practical_exercise_10 , Methods 3, 2021, autumn semester

Linus Backström

8.12.2021

```
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.decomposition import PCA
```

Exercises and objectives

1) Use principal component analysis to improve the classification of subjective experience

2) Use logistic regression with cross-validation to find the optimal number of principal components

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 4 and will be part of your final portfolio

EXERCISE 1 - Use principal component analysis to improve the classification of subjective experience

We will use the same files as we did in Assignment 3 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)
The function `equalize_targets` is supplied - this time, we will only work with an equalized data set. One motivation for this is that we have a well-defined chance level that we can compare against. Furthermore, we will look at a single time point to decrease the dimensionality of the problem

1) Create a covariance matrix, find the eigenvectors and the eigenvalues
    i. 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

```
data = np.load('C:/Users/linus/Documents/GitHub/github_methods_3/week_10/megmag_data.npy')
y = np.load('C:/Users/linus/Documents/GitHub/github_methods_3/week_10/pas_vector.npy')
```

ii. Equalize the number of targets in `y` and `data` using `equalize_targets`

```
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]
```

iii. Construct `times=np.arange(-200, 804, 4)` and find the index corresponding to 248 ms - then reduce

```
import numpy as np

times = np.arange(-200, 804, 4)

times[112] # 248 ms
```

248

```
X_all_eq.shape  # (396, 102, 251)
```

(396, 102, 251)

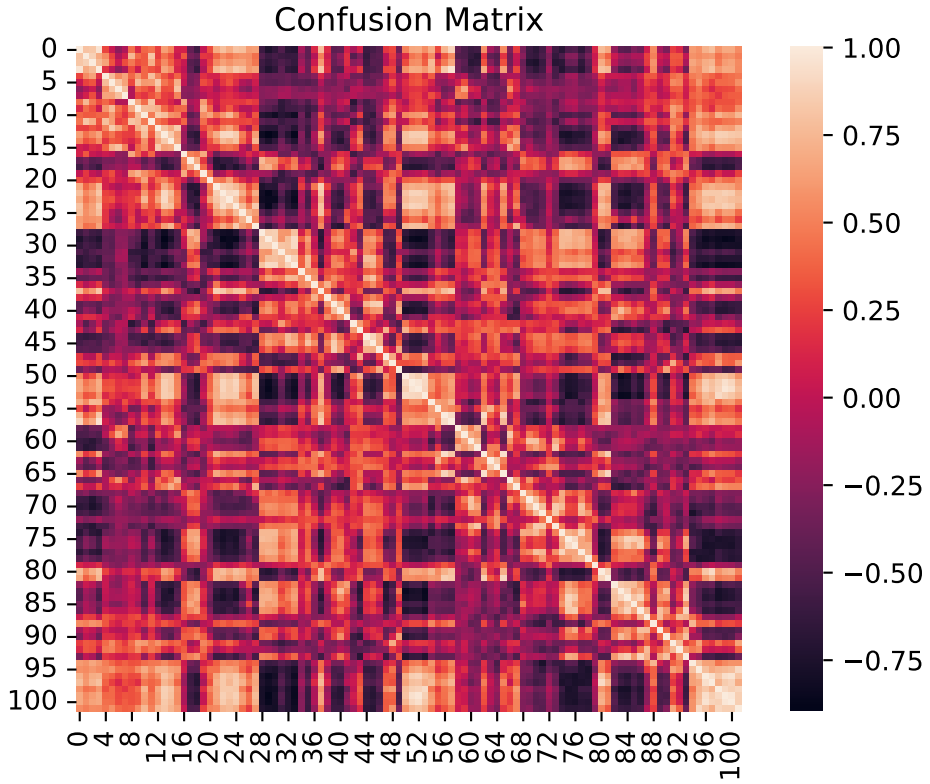
```
X_eq_rdc = X_all_eq[:, :, times[112]]
```

