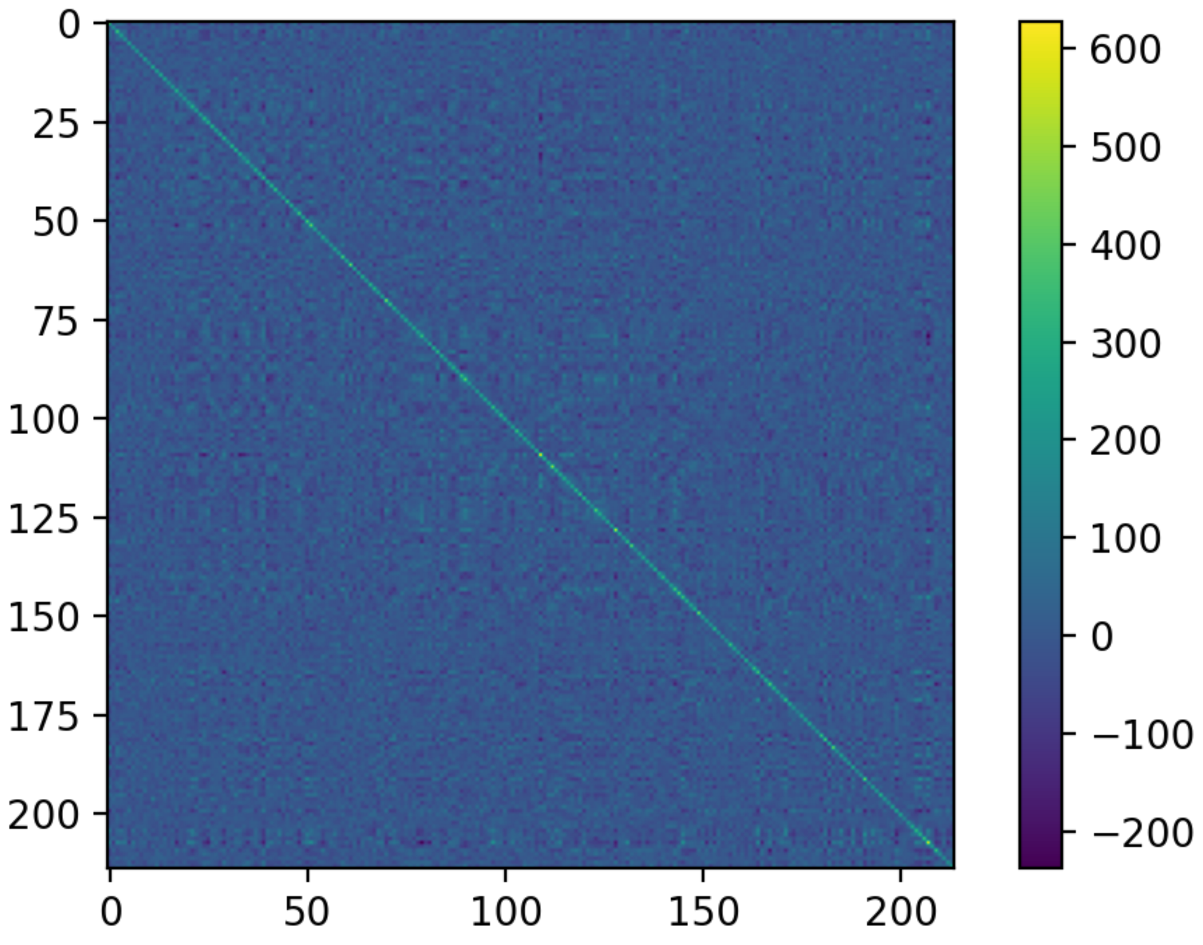
vi. Now apply the *L1* penalty instead - how many of the coefficients (`.coef_`) are non-zero after this?

In [ ]:
```
# Fitting
logR = LogisticRegression(penalty="l1", solver='liblinear', random_state=1) # With regularization
fit1 = logR.fit(X_1_2_scaled, y_1_2)
print(fit1)

# Finding nonzero coefs
print(np.sum(fit1.coef_ != 0), "coefficients are non-zero.")
```

```
LogisticRegression(penalty='l1', random_state=1, solver='liblinear')
282 coefficients are non-zero.
```

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?

