practical_exercise_8 , Methods 3, 2021, autumn semester

Linus Backström

1.12.2021

```
import numpy as np
import matplotlib.pyplot as plt
```

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

```
path = ('C:/Users\linus/Documents/GitHub/github_methods_3/week_08')

path = os.path.join(path, 'megmag_data.npy')

data = np.load(path)
```

```
i. The data is a 3-dimensional array. The first dimension is number of repetitions of a visual stimulus
```

```
data.shape
```

```
## (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 ever
```

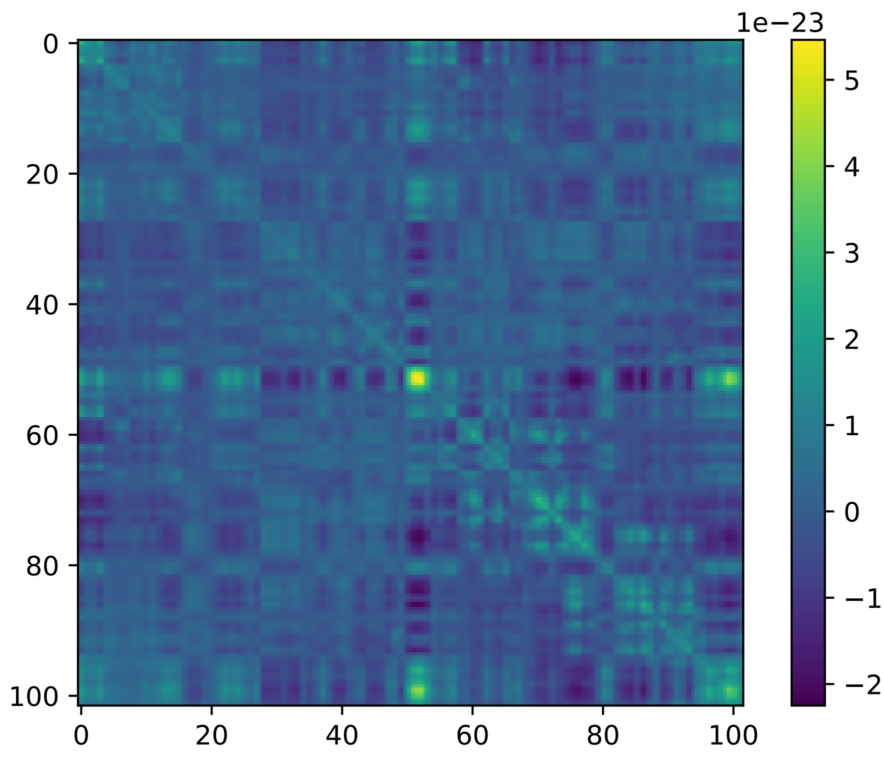
```
times = np.arange(-200, 801, 4)
```

```
iii. Create the sensor covariance matrix $\Sigma_{XX}$: $$\Sigma_{XX} = \frac 1 N \sum_{i=1}^N XX^T$$ $I
```

```
X = data[0,:,:]

n_rep = 682


poop = np.zeros(shape = (102, 102))

# creating the covariance matrix
for rep in range(0,682):
  poop += data[rep, :, :] @ data[rep, :, :].T

X = poop/682

import matplotlib.pyplot as plt
plt.figure()
plt.imshow(X)
plt.colorbar()
```

```
## <matplotlib.colorbar.Colorbar object at 0x00000000355E12E0>
```

```
plt.show()
```



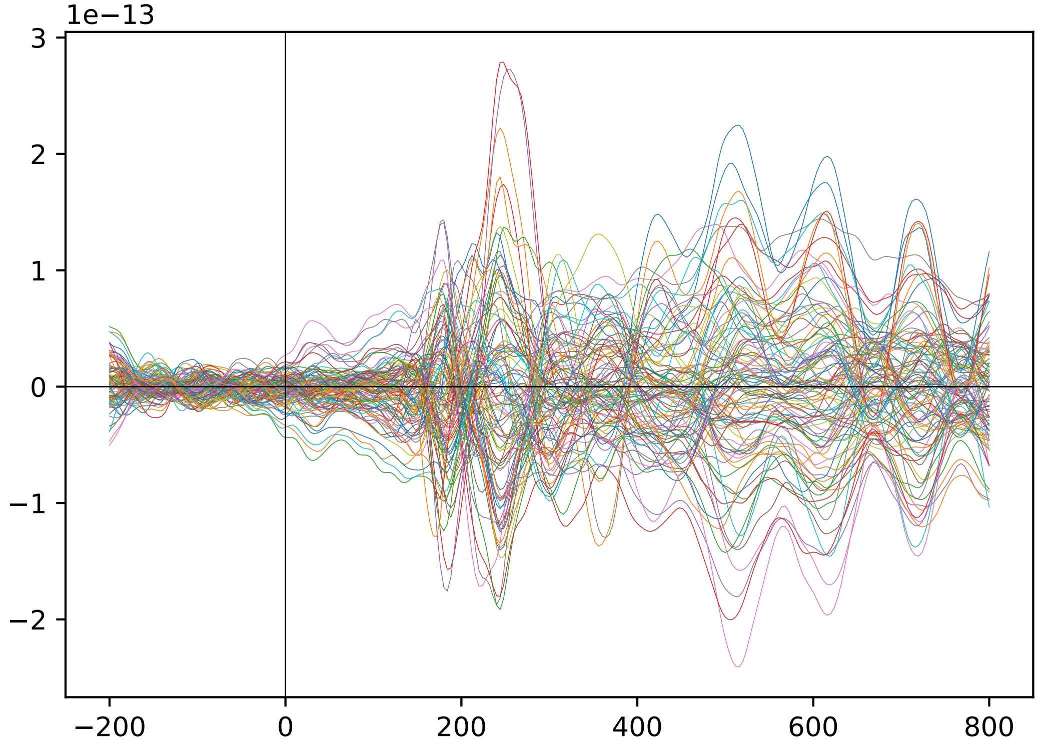
iv. Make an average over the repetition dimension using `np.mean` - use the `axis` argument. (The resul

```
rep_avg = np.mean(data, axis=0).T # so that repetition is the 1st dimension
rep_avg.shape
```