iv. Scale the data using `StandardScaler`

```
from sklearn.preprocessing import StandardScaler

scaledata = StandardScaler()

X_eq_rdc_sc = scaledata.fit_transform(X_eq_rdc)
```

v. Calculate the sample covariance matrix for the sensors (you can use `np.cov`) and plot it (either us

```
cov_mat = np.cov(X_eq_rdc_sc, rowvar = False) # rowvar because our rows and columns are a certain way

#We now plot the confusion matrix
import seaborn as sns
plt.figure()
heatmap = sns.heatmap(cov_mat, square=True)
heatmap.set_title('Confusion Matrix');
plt.show()
```



vi. What does the off-diagonal activation imply about the independence of the signals measured by the 1

It implies that the sensors are not completely independent of each other.

vii. Run `np.linalg.matrix_rank` on the covariance matrix - what integer value do you get? (we'll use t

```
np.linalg.matrix_rank(cov_mat) # 97, i.e. 5 sensors are completely linearly dependent of some other sen
```

```
## 97
```

viii. Find the eigenvalues and eigenvectors of the covariance matrix using `np.linalg.eig` - note that

```
eig_val, eig_vec = np.linalg.eig(cov_mat)

eig_val = np.real(eig_val)
eig_vec = np.real(eig_vec)
```

2) Create the weighting matrix W and the projected data, Z

   i. We need to sort the eigenvectors and eigenvalues according to the absolute values of the eigenvalues (use `np.abs` on the eigenvalues).

```
eig_val_abs = np.abs(eig_val)
```

ii. Then, we will find the correct ordering of the indices and create an array, e.g. `sorted_indices` t

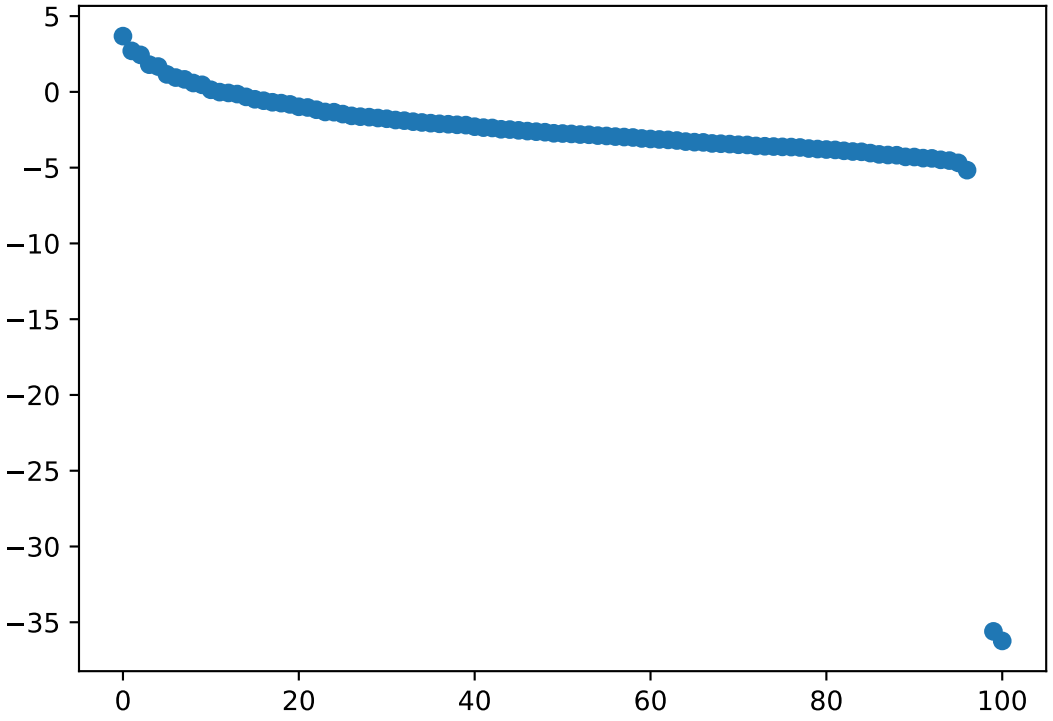
```
sorted_indices = np.argsort(eig_val_abs)
sorted_indices = np.flip(sorted_indices)
```