In [ ]:
```
coefs = logR.coef_.flatten()
non_zero = coefs != 0
X_reduced = X_1_2_scaled[:, non_zero]

# Non-zero coefficients covariance matrix
coef_covar = X_reduced @ np.transpose(X_reduced)

plt.close("all")
plt.figure()
plt.imshow(coef_covar)
plt.colorbar()
plt.show()
```



We see less covariance overall, the diagonal is very clear and defined. There is still covariance of the non-zero coefficients, but it is better than the sensor covariance matrix in 1.1.iii.

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`

In [ ]:
```
from sklearn.model_selection import cross_val_score, StratifiedKFold
```

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. 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` !

In [ ]:
```
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

# Making equal data with function
y_1_2 = np.array(y_1_2) # Has to be array instead of list, thats why
data_1_2_equal, y_1_2_equal = equalize_targets_binary(data_1_2, y_1_2) # Assigning new data
X_1_2_equal = data_1_2_equal.reshape(198, -1)
X_1_2_equal = sc.fit_transform(X_1_2_equal)
```

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

In [ ]:
```
cv = StratifiedKFold()

logR = LogisticRegression()
logR.fit(X_1_2_equal, y_1_2_equal)

scores = cross_val_score(logR, X_1_2_equal, y_1_2_equal, cv=5)
print(np.mean(scores))
```

```
0.5402564102564102
```

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)?

In [ ]:
```
# With C = 1e5
cv = StratifiedKFold()

logR = LogisticRegression(C=1e5, penalty="l2")
logR.fit(X_1_2_equal, y_1_2_equal)

scores = cross_val_score(logR, X_1_2_equal, y_1_2_equal, cv=5)
print(np.mean(scores))

# With C = 1e1
cv = StratifiedKFold()

logR = LogisticRegression(C=1e1, penalty="l2")
logR.fit(X_1_2_equal, y_1_2_equal)

scores = cross_val_score(logR, X_1_2_equal, y_1_2_equal, cv=5)
print(np.mean(scores))

# With C = 1e-5
cv = StratifiedKFold()

logR = LogisticRegression(C=1e-5, penalty="l2")
logR.fit(X_1_2_equal, y_1_2_equal)

scores = cross_val_score(logR, X_1_2_equal, y_1_2_equal, cv=5)
print(np.mean(scores))
```

```
0.5353846153846155
0.5252564102564102
0.5956410256410256
```

In [ ]:
```
# Difference between models
0.5956410256410256-0.5252564102564102
```

```
0.07038461538461538
```

Out[ ]:

The best model (C=1e-5) is on average about 7% better at prediciting than with C=1e1.