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

v. Plot the magnetic field (based on the average) as it evolves over time for each of the sensors (a lin

```python
plt.figure()
plt.plot(times, rep_avg, linewidth=.3)
plt.axvline(x = 0, color='black', linewidth=.5)
plt.axhline(y = 0, color='black', linewidth=.5)
plt.show()
```



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

```python
perse = np.argmax(rep_avg)  # 18435
# it shows the linear index of the max value

paska = rep_avg.shape  # 251, 102

max_mf_ind = np.unravel_index(perse, paska)
print(max_mf_ind)
```

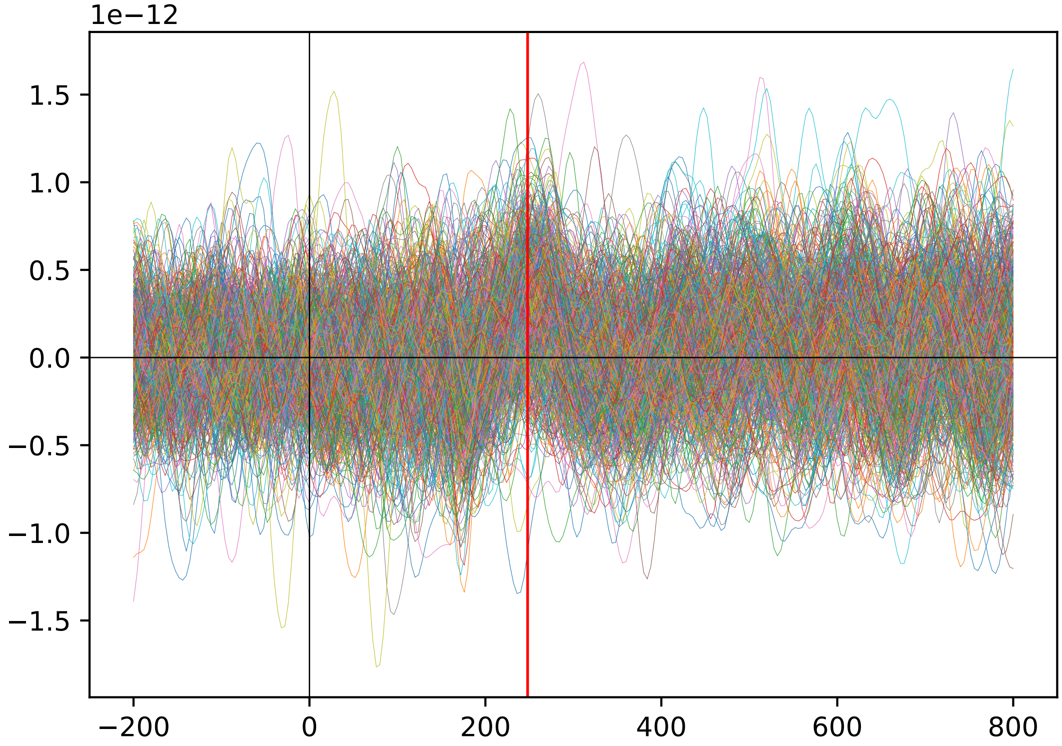
```
## (112, 73)
```

vii. Plot the magnetic field for each of the repetitions (a line for each) for the sensor that has the m

```python
max_sensor = max_mf_ind[1]
max_time = max_mf_ind[0]

sensor_matrix = data[:, max_sensor, :]
sensor_matrix.shape # 682, 251
```

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

```
plt.figure()
plt.plot(times, sensor_matrix.T, linewidth=.2)
plt.axvline(x = 0, color = 'black', linewidth=.5)
plt.axvline(x = times[max_time], color = 'red', linewidth=1)
plt.axhline(y = 0, color = 'black', linewidth=.5)
plt.show()
```



`viii. Describe in your own words how the response found in the average is represented in the single repe`

We use the average plot to see that there is one sensor that has the highest activation. We then plot only that sensor, showing all repetitions individually. By singling out sensor 73, we filter out the noise of the rest of the sensors, increasing the signal-to-noise ratio. This helps us to see that there is a clear peak for nearly all repetitions at 248 ms in the sensor that recorded the highest activation overall. It can also be seen that the minimum values are clearly higher around 248 ms.

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

```
path2 = ('C:/Users\linus/Documents/GitHub/github_methods_3/week_08')

path2 = os.path.join(path2, 'pas_vector.npy')

y = np.load(path2)

y.shape
```