iii. Finally, create arrays of sorted eigenvalues and eigenvectors using the `sorted_indices` array jus

```
eigenvalues = eig_val[sorted_indices]
eigenvectors = eig_vec[:, sorted_indices]
```

iv. Plot the log, `np.log`, of the eigenvalues, `plt.plot(np.log(eigenvalues), 'o')` - are there some va

```
plt.figure()
plt.plot(np.log(eigenvalues), 'o')
plt.show()
```



Two values are much lower than the rest, signifying eigenvalues that are very close to zero, which results in a negative value of around -35 for the log of the eigenvalues.

v. Create the weighting matrix, `W` (it is the sorted eigenvectors)

```
W = eigenvectors
```

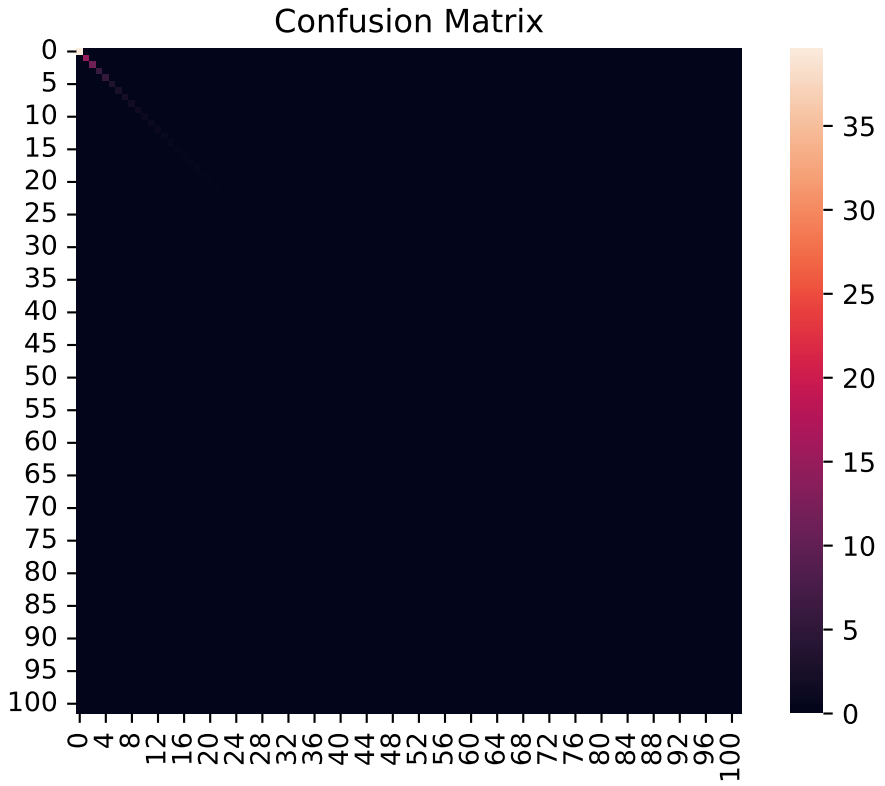
vi. Create the projected data, `Z`, $Z = XW$ - (you can check you did everything right by checking whetl

```
Z = X_eq_rdc_sc @ W

np.isclose(Z @ W.T, X_eq_rdc_sc) # it matches!
```

```
## array([[ True,  True,  True, ...,  True,  True,  True],
##        [ True,  True,  True, ...,  True,  True,  True],
##        [ True,  True,  True, ...,  True,  True,  True],
##        ...,
##        [ True,  True,  True, ...,  True,  True,  True],
##        [ True,  True,  True, ...,  True,  True,  True],
##        [ True,  True,  True, ...,  True,  True,  True]])
```

vii. Create a new covariance matrix of the principal components (n=102) - plot it! What has happened of

```
cov_mat2 = np.cov(Z, rowvar = False) # rowvar because our rows and columns are a certain way

#We now plot the confusion matrix
import seaborn as sns
plt.figure()
heatmap = sns.heatmap(cov_mat2, square=True)
heatmap.set_title('Confusion Matrix');
plt.show()
```



There appears to be no collinearity on the off-diagonal.