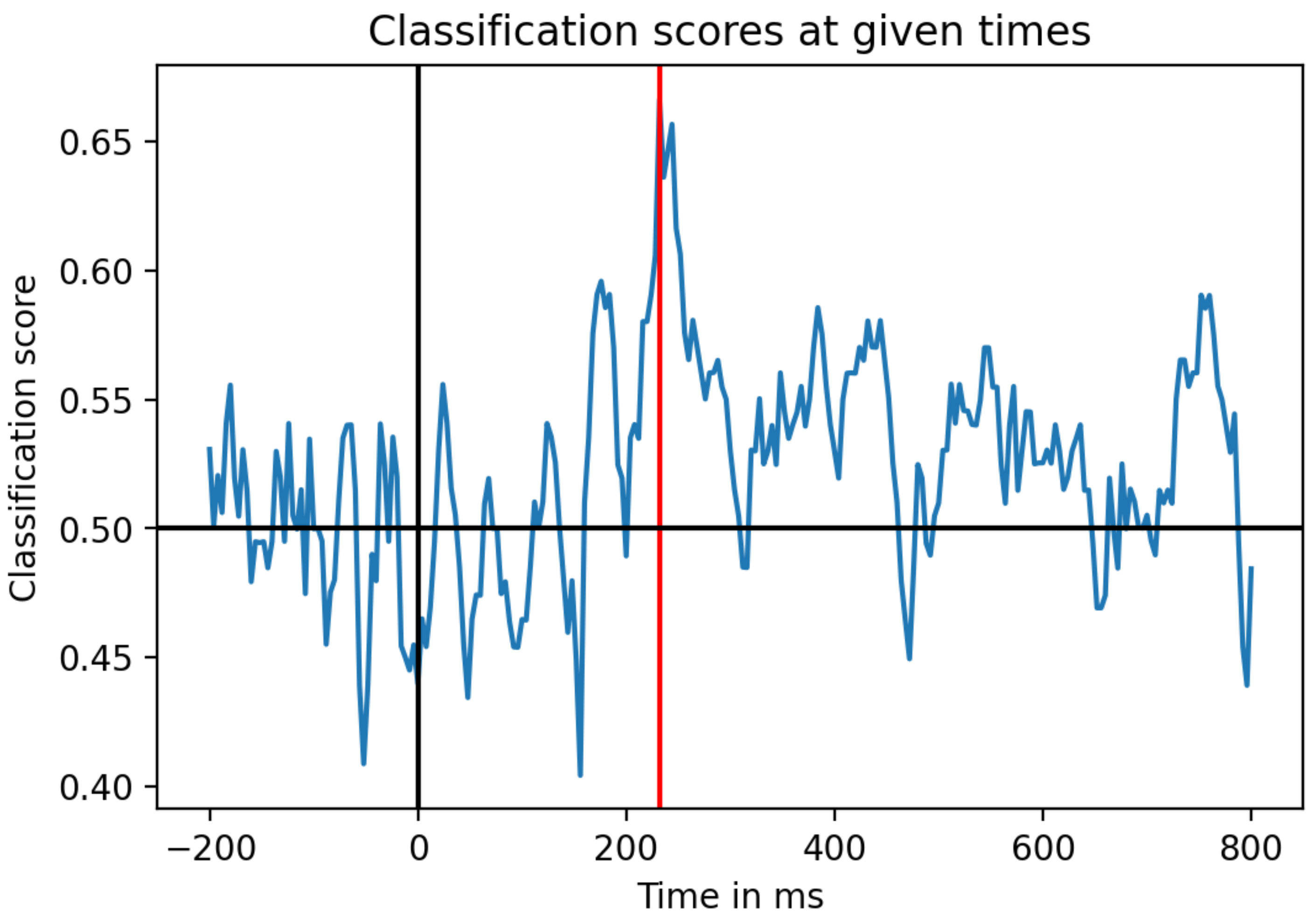
v. Instead of fitting a model on all `n_sensors * n_samples` features, fit a logistic regression (same kind as in Exercise 2.2.iv (use the `C` that resulted in the best prediction)) for each time sample and use the same cross-validation as in Exercise 2.2.iii. What are the time points where classification 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?)

In [ ]:
```
# I need 251 models.
cv = StratifiedKFold()
logR = LogisticRegression(C=1e-5, penalty="l2", solver = "liblinear")
cv_scores = []

# Subsetting time
for i in range(251):
    t = sc.fit_transform(data_1_2_equal[:,:,i])
    logR.fit(t, y_1_2_equal)
    scores = cross_val_score(logR, t, y_1_2_equal, cv=5)
    cv_scores.append(np.mean(scores))
```

In [ ]:
```
# Picking highest score
print(np.amax(cv_scores)) #  What is the max value?
print(np.argmax(cv_scores)) # Where is the max value? (Indeci)
```

```
0.6661538461538462
108
```

In [ ]:
```
plt.figure()
plt.plot(times, cv_scores)
plt.axvline(x = 0, color = "black")
plt.axvline(times[108], color = "red")
plt.axhline(y = 0.5, color = "black") # Chance level is 50% for binary classification
plt.xlabel("Time in ms")
plt.ylabel("Classification score")
plt.title("Classification scores at given times")
plt.show()

print(times[108]) # Where classification is best

# Classification is best at 232 ms.
```



```
232
```

Classification is best between 200-250 and 300-450 ms.