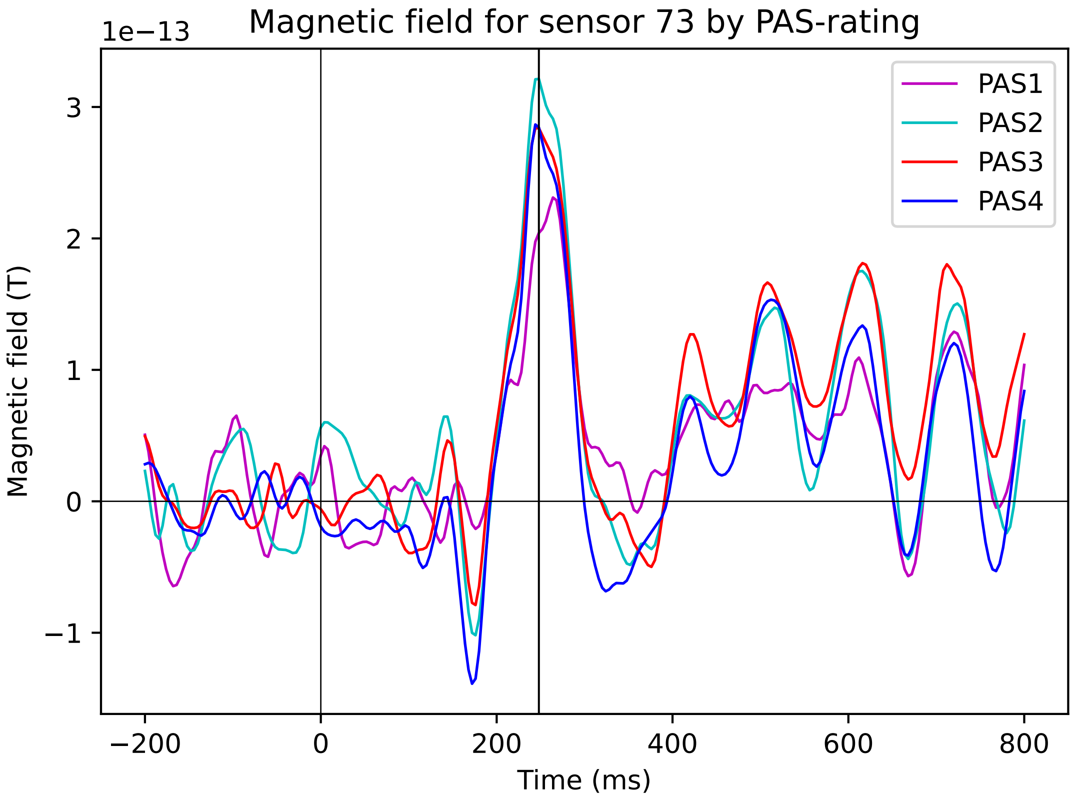
```
## (682,)
```

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

It is the same in all dimensions.

```
ii. Now make four averages (As in Exercise 1.1.iii), one for each PAS rating, and plot the four time co
```

```
# variables for each of the PAS-ratings
pas1 = data[np.where(y==1)]
pas2 = data[np.where(y==2)]
pas3 = data[np.where(y==3)]
pas4 = data[np.where(y==4)]

# get mean
pas1_mean = np.mean(pas1, axis=0)
pas2_mean = np.mean(pas2, axis=0)
pas3_mean = np.mean(pas3, axis=0)
pas4_mean = np.mean(pas4, axis=0)

plt.figure()
plt.plot(times, pas1_mean[max_sensor,], 'm-', linewidth = 1)
plt.plot(times, pas2_mean[max_sensor,], 'c-', linewidth = 1)
plt.plot(times, pas3_mean[max_sensor,], 'r-', linewidth = 1)
plt.plot(times, pas4_mean[max_sensor,], 'b-', linewidth = 1)
plt.axvline(x=0, color = 'k', linewidth=.5)
plt.axvline(x=times[max_time], color = 'k', linewidth=.75)
plt.axhline(y=0, color = 'k', linewidth=.5)
plt.xlabel('Time (ms)')
plt.ylabel('Magnetic field (T)')
plt.title('Magnetic field for sensor 73 by PAS-rating')
plt.legend(['PAS1', 'PAS2', 'PAS3', 'PAS4'])
plt.show()
```



iii. Notice that there are two early peaks (measuring visual activity from the brain), one before 200 m

The activations for PAS1 are lower than the rest, as expected. Perhaps surprisingly though, PAS2-ratings show the highest activation of all. One explanations could be that ratings of PAS2 mean that there is barely enough information to form an accurate perception, meaning the visual areas of the brain have to work harder.

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

```
data_1_2 = data[np.where((y==1) | (y==2))]
data_1_2.shape
```

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

```
y_1_2 = y[np.where((y==1) | (y==2))]
y_1_2.shape
```

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

ii. Scikit-learn expects our observations (`data_1_2`) to be in a 2d-array, which has samples (repetiti

```
X_1_2 = np.reshape(data_1_2, newshape = (214, 102*251))
X_1_2.shape
```

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

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

```
from sklearn.preprocessing import StandardScaler

scaledata = StandardScaler()
X_1_2_scaled = scaledata.fit_transform(X_1_2)
# standardized so it's around 0 and easier to compare
```

iv. Do a standard `LogisticRegression` - can be imported from `sklearn.linear_model` - make sure there

```
from sklearn.linear_model import LogisticRegression

# Higher value for C means less penalizing

log_reg = LogisticRegression(penalty='none', solver='lbfgs').fit(X_1_2_scaled, y_1_2)

log_reg.coef_[0, 1212]
```

```
## 0.013714444657457942
```

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

```
log_reg.score(X_1_2_scaled, y_1_2)
```

```
## 1.0
```

There is clearly overfitting, because 100 % of the labels are classified correctly.

vi. Now apply the _L1_ penalty instead - how many of the coefficients (`.coef_`) are non-zero after thi

```
log_reg_l1 = LogisticRegression(penalty='l1', solver='liblinear').fit(X_1_2_scaled, y_1_2)

log_reg_l1.score(X_1_2_scaled, y_1_2)
```

```
## 1.0
```

vii. Create a new reduced X that only includes the non-zero coefficients - show the covariance of the

```
np.random.seed(69)

coefficients = log_reg_l1.coef_.flatten()

non_zero = coefficients !=0

X_1_2.shape  # 214, 25602
```

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

```
X_1_2_reduced = X_1_2_scaled[:, non_zero]
# takes all 214 rows, and picks the columns with non-zero values

X_1_2_reduced.shape # 214, 25602

# Remember that 6x2 @ 2x6 = 6x6 matrix,
# while         2x6 @ 6x6 = 2x2 matrix
```