EXERCISE 2 - Use logistic regression with cross-validation to find the optimal number of principal components

1) We are going to run logistic regression with in-sample validation
    i. First, run standard logistic regression (no regularization) based on $Z_{d\times k}$ and `y` (the target vector). Fit (`.fit`) 102 models based on: $k = [1, 2, ..., 101, 102]$ and $d = 102$. For each fit get the classification accuracy, (`.score`), when applied to $Z_{d\times k}$ and y. This is an in-sample validation. Use the solver `newton-cg` if the default solver doesn't converge

```
from sklearn.linear_model import LogisticRegression

Z.shape # 396, 102
```

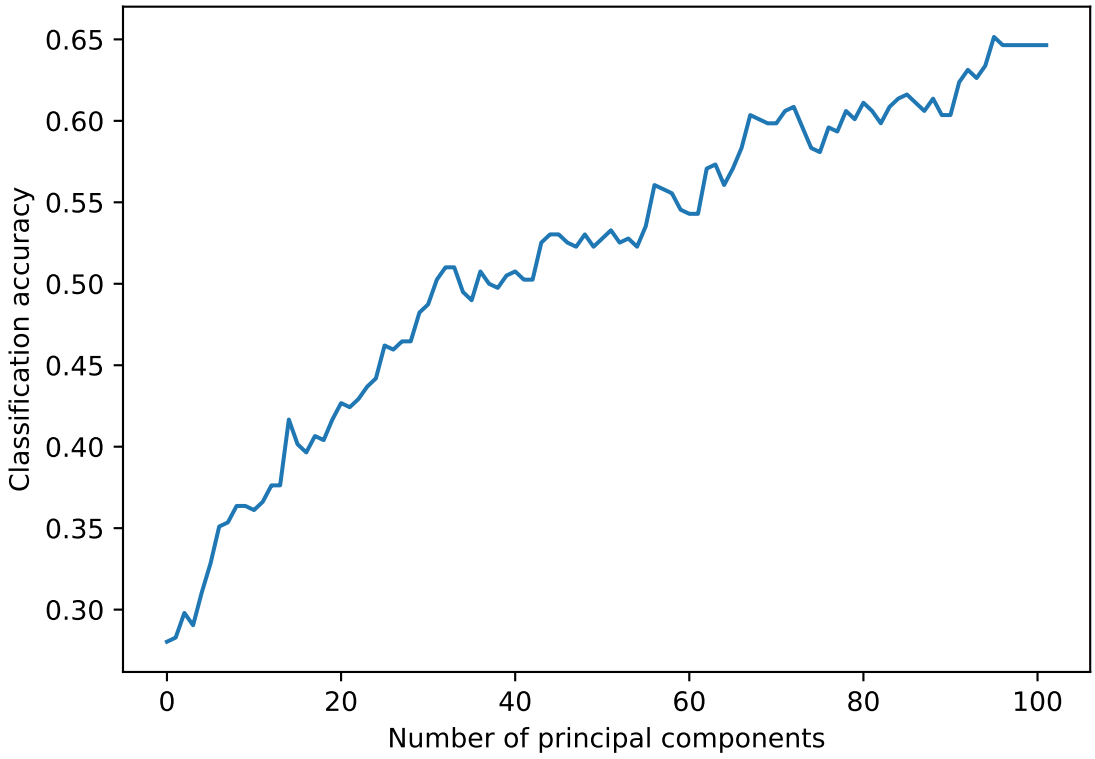
```
## (396, 102)
```

```
log_reg = LogisticRegression(penalty='none', solver='newton-cg')

scores = np.zeros(shape=(102))

for i in range(102):
    if i == 0:
        X = Z[:, 0]
        X = X.reshape(-1, 1)
    elif i == 102:
        X = Z[:, 0]
    else:
        X = Z[:, 0:i+1]
    log_model = log_reg.fit(X, y_all_eq)
    score = log_model.score(X, y_all_eq)
    scores[i] = score

scores
```

```
## array([0.28030303, 0.28282828, 0.2979798 , 0.29040404, 0.31060606,
##        0.32828283, 0.3510101 , 0.35353535, 0.36363636, 0.36363636,
##        0.36111111, 0.36616162, 0.37626263, 0.37626263, 0.41666667,
##        0.40151515, 0.39646465, 0.40656566, 0.4040404 , 0.41666667,
##        0.42676768, 0.42424242, 0.42929293, 0.43686869, 0.44191919,
##        0.46212121, 0.45959596, 0.46464646, 0.46464646, 0.48232323,
##        0.48737374, 0.50252525, 0.51010101, 0.51010101, 0.49494949,
##        0.48989899, 0.50757576, 0.5       , 0.49747475, 0.50505051,
##        0.50757576, 0.50252525, 0.50252525, 0.52525253, 0.53030303,
##        0.53030303, 0.52525253, 0.52272727, 0.53030303, 0.52272727,
##        0.52777778, 0.53282828, 0.52525253, 0.52777778, 0.52272727,
##        0.53535354, 0.56060606, 0.55808081, 0.55555556, 0.54545455,
##        0.54292929, 0.54292929, 0.57070707, 0.57323232, 0.56060606,
##        0.57070707, 0.58333333, 0.60353535, 0.6010101 , 0.59848485,
##        0.59848485, 0.60606061, 0.60858586, 0.5959596 , 0.58333333,
##        0.58080808, 0.5959596 , 0.59343434, 0.60606061, 0.6010101 ,
##        0.61111111, 0.60606061, 0.59848485, 0.60858586, 0.61363636,
##        0.61616162, 0.61111111, 0.60606061, 0.61363636, 0.60353535,
##        0.60353535, 0.62373737, 0.63131313, 0.62626263, 0.63383838,
##        0.65151515, 0.64646465, 0.64646465, 0.64646465, 0.64646465,
##        0.64646465, 0.64646465])
```