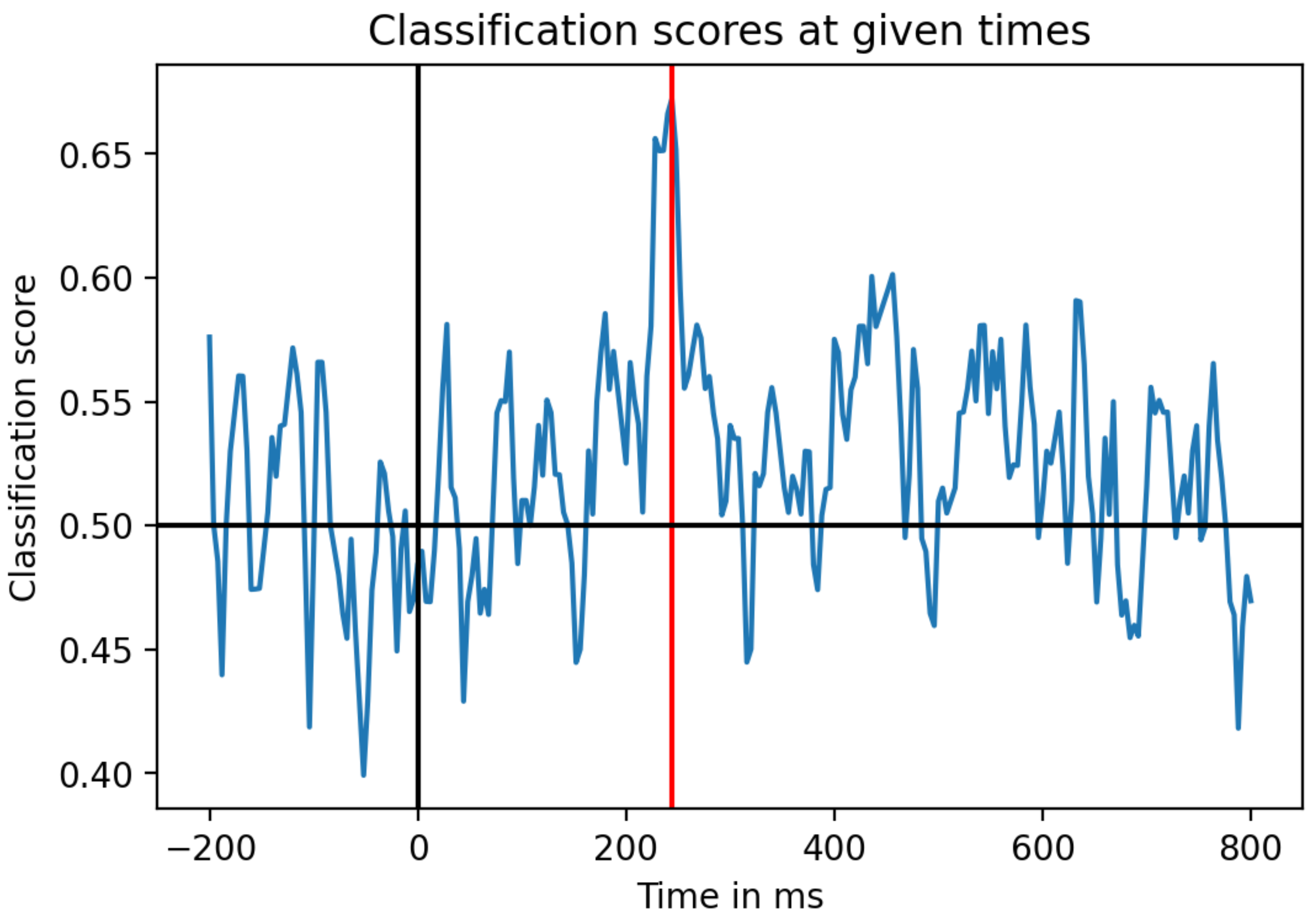
vi. Now do the same, but with L1 regression - set `C=1e-1` - what are the time points when classification is best? (make a plot)?

In [ ]:
```
cv = StratifiedKFold()
logR = LogisticRegression(C=1e-1, penalty="l1", solver = "liblinear")
cv_scores = []

# Subsetting time
for i in range(251):
    t = sc.fit_transform(data_1_2_equal[:,:,i])
    logR.fit(t, y_1_2_equal)
    scores = cross_val_score(logR, t, y_1_2_equal, cv=5)
    cv_scores.append(np.mean(scores))
```

In [ ]:
```
# Picking highest score
print(np.amax(cv_scores))
print(np.argmax(cv_scores)) # Indeci with highest classification
```

```
0.6720512820512822
111
```

In [ ]:
```
plt.figure()
plt.plot(times, cv_scores)
plt.axvline(x = 0, color = "black")
plt.axvline(times[111], color = "red")
plt.axhline(y = 0.5, color = "black") # Chance level is 50% for binary classification
plt.xlabel("Time in ms")
plt.ylabel("Classification score")
plt.title("Classification scores at given times")
plt.show()

times[111]
```



Out[ ]:
```
244
```

Classification is best at 244 ms.

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?)