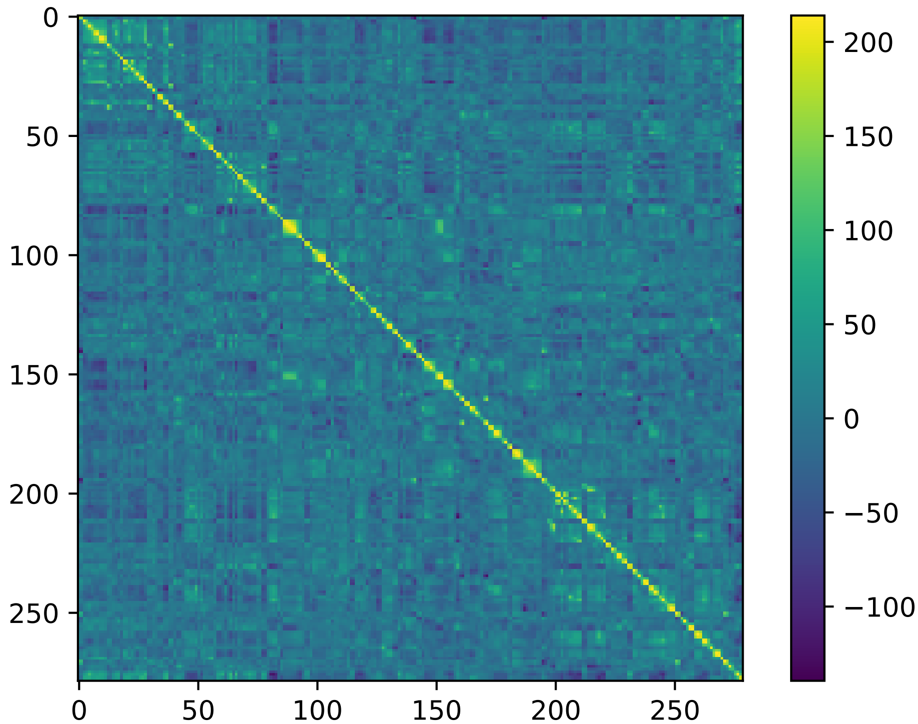
```
## (214, 279)
```

```
X_lol = X_1_2_reduced.T @ X_1_2_reduced
```

```
plt.figure()
plt.imshow(X_lol)
plt.colorbar()
```

```
## <matplotlib.colorbar.Colorbar object at 0x0000000048E4BF70>
```

```
plt.show()
```



No, we see more covariance.

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`

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold as skf
```

```
ii. To make sure that our training data sets are not biased to one target (PAS) or the other, create `y
```

```
# Exercise 2.2.ii
def equalize_targets_binary(data, y):
    np.random.seed(69)
    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

equal = equalize_targets_binary(data_1_2, y_1_2)

y_1_2_equal = equal[1]
X_1_2_equal = equal[0]
```

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

```
X_1_2_equal.shape # 198, 102, 251

# reshape into two dimensions
```

(198, 102, 251)

```
X_1_2_equal_rs = np.reshape(X_1_2_equal, newshape = (198, 102*251))

# standardize
X_1_2_equal_rs_scaled = scaledata.fit_transform(X_1_2_equal_rs)

# create model
log_reg_cv = LogisticRegression(penalty='none', solver='lbfgs').fit(X_1_2_equal_rs_scaled, y_1_2_equal)
# solver is lbfgs to be able to specify penalty as none

# five folds
cv = skf(n_splits = 5)

# cross-validate
score = cross_val_score(log_reg_cv, X_1_2_equal_rs_scaled, y_1_2_equal, cv = cv)
print(np.mean(score))
```

0.5007692307692307

iv. Do L2-regularisation with the following `Cs=  [1e5, 1e1, 1e-5]`. Use the same kind of cross-validati

```
log_reg_l2_c1e5 = LogisticRegression(penalty='l2', solver='lbfgs', C = 1e5)

log_reg_l2_c1e1 = LogisticRegression(penalty='l2', solver='lbfgs', C = 1e1)

log_reg_l2_c1em5 = LogisticRegression(penalty='l2', solver='lbfgs', C = 1e-5)

score_1 = cross_val_score(log_reg_l2_c1e5, X_1_2_equal_rs_scaled, y_1_2_equal, cv=cv)
score_2 = cross_val_score(log_reg_l2_c1e1, X_1_2_equal_rs_scaled, y_1_2_equal, cv=cv)
score_3 = cross_val_score(log_reg_l2_c1em5, X_1_2_equal_rs_scaled, y_1_2_equal, cv=cv)
```

```
print(np.mean(score_1))
```

```
## 0.5007692307692307
```

```
print(np.mean(score_2))
```

```
## 0.49551282051282053
```

```
print(np.mean(score_3))
```

```
## 0.5465384615384615
```

??????????

```
v. Instead of fitting a model on all `n_sensors * n_samples` features, fit  a logistic regression (same
```

```
log_reg_l2_c1em5 = LogisticRegression(penalty='l2', solver='lbfgs', C = 1e-5)

horo = np.zeros(shape = (251))

for i in range(0, 251):
    X_1_2_time = X_1_2_equal[:, :, i]
    X_1_2_time = scaledata.fit_transform(X_1_2_time)
    sieni = log_reg_l2_c1em5.fit(X_1_2_time, y_1_2_equal)

