Portfolio Assignment 3

Study Group 8, Emma Rose Hahn (EH), Viara Krasteva (VK), Kristian Nøhr Villebro (KV), Luke Ring (LR)

Student numbers

- Emma Rose Hahn: 202004249
- Kristian Nøhr Villebro: 202005289
- Luke Ring: 202009983
- Viara Krasteva: 202005673

In [ ]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay
import seaborn as sns

# surpress warnings for final output
import warnings
warnings.filterwarnings('ignore')
```

Exercise 1

1.1 Describe data structure (EH)

In [ ]:
```
data = np.load('megmag_data.npy')
dshape  = data.shape
print('Megmag data shape: {}'.format(dshape))
```

```
Megmag data shape: (682, 102, 251)
```

1.1.i There are 682 repetitions, 102 sensors and 251 timestamps

1.1.ii) Add time offset (ms) (EH)

In [ ]:
```
times = np.arange(-200, 801, 4)
print('Time offset array shape: {}'.format(times.shape))
```

```
Time offset array shape: (251,)
```

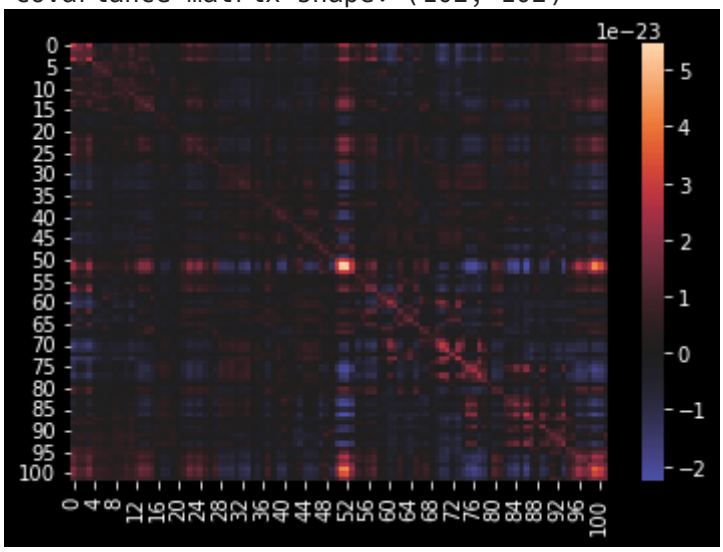
1.1.iii) Create covariance matrix, do the sensors pick up independent signals? (VK)

In [ ]:
```
output = []
for i in range(len(data)):
  X = data[i]
```

```
  Xt = X.T
  output.append(np.matrix(np.dot(X,Xt)))
cov_mat = 1  / dshape[0] * np.sum(i for i in output)

print('Covariance matrix shape: {}'.format(cov_mat.shape))
sns.heatmap(cov_mat, cmap='icefire', center = 0)
# plt.imshow(cov_mat)
plt.show()
```

```
Covariance matrix shape: (102, 102)
```



1.1.iii It looks like there is a lot of covariance (which would make sense), however it is quite difficult to get a clear impression with such big covariance matrix.

1.1.iv - 1.1.v) Create signal averages across repititions (KV)

In [ ]:

```
avg_for_reps = np.mean(data, axis=0).T
print('Average for each repetition shape: {}'.format(avg_for_reps.shape))
```

```
Average for each repetition shape: (251, 102)
```

1.1.vi - 1.1.vii Find and plot maximum response channel (KV)

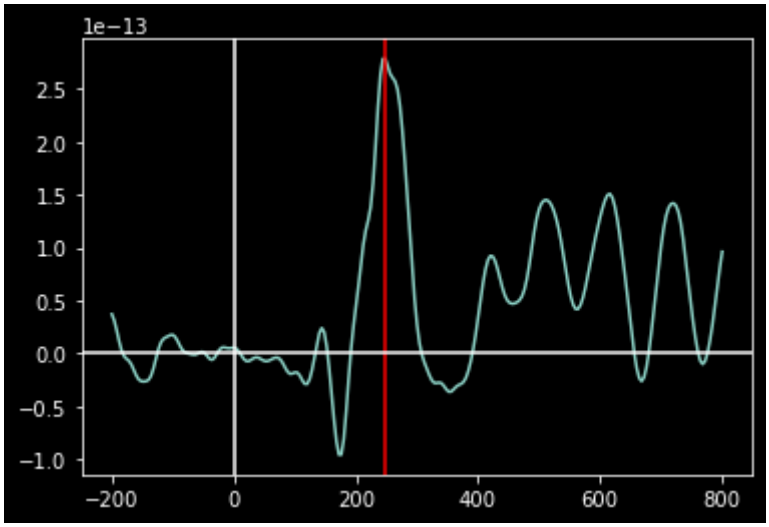
In [ ]:

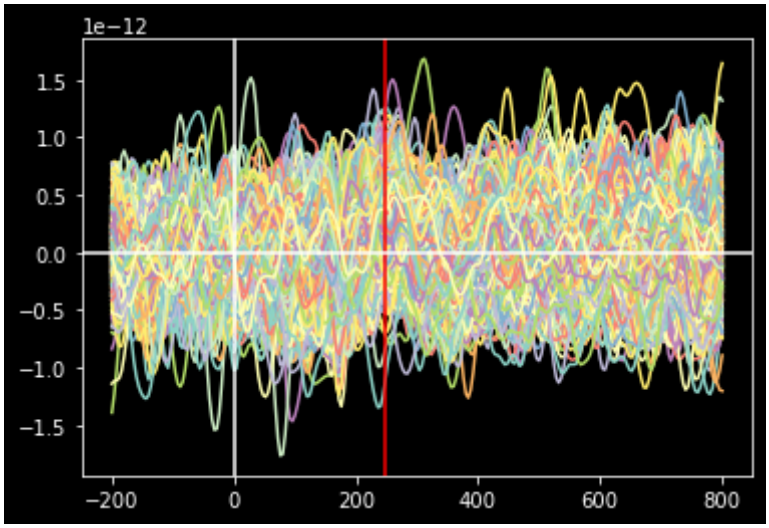
```
max_resp = np.unravel_index(np.argmax(avg_for_reps), avg_for_reps.shape)
print('Max response index: {}, data shape: {}'.format(max_resp,
data.shape))
#plot average for all repititions
plt.plot(times, avg_for_reps[:, max_resp[1]])
plt.axvline()
plt.axvline(x = times[max_resp[0]], color='r', label='max response')
plt.axhline()
plt.show()