In [ ]:
```
# Prepare array
data_1_4 = data[(y != 2) & (y != 3),:,:]
print(data_1_4.shape)
```

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

In [ ]:
```
# Prepare target vector
y_1_4 = []
```

```
for i in range(len(y)):
    if y[i] == 1:
        y_1_4.append(1)
    if y[i] == 4:
        y_1_4.append(4)

# repetition as rows, and sensor and time as columns
X_1_4 = data_1_4.reshape(359, -1)
X_1_4.shape
```

Out[ ]:

```
(359, 25602)
```

In [ ]:

```
# Scaling data
X_1_4_scaled = sc.fit_transform(X_1_4)
```

In [ ]:

```
# Making equal data with function
y_1_4 = np.array(y_1_4) # Has to be array instead of list, thats why
data_1_4_equal, y_1_4_equal = equalize_targets_binary(data_1_4, y_1_4) # Assigning new data
X_1_4_equal = data_1_4_equal.reshape(198, -1)
X_1_4_equal = sc.fit_transform(X_1_4_equal)
```

In [ ]:

```
cv = StratifiedKFold()
logR = LogisticRegression(C=1e-1, penalty="l1", solver = "liblinear")
cv_scores_1_4 = []

# Subsetting time
for i in range(251):
    t = sc.fit_transform(data_1_4_equal[:,:,i])
    logR.fit(t, y_1_4_equal)
    scores = cross_val_score(logR, t, y_1_4_equal, cv=5)
    cv_scores_1_4.append(np.mean(scores))
```

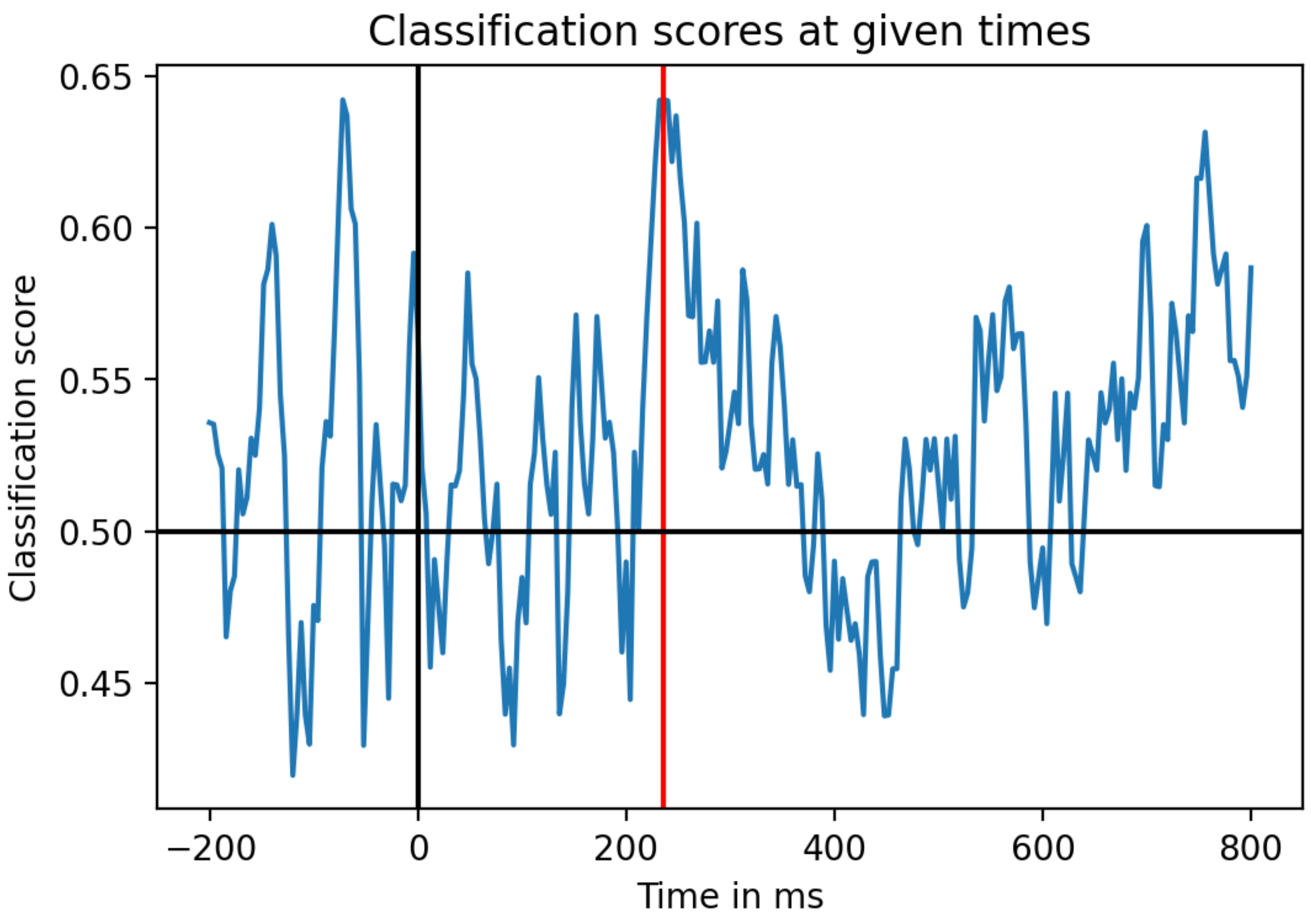
In [ ]:

```
# Picking highest score
print(np.amax(cv_scores_1_4))
print(np.argmax(cv_scores_1_4)) # Indeci with highest classification
```

```
0.6423076923076924
109
```