ii. Make a plot with the number of principal components on the _x_-axis and classification accuracy on

```
plt.figure()
plt.plot(scores)
plt.xlabel('Number of principal components')
plt.ylabel('Classification accuracy')
plt.show()
```



We can observe that the number of principal components increases classification accuracy, with diminishing returns. More principal components means more information to base the classification upon. Diminishing returns occurs because we have ordered the principal components so that the most impactful ones are first.

`iii. In terms of classification accuracy, what is the effect of adding the five last components? Why do`

The last five components do not increase the accuracy at all, because they are dependent on other components and therefore do not add extra information to help with the classification.

2) Now, we are going to use cross-validation - we are using `cross_val_score` and `StratifiedKFold` from `sklearn.model_selection`

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

`i. Define the variable: `cv = StratifiedKFold()` and run `cross_val_score` (remember to set the `cv` ar`

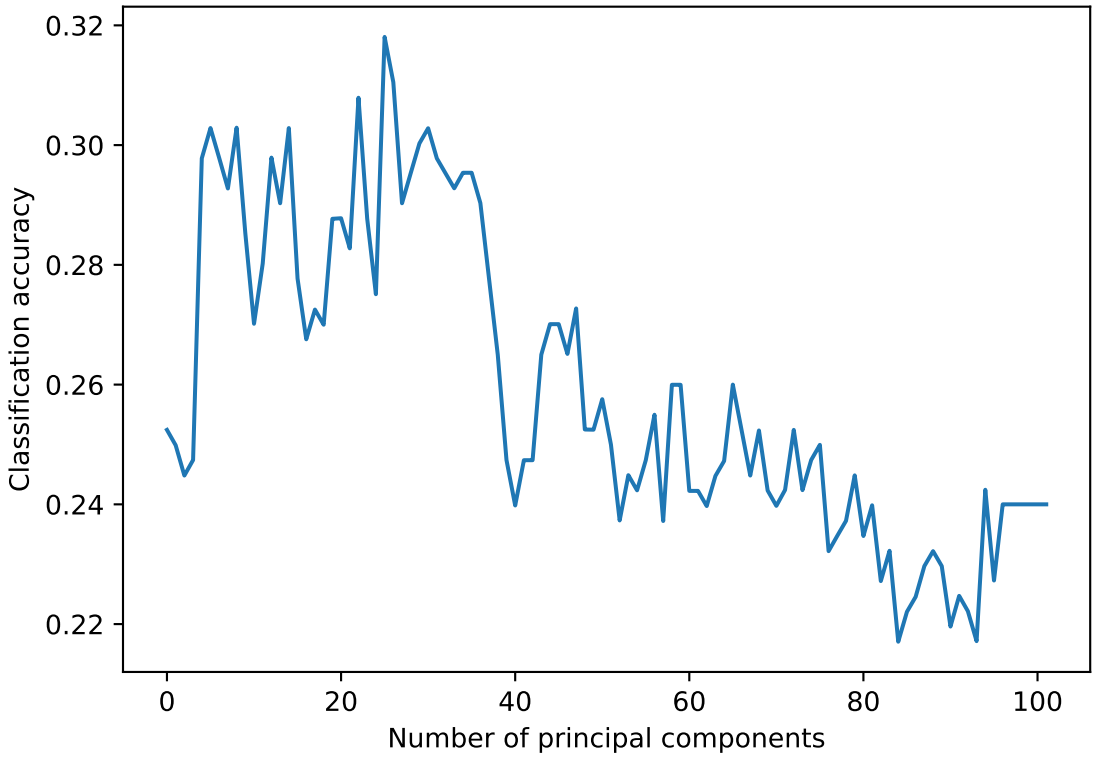
```
log_reg = LogisticRegression(penalty='none', solver='newton-cg')

cv = StratifiedKFold(n_splits = 5)

scores = np.zeros(shape=(102))

for i in range(102):
    if i == 0:
        X = Z[:, 0]
        X = X.reshape(-1, 1)
    elif i == 102:
        X = Z[:, 0]
```

```
    else:
        X = Z[:, 0:i+1]
    log_model = log_reg.fit(X, y_all_eq)