    X_wow = cross_val_score(sieni, X_1_2_time, y_1_2_equal, cv=cv)
    X_wow_avg = np.mean(X_wow)

    horo[i] = X_wow_avg
```

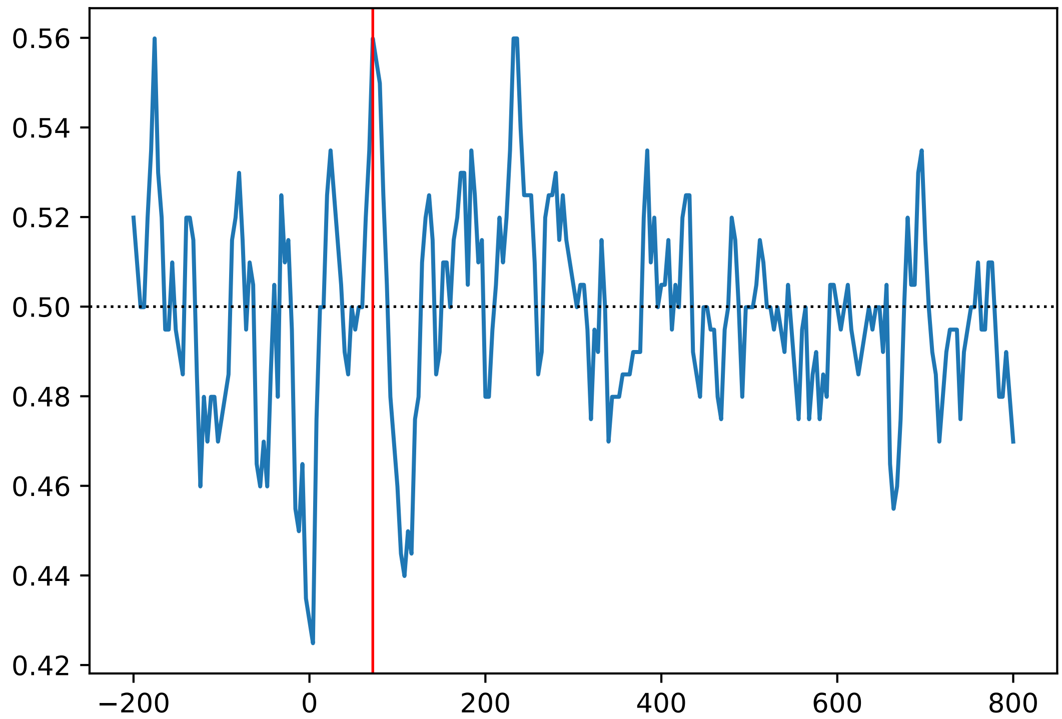
```
# find index of highest value, i.e. highest accuracy
max_0 = np.argmax(horo) # 68
times[68] # 72 ms
```

```
## 72
```

```
horo[68]
```

```
## 0.559871794871795
```

```
plt.figure()
plt.plot(times, horo)
plt.axhline(y = .5, color='black', linewidth=1, linestyle=':')
plt.axvline(x=times[max_0], color='red', linewidth=1)
plt.show()
```



Classification is best at 72 ms.

The chance level is set at .5, because there is only two options in the classification.

vi. Now do the same, but with L1 regression - set `C=1e-1` - what are the time points when classificati

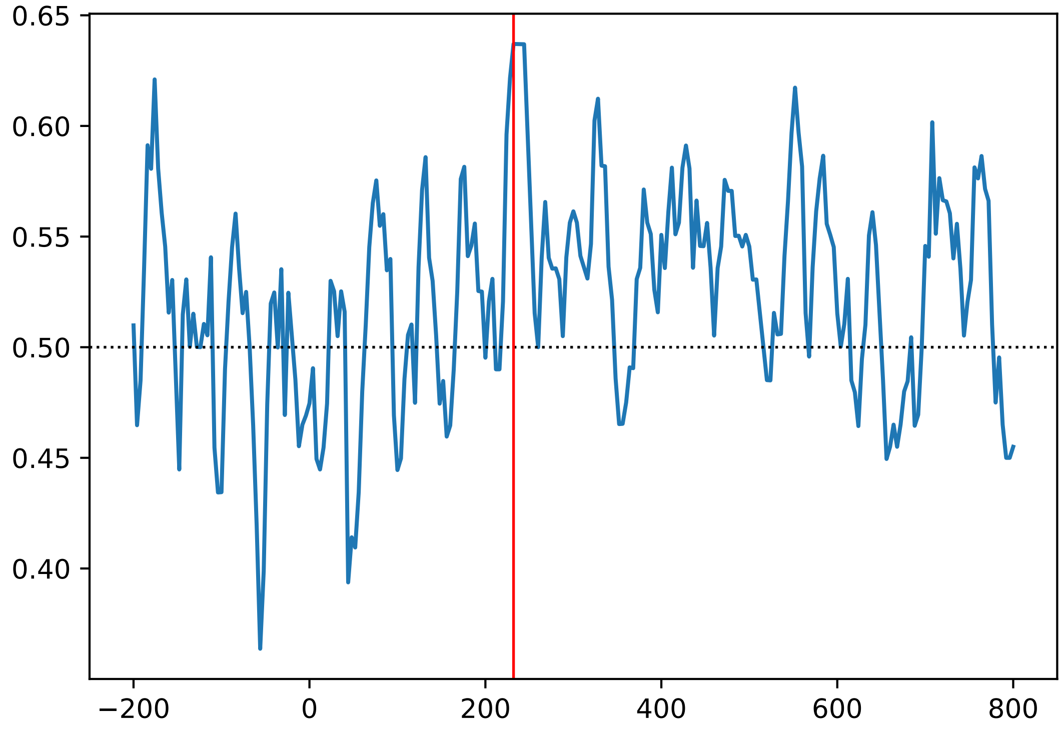
```
log_reg_l1 = LogisticRegression(penalty='l1', solver='liblinear', C = 1e-1)

horo = np.zeros(shape = (251))

for i in range(0, 251):
    X_1_2_time = X_1_2_equal[:, :, i]
    X_1_2_time = scaledata.fit_transform(X_1_2_time)
    sieni = log_reg_l1.fit(X_1_2_time, y_1_2_equal)

    X_wow = cross_val_score(sieni, X_1_2_time, y_1_2_equal, cv=cv)
    X_wow_avg = np.mean(X_wow)

    horo[i] = X_wow_avg

# find index of highest value, i.e. highest accuracy
max_0 = np.argmax(horo) # 108
times[108] # 232 ms
```

```
## 232
```

```
horo[108]
```

```
## 0.6370512820512821
```

```
plt.figure()
plt.plot(times, horo)
plt.axhline(y = .5, color='black', linewidth=1, linestyle=':')
plt.axvline(x=times[max_0], color='red', linewidth=1)
plt.show()
```



Classification is best at 232 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 da