In [ ]:

```
# Plotting
plt.figure()
plt.plot(times, cv_scores_1_4)
plt.axvline(x = 0, color = "black")
plt.axvline(times[109], color = "red")
plt.axhline(y = 0.5, color = "black") # Chance level is 50% for binary classification
plt.xlabel("Time in ms")
plt.ylabel("Classification score")
plt.title("Classification scores at given times")
plt.show()

times[109]
```



```
Out[ ]: 236
```

Classification is best at 236 ms.

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?

It appears that it is more succesful to classify 1 vs 2 compared to 1 vs 4, which is quite surprising as PAS-score can be viewed as a pseudo-variable where scores of 1 should be more similar to 2 than to 4, which is also apparent from the plot of PAS-ratings and magnetic field from sensor 73. The difference between the two max-scores is only about 3%, which isn't huge. I would say that pairwise classification for PAS is somewhat possible in the 200-400 ms range, but the scores are still low.

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

```
In [ ]:
# Define function
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
```

```
In [ ]:
# Making data equal
data_equal, y_equal = equalize_targets(data, y)
```

```
print(data_equal.shape)
print(y_equal.shape)
```

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

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?

In [ ]:
```
# Import
from sklearn.svm import SVC

# Making classes
svm_linear = SVC(kernel="linear")
svm_radial = SVC(kernel="rbf")

# Converting to 2d array
X_equal = data_equal.reshape(396, -1)
print(X_equal.shape)

# Scaling data
X_equal_scaled = sc.fit_transform(X_equal)

# Fitting
scores_svm_linear = cross_val_score(svm_linear, X_equal_scaled, y_equal, cv=cv)
print("The linear basis classifier score is", np.mean(scores_svm_linear))

# Fitting
scores_svm_radial = cross_val_score(svm_radial, X_equal_scaled, y_equal, cv=cv)
print("The radial basis classifier score is", np.mean(scores_svm_radial))
```