# plot all repititions for max response channel
plt.plot(times, data[:, max_resp[1], :].squeeze().T)
plt.axvline()
plt.axvline(x = times[max_resp[0]], color='r', label='max response')
plt.axhline()

plt.show()
```

```
Max response index: (112, 73), data shape: (682, 102, 251)
```





1.1.vi We found that the maximum response signal was received by sensor 74 (as python index starts at 0)

1.1.viii The messy plot showing the single repetitions includes a lot of fluctuation around the y-axis, thus including lots of noise, making it difficult to assess anything from. Even though the messy plot looks as if the activity is more or less the same across the whole time period, the averaged plot reveals that activation is quite different over time. As the signal noise goes in both negative and positive direction, they cancel each other out thus resulting in the averaged plot we have seen.

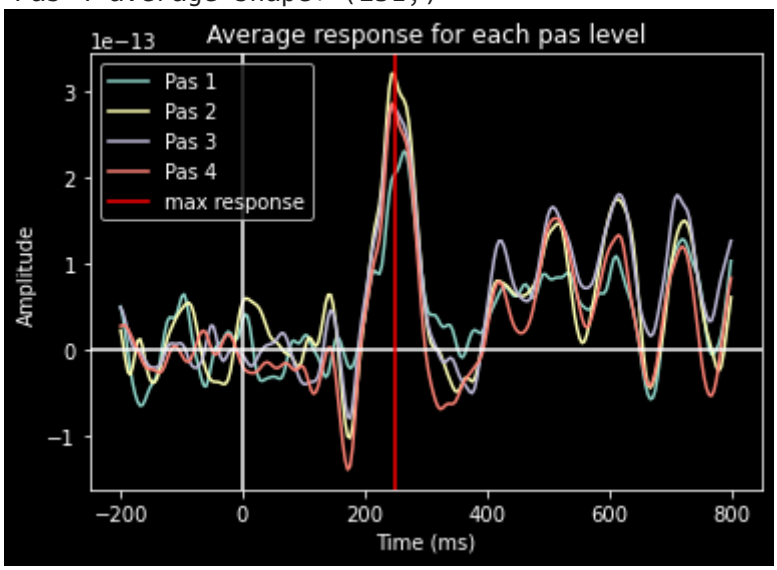
1.2) load pas_vector.npy (LR)

In [ ]:

```
y = np.load('pas_vector.npy')
# same as number of repitititions
print('Pas vector shape: {}'.format(y.shape))
pas_data = []
pas_index = {}
for i in np.unique(y):
  print('Creating pas slice for pas {}'.format(i))
  pas_slice = np.argwhere(y == i)
  data_slice = data[pas_slice].squeeze()
  pas_data.append({
    'paslevel': i,
    'data': data_slice,
  })
  pas_index[i] = len(pas_data) - 1
for i in pas_data:
  print('Pas {} shape: {}'.format(i['paslevel'], i['data'].shape))
```

```
  avg_for_pas = np.mean(i['data'][:,max_resp[1],:], axis=0).T
  print('Pas {} average shape: {}'.format(i['paslevel'],
avg_for_pas.shape))
  plt.plot(times, avg_for_pas, label='Pas {}'.format(i['paslevel']))
plt.axvline()
plt.axhline()
plt.axvline(x = times[max_resp[0]], color='r', label='max response')
plt.legend(loc="upper left")
plt.title('Average response for each pas level')
plt.xlabel('Time (ms)')
plt.ylabel('Amplitude')
plt.show()
```

```
Pas vector shape: (682,)
Creating pas slice for pas 1
Creating pas slice for pas 2
Creating pas slice for pas 3
Creating pas slice for pas 4
Pas 1 shape: (99, 102, 251)
Pas 1 average shape: (251,)
Pas 2 shape: (115, 102, 251)
Pas 2 average shape: (251,)
Pas 3 shape: (208, 102, 251)
Pas 3 average shape: (251,)
Pas 4 shape: (260, 102, 251)
Pas 4 average shape: (251,)
```



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

The pas_vector has length 682, thus having the same length as repetitions.

1.2.iii Describe how the amplitudes of responses are related to the four PAS-scores. Does PAS 2 behave differently than expected?

One would expect that the lower the PAS level the lower the response signal – e.g., PAS 1, not perceiving anything, would result in a low signal, and PAS 4 would result in the highest signal. This seems to correlate with what we see in plot. However, surprisingly PAS 2 has a higher max at 250ms than PAS 4 (which we would expect would have the highest).

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

2.1 Logistic regression (EH)

In [ ]:

```
# create a new array called data_1_2 that only contains PAS responses 1 and
2
data_1_2 = np.concatenate((pas_data[pas_index[1]]['data'],
pas_data[pas_index[2]]['data']), axis = 0)
print('Data for pas 1 and 2 shape: {}'.format(data_1_2.shape))
# Similarly, create a y_1_2 for the target vector
y_1_2 = y[np.where((y == 1) | (y == 2))]
print('y for pas 1 and 2 shape: {}'.format(y_1_2.shape))
#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.
X_1_2 = np.reshape(data_1_2, (data_1_2.shape[0], data_1_2.shape[1] *
data_1_2.shape[2]))
print('Reshaped data for pas 1 and 2 shape: {}'.format(X_1_2.shape))
#  and scale X_1_2
sc = StandardScaler()
sc.fit(X_1_2)
X_1_2_std = sc.transform(X_1_2)
# Do a standard LogisticRegression - make sure there is no penalty applied
lr = LogisticRegression(penalty='none', solver='lbfgs', multi_class='ovr',
n_jobs=-1)
lr.fit(X_1_2_std, y_1_2)
# we get a score of 1.0...that'd definitely an overfit, right? haha
print('Logistic regression score with no penalty:
{}'.format(lr.score(X_1_2_std, y_1_2)))
#found classes
print('Classes found in regression: {}'.format(lr.classes_))
# total coefs
coefs_orig = lr.coef_
coefs = len(lr.coef_[0])
print('Total coefs: {}'.format(coefs))
# non-zero coefs
coefs_nonzero = len(lr.coef_[0].nonzero()[0])
print('Nonzero coefs: {}'.format(coefs_nonzero))
# zero coefs
coefs_zero = coefs - coefs_nonzero
print('Zero coefs: {}'.format(coefs_zero))
# apply l1 penalty
lr = LogisticRegression(penalty='l1', solver='liblinear',
multi_class='ovr')
lr.fit(X_1_2_std, y_1_2)
print('Logistic regression score with l1 penalty:
{}'.format(lr.score(X_1_2_std, y_1_2)))
# default_threshold = np.get_printoptions()['threshold']
# np.set_printoptions(threshold=sys.maxsize)
# print(lr.coef_)
# np.set_printoptions(threshold=default_threshold)
#found classes
print('Classes found in regression: {}'.format(lr.classes_))
# total coefs
print('Total coefs: {}'.format(lr.coef_.shape))
coefs_orig = lr.coef_[0]
coefs = len(lr.coef_[0])
print('Total coefs: {}'.format(coefs))
# non-zero coefs
coefs_nonzero = len(lr.coef_[0].nonzero()[0])
print('Nonzero coefs: {}'.format(coefs_nonzero))
# zero coefs
coefs_zero = coefs - coefs_nonzero
print('Zero coefs: {}'.format(coefs_zero))
# part 2.1.3: something is wrong with the above code because there is only
```

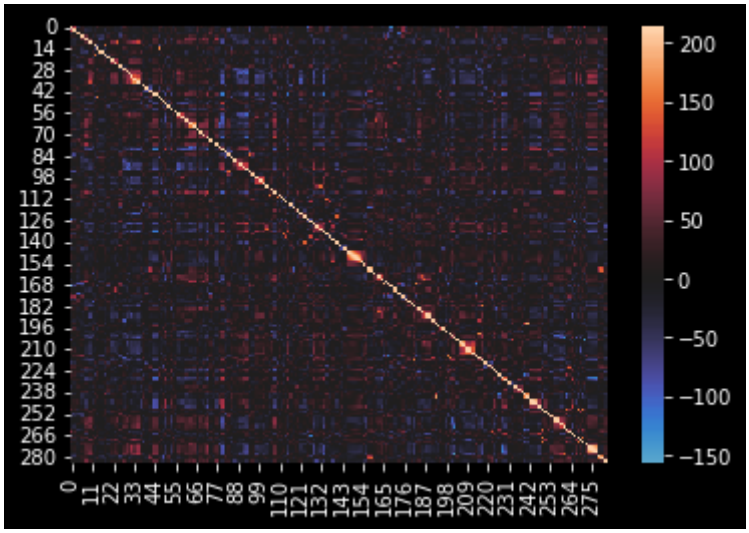
```
one non-zero coefficient.

cov_nz_index = lr.coef_[0].nonzero()[0]
X_reduced = X_1_2_std[:,cov_nz_index]

cmat = X_reduced.T @ X_reduced

sns.heatmap(cmat, cmap='icefire', center = 0)
plt.show()
```

```
Data for pas 1 and 2 shape: (214, 102, 251)
y for pas 1 and 2 shape: (214,)
Reshaped data for pas 1 and 2 shape: (214, 25602)
Logistic regression score with no penalty: 1.0
Classes found in regression: [1 2]
Total coefs: 25602
Nonzero coefs: 25602
Zero coefs: 0
Logistic regression score with l1 penalty: 1.0
Classes found in regression: [1 2]
Total coefs: (1, 25602)
Total coefs: 25602
Nonzero coefs: 284
Zero coefs: 25318
```



2.1.v Are we overfitting? Besides the score, what would make you suspect that we are overfitting?

As we get a score of 1.0, it seems like an overfitting situation – besides the score of 1.0 we also suspect overfitting based on the lack of model-penalties.

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

We found there to be 215 non-zero coefficients.

2.1.vii Plot the covariance of the features using plt.imshow. Compared to the plot from 1.1.iii, do we see less covariance?

It could seem as there is less covariance, it is however super difficult to assess this from these complex plots still. <3

Exercise 2.2 cross validation (VK)

In [ ]:

```
def equalize_targets_class(data, y):
    np.random.seed(7)
    targets = np.unique(y) ## find the number of targets
    if len(targets) < 2:
        raise NameError("at least 2 targets required")
    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
    choices = []
    for i in range(len(targets)):
        choices.append(np.random.choice(indices[i], size=min_count,
replace=False))

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

    return new_data, new_y
```

2.2.iii Stratified 5-fold cross validation logistic model (KV)

In [ ]:

```
X_1_2_equal, y_1_2_equal = equalize_targets_class(data_1_2, y_1_2)
X_1_2_equal = np.reshape(X_1_2_equal, (X_1_2_equal.shape[0],
X_1_2_equal.shape[1] * X_1_2_equal.shape[2]))
sc.fit(X_1_2_equal)
X_1_2_equal_std = sc.transform(X_1_2_equal)
cv = StratifiedKFold(n_splits=5)
lr = LogisticRegression(penalty='none', solver='lbfgs', multi_class='ovr',
n_jobs=-1) # no regularisation
print('Getting cross validation score (Stratified 5-fold)...')
scores = cross_val_score(lr, X_1_2_equal_std, y_1_2_equal, cv=cv)
print('Cross validation score (Stratified 5-fold):
{}'.format(np.mean(scores)))
```

```
Getting cross validation score (Stratified 5-fold)...
Cross validation score (Stratified 5-fold): 0.4746153846153846
```

2.2.iv Stratified 5-fold cross validation logistic model with l2 (LR)

In [ ]:

```
best_score = 0.0
for c in [1e5, 1e1, 1e-5]:
  print('Getting cross validation score (Stratified 5-fold, C=
{})...'.format(c))
  lr = LogisticRegression(penalty='l2', solver='lbfgs', multi_class='ovr',
C=c, n_jobs=-1)
  scores = cross_val_score(lr, X_1_2_equal_std, y_1_2_equal, cv=cv)
  score = np.mean(scores)
  print('Cross validation score (Stratified 5-fold, C={}): {}'.format(c,
score))
  if score >= best_score:
    best_score = score
    best_c = c
print('Best C: {}'.format(best_c))
```

```
Getting cross validation score (Stratified 5-fold, C=100000.0)...
Cross validation score (Stratified 5-fold, C=100000.0): 0.4746153846153846
```

```
Getting cross validation score (Stratified 5-fold, C=10.0)...
Cross validation score (Stratified 5-fold, C=10.0): 0.4746153846153846
Getting cross validation score (Stratified 5-fold, C=1e-05)...
Cross validation score (Stratified 5-fold, C=1e-05): 0.45487179487179485
Best C: 10.0
```

2.2.iv In the best-scoring of these models, how many more/fewer predictions are correct (on average)?

It seems as if C=1e-05 gives the worst score at 0.455 and C=10.0 gives the best score at 0.475. The difference between them is thus 0.02. It is however surprising that C=1e5 seems to give almost the exact same number as C=1e1. This indicates that reularization is not making a large difference for these data.

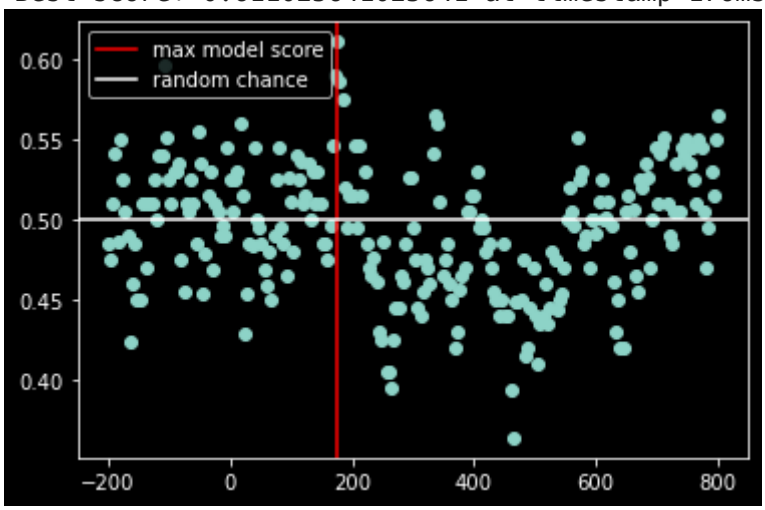
2.2.v now fit a model for each time sample using best c (EH)

In [ ]:

```
# 2.2.v
scores = []
lr = LogisticRegression(penalty='l2', solver='newton-cg',
multi_class='ovr', C=best_c, n_jobs=-1)
X_1_2_equal, y_1_2_equal = equalize_targets_class(data_1_2, y_1_2)
print('Fitting l2 model (c={}) for time sample: '.format(best_c), end='',
flush=True)
for time_ofs in range(data_1_2.shape[2]):
  X_1_2_equal_time = X_1_2_equal[:, :, time_ofs].squeeze()
  print('{}'.format(time_ofs), end='..', flush=True)
  # print('Shape of X: {}'.format(X_1_2_equal_time.shape))
  sc.fit(X_1_2_equal_time)
  X_1_2_equal_std = sc.transform(X_1_2_equal_time)
  # print('Getting Cross validation score (Stratified 5-fold, C={}, time=
{}ms)'.format(best_c, times[time_ofs]))
  time_scores = cross_val_score(lr, X_1_2_equal_std, y_1_2_equal, cv=cv)
  score = np.mean(time_scores)
  # print('Cross validation score (Stratified 5-fold, C={}, time={}ms):
{}'.format(best_c, times[time_ofs], score))
  scores.append(score)
print('\nCompleted fitting time-based models')
scores = np.array(scores)
max_score = np.unravel_index(np.argmax(scores), scores.shape)
print('Best score: {} at timestamp {}ms'.format(scores[max_score],
times[max_score[0]]))
plt.scatter(times, scores)
plt.axvline(x = times[max_score[0]], color='r', label='max model score')
plt.axhline(y = 0.5, label = 'random chance', color='w')
plt.legend(loc="upper left")
plt.show()
```