```
# only choose PAS-ratings of 1 and 4
data_1_4 = data[np.where((y==1) | (y==4))]
data_1_4.shape
```

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

```
y_1_4 = y[np.where((y==1) | (y==4))]
y_1_4.shape

# equalize so there's as an equal amount of PAS-ratings
```

```
## (359,)
```

```
equal2 = equalize_targets_binary(data_1_4, y_1_4)

X_1_4_equal = equal2[0]
X_1_4_equal.shape
```

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

```
y_1_4_equal = equal2[1]
y_1_4_equal.shape
```

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

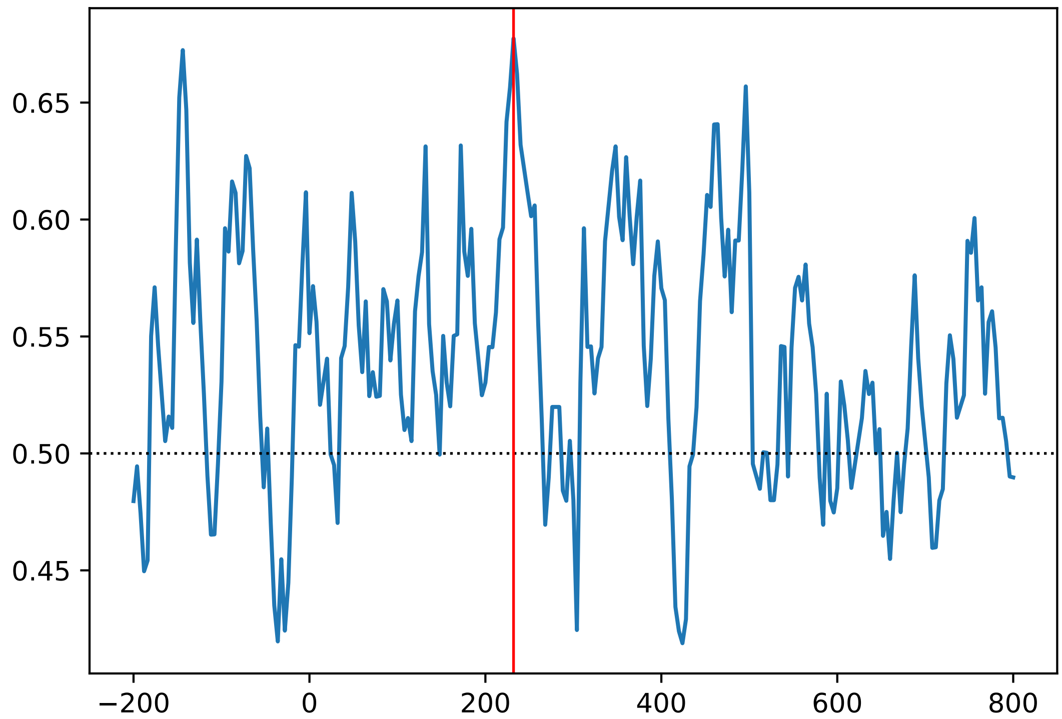
```
log_reg_l1 = LogisticRegression(penalty='l1', solver='liblinear', C = 1e-1)

horo = np.zeros(shape = (251))

for i in range(0, 251):
    X_1_4_time = X_1_4_equal[:, :, i]
    X_1_4_time = scaledata.fit_transform(X_1_4_time)
    sieni = log_reg_l1.fit(X_1_4_time, y_1_4_equal)

    X_wow = cross_val_score(sieni, X_1_4_time, y_1_4_equal, cv=cv)
    X_wow_avg = np.mean(X_wow)

    horo[i] = X_wow_avg

# find index of highest value, i.e. highest accuracy
max_0 = np.argmax(horo) # 108
times[108] # 232 ms
```

```
## 232
```

```
horo[108]
```

```
## 0.6774358974358975
```

```
plt.figure()
plt.plot(times, horo)
plt.axhline(y = .5, color='black', linewidth=1, linestyle=':')
plt.axvline(x=times[max_0], color='red', linewidth=1)
plt.show()
```



Classification is best at 232 ms, the exact same as for PAS 1 vs 4.

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?

With PAS-1 vs PAS-4, the classification is better, as we can see more values above our chance level of .5. We can also see a higher maximum value for PAS-1 vs PAS-4, and a lower minimum value for PAS-1 vs PAS-2.

With PAS-1 vs PAS-2, the classification is thus worse, which is intuitive considering that they are closer to each other in subjective experience.

We have learned that pairwise classification is dependant on 1) the timestamp, and 2) the "length" between the pairs of perception, e.g. PAS-1 vs PAS-2 or PAS-1 vs PAS-4.

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

```
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

eq = equalize_targets(data, y)

X_all_eq = eq[0]
y_all_eq = eq[1]
```

ii. Run two classifiers, one with a linear kernel and one with a radial basis (other options should be l

```
#from sklearn.model_selection import cross_val_score as cvs
#from sklearn.model_selection import StratifiedKFold as skf

X_all_eq_rs = np.reshape(X_all_eq, newshape = (396, 102*251))

X_all_eq_rs_scaled = scaledata.fit_transform(X_all_eq_rs)

from sklearn.svm import SVC

linear_classifier = SVC(kernel='linear')
lc_cv_score = cross_val_score(linear_classifier, X_all_eq_rs_scaled, y_all_eq, cv=cv)
np.mean(lc_cv_score)
```

```
## 0.2928164556962025
```

```
radial_classifier = SVC(kernel='rbf')
rc_cv_score = cross_val_score(radial_classifier, X_all_eq_rs_scaled, y_all_eq, cv=cv)
np.mean(rc_cv_score)
```