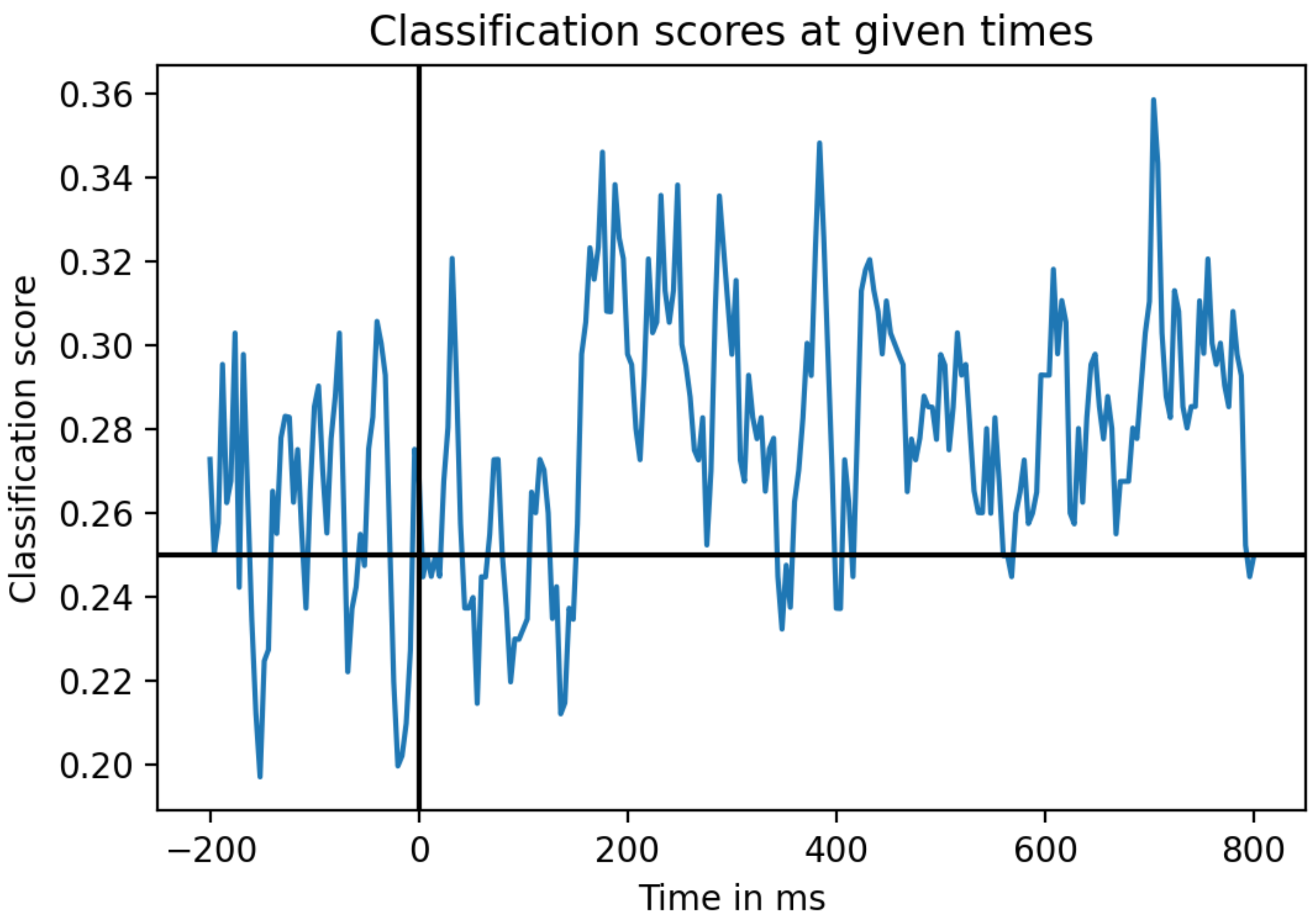
```
(396, 25602)
The linear basis classifier score is 0.2928164556962025
The radial basis classifier score is 0.3333544303797468
```

It appears that the radial classifier is best with mean scores of 0,33. Though chance level is at 0,25, so this is a pretty bad performance.

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?)

In [ ]:
```
# I need 251 models.
cv = StratifiedKFold()
svm_radial = SVC(kernel="rbf")
cv_scores_svm = []

# Subsetting time
for i in range(251):
    t = sc.fit_transform(data_equal[:,:,i])
    logR.fit(t, y_equal)
    scores = cross_val_score(svm_radial, t, y_equal, cv=5)
    cv_scores_svm.append(np.mean(scores))
```

In [ ]:
```
# Plotting
plt.figure()
plt.plot(times, cv_scores_svm)
plt.axvline(x = 0, color = "black")
plt.axhline(y = 0.25, color = "black") # Chance level is 25% for classification with 4 classes
plt.xlabel("Time in ms")
plt.ylabel("Classification score")
plt.title("Classification scores at given times")
plt.show()
```



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

- It is better than chance level at this time, with classification scores around 0.30-0.34.

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`

In [ ]:
```
# Import
from sklearn.model_selection import train_test_split

# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X_equal, y_equal, test_size=0.30)
```