    score = cross_val_score(log_model, X, y_all_eq, cv=cv)
    mean = np.mean(score)
    scores[i] = mean

scores
```

```
## array([0.25243671, 0.24987342, 0.24481013, 0.24740506, 0.29778481,
##        0.3028481 , 0.2978481 , 0.29275316, 0.30291139, 0.28528481,
##        0.27015823, 0.28022152, 0.29791139, 0.29028481, 0.3028481 ,
##        0.27772152, 0.26756329, 0.27253165, 0.27      , 0.28768987,
##        0.28778481, 0.28275316, 0.30791139, 0.28772152, 0.27509494,
##        0.31806962, 0.31050633, 0.29028481, 0.29531646, 0.30028481,
##        0.30281646, 0.29778481, 0.29528481, 0.29278481, 0.29537975,
##        0.29537975, 0.29031646, 0.27762658, 0.26496835, 0.24740506,
##        0.23981013, 0.24737342, 0.24737342, 0.265     , 0.27009494,
##        0.27009494, 0.26512658, 0.27272152, 0.2525    , 0.25246835,
##        0.25756329, 0.24996835, 0.23731013, 0.24487342, 0.24234177,
##        0.24737342, 0.25496835, 0.23721519, 0.25996835, 0.25996835,
##        0.24224684, 0.24224684, 0.23971519, 0.24474684, 0.24724684,
##        0.26      , 0.25243671, 0.24481013, 0.25234177, 0.24231013,
##        0.23974684, 0.24240506, 0.25243671, 0.24237342, 0.24740506,
##        0.24993671, 0.23218354, 0.23474684, 0.23724684, 0.24484177,
##        0.23471519, 0.23984177, 0.22718354, 0.23224684, 0.21705696,
##        0.22205696, 0.22455696, 0.22968354, 0.23218354, 0.22968354,
##        0.21958861, 0.22471519, 0.2221519 , 0.2171519 , 0.24243671,
##        0.22727848, 0.24      , 0.24      , 0.24      , 0.24      ,
##        0.24      , 0.24      ])
```