```
## 0.3333544303797468
```

iii. Run the sample-by-sample analysis (similar to Exercise 2.2.v) with the best kernel (from Exercise 3

```
horo = np.zeros(shape = (251))

for i in range(0, 251):
    pepe = X_all_eq[:, :, i]
    pepe_s = scaledata.fit_transform(pepe)

    rc_cv_score = cross_val_score(radial_classifier, pepe_s, y_all_eq, cv=cv)
    rc_cv_score_mean = np.mean(rc_cv_score)

    horo[i] = rc_cv_score_mean
```

```
print (pepe_s)
```

```
## [[-0.56439705  0.01674436 -0.57831741 ... -0.47997217 -0.65842522
##   -0.94390272]
##  [-0.34225372 -0.50299186 -0.40130082 ... -0.37658195 -0.35970541
##   -1.09766649]
##  [-0.52929071 -0.18619107 -0.04398457 ... -0.50821576 -0.28559389
##   -0.41736483]
##  ...
##  [ 0.74578682  0.61364947  0.94946873 ...  2.24400428  2.54631478
##    2.35582733]
##  [ 0.98537619  1.22176119  1.49060886 ...  0.57566884  0.75385598
##    0.31740526]
##  [ 0.68141374  0.35570286  0.19514428 ...  0.81996179  0.74436929
##    0.86867203]]
```

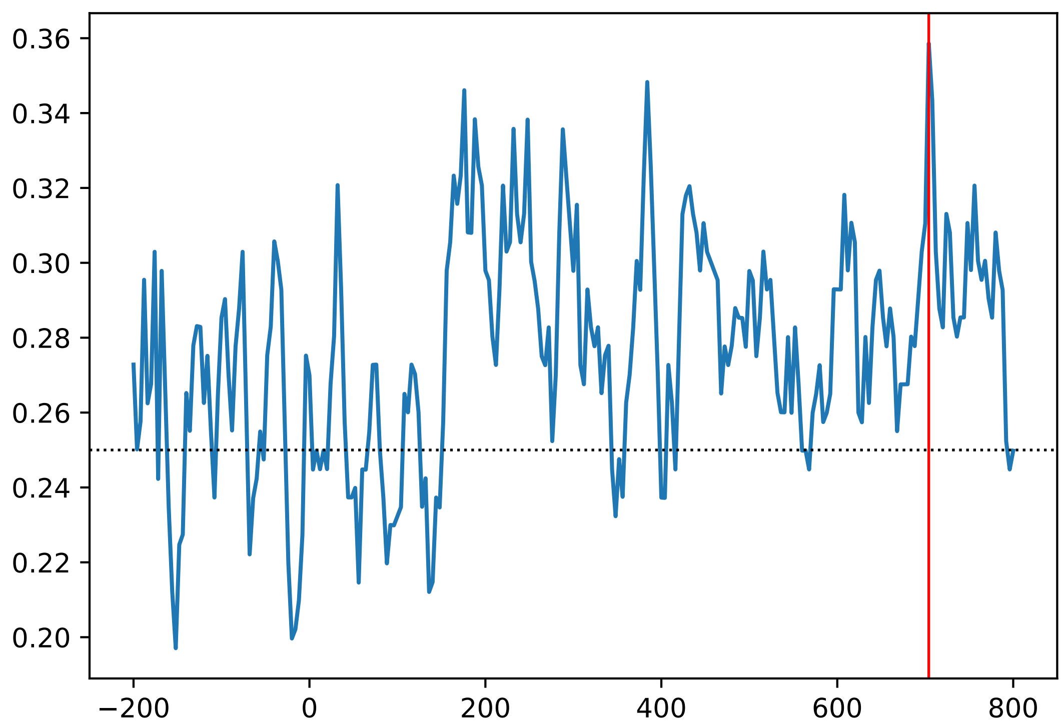
```
# find index of highest value, i.e. highest accuracy
max_0 = np.argmax(horo) # 68
times[68]
```

```
## 72
```

```
horo[68]
```

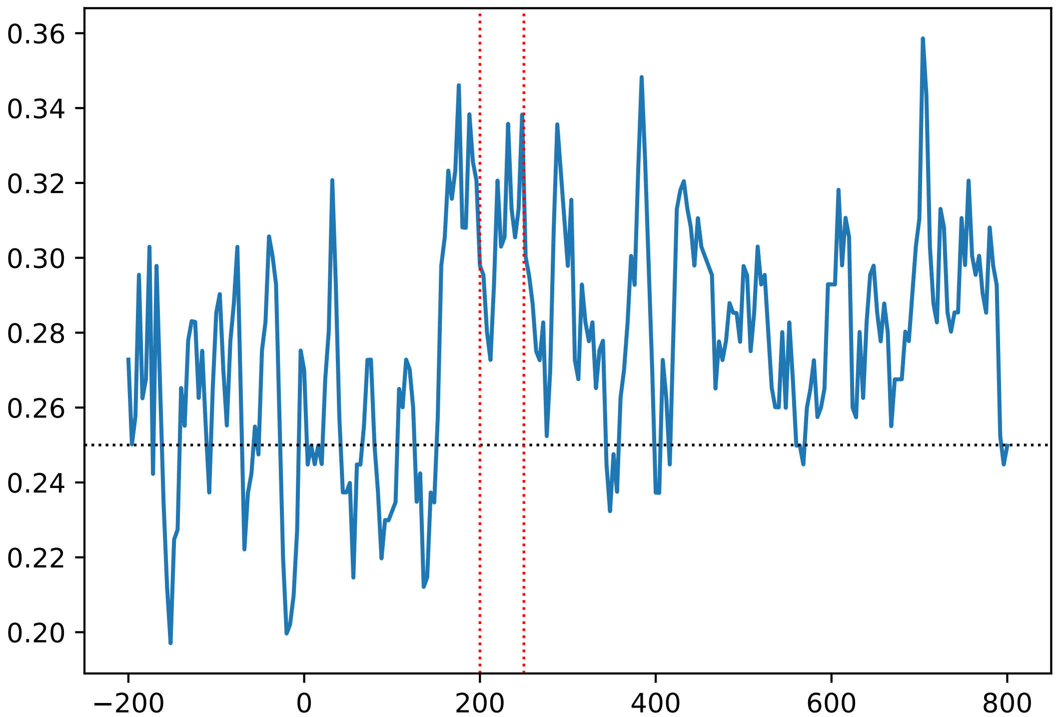
```
## 0.27275316455696197
```