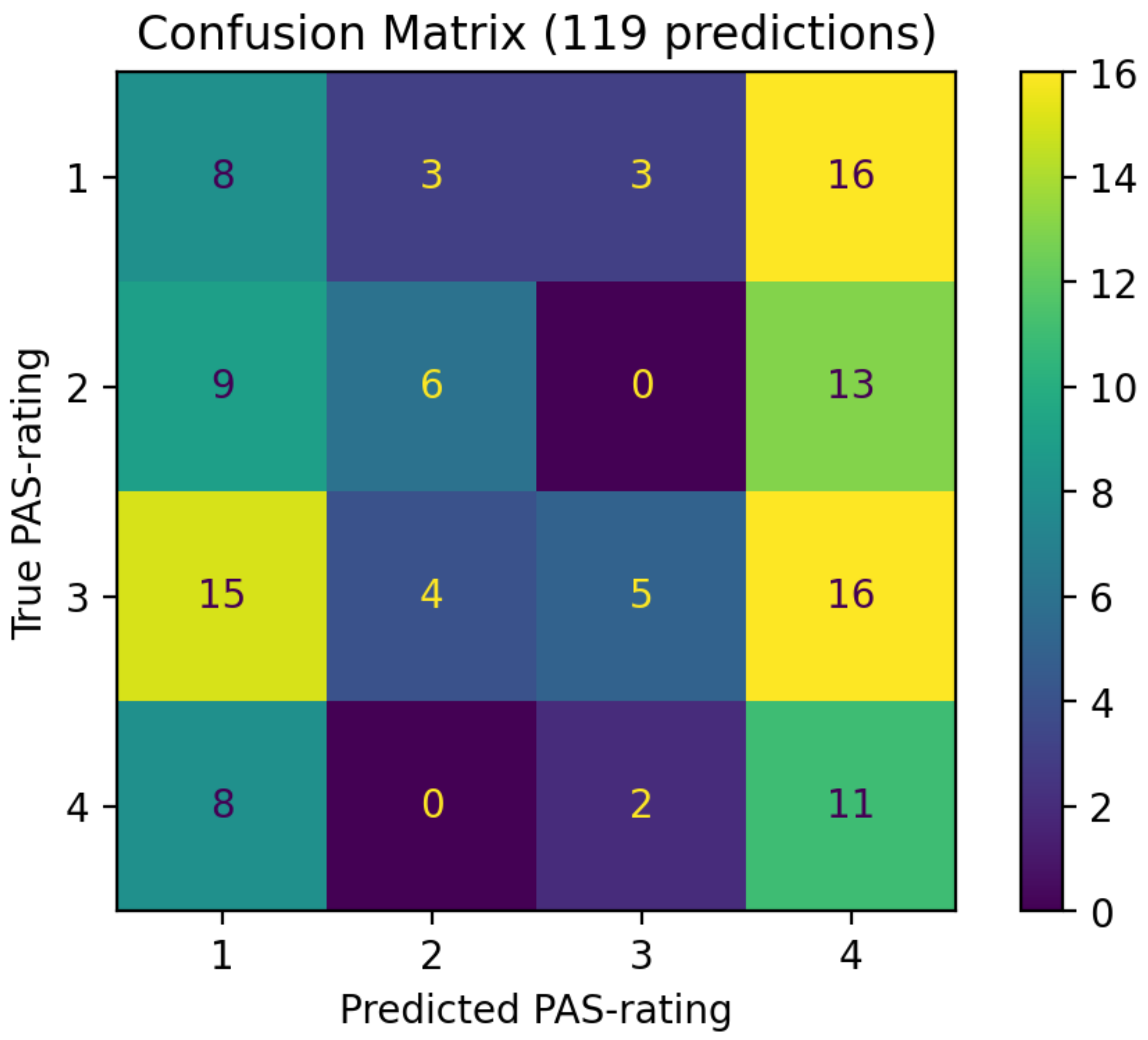
i. Use the kernel that resulted in the best classification in Exercise 3.1.ii and `fit` the training set and `predict` on the test set. This time your features are the number of sensors multiplied by the number of samples.

In [ ]:
```
# Setting up class
svm_radial = SVC(kernel="rbf")

# Fitting
svm_radial.fit(X_train, y_train)

# Predicting
predictions = svm_radial.predict(X_test)
print(predictions)
```

```
[2 4 4 1 4 1 4 4 4 4 1 3 4 4 2 1 2 4 1 4 4 4 1 2 4 4 4 1 4 1 1 2 1 1 3 4 4
 2 4 1 1 2 1 4 4 1 2 2 4 4 1 4 1 3 1 3 4 1 4 4 4 1 4 1 1 3 1 4 1 1 1 2 4 4
 4 4 4 4 3 4 1 1 4 4 3 1 4 1 1 1 4 1 2 4 1 4 4 1 4 4 3 1 1 4 4 4 1 2 3 4 4
 1 4 3 4 4 2 1 4]
```

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

In [ ]:
```
# Import
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay as cmd

# Plot
plt.close("all")
cmd.from_estimator(svm_radial, X_test, y_test) # Plotting given estimator, test data and true labels.
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?

- It appears that all PAS-ratings were often classified as PAS 4, but also quite often as 1, and is therefore biased towards these. It also means that it often correctly classifies PAS 1 and 4, but at the expense of misclassifying PAS 2 and 3. I would also assume that PAS ratings would more often be misclassified as the adjacent ratings, e.g. PAS 2 would more often be misclassified as 1 or 3 than 4. This doesn't seem to be the case though.