ii. Make a plot with the number of principal components on the _x_-axis and classification accuracy on

```
plt.figure()
plt.plot(scores)
plt.xlabel('Number of principal components')
plt.ylabel('Classification accuracy')
plt.show()
```



The general trend is downwards.

iii. What is the number of principal components, $k_{max\_accuracy}$, that results in the greatest class

```
np.argmax(scores) # 25
```

```
## 25
```

iv. How many percentage points is the classification accuracy increased with relative to the to the full

```
scores[25]
```

```
## 0.31806962025316454
```

```
log_reg = LogisticRegression(penalty='none', solver='newton-cg')
#cv = StratifiedKFold(n_splits = 5) REMOVE THIS
fit_all = log_reg.fit(Z, y_all_eq)
mean_all = np.mean(score)

improvement = (scores[25])/ mean_all
improvement
```

```
## 1.3252900843881854
```

v. How do the analyses in Exercises 2.1 and 2.2 differ from one another? Make sure to comment on the di

3) We now make the assumption that $k_{max\_accuracy}$ is representative for each time sample (we only tested for 248 ms). We will use the PCA implementation from *scikit-learn*, i.e. import `PCA` from `sklearn.decomposition`.
   i. For **each** of the 251 time samples, use the same estimator and cross-validation as in Exercises

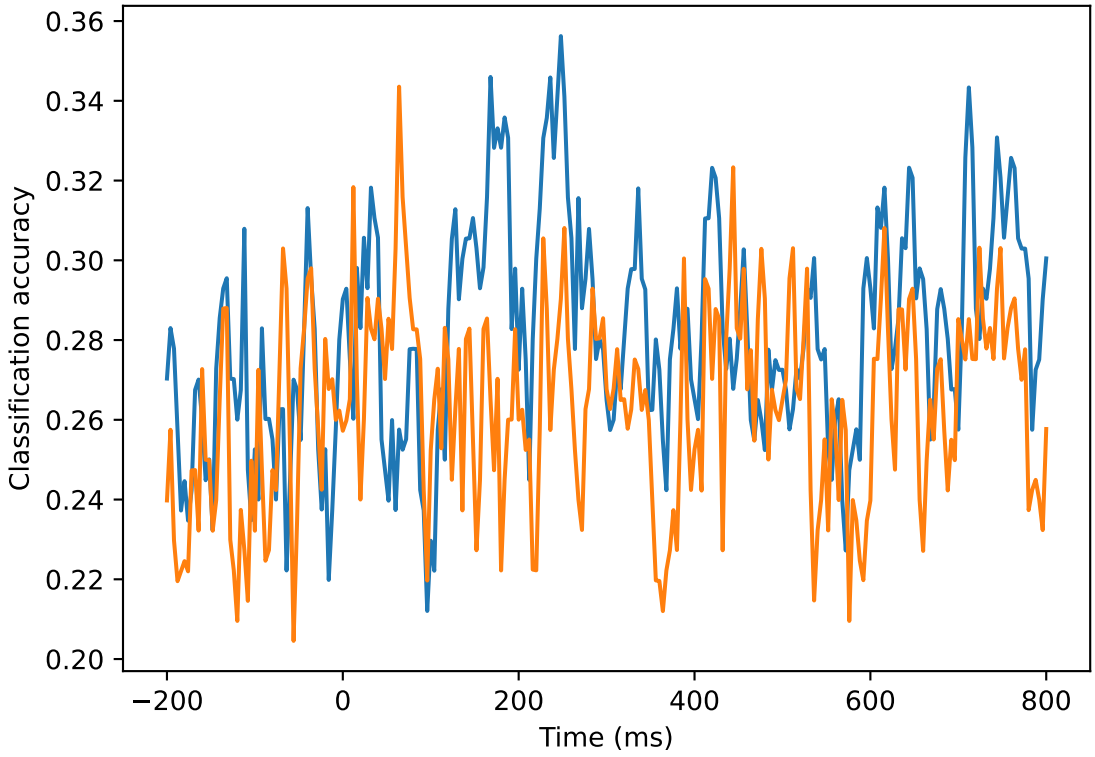
2.1.i and 2.2.i. Run two analyses - one where you reduce the dimensionality to $k_{max\_accuracy}$ dimensions using `PCA` and one where you use the full data. Remember to scale the data (for now, ignore if you get some convergence warnings - you can try to increase the number of iterations, but this is not obligatory)