```
Fitting l2 model (c=10.0) for time sample: 0..1..2..3..4..5..6..7..8..9..10..
11..12..13..14..15..16..17..18..19..20..21..22..23..24..25..26..27..28..29..3
0..31..32..33..34..35..36..37..38..39..40..41..42..43..44..45..46..47..48..4
9..50..51..52..53..54..55..56..57..58..59..60..61..62..63..64..65..66..67..6
8..69..70..71..72..73..74..75..76..77..78..79..80..81..82..83..84..85..86..8
7..88..89..90..91..92..93..94..95..96..97..98..99..100..101..102..103..104..1
05..106..107..108..109..110..111..112..113..114..115..116..117..118..119..12
0..121..122..123..124..125..126..127..128..129..130..131..132..133..134..13
5..136..137..138..139..140..141..142..143..144..145..146..147..148..149..15
0..151..152..153..154..155..156..157..158..159..160..161..162..163..164..16
5..166..167..168..169..170..171..172..173..174..175..176..177..178..179..18
0..181..182..183..184..185..186..187..188..189..190..191..192..193..194..19
5..196..197..198..199..200..201..202..203..204..205..206..207..208..209..21
0..211..212..213..214..215..216..217..218..219..220..221..222..223..224..22
5..226..227..228..229..230..231..232..233..234..235..236..237..238..239..24
```

```
0..241..242..243..244..245..246..247..248..249..250..
Completed fitting time-based models
Best score: 0.611025641025641 at timestamp 176ms
```



2.2.v 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?)

The best score 61.1% is at timestamp = 176 ms. Chance level should be at 50% as we are predicting PAS 1 and PAS 2

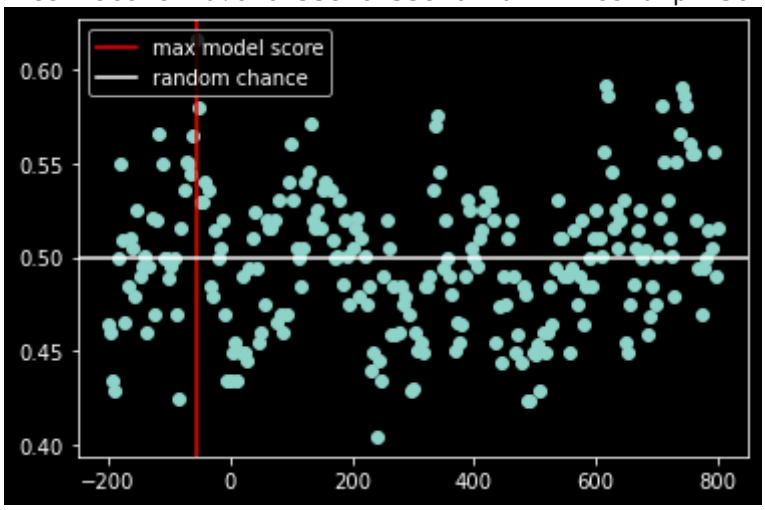
2.2.vi now fit a model for each time sample using c=1e-1 and l1 (VK)

In [ ]:

```
scores = []
lr = LogisticRegression(penalty='l1', solver='liblinear',
multi_class='ovr', C=1e-1)
X_1_2_equal, y_1_2_equal = equalize_targets_class(data_1_2, y_1_2)
print('Fitting l1 model (c=1e-1) for time sample: ', end='', flush=True)
for time_ofs in range(data_1_2.shape[2]):
  X_1_2_equal_time = X_1_2_equal[:, :, time_ofs].squeeze()
  print('{}'.format(time_ofs), end='..', flush=True)
  # print('Shape of X: {}'.format(X_1_2_equal_time.shape))
  sc.fit(X_1_2_equal_time)
  X_1_2_equal_std = sc.transform(X_1_2_equal_time)
  # print('Getting Cross validation score (Stratified 5-fold, C={}, time=
{}ms)'.format(best_c, times[time_ofs]))
  time_scores = cross_val_score(lr, X_1_2_equal_std, y_1_2_equal, cv=cv)
  score = np.mean(time_scores)
  # print('Cross validation score (Stratified 5-fold, C={}, time={}ms):
{}'.format(best_c, times[time_ofs], score))
  scores.append(score)
print('\nCompleted fitting time-based models')
scores = np.array(scores)
max_score = np.unravel_index(np.argmax(scores), scores.shape)
print('Best score: {} at timestamp {}ms'.format(scores[max_score],
times[max_score[0]]))
plt.scatter(times, scores)
plt.axvline(x = times[max_score[0]], color='r', label='max model score')
plt.axhline(y = 0.5, label = 'random chance', color='w')
plt.legend(loc="upper left")
plt.show()
```