```
plt.figure()
plt.plot(times, horo)
plt.axhline(y = .25, color='black', linewidth=1, linestyle=':')
plt.axvline(x=times[max_0], color='red', linewidth=1)
plt.show()
```



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

```
plt.figure()
plt.plot(times, horo)
plt.axhline(y = .25, color='black', linewidth=1, linestyle=':')
plt.axvline(x=200, color='red', linewidth=1, linestyle=':')
plt.axvline(x=250, color='red', linewidth=1, linestyle=':')
plt.show()
```



From visual inspection of the plot, we can observe that classification around 200-250 ms is above chance.

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`

```
from sklearn.model_selection import train_test_split as tts

X_train, X_test, y_train, y_test = tts(X_all_eq_rs, y_all_eq, test_size=0.3, random_state = 0)

X_train_scaled = scaledata.fit_transform(X_train)
X_test_scaled = scaledata.fit_transform(X_test)
```

```
i. Use the kernel that resulted in the best classification in Exercise 3.1.ii and `fit`the training set
```

```
train_score = cross_val_score(radial_classifier, X_train_scaled, y_train, cv=cv)
train_score_mean = np.mean(train_score)

radial_classifier.fit(X_train_scaled, y_train)
```

```
## SVC()
```

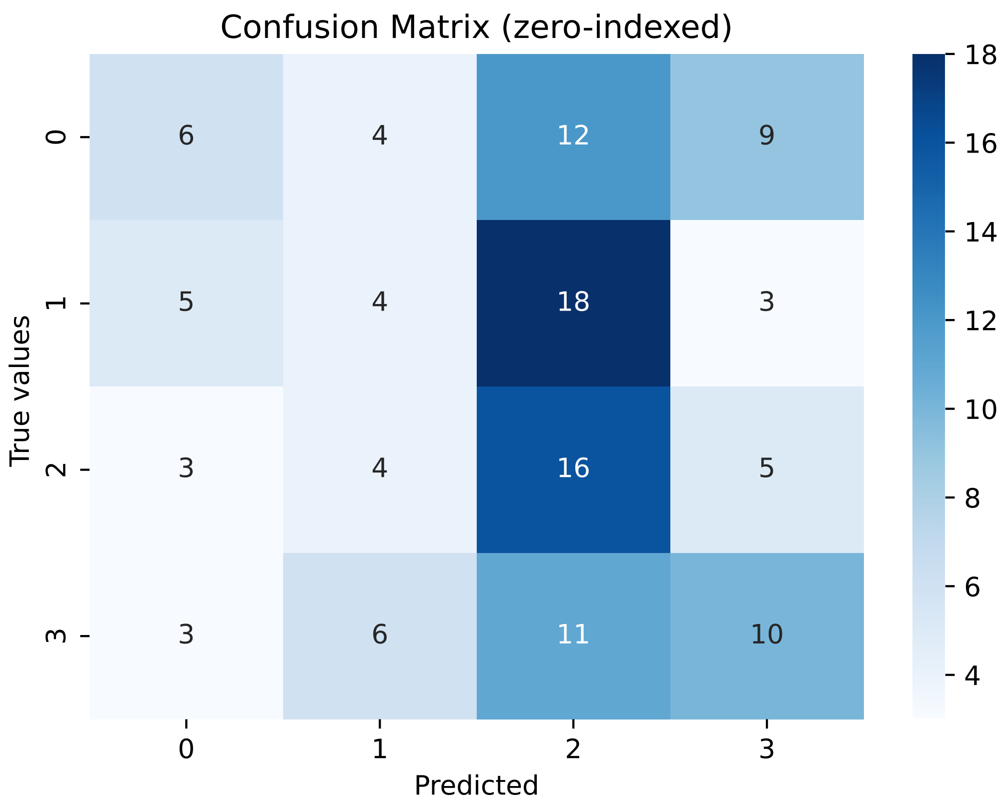
```
y_predict = radial_classifier.predict(X_test_scaled)
```

```
ii. Create a _confusion matrix_. It is a 4x4 matrix. The row names and the column names are the PAS-sco
```

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_true = y_test, y_pred = y_predict)
cm
```

```
## array([[ 6,  4, 12,  9],
##        [ 5,  4, 18,  3],
##        [ 3,  4, 16,  5],
##        [ 3,  6, 11, 10]], dtype=int64)
```

```
import seaborn as sns
plt.figure()
heatmap = sns.heatmap(cm, annot=True, cmap='Blues')
heatmap.set_title('Confusion Matrix (zero-indexed)');
heatmap.set_xlabel('Predicted')
heatmap.set_ylabel('True values ');
plt.show()
```



`iii. Based on the confusion matrix, describe how ratings are misclassified and if that makes sense given`

Based on the confusion matrix, our classifier is not doing a very good job. For PAS-1 and PAS-2, two other values are predicted more than the true value; for PAS-3 it predicts the majority correctly; for PAS-4, one other value is predicted more than the true value. Another problem is that for PAS-1, the misclassifications are not the "closest neighbour" to the true value, but instead it predicts them as PAS-3 and PAS-4. Finally, we can observe that the classifier is strongly biased towards predicting ratings of PAS-3, and even then its most frequent PAS-3 -predictions are actually ratings of PAS-2.