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 25)
scores_max = np.zeros(shape=(251))
log_reg = LogisticRegression(penalty='none', solver='newton-cg')
cv = StratifiedKFold(n_splits = 5)
sc = StandardScaler()

for i in range(251):
    X_time = X_all_eq[:, :, i]
    X_time_std = sc.fit_transform(X_time)
    X_pca = pca.fit_transform(X_time_std)
    log_model = log_reg.fit(X_pca, y_all_eq)
    score = cross_val_score(log_model, X_pca, y_all_eq, cv=cv)
    mean = np.mean(score)
    scores_max[i] = mean

scores_all = np.zeros(shape=(251))
log_reg = LogisticRegression(penalty='none', solver='newton-cg')
cv = StratifiedKFold(n_splits = 5)
sc = StandardScaler()

for i in range(251):
    X_time = X_all_eq[:, :, i]
    X_time_std = sc.fit_transform(X_time)
    log_model = log_reg.fit(X_time_std, y_all_eq)
    score = cross_val_score(log_model, X_time_std, y_all_eq, cv=cv)
    mean = np.mean(score)
    scores_all[i] = mean
```

```
## C:\Users\linus\MINICO~1\envs\methods3\lib\site-packages\sklearn\utils\optimize.py:210: ConvergenceWa
##   warnings.warn(
## C:\Users\linus\MINICO~1\envs\methods3\lib\site-packages\sklearn\utils\optimize.py:210: ConvergenceWa
##   warnings.warn(
## C:\Users\linus\MINICO~1\envs\methods3\lib\site-packages\sklearn\utils\optimize.py:210: ConvergenceWa
##   warnings.warn(
## C:\Users\linus\MINICO~1\envs\methods3\lib\site-packages\sklearn\utils\optimize.py:210: ConvergenceWa
##   warnings.warn(
## C:\Users\linus\MINICO~1\envs\methods3\lib\site-packages\sklearn\utils\optimize.py:210: ConvergenceWa
##   warnings.warn(
## C:\Users\linus\MINICO~1\envs\methods3\lib\site-packages\sklearn\utils\optimize.py:210: ConvergenceWa
##   warnings.warn(
```

```
ii. Plot the classification accuracies for each time sample for the analysis with PCA and for the one w
```

```
plt.figure()
plt.plot(times, scores_max)
plt.plot(times, scores_all)
plt.xlabel('Time (ms)')
plt.ylabel('Classification accuracy')
plt.show()
```



iii. Describe the differences between the two analyses - focus on the time interval between 0 ms and 40