```
Fitting l1 model (c=1e-1) for time sample: 0..1..2..3..4..5..6..7..8..9..10..
11..12..13..14..15..16..17..18..19..20..21..22..23..24..25..26..27..28..29..3
0..31..32..33..34..35..36..37..38..39..40..41..42..43..44..45..46..47..48..4
9..50..51..52..53..54..55..56..57..58..59..60..61..62..63..64..65..66..67..6
8..69..70..71..72..73..74..75..76..77..78..79..80..81..82..83..84..85..86..8
```

```
7..88..89..90..91..92..93..94..95..96..97..98..99..100..101..102..103..104..1
05..106..107..108..109..110..111..112..113..114..115..116..117..118..119..12
0..121..122..123..124..125..126..127..128..129..130..131..132..133..134..13
5..136..137..138..139..140..141..142..143..144..145..146..147..148..149..15
0..151..152..153..154..155..156..157..158..159..160..161..162..163..164..16
5..166..167..168..169..170..171..172..173..174..175..176..177..178..179..18
0..181..182..183..184..185..186..187..188..189..190..191..192..193..194..19
5..196..197..198..199..200..201..202..203..204..205..206..207..208..209..21
0..211..212..213..214..215..216..217..218..219..220..221..222..223..224..22
5..226..227..228..229..230..231..232..233..234..235..236..237..238..239..24
0..241..242..243..244..245..246..247..248..249..250..
Completed fitting time-based models
Best score: 0.6161538461538462 at timestamp -56ms
```



2.2.vi what are the time points when classification is best? (make a plot)?

classification accuracy was best (61.62%) at time point -56ms.

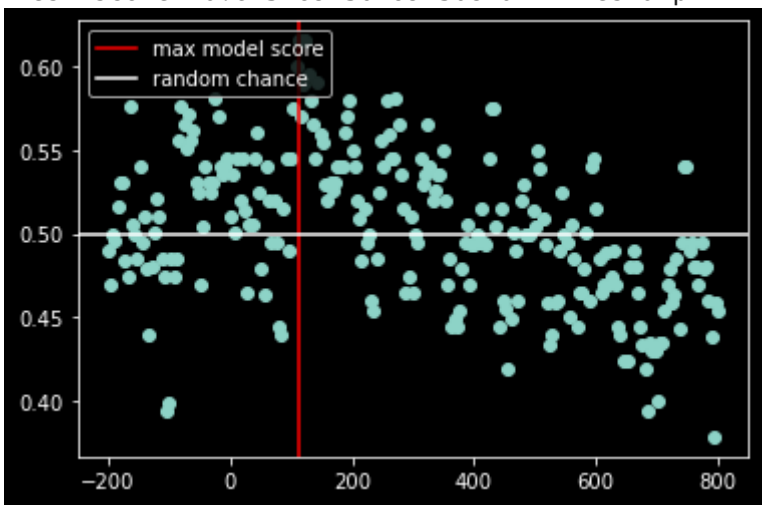
2.2.vii fit again but for PAS 1 and 4 (KV)

In [ ]:

```
data_1_4 = np.concatenate((pas_data[pas_index[1]]['data'],
pas_data[pas_index[4]]['data']), axis = 0)
print('Data for pas 1 and  4 shape: {}'.format(data_1_4.shape))
y_1_4 = y[np.where((y == 1) | (y == 2))]
print('y for pas 1 and 4 shape: {}'.format(y_1_4.shape))
#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.
X_1_4 = np.reshape(data_1_4, (data_1_4.shape[0], data_1_4.shape[1] *
data_1_4.shape[2]))
scores = []
lr = LogisticRegression(penalty='l1', solver='liblinear',
multi_class='ovr', C=1e-1)
X_1_4_equal, y_1_4_equal = equalize_targets_class(data_1_4, y_1_4)
print('Fitting l1 model (c=1e-1) for time sample: ', end='', flush=True)
for time_ofs in range(data_1_4.shape[2]):
  X_1_4_equal_time = X_1_4_equal[:, :, time_ofs].squeeze()
  print('{}'.format(time_ofs), end='..', flush=True)
  # print('Shape of X: {}'.format(X_1_4_equal_time.shape))
  sc.fit(X_1_4_equal_time)
  X_1_4_equal_std = sc.transform(X_1_4_equal_time)
  # print('Getting Cross validation score (Stratified 5-fold, C={}, time=
{}ms)'.format(best_c, times[time_ofs]))
  time_scores = cross_val_score(lr, X_1_4_equal_std, y_1_4_equal, cv=cv)
  score = np.mean(time_scores)
  # print('Cross validation score (Stratified 5-fold, C={}, time={}ms):
{}'.format(best_c, times[time_ofs], score))
```

```
  scores.append(score)
print('\nCompleted fitting time-based models')
scores = np.array(scores)
max_score = np.unravel_index(np.argmax(scores), scores.shape)
print('Best score: {} at timestamp {}ms'.format(scores[max_score],
times[max_score[0]]))
plt.scatter(times, scores)
plt.axvline(x = times[max_score[0]], color='r', label='max model score')
plt.axhline(y = 0.5, label = 'random chance', color='w')
plt.legend(loc="upper left")
plt.show()
```

```
Data for pas 1 and  4 shape: (359, 102, 251)
y for pas 1 and 4 shape: (214,)
Fitting l1 model (c=1e-1) for time sample: 0..1..2..3..4..5..6..7..8..9..10..
11..12..13..14..15..16..17..18..19..20..21..22..23..24..25..26..27..28..29..3
0..31..32..33..34..35..36..37..38..39..40..41..42..43..44..45..46..47..48..4
9..50..51..52..53..54..55..56..57..58..59..60..61..62..63..64..65..66..67..6
8..69..70..71..72..73..74..75..76..77..78..79..80..81..82..83..84..85..86..8
7..88..89..90..91..92..93..94..95..96..97..98..99..100..101..102..103..104..1
05..106..107..108..109..110..111..112..113..114..115..116..117..118..119..12
0..121..122..123..124..125..126..127..128..129..130..131..132..133..134..13
5..136..137..138..139..140..141..142..143..144..145..146..147..148..149..15
0..151..152..153..154..155..156..157..158..159..160..161..162..163..164..16
5..166..167..168..169..170..171..172..173..174..175..176..177..178..179..18
0..181..182..183..184..185..186..187..188..189..190..191..192..193..194..19
5..196..197..198..199..200..201..202..203..204..205..206..207..208..209..21
0..211..212..213..214..215..216..217..218..219..220..221..222..223..224..22
5..226..227..228..229..230..231..232..233..234..235..236..237..238..239..24
0..241..242..243..244..245..246..247..248..249..250..
Completed fitting time-based models
Best score: 0.6157692307692308 at timestamp 112ms
```



2.2.vii What are the time points when classification is best? ... what is the chance level for this analysis?

Best classification (61.58%) is at time point 112ms. Chance level for this analysis should be 50% as we are working with PAS 1 and PAS 4.

2.3 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?

As our scores is all above chance level (however not a lot) it seems to be possible to do pairwise classification. It is quite surprising that the accuracy level of PAS 1 vs 2 and PAS 1 vs 4 are almost

identical, though at very different time points.

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

3.1 Support Vector Machine Classification (LR)

3.1.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 [ ]:
```
X_equal, y_equal = equalize_targets_class(data, y)
print('X shape: {}'.format(X_equal.shape))
print('y shape: {}'.format(y_equal.shape))

X_equal_f = np.reshape(X_equal, (X_equal.shape[0], X_equal.shape[1] *
X_equal.shape[2]))

sc.fit(X_equal_f)
X_equal_std = sc.transform(X_equal_f)

print('X reshaped shape: {}'.format(X_equal_std.shape))
```

```
X shape: (396, 102, 251)
y shape: (396,)
X reshaped shape: (396, 25602)
```

3.1.ii Linear kernel classifier (LR)

In [ ]:
```
cv = StratifiedKFold()
svm = SVC(kernel='linear')
scores_svm = cross_val_score(svm, X_equal_std, y_equal, cv=cv)
print(np.mean(scores_svm))
```

```
0.2928164556962025
```

3.1.ii Radial basis (EH)

In [ ]:
```
cv = StratifiedKFold()
svm = SVC(kernel='rbf')
scores_svm = cross_val_score(svm, X_equal_std, y_equal, cv=cv)
print(np.mean(scores_svm))
```

```
0.3333544303797468
```

3.1.ii) Which one is better predicting the category?

Based on the cross-validation results, the linear kernel classifier achieved 29.9% accuracy, and the radial basis kernel classifier achieved 33.4% accuracy, so for these data the radial basis kernel classifier is the best predictor.

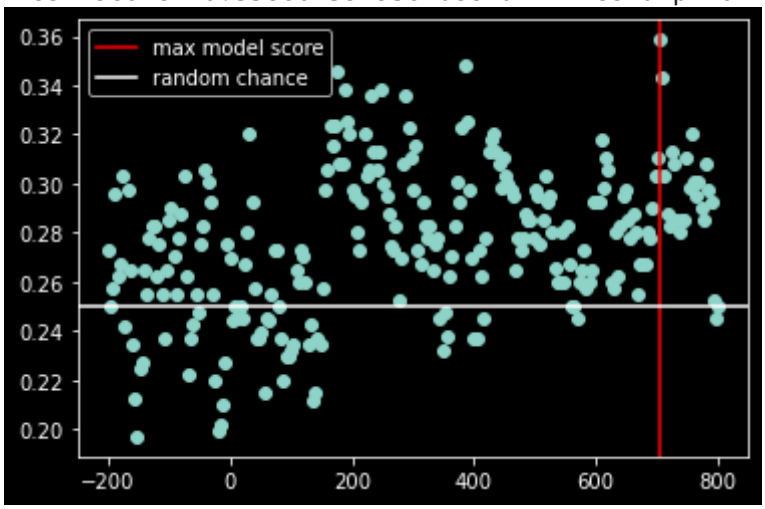
3.3.iv Sample-by-sample analysis (VK)

In [ ]:
```
# radial basis score better than linear
cv = StratifiedKFold()
svm = SVC(kernel='rbf')

scores = []
print(X_equal.shape)
```

```
print('Fitting rbv svm model time sample: ', end='', flush=True)
for time_ofs in range(X_equal.shape[2]):
  X_equal_time = X_equal[:, :, time_ofs].squeeze()
  print('{}'.format(time_ofs), end='..', flush=True)
  # print('Shape of X: {}'.format(X_1_2_equal_time.shape))
  sc.fit(X_equal_time)
  X_equal_std = sc.transform(X_equal_time)
  # print('Getting Cross validation score (Stratified 5-fold, C={}, time=
{}ms)'.format(best_c, times[time_ofs]))
  time_scores = cross_val_score(svm, X_equal_std, y_equal, cv=cv)
  score = np.mean(time_scores)
  # print('Cross validation score (Stratified 5-fold, C={}, time={}ms):
{}'.format(best_c, times[time_ofs], score))
  scores.append(score)
print('\nCompleted fitting time-based models')
scores = np.array(scores)
max_score = np.unravel_index(np.argmax(scores), scores.shape)
print('Best score: {} at timestamp {}ms'.format(scores[max_score],
times[max_score[0]]))
plt.scatter(times, scores)
# random chance is 1/4
plt.axvline(x = times[max_score[0]], color='r', label='max model score')
plt.axhline(y = 0.25, label = 'random chance', color='w')
plt.legend(loc="upper left")
plt.show()
```

```
(396, 102, 251)
Fitting rbv svm model time sample: 0..1..2..3..4..5..6..7..8..9..10..11..12..
13..14..15..16..17..18..19..20..21..22..23..24..25..26..27..28..29..30..31..3
2..33..34..35..36..37..38..39..40..41..42..43..44..45..46..47..48..49..50..5
1..52..53..54..55..56..57..58..59..60..61..62..63..64..65..66..67..68..69..7
0..71..72..73..74..75..76..77..78..79..80..81..82..83..84..85..86..87..88..8
9..90..91..92..93..94..95..96..97..98..99..100..101..102..103..104..105..10
6..107..108..109..110..111..112..113..114..115..116..117..118..119..120..12
1..122..123..124..125..126..127..128..129..130..131..132..133..134..135..13
6..137..138..139..140..141..142..143..144..145..146..147..148..149..150..15
1..152..153..154..155..156..157..158..159..160..161..162..163..164..165..16
6..167..168..169..170..171..172..173..174..175..176..177..178..179..180..18
1..182..183..184..185..186..187..188..189..190..191..192..193..194..195..19
6..197..198..199..200..201..202..203..204..205..206..207..208..209..210..21
1..212..213..214..215..216..217..218..219..220..221..222..223..224..225..22
6..227..228..229..230..231..232..233..234..235..236..237..238..239..240..24
1..242..243..244..245..246..247..248..249..250..
Completed fitting time-based models
Best score: 0.3586075949367088 at timestamp 704ms
```



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

It seems that the model would perform above random chance at classification of subjective experience for 200-250ms although the accuracy is still not high, ranging from around 0.27-0.34.

Exercise 3.2 Test/train (KV)

In [ ]:

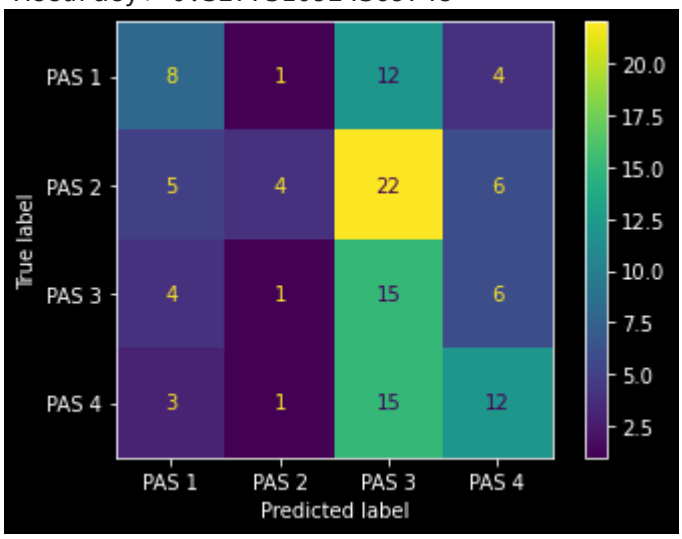
```
X_train, X_test, y_train, y_test = train_test_split(X_equal, y_equal,
test_size=0.3)
print(X_train.shape)
print(y_train.shape)
# assignment says: This time your features are the number of sensors
multiplied by the number of samples.
# So, we need to reshape the data to be (n_samples, n_features)
X_train_f = np.reshape(X_train, (X_train.shape[0], X_train.shape[1] *
X_train.shape[2]))
X_test_f = np.reshape(X_test, (X_test.shape[0], X_test.shape[1] *
X_test.shape[2]))

sc.fit(X_train_f)
X_train_f_std = sc.transform(X_train_f)
X_test_f_std = sc.transform(X_test_f)

svm = SVC(kernel='rbf')

svm.fit(X_train_f_std, y_train)
predictions = svm.predict(X_test_f_std)
print('Accuracy: {}'.format(accuracy_score(y_test, predictions)))
# confusion matrix
# print(confusion_matrix(y_test, predictions))
ConfusionMatrixDisplay.from_predictions(y_test, predictions,
display_labels=['PAS 1', 'PAS 2', 'PAS 3', 'PAS 4'])
plt.show()
```

```
(277, 102, 251)
(277,)
Accuracy: 0.3277310924369748
```



3.2.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?

The confusion matrix shows a bias towards predicting PAS 3, which can be seen from the PAS 3 predicted label on the x axis, this should not come from PAS 3 being the most frequent rating, because the samples were stratified. Of the predicted labels classified as PAS 1, 2 and 4, the

prediction was correct most frequently, which fits with subjects indicating that they either not aware of the stimulus (PAS 1) or did not doubt their experience of the stimulus (PAS 4). The PAS 3 label was most frequently misclassified, especially when the true label was PAS 2, but both 2 and 3 ratings that relate to uncertainty and ambiguity.